The Influence of Grid Shape and Asynchronicity on Cellular Evolutionary Algorithms

Bernabé Dorronsoro
Computing Central Services
University of Málaga
Málaga, Spain
dorronsoro@uma.es

Enrique Alba
Dept. of Languages and Computer Science
University of Málaga
Málaga, Spain
eat@lcc.uma.es

Mario Giacobini
Information Systems Department
University of Lausanne
Lausanne, Switzerland
Mario.Giacobini@hec.unil.ch

Marco Tomassini
Information Systems Department
University of Lausanne
Lausanne, Switzerland
Marco.Tomassini@hec.unil.ch

***Abstract*— In this paper we study cellular evolutionary algorithms, a kind of decentralized heuristics, and the importance of their induced exploration/exploitation balance on different problems. It is shown that, by choosing synchronous or asynchronous update policies, the selection pressure, and thus the exploration/exploitation tradeoff, can be influenced directly, without using additional ad hoc parameters. Synchronous algorithms of different neighborhood-to-topology ratio, and asynchronous update policies are applied to a set of benchmark problems. Our conclusions show that the update methods of the asynchronous versions, as well as the ratio of the decentralized algorithm, have a marked influence on its convergence and on its accuracy.**

I. Introduction

This paper focusses on the class of algorithms called cellular evolutionary algorithms (cEAs). We here present the canonical algorithm and suggest interesting variants targeted to solve complex problems accurately with a minimum effort for customization. These techniques, also called diffusion or fine-grained models, have been popularized, among others, by early work of Gorges-Schleuter [1] and Manderick and Spiessen [2]. The basic idea behind their behavior is to add some structure to the population of tentative solutions. The pursued effect is to improve on the diversity and exploration capabilities of the algorithm while still admitting an easy combination with local search and other search techniques to improve exploitation.

The above mentioned structured models are based on a spatially distributed population in which genetic operations may only take place in a small neighborhood of each individual. Usually, individuals are arranged on a regular grid of dimensions $d = 1, 2$ or 3. Cellular EAs are a kind of decentralized EA model [3]. They are not just a parallel implementation of an EA; in fact, although parallelism could be used to speed up the search, we do not address parallel implementations in this work. However, it is worth remarking that, although SIMD (single instruction stream - multiple data streams) machine implementations were popular a decade ago, this is no longer true, and today the best distributed implementation of a cEA should make use of domain decomposition on clusters of networked machines.

Although fundamental theory is still an open research line for cEAs, 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* EAs (here panmictic means that any individual may mate with any other in the population). The influence of the selection method, neighborhood, and grid topology on the efficiency of cEAs in comparison to other EAs have all been investigated in detail in [4], [5], [6], [7], and tested on different applications such as combinatorial and numerical optimization.

Cellular EAs can be seen as stochastic cellular automata (CAs) [8], [9] where the cardinality of the set of states is equal to the number of points in the search space. CAs, as well as cEAs, 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* CAs have shown that sequential update policies have a marked effect on their dynamics (see e.g. [10], [11]). In addition, the shape of the structure in which individuals evolve has a deep impact on the performance of the cEA. The algorithm admits a specially easy modulation of its shape that can sharpen the exploration or the exploitation capabilities of the canonical technique, as shown in [7]. Thus, it is interesting to investigate asynchronous cEAs and non-square shaped cEAs, in order to analyze their problem solving capabilities, which is the subject of the second part of this paper.

This work is organized as follows. The next section contains some background on synchronous and asynchronous cEAs. In Section III we discuss the ability of cEAs for changing their behavior depending on the population shape. We briefly study in Section IV the theoretical behavior of the proposed algorithms. In sections V and VI we deal with a set of benchmark problems with the goal of illustrating the actual computational power of these algorithms for optimization. Finally, section VII offers our conclusions, as well as some comments on future work.

II. Synchronous and Asynchronous cEAs

A cEA starts with the cells (individuals) in a random state and proceeds by successively updating them using evolutionary operators, until a termination condition is met. Updating a cell in a cellular EA means selecting two parents in the individual's neighborhood (including the individual itself),

applying genetic operators to them, and finally replacing the individual if an offspring has a better fitness or using another replacement policy. The following pseudo-code is a high-level description of the algorithm for a two-dimensional grid of size HEIGHT×WIDTH and for formally simultaneous update of all the cells. In fact, true simultaneous update could be performed in a parallel computer, but usually this is simulated by using a sequential machine and a second array to hold the updated cells:

Algorithm 1 Pseudocode of a synchronous cGA

```
1: proc Steps_Up(cga)      //Algorithm parameters in 'cga'
2: for s ← 1 to MAX_STEPS do
3:   for x ← 1 to WIDTH do
4:     for y ← 1 to HEIGHT do
5:       n_list←Get_Neighborhood(cga,position(x,y));
6:       parents←Local_Select(n_list);
7:       aux_indiv←Recombination(cga.Pc,parents);
8:       aux_indiv←Mutation(cga.Pm,aux_indiv);
9:       aux_indiv←Local_Search(cga.Pl,aux_indiv);
10:      Evaluate_Fitness(aux_indiv);
11:      Insert_If_Better(position(x,y),aux_indiv,cga,aux_pop);
12:    end for
13:  end for
14:  cga.pop←aux_pop;
15:  Update_Statistics(cga);
16: end for
17: end_proc Steps_Up;
```

Cells can be updated *synchronously* or *asynchronously*. In synchronous (parallel) update all the cells change their states simultaneously, while in asynchronous, or sequential, update cells are updated one at a time in some order. There exist many ways for sequentially updating the cells of a cEA (an excellent discussion of asynchronous update in cellular automata, which are essentially the same system as a cEA, is available in [10]). We consider four asynchronous update methods: *fixed line sweep* (LS), *fixed random sweep* (FRS), *new random sweep* (NRS), and *uniform choice* (UC).

- In *fixed line sweep* (LS), the simplest method, the n grid cells are updated sequentially $(1, 2 \ldots n)$ line after line.
- In *fixed random sweep* (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 all update cycles.
- The *new random sweep* method (NRS) works like FRS, except that a new random cell permutation is used for each sweep through the array.
- In *uniform choice* (UC), the next cell to be updated is chosen at random with uniform probability and with replacement. This corresponds to a binomial distribution for the update probability.

A *time step* is defined as updating n times sequentially, 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 UC method, since some cells might be updated more than once in a single time step. It should be noted that, with the exception of fixed line sweep, the other asynchronous updating policies are stochastic, representing an additional source of non-determinism besides that of the genetic operators.

III. New cEA Variants Based on a Modified Ratio

After explaining our basic algorithm and the asynchronous variants in the previous section, we now proceed to characterize the population grid itself. For this goal, we use the "radius" definition given in [7], which is refined from the seminal one appeared in [6] to account for non square grids. The grid is considered to have a radius equal to the dispersion of n^* points in a circle centered in $(\overline{x}, \overline{y})$ (Eq. 1). This definition always assigns different numerical values to different grids.

$$rad = \sqrt{\frac{\sum (x_i - \overline{x})^2 + \sum (y_i - \overline{y})^2}{n^*}}, \qquad (1)$$
$$\overline{x} = \frac{\sum_{i=1}^{n^*} x_i}{n^*},\ \overline{y} = \frac{\sum_{i=1}^{n^*} y_i}{n^*}$$

Although it is called a "radius", rad measures the dispersion of n^* patterns. Other possible measures for symmetrical topologies would allocate the same numeric value to different topologies (which is undesirable). Two examples are the radius of a circle surrounding a rectangle containing the topology, or an *asymmetry coefficient*. The definition (1) does not only characterize the grid shape but it also can provide a radius value for the neighborhood. As proposed in [6], the grid-to-neighborhood relationship can be quantified by the ratio between their radii (Eq. 2).

$$ratio_{cEA} = \frac{rad_{Neighborhood}}{rad_{Topology}} \qquad (2)$$

When solving a given problem with a constant number of individuals ($n = n^*$, for making fair comparisons) the topology radius will increase as the grid gets thinner (Fig. 1b). Since the neighborhood is kept constant in size and shape throughout this paper (we always use linear5 –L5–, Fig. 1a), the ratio will be smaller as the grid gets thinner.

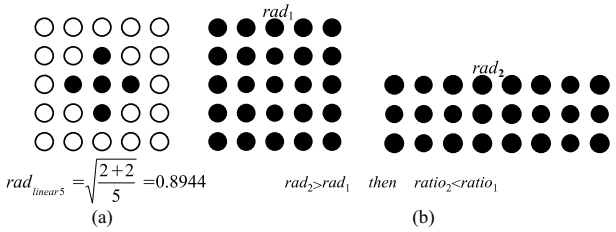

Fig. 1. (a) Radius of neighborhood L5. (b) $5\times5=25$ and $3\times8\approx25$ grids; approximately equal number of individuals with two different ratios

After presenting this characterization of the radius and topology by means of a ratio value the main question still remains to be posed: what is the importance of such a ratio measure? The answer comes when we get into the actual meaning of the ratio, i.e., reducing the ratio means reducing the global selection intensity on the population (see next section), thus promoting *exploration*. This is expected to allow for a higher diversity in the genotype that could improve the results

in difficult problems (like in multimodal or epistatic tasks). On the other hand, the search performed inside each neighborhood is guiding the *exploitation* of the algorithm. We study in this paper how the ratio affects the search efficiency over a variety of domains. Changing the ratio during the search is a unique feature of cEAs that can be used to shift from exploration to exploitation at a minimum complexity without introducing just another new algorithm family in the literature.

Many techniques for managing the exploration/exploitation tradeoff are possible. Among them, it is worth to make a special mention to heterogeneous EAs [12] [13], in which algorithms with different features run in separate subpopulations and collaborate in order to avoid premature convergence. A different alternative is using *Memetic Algorithms* [14], in which *local search* is combined with the genetic operators in order to promote local exploitation.

IV. Selection Pressure, Grid Shape and Time

Selection pressure is related to the concept of *takeover time*, which is the time it takes for a single best individual to colonize the whole population with copies of itself under the effects of selection only [15]. Shorter takeover times mean a stronger selection.

We plot in figures 2, 3 and 4 the takeover time for some synchronous and asynchronous cEAs. For that, we only maintain selection active, and we also use an elitist replacement of the best parent. Nor mutation neither crossover are applied during the evolution. Initially, we place only one optimal individual in a random location of the grid, and we let it be selected and copied until it takes over all the grid positions. The axes of the graphics represent the proportion of the best individual in the population (x axis), and the number of generations or time steps (y axis).

Algorithms with similar ratio show a similar selection pressure, as stated in [5]. In Fig. 2 we plot such a similar behavior for two algorithms with different neighborhood and population radii, but having two similar ratio values. The cases plotted are those using a L5 neighborhood with a 32×32 population, and a compact21 —C21— neighborhood with a population of 64×64 individuals. In the C21 neighborhood a central cell is surrounded by two cells in all directions, including the diagonals, and the four corner cells are cut out.

Hence, it may be interesting to see how the shape of the grid influences the search of the algorithm. The selection pressure for different cEAs using L5 neighborhood and 6 possible grid shapes are plotted in Fig. 3 for a population of 1024 individuals. Note that the selection pressure induced in synchronous rectangular grids falls under the curve for a synchronous square grid (32×32 population), which means that thinner grids favor a more explorative style of search.

If we now keep the shape of the grid constant (say a square) but we allow the cell update mode to change, we observe a similar effect on the selection pressure. In fact, it is found that the global selection pressure induced by the various asynchronous policies fall between the low synchronous limit and the high panmictic bound (see Fig. 4 and [16]). Thus, by varying the update policies it is possible to influence the explorative or exploitative character of the search.

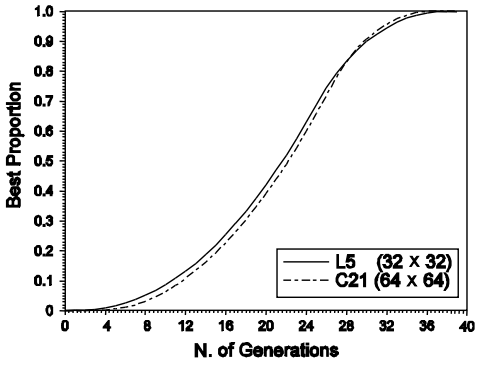

Fig. 2. Growth curves of the best individual for two cEAs with different neighborhood and population shapes, but similar ratio values. The vertical axis represents the proportion of population consisting of best individual as a function of time

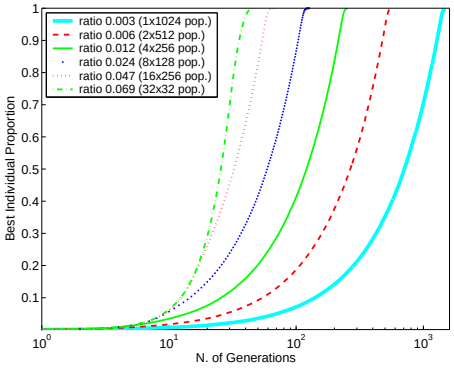

Fig. 3. Takeover times with tournament selection using a L5 neighborhood in populations of 1024 individuals with different grid shapes. Mean values over 100 runs. The vertical axis represents the proportion of population consisting of best individual as a function of time. Horizontal axis is in logarithmic scale

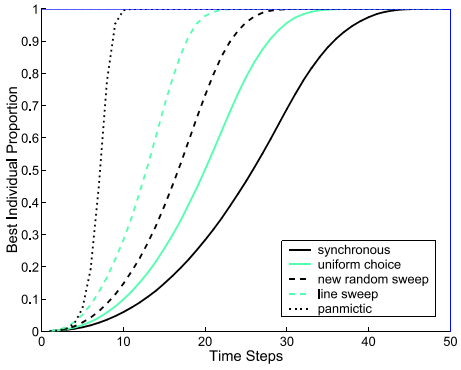

Fig. 4. Takeover times with tournament selection using a L5 neighborhood in a 32×32 grid. Mean values over 100 runs. The vertical axis represents the proportion of population consisting of best individual as a function of time

V. Test Suite

In this section we present the set of problems chosen for this study. The benchmark is representative because it contains

many different interesting features found in optimization, such as epistasis, multimodality, deceptiveness, and problem generators. These are important ingredients in any work trying to evaluate algorithmic approaches with the objective of getting reliable results, as stated by Whitley et al. in [17].

We experiment with the massively multimodal deceptive problem (MMDP), and the multimodal problem generator P-PEAKS, which are included in the set of problems studied in [7]; next we will extend this basic two-problem benchmark with error correcting code design (ECC), and maximum cut of a graph (MAXCUT). The choice of this set of problems is justified by both their difficulty and their application domains (telecommunications, combinatorial optimization, etc.). This gives us a high level of confidence in the results, although the evaluation of conclusions will result more laborious than with a small test suite.

The selected problems are explained in subsections V-A to V-D. We include the explanations in this paper to make it self-contained and to avoid the typical small lacks that could preclude other researchers from reproducing the results.

A. Massively Multimodal Deceptive Problem (MMDP)

The MMDP is a problem that has been specifically designed to be difficult for an EA [18]. It is made up of k deceptive subproblems (s_i) of 6 bits each one, whose value depends on the number of ones (*unitation*) a binary string has (see Fig. 5). It is easy to see (graphic of Fig. 5) that these subfunctions have two global maxima and a deceptive attractor in the middle point.

Unitation	Subfunction value
0	1,000000
1	0,000000
2	0,360384
3	0,640576
4	0,360384
5	0,000000
6	1,000000

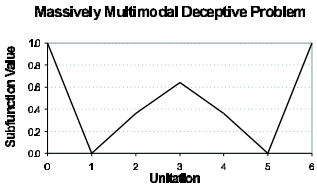

Fig. 5. Basic deceptive bipolar function (s_i) for MMDP

In MMDP each subproblem s_i contributes to the fitness value according to its *unitation* (Fig. 5). The global optimum has a value of k and it is attained when every subproblem is composed of zero or six ones. The number of local optima is quite large (22^k), while there are only 2^k global solutions. Therefore, the degree of multimodality is regulated by the k parameter. We use here a considerably large instance of $k = 40$ subproblems. The instance we try to maximize for solving the problem is shown in Eq. 3, and its maximum value is equal to k.

$$f_{MMDP}(\vec{s}) = \sum_{i=1}^{k} fitness_{s_i} \quad (3)$$

B. Multimodal Problem Generator (P-PEAKS)

The P-PEAKS problem [19] is a multimodal problem generator. A problem generator is an easily parameterizable task which has a tunable degree of epistasis, thus admitting to derive instances with growing difficulty at will. Also, using a problem generator removes the opportunity to hand-tune algorithms to a particular problem, therefore allowing a larger fairness when comparing algorithms. With a problem generator we evaluate our algorithms on a high number of random problem instances, since a different instance is solved each time the algorithm runs, then the predictive power of the results for the problem class as a whole is increased.

The idea of P-PEAKS is to generate P random N-bit strings that represent the location of P peaks in the search space. The fitness value of a string is the number of bits the string has in common with the nearest peak in that space, divided by N (as shown in Eq. 4). By using a small/large number of peaks we can get weakly/strongly epistatic problems. In this paper we have used an instance of $P = 100$ peaks of length $N = 100$ bits each, which represents a medium/high epistasis level [7]. The maximum fitness value for this problem is 1.0.

$$f_{P-PEAKS}(\vec{x}) = \frac{1}{N} \max_{1 \le i \le p} \{N - HammingD(\vec{x}, Peak_i)\} \quad (4)$$

C. Error Correcting Code Design Problem (ECC)

The ECC problem was presented in [20]. We will consider a three-tuple (n, M, d), where n is the length of each codeword (number of bits), M is the number of codewords, and d is the minimum Hamming distance between any pair of codewords. Our objective will be to find a code which has a value for d as large as possible (reflecting greater tolerance to noise and errors), given previously fixed values for n and M. The problem we have studied is a simplified version of that in [20]. In our case we search half of the codewords ($M/2$) that will compose the code, and the other half is made up by the complement of the codewords computed by the algorithm.

The fitness function to be maximized is:

$$f_{ECC} = \frac{1}{\sum_{i=1}^{M} \sum_{j=1, i \neq j}^{M} d_{ij}^{-2}} \quad (5)$$

where d_{ij} represents the Hamming distance between codewords i and j in the code *C* (made up of M codewords, each of length n). We consider in the present paper an instance where $M = 24$ and $n = 12$. The search space is of size $\binom{4096}{24}$, which is approximately 10^{87}. The optimum solution for $M = 24$ and $n = 12$ has a fitness value of 0.0674 [21].

D. Maximum Cut of a Graph (MAXCUT)

The MAXCUT problem looks for a partition of the set of vertices (V) of a weighted graph $G = (V, E)$ into two disjoint subsets V_0 and V_1 so that the sum of the weights of the edges with one endpoint in V_0 and the other one in V_1 is maximized. For encoding the problem we use a binary string $(x_1, x_2, \ldots, x_n)$ of length n where each digit corresponds to a vertex. If a digit is 1 then the corresponding vertex is in set V_1; if it is 0 then the corresponding vertex is in set V_0. The function to be maximized [22] is:

$$f_{MAXCUT}(\vec{x}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} \cdot \left[x_i \cdot (1 - x_j) + x_j \cdot (1 - x_i) \right] \quad (6)$$

Note that w_{ij} contributes to the sum only if nodes i and j are in different partitions. While one can generate different random graph instances to test the algorithm, here we have used the case "cut20.09", with 20 vertices and a probability of 0.9 of having an edge between any two randomly chosen vertices. The maximum fitness value for this instance is 56.740064.

VI. Experimental Analysis

Although a full-length study of the problems presented in the previous section is beyond the scope of this work, we present results comparing synchronous and asynchronous cEAs, and also cEAs having different values of the ratio, always with a constant neighborhood shape (L5). Note that it is not our aim here to compare cEAs performance with state-of-the-art algorithms and heuristics for combinatorial and numerical optimization. To this end, we should at least tune the parameters and include local search capabilities in the algorithm, which is not the case. Thus, the results only pertain to the relative performance of the different cEA update methods and ratios among themselves.

Here we present the results of solving several problems using JCell v1.0, our custom simulation program written in Java, with three different static ratios, and the four asynchronous update modes previously described. The configuration of the algorithm is detailed in Table I, while the static ratios used are shown in Table II.

TABLE I
PARAMETERIZATION USED IN THE ALGORITHM

Population Size	400 individuals
Selection of Parents	binary tournament + binary tournament
Recombination	DPX, $p_c = 1.0$
Bit Mutation	Bit-flip, $p_m = 1/L$
Individual Length	L
Replacement	Rep_if_Better

TABLE II
STUDIED RATIOS

Name	(shape of population)	Value of ratio
Square	(20×20 individuals)	0.11
Rectangular	(10×40 individuals)	0.075
Narrow	(4×100 individuals)	0.031

We show in the following tables the results for the problems before mentioned: MMDP (Table III), P-PEAKS (Table IV), ECC (Table V), and MAXCUT (Table VI). We have performed 100 independent runs for any algorithm and for every problem in the test-suite. In these tables we report the average of the final best fitness on every run, the average number of time steps to obtain the optimum value (if obtained) and the hit rate (percentage of successful runs). Therefore, we are analyzing the final distance to the optimum (especially interesting when the optimum is not found), the effort of the algorithm, and its expected efficacy, respectively.

From the inspection of these tables some conclusions can be clearly drawn. First, the studied asynchronous algorithms

TABLE III
MMDP PROBLEM WITH A MAXIMUM OF 1000 GENERATIONS

Algorithm	Avg. Solution (best=20)	Avg. Generations	Hit Rate
Square	19.813	214.18	57%
Rectangular	19.824	236.10	58%
Narrow	19.842	299.67	61%
LS	19.518	343.52	23%
FRS	19.601	209.94	31%
NRS	19.536	152.93	28%
UC	19.615	295.72	36%

TABLE IV
P-PEAKS PROBLEM WITH A MAXIMUM OF 100 GENERATIONS

Algorithm	Avg. Solution (best=1)	Avg. Generations	Hit Rate
Square	1.0	51.84	100%
Rectangular	1.0	50.43	100%
Narrow	1.0	53.94	100%
LS	1.0	34.75	100%
FRS	1.0	38.39	100%
NRS	1.0	38.78	100%
UC	1.0	40.14	100%

TABLE V
ECC PROBLEM WITH A MAXIMUM OF 500 GENERATIONS

Algorithm	Avg. Solution (best=0.0674)	Avg. Generations	Hit Rate
Square	0.0670	93.92	85%
Rectangular	0.0671	93.35	88%
Narrow	0.0673	104.16	94%
LS	0.0672	79.66	89%
FRS	0.0672	82.38	90%
NRS	0.0672	79.46	89%
UC	0.0671	87.27	86%

TABLE VI
MAXCUT PROBLEM WITH A MAXIMUM OF 100 GENERATIONS

Algorithm	Avg. Solution (best=56.74)	Avg. Generations	Hit Rate
Square	56.74	11.26	100%
Rectangular	56.74	11.03	100%
Narrow	56.74	11.88	100%
LS	56.74	9.46	100%
FRS	56.74	9.69	100%
NRS	56.74	9.55	100%
UC	56.74	9.58	100%

tend to need a smaller number of generations than the synchronous ones to locate an optimum, in general. Moreover, referring to tables in Appendix where t-tests are reported (character '+' stands for significant values, while '-' means no significance), the reader will confirm that the differences among asynchronous and synchronous algorithms are statistically significant, thus indicating that the asynchronous versions perform more efficiently with respect to cEAs with different static ratios. This result conforms to our study of Section IV since asynchronous algorithms have a lesser takeover time. There are however punctual exceptions, like in MMDP.

Conversely, if we pay attention to the success (hit) rate, it can be concluded that the synchronous policies with changing ratios outperform the asynchronous algorithms (except for ECC): slightly in terms of the average final fitness, and clearly in terms of probability of finding a solution (i.e., frequency of optimum location).

Another interesting result is the fact that we can define two classes of problems: those solved by any method to optimality

(100% hit rate) and those in which no 100% rate is achieved at all. The latter seem to be suitable for cEAs directly, while the former need some help, e.g., by including local search.

In order to summarize the large set of results and get some useful conclusions we present a ranking with the best algorithms by following three different metrics: average best final solution, average number of generations on success, and hit rate. Table VII shows the three mentioned rankings. These rankings have been computed by adding the position (from better to worse: 1, 2, 3 ...) that algorithms are allocated for the previous results presented from Table III to Table VI, according to the three criteria.

TABLE VII

RANKING OF THE ALGORITHMS

Avg. Solution		Avg. Generations		Hit Rate	
1 **Narrow**	4	1 **NRS**	8	1 **Narrow**	4
2 **Rectangular**	9	2 **LS**	10	2 **Rectangular**	9
2 **FRS**	9	3 **FRS**	11	2 **FRS**	9
4 **NRS**	10	4 **UC**	16	4 **NRS**	11
5 **UC**	11	5 **Rectangular**	19	5 **Square**	12
5 **LS**	11	6 **Square**	21	5 **UC**	12
7 **Square**	12	7 **Narrow**	27	5 **LS**	12

As we would expect after the previous comments, according to the average final best fitness and hit rate criteria, synchronous algorithms with narrow and rectangular ratios are in general more accurate than all the asynchronous ones for our test problems, with a special leading position for narrow population grids. On the other hand, asynchronous versions clearly outperform any of the synchronous algorithms in terms of the average number of generations, with a trend towards NRS as the best ranked flavor of cEA for our test suite.

VII. CONCLUSIONS

In the first part of this paper we have described several asynchronous update policies for the population of a cEA, followed by some ratio policies, all of them inducing a different kind of search in the cellular EA. One can tune the selection intensity of a cEA by choosing the update policy and/or grid ratio without having to deal with additional numerical parameter settings. This is a clear advantage of the algorithms proposed in this study.

In the second part of the paper we have applied our extended cEAs to a set of test problems. Although our goal has not been that of obtaining solvers able to compete with state-of-the art specialized heuristics, the results point in that sense: cEAs are very efficient optimization techniques, that could be further improved by being hybridized with local search techniques [23]. The results on the test problems largely confirm, with some small exceptions, that the solving abilities using the various update/ratio modes are directly linked to their induced selection pressures, showing that exploitation plays an important role. It is clear that the role of exploration might be more important on even harder problem instances, but this aspect can be addressed in our algorithms by using more explorative settings, as well as by using different cEA strategies at different times during the search dynamically [24].

In summary, we have found asynchronous algorithms to be numerically more efficient (faster) than synchronous ones (with statistically significance) for P-PEAKS, ECC, and MAXCUT, but not for MMDP. On the other hand, synchronous algorithms outperform asynchronous ones in terms of the hit rate for our benchmark, which could be a very important issue for many applications. As a future work, it would be interesting to give a unified study of the different selection intensities and their influence in the resolution of each problem considered for the analysis.

APPENDIX

In this appendix we show a statistical comparison of the studied algorithms by performing *t*-tests on the results of all the algorithms. Tables VIII to XIII contain the values of our statistical comparison in terms of the solutions found and the number of generations. No tables are provided for those cases in which the optimum is found every run (MAXCUT and P-PEAKS). On the following tables, statistical significance (5% level) is shown by using symbol '+', while absence of statistical significance is marked with '-'.

TABLE VIII

P-VALUES OF THE AVG. FITNESS FOR MMDP

Algorithm	**Square**	**Rectangular**	**Narrow**	**LS**	**FRS**	**NRS**	**UC**
Square	•	−	−	+	+	+	+
Rectangular	−	•	−	+	+	+	+
Narrow	−	−	•	+	+	+	+
LS	+	+	+	•	−	−	−
FRS	+	+	+	−	•	−	−
NRS	+	+	+	−	−	•	−
UC	+	+	+	−	−	−	•

TABLE IX

P-VALUES OF THE GENERATIONS FOR MMDP

Algorithm	**Square**	**Rectangular**	**Narrow**	**LS**	**FRS**	**NRS**	**UC**
Square	•	−	+	+	−	−	−
Rectangular	−	•	−	−	−	−	−
Narrow	+	−	•	−	−	+	−
LS	+	−	−	•	−	+	−
FRS	−	−	−	−	•	−	−
NRS	−	−	+	+	−	•	+
UC	−	−	−	−	−	+	•

TABLE X

P-VALUES OF THE GENERATIONS FOR P-PEAKS

Algorithm	**Square**	**Rectangular**	**Narrow**	**LS**	**FRS**	**NRS**	**UC**
Square	•	+	+	+	+	+	+
Rectangular	+	•	+	+	+	+	+
Narrow	+	+	•	+	+	+	+
LS	+	+	+	•	+	+	+
FRS	+	+	+	+	•	−	+
NRS	+	+	+	+	−	•	+
UC	+	+	+	+	+	+	•

TABLE XI

P-VALUES OF THE AVG. FITNESS FOR ECC

Algorithm	Square	Rectangular	Narrow	LS	FRS	NRS	UC
Square	•	−	+	−	−	−	−
Rectangular	−	•	−	−	−	−	−
Narrow	+	−	•	−	−	−	+
LS	−	−	−	•	−	−	−
FRS	−	−	−	−	•	−	−
NRS	−	−	−	−	−	•	−
UC	−	−	+	−	−	−	•

TABLE XII

P-VALUES OF THE GENERATIONS FOR ECC

Algorithm	Square	Rectangular	Narrow	LS	FRS	NRS	UC
Square	•	−	+	+	+	+	+
Rectangular	−	•	+	+	+	+	+
Narrow	+	+	•	+	+	+	+
LS	+	+	+	•	−	−	+
FRS	+	+	+	−	•	−	+
NRS	+	+	+	−	−	•	+
UC	+	+	+	+	+	+	•

TABLE XIII

P-VALUES OF THE GENERATIONS FOR MAXCUT

Algorithm	Square	Rectangular	Narrow	LS	FRS	NRS	UC
Square	•	−	−	+	+	+	+
Rectangular	−	•	−	+	+	+	+
Narrow	−	−	•	+	+	+	+
LS	+	+	+	•	−	−	−
FRS	+	+	+	−	•	−	−
NRS	+	+	+	−	−	•	−
UC	+	+	+	−	−	−	•

ACKNOWLEDGEMENTS

This work has been partially funded by the Ministry of Science and Technology (MCYT) and Regional Development European Found (FEDER) under contract TIC2002-04498-C05-02 (the TRACER project) `http://tracer.lcc.uma.es`.

REFERENCES

[1] M. Gorges-Schleuter, "ASPARAGOS an asynchronous parallel genetic optimisation strategy," in *Proc. of the 3th Int. Conference on Genetic Algorithms*, J. D. Schaffer, Ed. 1989, pp. 422–427, Morgan Kaufmann.

[2] B. Manderick and P. Spiessens, "Fine-grained parallel genetic algorithms," in *Proc. of the 3th Int. Conference on Genetic Algorithms*, J. D. Schaffer, Ed. 1989, pp. 428–433, Morgan Kaufmann.

[3] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE-TEC*, vol. 6, no. 5, pp. 443–462, October 2002.

[4] M. Gorges-Schleuter, "An analysis of local selection in evolution strategies," in *Genetic and evolutionary conference, GECCO99*. 1999, vol. 1, pp. 847–854, Morgan Kaufmann, San Francisco, CA.

[5] J. Sarma and K. A. De Jong, "An analysis of local selection algorithms in a spatially structured evolutionary algorithm," in *Proceedings of the Seventh International Conference on Genetic Algorithms*, T. Bäck, Ed. 1997, pp. 181–186, Morgan Kaufmann.

[6] J. Sarma and K. A. De Jong, "An analysis of the effect of the neighborhood size and shape on local selection algorithms," in *Parallel Problem Solving from Nature (PPSN IV)*, H. M. Voigt, W. Ebeling, I. Rechenberg, and H. P. Schwefel, Eds. 1996, vol. 1141 of *Lecture Notes in Computer Science*, pp. 236–244, Springer-Verlag, Heidelberg.

[7] E. Alba and J. M. Troya, "Cellular evolutionary algorithms: Evaluating the influence of ratio," in *Parallel Problem Solving from Nature, PPSN VI*, M. Schoenauer et al., Ed. 2000, vol. 1917 of *Lecture Notes in Computer Science*, pp. 29–38, Springer-Verlag.

[8] M. Tomassini, "The parallel genetic cellular automata: Application to global function optimization," in *Proceedings of the Int. Conference on Artificial Neural Networks and Genetic Algorithms*, R. F. Albrecht, C. R. Reeves, and N. C. Steele, Eds. 1993, pp. 385–391, Springer-Verlag.

[9] D. Whitley, "Cellular genetic algorithms," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest, Ed. 1993, p. 658, Morgan Kaufmann Publishers, San Mateo, California.

[10] B. Schönfisch and A. de Roos, "Synchronous and asynchronous updating in cellular automata," *BioSystems*, vol. 51, pp. 123–143, 1999.

[11] M. Sipper, M. Tomassini, and M. S. Capcarrere, "Evolving asynchronous and scalable non-uniform cellular automata," in *Proceedings of International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA97)*. 1998, pp. 67–71, Springer-Verlag KG, Vienna.

[12] F. Herrera and M. Lozano, "Gradual distributed real-coded genetic algorithms," *IEEE-EC*, vol. 4, no. 1, pp. 43–62, April 2000.

[13] P. Adamidis and V. Petridis, "Co-operating populations with different evolution behaviours," in *3rd IEEE Conf. Evolutionary Computation*. 1996, pp. 188–191, IEEE Press.

[14] P. Moscato, *Handbook of Applied Optimization*, chapter Memetic Algorithms, Oxford University Press, 2000.

[15] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. 1991, pp. 69–93, Morgan Kaufmann.

[16] M. Giacobini, E. Alba, and M. Tomassini, "Selection intensity in asynchronous cellular evolutionary algorithms," in *Proceedings of the genetic and evolutionary computation conference GECCO'03*, E. Cantú-Paz et al., Ed. 2003, pp. 955–966, Springer Verlag, Berlin.

[17] D. Whitley, S. Rana, J. Dzubera, and K. E. Mathias, "Evaluating evolutionary algorithms," *Artif. Intelligence*, vol. 85, pp. 245–276, 1997.

[18] D. E. Goldberg, K. Deb, and J. Horn, "Massively multimodality, deception and genetic algorithms," in *Parallel Prob. Solving from Nature II*, R. Männer and B. Manderick, Eds. 1992, pp. 37–46, North-Holland.

[19] K. A. De Jong, M. A. Potter, and W. M. Spears, "Using problem generators to explore the effects of epistasis," in *Proceedings of the Seventh ICGA*, T. Bäck, Ed. 1997, pp. 338–345, Morgan Kaufmann.

[20] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, 1977.

[21] H. Chen, N. S. Flann, and D. W. Watson, "Parallel genetic simulated annealing: A massively parallel SIMD algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 2, pp. 126–136, 1998.

[22] S. Khuri, T. Bäck, and J. Heitkötter, "An evolutionary approach to combinatorial optimization problems," in *Proc. of the 22nd ACM Comp. Science Conference*, Phoenix, Arizona, 1994, pp. 66–73, ACM Press.

[23] G. Folino, C. Pizzuti, and G. Spezzano, "Parallel hybrid method for SAT that couples genetic algorithms and local search," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 323–334, Aug. 2001.

[24] E. Alba and B. Dorronsoro, "The exploration/exploitation tradeoff in dynamic cellular evolutionary algorithms," Submitted to a Journal, 2003.