Comparing Synchronous and Asynchronous Cellular Genetic Algorithms

Enrique Alba[1], Mario Giacobini[2], Marco Tomassini[2], and Sergio Romero

[1] Department of Lenguajes y CC.CC., University of Málaga, Málaga, Spain
[2] Computer Science Institute, University of Lausanne, Lausanne, Switzerland

`eat@lcc.uma.es, Mario.Giacobini/Marco.Tomassini@iis.unil.ch`

Abstract. This paper presents a comparative study of several asynchronous policies for updating the population in a cellular genetic algorithm (cGA). Cellular GA's are regular GA's with the important exception that individuals are placed in a given geographical distribution (usually a 2-d grid). Operators are applied locally on a set made of each individual and the surrounding neighbors, thus promoting intra-neighborhood exploitation and inter-neighborhood exploration of the search space. Here, we analyze the respective advantages and drawbacks of dealing with this decentralized population in the traditional synchronous manner or in several possible asynchronous update policies. Asynchronous behavior has proven to be better in many domains such as cellular automata and distributed GA's, which, in turn, is also the main conclusion of this work. We will undergo a structured analysis on a set of problems with different features in order to get well grounded conclusions.

1 Introduction

Cellular evolutionary algorithms (cEA) models, also called *diffusion* or *fine-grained* models, are based on a spatially distributed population in which genetic interactions may only take place in a small neighborhood of each individual. Individuals are usually disposed on a regular grid of dimensions $d = 1, 2$ or 3. Cellular evolutionary algorithms were popularized by early work of Gorges-Schleuter [5], and by Manderick and Spiessen [9]. However, we here want to stress the difference between the model and its implementation, and this is why we call them *cellular* and not fine-grained EA's. Cellular EA's are just a new kind of algorithm, and not a parallel implementation on massively parallel machines.

Although fundamental theory is still an open research line for cEA's, they have been empirically reported as being useful in maintaining diversity and promoting slow diffusion of solutions through the grid. Part of their behavior is due to a lower selection pressure compared to that of panmictic EA's. The influence of the neighborhood, grid topology, and grid size/shape on the induced selection pressure has been investigated in detail in [1, 6, 11, 12] (and tested on different applications such as combinatorial and numerical optimization).

Let us analyze a typical cGA, an important kind of cEA. The cGA iteratively considers groups of individuals belonging to the same neighborhood to work with. In a North-East-West-South (NEWS or Von Newmann) neighborhood type, the central individual plus its 4 neighbors make up a small pool to apply operators on. The cGA iterates through the population in various generations. In each generation, it considers as a central string every individual in the population. Since a string belongs to several neighborhoods, a change in its contents affects its neighbors in a smooth manner, representing a good tradeoff between slow convergence and good exploration of the search space. In a synchronous cGA, we compute the full new generation incrementally onto a temporary population, and them replace the full old population with the new one.

Synchronous Cellular Genetic Algorithm (cGA)

```
proc Reproductive_Cycle (ga):
    for s=1 to MAX_STEPS do
        for x=1 to WIDTH do
            for y=1 to HEIGHT do
                n_list  = Calculate_neigbors  (ga, position (x,y) );
                parent1 = Select (n_list);
                parent2 = Select (n_list);
                Crossover(ga.Pc, n_list[parent1], n_list[parent2], ind_aux.chrom);
                Mutate(ga.Pm, ind_aux.chrom);
                ind_aux.fitness = ga.Evaluate ( Decode ( ind_aux.chrom) ) ;
                Insert_New_Ind(position(x,y),ind_aux,[if better | always], ga, pop_aux);
            end_for;
        end_for;
        ga.pop=pop_aux;
        Collect_Statistics (ga);
    end_for;
end_proc Reproductive_Cycle;
```

Cellular EA's can also be seen as stochastic cellular automata (CA's) [15, 16] where the cardinality of the set of states is equal to the number of points in the search space. CA's, as well as cEA's, usually assume a *synchronous* or parallel update policy, in which *all the cells* are formally updated simultaneously. However, this is not the only option available. Indeed, several works on *asynchronous* CA's have shown that sequential update policies have a marked effect on their dynamics (see e.g. [7, 13, 14]). While asynchronous updating is physically more realistic for CA's due to their finite signal propagation speed, this is not an issue for cEA, unless they are implemented on an actual massively parallel cellular machine, which is seldom the case in practice. However, it would be interesting to investigate asynchronous cEA's and their problem solving capabilities. To our knowledge, the present paper is the first step in that direction. We will thus present a few asynchronous update policies for a cEA, and compare them with the customary synchronous updating on a set of test functions.

The paper is structured as follows. The next section contains some background on asynchronous cEA's and explains the techniques we are considering. Section 3 describes the test problems used, while section 4 gives details on the cEA parameters employed in the simulations, the performance measures, and the statistics used. Section 5 contains a discussion of the experimental results, and section 6 offers our conclusions, as well as some comments on the future work.

2 Asynchronous Cellular Evolutionary Algorithms

There are many ways for sequentially updating the cells of a cEA with a population on a 2-d grid (see an excellent discussion of asynchronous CA's in [13]). The most general one is independent random ordering of updates in time, which consists in randomly choosing the cell to be updated next, with replacement. This corresponds to a binomial distribution for the update probability. The limiting case of such distribution for large n is the Poisson distribution (where n is the number of cells, or individuals, in the grid). This update policy will be called *uniform choice* (UC) in the following.

For comparison purposes we also consider three other update methods: *fixed line sweep*, *fixed random sweep* and *random new sweep* (we employ the same terms as in [13]).

– In *fixed line sweep* (LS), the simplest method, grid cells are updated sequentially $(1, 2 \ldots n)$, line by line of the 2-d grid.
– In the *fixed random sweep* update (FRS), the next cell to be updated is chosen with uniform probability without replacement; this will produce a certain update sequence $(c_1^j, c_2^k, \ldots, c_n^m)$, where c_q^p means that cell number p is updated at time q and $(j, k, \ldots, m)$ is a permutation of the n cells. The same permutation is then used for the following update cycles.
– The *new random sweep* method (NRS) works like FRS, except that a random new cell permutation is chosen anew for each sweep through the array.

A *time step* is defined to be the action of updating n times, which corresponds to updating *all* the n cells in the grid for LS, FRS and NRS, and possibly less than n different cells in the uniform choice method, since some cells might be updated more than once. It should be noted that, with the exception of fixed line sweep, the other asynchronous updating policies are non-deterministic, representing an additional source of non-determinism besides that of the genetic operators. An asynchronous parallel implementation could be easily derived for these algorithms, although we do not explore physical parallelism in this work.

3 Description of the Test Problems

To test the differences between the synchronous and the four asynchronous update models we have decided to use problems representing three large classes of difficulty with interest in evolutionary computation, namely: deception, multimodality, and epistasis. Similarly to the work of Alba and Troya [1], the choice has been driven to the Massive Multimodal Deceptive Problem (MMDP), the Frequency Modulation Sounds Problem (FMS), and the P-PEAKS multimodal generator.

The MMDP has been specifically designed by Goldberg *et al.* [4] to be difficult for an EA. It is made up of k subproblems of 6 bits each (see equation 1). The optimum has a value of k and is attained when the *unitation* of each subproblem, i.e. the number of ones of its 6-bits defining string, is 0 or 6. Thus every subproblem $x_i =< x_{i_1}, ..., x_{i_6} >$ $(i = 1, ..., k)$ contributes to the fitness of a possible solution $\overrightarrow{x} =< x_1...x_k >$ according to its *unitation*:

$$f_{MMDP}(\overrightarrow{x}) = \sum_{i=1}^{k} g(unitation(x_i)), \tag{1}$$

where g is such that $g(0) = g(6) = 1$, $g(1) = g(5) = 0$, $g(2) = g(4) = 0.36$, $g(3) = 0.64$. Such function has a quite large number of local optima (22^k), while only 2^k are global solutions, and therefore its degree of multimodality is defined by k. Here we set $k = 40$, obtaining a considerably large degree of multimodality.

Proposed by Tsutsui *et al.* [10], the Frequency Modulation Sounds parameter identification problem (FMS) consists in adjusting a general model $y(t)$ (equation 2) to a basic sound function $y_0(t)$ (equation 3). The problem is to evolve a solution $\overrightarrow{x}$ consisting in 6 real parameters ($\overrightarrow{x} =< a_1, w_1, a_2, w_2, a_3, w_3 >$) each one encoded with 32 bits in the range $[-6.4, 6.35]$, in order $y(t)$ to fit the target function $y_0(t)$.

$$y(t) = a_1 \sin(w_1 t\theta + a_2 \sin(w_2 t\theta + a_3 \sin(w_3 t\theta))), \tag{2}$$

$$y_0(t) = 1.0 \sin(5.0 t\theta + 1.5 \sin(4.8 t\theta + 2.0 \sin(4.9 t\theta))). \tag{3}$$

The goal is, therefore, to minimize the sum of square errors (equation 4):

$$f_{FMS}(\overrightarrow{x}) = \sum_{t=0}^{100} (y(t) - y_0(t))^2. \tag{4}$$

The resulting problem is a complex multimodal function having strong epistasis with minimum value in $\overrightarrow{z}$, where $f(\overrightarrow{z}) = 0$. For our calculations, we consider the algorithm having found an optimum when the error falls below 10^{-2}.

The last optimization task solved in this paper is a problem generator proposed by De Jong *et al.* [8]. This problem generator is an easily parameterizable task which has a tunable degree of epistasis, thus allowing to derive instances with growing difficulty at will. With a problem generator we evaluate our algorithms on a high number of random problem instances, thus increasing the predictive power of the results for the problem class as a whole. Such a characteristic allows a larger fairness when comparing algorithms, since it implicitly removes the opportunity to hand-tune algorithms to a particular problem. In this paper we use the multimodal generator called P-PEAKS (see equation 5).

$$f_{P-PEAKS}(\overrightarrow{x}) = \frac{1}{N} max_{i=1}^{p} \{N - HammingD(\overrightarrow{x}, Peak_i)\} \tag{5}$$

The idea is to generate P strings, each of N random bits, that represent the location of the global optima in the search space. The fitness of a possible solution (a bit string of length N) is the number of bits it has in common with the nearest peak in the Hamming space, divided by the length N of the strings. Problems with a small/large number of peaks are weakly/strongly epistatic. The instance we use has $P = 100$ peaks of $N = 100$ bits each, which represents a medium-high epistasis level.

4 Parameters and Statistics Used

As we said in the introduction, many authors have already investigated the influence of the neighborhood, grid topology and grid dimensions on the induced selection pressure [1, 6, 11, 12]. Reduced grid dimensions of 20×20 have been used to design efficient cGA's [1], but, most of the time, larger grids are preferred for the analysis. For the three problems described in section 3 we have investigated four different grid sizes, so as to choose the dimension that, for all the problems, guarantees significant success rate of the five update methods. Each grid size has been tested 50 times for each update method.

MMDP	**Synchronous**	**Line Sweep**	**Fixed Random Sweep**	**New Random Sweep**	**Uniform Choice**
20×20	0%	0%	0%	0%	0%
32×32	74%	32%	38%	44%	56%
40×40	98%	86%	92%	84%	94%
50×50	100%	98%	100%	100%	100%

Table 1. Success rate for the Massive Multimodal Deceptive Problem of the synchronous and the four asynchronous update methods (horizontally) with different grid dimensions (vertically).

All the results are summarized in tables 1, 2 and 3 and were obtained with the same internal parameters. For the parent's selection we have used a roulette wheel operator among the five individuals in the von Neumann neighborhood (the central plus the four neighbors in the North, East, West and South positions). A two point crossover is applied to the two selected parents with probability 1.0, thus producing an offspring individual (the one with the largest proportion of the best parent). Such a new solution is then mutated with different mutation probabilities p_m for the three problems. The obtained offspring will replace the considered individual only if it has a better fitness. We stop the algorithm when a global optimum is found, and we analyze the cost of a successful run of a cEA by measuring the number of evaluations done. In the MMDP, an individual is encoded by a $40 \times 6 = 240$ bit string, in the FMS problem by a $6 \times 32 = 192$ binary chromosome, and in the P-PEAKS problem by a binary vector of length 100. The problems' differences and the different chromosomal lengths determine three mutation probabilities: for the MMDP and the P-PEAKS problem we set p_m to $1/L$, having respectively $p_m = 0.0042$ and $p_m = 0.01$, while for the FMS problem p_m is set to $10/L$, i.e. $p_m = 0.052$.

FMS	Synchronous	Line Sweep	Fixed Random Sweep	New Random Sweep	Uniform Choice
20×20	0%	0%	0%	2%	0%
32×32	4%	0%	2%	0%	2%
40×40	8%	8%	6%	8%	6%
50×50	24%	22%	12%	12%	12%

Table 2. Success rate for the Frequency Modulation Sounds problem of the synchronous and the four asynchronous update methods (horizontally) with different grid dimensions (vertically).

P-PEAKS	Synchronous	Line Sweep	Fixed Random Sweep	New Random Sweep	Uniform Choice
20×20	52%	26%	36%	34%	36%
32×32	100%	100%	100%	100%	100%
40×40	100%	100%	100%	100%	100%
50×50	100%	100%	100%	100%	100%

Table 3. Success rate for the P-PEAKS problem of the synchronous and the four asynchronous update methods (horizontally) with different grid dimensions (vertically).

Capcarrère *et al.* defined a number of statistical measures that are useful for understanding the dynamical behavior of cellular evolutionary algorithms. Two kinds of statistics were used: *genotypic* and *phenotypic*. Genotypic measures embody aspects related to the genotypes of individuals in a population. Phenotypic statistics concern properties of individual performance, essentially fitness (see [3] for the exact definitions). Here, we use the variance and the mean fitness as phenotypic statistics, and the entropy as a statistics pertaining to the genotype (a phenotypic diversity index based on fitness entropy can also be defined but it will not be used here). Differently from the paper of Capcarrère *et al.*, we calculate the entropy in the interval $[0, 1]$, instead of in the interval $[0, \log(N)]$ (where N is the population size). Such a result is obtained setting the multiplicative constant in the entropy formula to $1/\log(N)$ instead of using 1.

5 Experimental Results

In order to have comparable results for all the three problems, we have decided to set the population to the 50×50 grid size (non-square grids analyzed in [1]). The results, for all the runs, are summarized in table 4: for each update method the average number of evaluations needed to solve the three problems is shown.

For the MMDP and the P-PEAKS problem, where success rate is 100%, we can see that the synchronous update method is more expensive than every asynchronous method. We can therefore deduce that for multimodal, deceptive (MMDP) and highly epistatic (P-PEAKS) problems it should be more efficient to implement an asynchronous update method rather than a synchronous one.

	Synchronous	Line Sweep	Fixed Random Sweep	New Random Sweep	Uniform Choice
MMDP	277950	201887	216300	217850	238850
FMS	560760	543928	427291	480500	534772
P-PEAKS	243300	189550	191700	201400	203350

Table 4. Mean number of evaluations for the three problems (vertically) and the five update methods (horizontally).

The behavior of these two problems is slightly different for each asynchronous updating policy. For MMDP and P-PEAKS the Line Sweep method is the fastest policy, and the New Random Sweep is faster than Uniform Choice. The Fixed Random Sweep policy has a convergence speed similar to the Line Sweep for the highly epistatic problem, and to the New Random Sweep for the multimodal and deceptive problems. Such a different speed can be seen in figure 1, where a sample curve is drawn for each update method, both for MMDP (a) and for P-PEAKS (b).

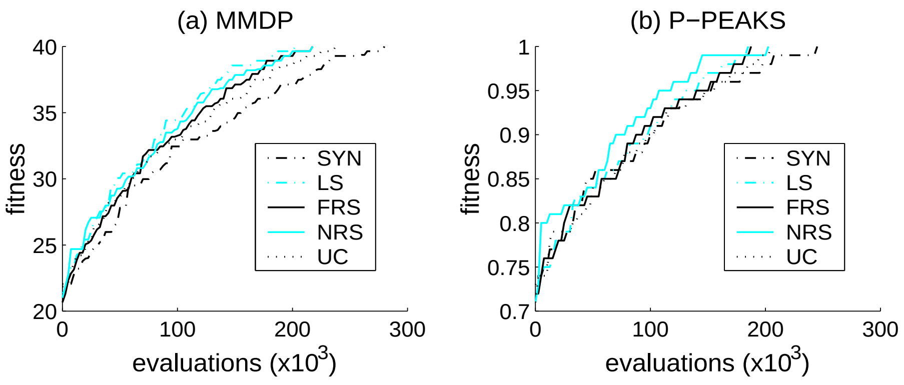

Fig. 1. A sample run of each update method for the MMDP (a), and for the P-PEAKS problem (b).

This difference in the speed of convergence is followed by a consistent behavior of the entropy curves (see figure 2): the faster the convergence speed is, the lower the entropy is. Such a result confirms the intuitive idea that the genotypic diversity decreases proportionally to the convergence speed, i.e. the faster a cEA is, the bigger the upward thrust of the populations is. The variance and the standard deviation values are consistent with the described behaviors.

For the FMS problem the comparison between the convergence speeds of the different update methods must be coupled with their different success rates. In fact, as it can be seen comparing tables 2 and 4, the synchronous and the Line

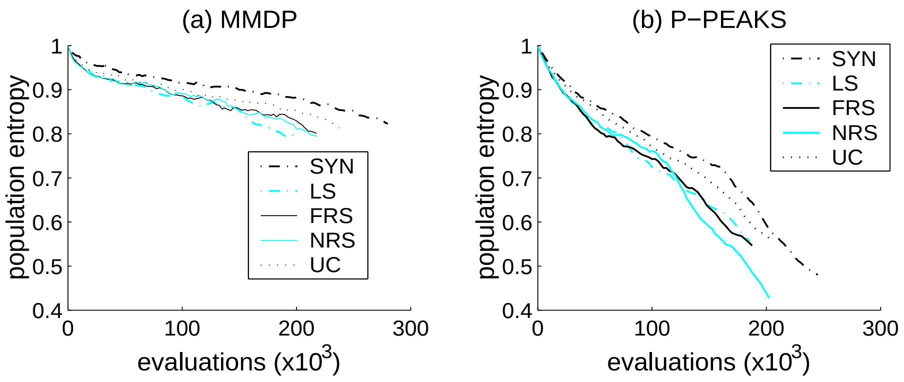

Fig. 2. A sample curve of the entropy of each update method for the MMDP (a) and for the P-PEAKS problem (b).

Sweep methods are slower than the Fixed Random Sweep, the New Random Sweep and the Uniform Choice methods (see also figure 3), but their success rate are twice the percentage of the other three asynchronous update methods. So, Line Sweep seems a good tradeoff between speed and accuracy, at least for problems similar to FMS. FMS showed to be the most difficult problem on which our cEA's have been tested in this study, due to its huge and complex search space. The entropy at the end (not shown here) is always very high, in the interval $[0.9, 0.92]$ for every algorithm.

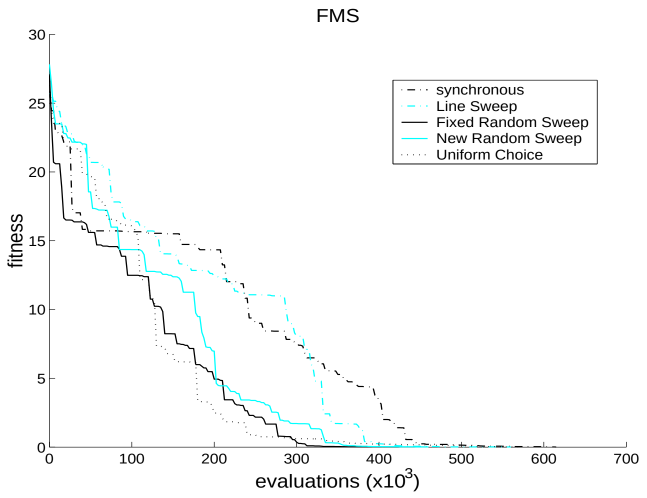

Fig. 3. A sample run of each update method for the FMS problem.

We have seen that, contrary to the behavior of cellular automata (CA's), the simple Line Sweep policy performs better than the rest in cEA's. Such a result can be explained by the fact that, in cEA's, we don't have a propagation of signals like in CA's, having instead a propagation of information on the solution of the problem. It is true that Line Sweep fixes a preferred direction of propagation in the axes of the grid, but such an order speeds the propagation of information in the population. In fact, if we take our 50×50 toroidal population grids with the chosen von Neumann neighborhood, in synchronous cEA's the information of an individual will take at least 50 time steps (i.e. 125000 evaluations) to reach the farthest individual in the grid, while in a cEA with asynchronous Line Sweep update method, it can take a small value such as 1 time step (i.e. 2500 evaluations).

6 Conclusions

In this paper we have analyzed the behavior of three alternative policies for asynchronously updating the population of a decentralized cellular GA. We have initiated this research line because we had some preliminary expectations relating asynchronous policies in the field of cellular automata [14] and distributed GA's [2]. We have tackled this study by considering three representative problems: deceptive (MMDP), epistatic (P-PEAKS) and hard (FMS) problems. Our first conclusion is that, for any size of the search grid, the synchronous update policy is the best in terms of percentage of hits, because it always provides an equal or larger success rate with respect to any of the asynchronous policies.

However, if we consider the number of evaluations needed to locate the optimum (efficiency) we got the opposite conclusion: asynchronous methods are faster (sometimes much faster) than the synchronous one. A simple asynchronous policy such that Line Sweep (LS) provides the faster convergence for two of the problems (MMDP and P-PEAKS), while it shows a high and desirable success rate (similar to that of the synchronous update). In the hard FMS problem, Fixed Random Sweep got a considerably faster solution, and can be pointed out as a good approach.

Globally stated, fast convergence means local optima in evolutionary algorithms, but cellular GA's in general, and the Line Sweep policy in particular, offers a good tradeoff solution to this problem without bothering researchers with a large number of parameters to be tuned (maybe only the ratio between the size of the grid and the neighborhood being used [1]).

As a future work, we will enlarge the set of considered problems, include a study of the influence of the shape of the 2-d grid containing the population, and try to better characterize the relationship between the performance measures and the kind of problems.

References

1. E. Alba and J. M. Troya. Cellular evolutionary algorithms: Evaluating the influence of ratio. In M. Schoenauer et al., editor, *Parallel Problem Solving from*

Nature, PPSN VI, volume 1917 of *Lecture Notes in Computer Science*, pages 29–38. Springer-Verlag, 2000.

2. E. Alba and J. M. Troya. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Generation Computer Systems*, 17:451–465, January 2001.
3. M. Capcarrère, M. Tomassini, A. Tettamanzi, and M. Sipper. A statistical study of a class of cellular evolutionary algorithms. *Evolutionary Computation*, 7(3):255–274, 1999.
4. K. Deb D.E. Goldberg and J. Horn. Massively multimodality, deception and genetic algorithms. In R. Männer and B. Manderick, editors, *Proceedings of the PPSN II*, pages 37–46. North-Holland, 1992.
5. M. Gorges-Schleuter. ASPARAGOS an asynchronous parallel genetic optimisation strategy. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 422–427. Morgan Kaufmann, 1989.
6. M. Gorges-Schleuter. An analysis of local selection in evolution strategies. In *Genetic and evolutionary conference, GECCO99*, volume 1, pages 847–854. Morgan Kaufmann, San Francisco, CA, 1999.
7. T. E. Ingerson and R. L. Buvel. Structure in asynchronous cellular automata. *Physica D*, 10:59–68, 1984.
8. K.A. De Jong, M.A. Potter, and W.M. Spears. Using problem generators to explore the effects of epistasis. In T. Bäck, editor, *Proceedings of the Seventh ICGA*, pages 338–345. Morgan Kaufmann, 1997.
9. B. Manderick and P. Spiessens. Fine-grained parallel genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 428–433. Morgan Kaufmann, 1989.
10. D. Corne S. Tsutsui, A. Ghosh and Y. Fujimoto. A real coded genetic algorithm with an explorer and an exploiter populations. In T. Bäck, editor, *Proceedings of the Seventh ICGA*, pages 238–245. Morgan Kaufmann, 1997.
11. J. Sarma and K. A. De Jong. An analysis of the effect of the neighborhood size and shape on local selection algorithms. In H. M. Voigt, W. Ebeling, I. Rechenberg, and H. P. Schwefel, editors, *Parallel Problem Solving from Nature (PPSN IV)*, volume 1141 of *Lecture Notes in Computer Science*, pages 236–244. Springer-Verlag, Heidelberg, 1996.
12. J. Sarma and K. A. De Jong. An analysis of local selection algorithms in a spatially structured evolutionary algorithm. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 181–186. Morgan Kaufmann, 1997.
13. B. Schönfisch and A. de Roos. Synchronous and asynchronous updating in cellular automata. *BioSystems*, 51:123–143, 1999.
14. M. Sipper, M. Tomassini, and M. S. Capcarrere. Evolving asynchronous and scalable non-uniform cellular automata. In G. D. Smith, N. C. Steele, and R. F. Albrecht, editors, *Proceedings of International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA97)*, pages 67–71. Springer-Verlag, Vienna, 1997.
15. M. Tomassini. The parallel genetic cellular automata: Application to global function optimization. In R. F. Albrecht, C. R. Reeves, and N. C. Steele, editors, *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 385–391. Springer-Verlag, 1993.
16. D. Whitley. Cellular genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, page 658. Morgan Kaufmann Publishers, San Mateo, California, 1993.