Advanced Models of Cellular Genetic Algorithms Evaluated on SAT

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

Hugo Alfonso
LISI, National University of La Pampa
General Pico, Argentine
alfonsoh@ing.unlpam.edu.ar

Bernabé Dorronsoro
Dept. of Languages and Computer Science
University of Málaga, Spain
dorronsoro@uma.es

ABSTRACT

Cellular genetic algorithms (cGAs) are mainly characterized by their spatially decentralized population, in which individuals can only interact with their neighbors. In this work, we study the behavior of a large number of different cGAs when solving the well-known 3-SAT problem. These cellular algorithms differ in the policy of individuals update and the population shape, since these two features affect the balance between exploration and exploitation of the algorithm. We study in this work both synchronous and asynchronous cGAs, having static and dynamically adaptive shapes for the population. Our main conclusion is that the proposed adaptive cGAs outperform other more traditional genetic algorithms for a well known benchmark of 3-SAT.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Performance measures*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms; Performance

Keywords

Adaptation, cellular genetic algorithms, synchronicity

1. INTRODUCTION

Genetic Algorithms (GAs) work over a set (*population*) of potential solutions (*individuals*) by applying on them some stochastic operators in order to search for the best solutions. Usually, most GAs use a single population (panmixia) of individuals and apply operators on them as a whole (see Figure 1). In contrast, there also exists some tradition in using structured GAs (where the population is decentralized somehow), especially in relation to their parallel implementation.

GECCO'05, June 25–29, 2005, Washington, DC, USA.

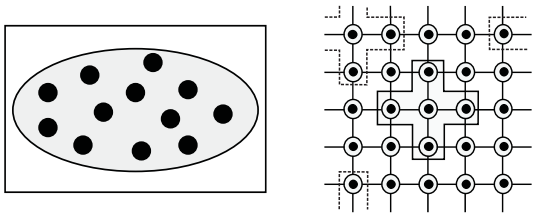

Figure 1: Panmictic and cellular populations.

Cellular GAs are a subclass of GAs in which the population is structured in a specified topology (usually a toroidal mesh of dimensions $d = 1, 2$, or 3). In a cGA, the genetic operations may only take place in a small neighborhood of each individual (see Figure 1). The pursued effect is to improve on the diversity and exploration capabilities of the algorithm (due to the presence of overlapped small neighborhoods) while still admitting an easy combination with local search to improve on exploitation at the level of each individual. Research in cGAs is a healthy field with recent advances in theory [1, 10, 11] and new results reporting their usefulness in maintaining diversity and promoting slow diffusion of solutions through the population grid [15].

Cellular GAs usually assume a *synchronous* (or "parallel") update policy, in which all the individuals are formally updated at the same time. However, this is not the only option available. The alternative is called in the literature an *asynchronous* (or "sequential") update method, and lies in placing the offsprings directly in the current population by following some rules [11] instead of updating all the individuals simultaneously. As it can be seen in [11], the update method has a marked effect on the behavior of the algorithm. Furthermore, the shape of the structure in which individuals evolve also has a deep impact in the performance of the cGA [3]. In [1], an adaptive cGA that dynamically can sharpen the exploration or the exploitation capabilities of the canonical technique by changing the shape of the population is proposed. In [6], the reader can find an study of the influence of some different asynchronous policies and grid shapes on the problem solving capabilities of cGAs.

The motivations for this work are basically two. Firstly, to check whether the results in [3] (non square grids are often more efficient than square ones) still hold for the 3-SAT problem. Secondly, to study whether changing the grid

shape during the evolution may be advantageous versus using an static one (as concluded in [1] for a large set of problems with synchronous cGAs) for an asynchronous updating of the individuals. An algorithm with changing shapes will dynamically select the most adequate grid shape during the search by itself, depending on the diversity of the population.

The contribution of this work is to deep in the study of the cGAs practice by merging two important trends in the search for a good exploration/exploitation tradeoff such as the synchronicity of the population updating [11], and the utilization of dynamically adaptive population shapes [1]. For that, we propose and evaluate 19 distinct cGAs in this paper, differing in the synchronicity of the population evolution and/or the population shape, which can be static or dynamically adaptive. In order to compare the behavior of all these algorithms, we have selected the satisfiability problem (SAT), a hard combinatorial problem, well-known in the literature, and having important practical applications.

This paper is organized as follows. In Section 2 we introduce the studied algorithms, explaining the concept of ratio and its effect on the behavior of the cGA. Section 3 briefly defines the SAT problem. Our results are summarized in Section 4, and Section 5 addresses our conclusions and future research activities.

2. CANONICAL AND ADVANCED CGAS

In this section we present the structure of a canonical cGA, as well as some possible methods for updating the population. After that, we introduce the concept of ratio (Section 2.1), and the importance of its value in the behavior of the algorithm (Section 2.2). Finally, we present in Section 2.3 the different policies we adopt for making the adaptive change in the shape of the dynamic populations.

In Algorithm 1 we show a pseudocode of a canonical cGA. As it can be seen, a cGA 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 cGA means selecting two parents in the individual's neighborhood (including the individual itself), applying the genetic operators to them, and finally replacing the individual following a given replacement policy in a new auxiliary population. After applying the reproductive cycle to all the individuals, the current population is replaced by the auxiliary one for the next generation. Algorithm 1 is a high-level description of the algorithm for a 2-D grid of size `HEIGHT`×`WIDTH` and for formally simultaneous update of all the cells.

Algorithm 1 Pseudocode of a synchronous cGA.

```
1: proc Steps_Up(cga)        //Algorithm parameters in 'cga'
2: while not Stop_Condition() 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:       Evaluate_Fitness(aux_indiv);
10:      Insert_If_Better(position(x,y),aux_indiv,cga,aux_pop);
11:    end for
12:  end for
13:  cga.pop←aux_pop;
14:  Update_Statistics(cga);
15: end while
16: 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 are many ways for sequentially updating the cells of a cGA (an excellent discussion of asynchronous update in cellular automata, which are essentially the same system as a cGA, is available in [17]). We consider four methods of asynchronous updating in this work:

- *Fixed Line Sweep* (LS): This is the simplest method. It updates the n grid cells sequentially $(1, 2, \ldots, n)$ line after line.
- *Fixed Random Sweep* (FRS): In this case, 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.
- *New Random Sweep* (NRS): Works like FRS, except that a new random cell permutation is used for each sweep through the array.
- *Uniform Choice* (UC): In this last method, 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* (or generation) 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 LS, the other asynchronous updating policies are stochastic, representing an additional source of non-determinism besides that of the genetic operators.

2.1 New cGA 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 [3], which accounts 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 grid shapes.

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

$$\overline{x} = \frac{\sum_{i=1}^{n^*} x_i}{n^*} \ , \ \overline{y} = \frac{\sum_{i=1}^{n^*} y_i}{n^*} \ .$$

The value given by Eq. 1 does not only characterize the grid shape but it can also provide a radius value for the neighborhood. The grid-to-neighborhood relationship can be quantified by the ratio between their radii (Eq. 2) [15].

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

After presenting this characterization of the radius and topology by means of the ratio, we wonder about the importance of such a measure. In fact, reducing the ratio means reducing the global selection intensity on the population (see Section 2.2), thus promoting *exploration*. This is expected to allow for a higher diversity in the genotype. Conversely, the search performed inside each neighborhood is guiding the *exploitation* of the algorithm. Changing the ratio during the search is a unique feature of cGAs that can be used to shift from exploration to exploitation at a minimum complexity without introducing just another new algorithm family in the literature.

2.2 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 using just selection [12][14]. Shorter takeover times mean a stronger selection.

In the field of cGAs, algorithms with a similar ratio show a similar selection pressure [16]. In Figure 2 we plot such a similar behavior for average executions of two cGAs with different neighborhood and population radii, but having two similar ratio values. The cases plotted are those using a linear5 —L5— neighborhood with a 32×32 population, and a compact21 —C21— neighborhood with a population of 64×64 individuals. As it is shown in Figure 2, the behavior of the two cGAs (having distinct neighborhood and population shapes, but similar ratio value) is almost the same. In [6] the authors show the effects of the ratio and the asynchronous update policy on the explorative/exploitative character of the search.

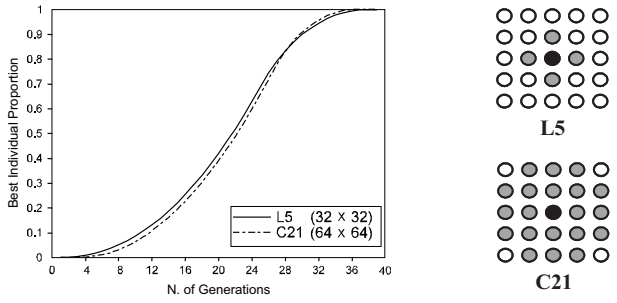

Figure 2: Selection pressure for two different cGAs with similar ratio values. The graph represents the proportion of best individuals in the population as a function of time.

2.3 Adaptive Algorithms

In this section we focus on a still new proposed class of cGA [1]. The core idea, like in most other adaptive algorithms, is to use some feedback mechanism that monitors the evolution and dynamically rewards or punishes parameters according to their impact on the solutions' quality. A main difference between our proposal and others in the literature is that both the monitoring feedback and the actions undertaken are computationally inexpensive in our case. We really believe that this is the capital feature of any adaptive algorithm for being useful for other researchers, since the literature contains many examples of interesting adaptive algorithms whose feedback control or adaptive actions are too expensive in some sense preventing somewhat their wide adoption in research.

Algorithm 2 Pattern for adaptive criteria.

```
1: if C1 then
2:    ChangeTo(square)      //exploit
3: else if C2 then
4:    ChangeTo(narrow)      //explore
5: end if
```

Our adaptive mechanisms (see Algorithm 2) modify the grid shape during the search as a function of the convergence speed, trying to maintain the diversity and to pursuit the global optimum during the run. For that, they increase the local exploitation by changing to the next more square grid shape whenever the algorithm evolves slowly, i.e., when the convergence speed decays below a given $\varepsilon \in [0, 1]$ value (condition C_1 of Algorithm 2). Conversely, if the search is proceeding too fast, diversity could be lost quickly, thus existing a risk of getting stuck in a local optimum. In this second case the population shape will be changed to a narrower grid (C_2 in Algorithm 2) promoting exploration and diversity in the forthcoming generations. This algorithm, which is executed every t_i generations, is a general search pattern, and multiple criteria could be used to control whether the algorithm should explore or exploit the individuals for the next t_i generations.

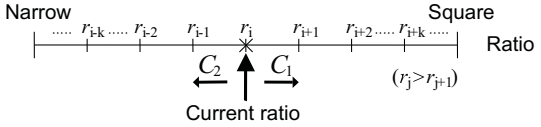

Figure 3: Performing the change in the ratio with our adaptive criteria.

Whenever the criterion C_1 (or C_2) is fulfilled, the adaptive search pattern performs a change in the grid shape to the immediately next more square (or narrow) allowed shape (function **ChangeTo**(square/narrow)). The bounding cases are the square and the completely linear (ring-like) shapes, as shown in Figure 3. When the change needed by the adaptive criteria exceeds those limits, the current population shape is maintained. Conversely, when the change is possible it is accomplished by computing a new position for every individual as shown in Figure 4. Of course, other changes could have been used, but, in our quest for efficiency, the proposed one is considered to introduce a negligible overhead.

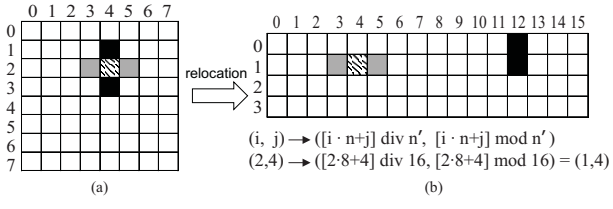

Figure 4: Relocation of individuals when the population changes from 8×8 to 4×16.

Once we have fixed the basic adaptive pattern and grid shape modification rules, we now proceed to explain the criteria used to determine when the population is going "too"

fast or "too" slow. We propose in this paper three different criteria for measuring the search speed. The measures are based on the *average fitness* (criterion AF), the *population entropy* (criterion PH), or a combination of both of them (criterion $AFPH$). Since these criteria check simple conditions over the average fitness and the population entropy (calculated in every run in the population statistics computation step), we can say that they are inexpensive to measure, and they are indicative of the search states at the same time too. The details on their internals are as follows:

- **AF**: This criterion is based on the *average fitness* of the population. Hence, we measure the convergence speed in terms of the diversity of performance. We define $\Delta\overline{f}_t$ as the difference between the average fitness values in generation t and the previous one: $\Delta\overline{f}_t = \overline{f}_t - \overline{f}_{t-1}$. The algorithm changes the ratio value to the immediately larger one (maintaining the population size) if the difference $\Delta\overline{f}_t$ between two contiguous generations (t and $t-1$) decreases at least in a factor of ε: $\Delta\overline{f}_t - \Delta\overline{f}_{t-1} < \varepsilon \cdot \Delta\overline{f}_{t-1}$ (condition C_1 of Algorithm 2). On the contrary, the ratio will be changed to its closer smaller value if that difference increases in a factor greater than $(1-\varepsilon)$: $\Delta\overline{f}_t - \Delta\overline{f}_{t-1} > (1-\varepsilon) \cdot \Delta\overline{f}_{t-1}$ (condition C_2). Thus, we define C_1 and C_2 as:

$$\begin{aligned} C_1 &::= \Delta\overline{f}_t < (1+\epsilon) \cdot \Delta\overline{f}_{t-1} \ , \\ C_2 &::= \Delta\overline{f}_t > (2-\epsilon) \cdot \Delta\overline{f}_{t-1} \ . \end{aligned}$$

- **PH**: We now propose to measure the convergence speed in terms of the genotypic diversity. The *population entropy* is the descriptor used in this case. The behavior of this criterion is similar to that of AF but it uses the change in the population entropy between two generations ($\Delta H_t = H_t - H_{t-1}$) instead of the average fitness variation. Consequently, conditions C_1 and C_2 are expressed as:

$$\begin{aligned} C_1 &::= \Delta H_t < (1+\epsilon) \cdot \Delta H_{t-1} \ , \\ C_2 &::= \Delta H_t > (2-\epsilon) \cdot \Delta H_{t-1} \ . \end{aligned}$$

- **AFPH**: This is the last proposed criterion for the adaptive algorithmic versions. It considers both the population entropy plus the average fitness acceleration by combining the two previous criteria (AF and PH). Thus, it relays on the phenotype and genotype diversity in order to obtain the best exploration/exploitation tradeoff. Condition C_1 in this case is the result of the logic `and` operation of conditions C_1 of the two preceding criteria. In the same way, C_2 will be the `and` operation of conditions C_2 of AF and PH:

$$\begin{aligned} C_1 &::= (\Delta\overline{f}_t < (1+\epsilon) \cdot \Delta\overline{f}_{t-1}) \ \textbf{and} \\ &\quad\ (\Delta H_t < (1+\epsilon) \cdot \Delta H_{t-1}) \ , \\ C_2 &::= (\Delta\overline{f}_t > (2-\epsilon) \cdot \Delta\overline{f}_{t-1}) \ \textbf{and} \\ &\quad\ (\Delta H_t > (2-\epsilon) \cdot \Delta H_{t-1}) \ . \end{aligned}$$

In short, we have proposed in this subsection three different adaptive criteria. The first one, AF, is based on the performance diversity, while PH is based on genotypic diversity. Finally, AFPH is a combined criterion accounting for diversity at these two levels.

3. SATISFIABILITY PROBLEMS

The satisfiability problem (SAT) has received much attention by the scientific community since any NP problem can be translated into an equivalent SAT problem in polynomial time (Cook theorem) [5]; while the inverse transformation may not always exist in polynomial time. The SAT problem was the first which was demonstrated to belong to the NP class of problems.

The SAT problem consists in assigning values to a set of n boolean variables $\vec{x} = (x_1, x_2, \ldots, x_n)$ such that they satisfy a given set of clauses $c_1(\vec{x}), c_2(\vec{x}), \ldots, c_m(\vec{x})$, where $c_i(\vec{x})$ is a disjunction of literals, and a literal is a variable or its negation. Hence, we can define SAT as a function $f : \mathbb{B}^n \rightarrow \mathbb{B}$, $\mathbb{B} = \{0, 1\}$ like:

$$f_{SAT}(\vec{x}) = c_1(\vec{x}) \wedge c_2(\vec{x}) \wedge \ldots \wedge c_m(\vec{x}) \ . \quad (3)$$

An instance of SAT, $\vec{x}$, is called satisfiable if $f_{SAT}(\vec{x}) = 1$, and unsatisfiable otherwise. A k-SAT instance is composed of clauses with length k, and when $k \geq 3$ the problem is NP-complete [9]. The fitness function we use is the *stepwise adaptation of weights* (SAW) [7]:

$$f_{SAW}(\vec{x}) = w_1 \cdot c_1(\vec{x}) + \ldots + w_m \cdot c_m(\vec{x}) \ . \quad (4)$$

This function weights the values of the clauses with $w_i \in \mathbb{N}$ in order to give more importance to those clauses which are not satisfied yet by the current best solution. These weights are adjusted dynamically according to $w_i = w_i + 1 - c_i(\vec{z})$, being $\vec{z}$ the current fittest individual.

4. RESULTS

In this section we analyze the results for all the proposed cGAs over the 12 hard instances (from $n = 30$ to 100 variables) of the selected test-suite [4]. These instances belong to the SAT phase transition of difficulty, where hardest instances are located, since they verify that $m = 4.3 * n$ [13] (being m the number of clauses).

In this paper we have studied three different synchronous cGAs, which use static shapes for the population. These algorithms are called Narrow, Rectang and Square, referring to the shape of the population used. Additionally, we have also studied 16 asynchronous cGAs. Four of them use a square static population (LS, FRS, NRS, and UC), and the other ones are the same algorithms but using the three presented adaptive criteria: AF (algorithms AF LS, AF FRS, AF NRS, and AF UC), PH (algorithms PH LS, PH FRS, PH NRS, and PH UC) and AFPH (algorithms AFPH LS, AFPH FRS, AFPH NRS, and AFPH UC).

We compare the efficiency and the effectiveness of the algorithms (details of the cGAs are given in Table 1). Therefore we present our results in tables 2 to 6 in terms of the Success Rate (SR) and the Average number of Evaluations to Solution (AES) after 50 independent runs. Hence, SR is the fraction of runs in which a solution was found, while AES is the number of evaluations made in those successful runs. It is clear that if SR = 0 then AES is not defined.

Analyzing the SR values of all the tables we can see that, for the tested instances, none of the 19 studied cGAs is able to find the optimum in every run. In particular, instances

Table 1: Parameterization used in cGAs.

	LS	FRS	NRS	UC	Square	Rectang	Narrow
Population Shape	12×12 (Ratio: 0.1832)					8×18 (Ratio: 0.1576)	4×36 (Ratio: 0.0856)
Population Size	144 Individuals						
Selection of Parents	Binary Tournament						
Recombination	DPX, $p_c = 1.0$						
Bit Mutation	Bit-flit, $p_m = 1/n$						
Replacement	Replace if Better						
Stop Condition	Find a solution or achieve 100.000 generations						

Table 2: Results for the synchronous cGAs with different static grid shapes.

Inst.		Narrow		Rectang		Square	
n	#	SR	AES	SR	AES	SR	AES
30	1	1.00	8243.2	1.00	7060.5	1.00	7781.8
	2	1.00	825546.2	1.00	633774.1	1.00	664741.4
	3	1.00	50941.4	1.00	58014.7	1.00	58757.8
40	4	1.00	19233.3	1.00	15585.3	1.00	12937.0
	5	1.00	10449.9	1.00	10245.1	1.00	10765.4
	6	0.76	2149524.2	0.76	2593805.5	0.68	1979966.1
50	7	1.00	15209.0	1.00	14863.4	1.00	13970.9
	8	1.00	64773.1	1.00	58416.6	1.00	66726.7
	9	0.96	873757.3	0.96	2065850.7	0.98	1657989.6
100	10	0.54	3850320.6	0.52	3669129.8	0.46	3459593.7
	11	1.00	162104.3	1.00	155289.6	1.00	128505.6
	12	1.00	334889.0	1.00	300922.9	1.00	264332.2

Table 3: Results for the asynchronous cGAs using a square grid shape.

Inst.		LS		FRS		NRS		UC	
n	#	SR	AES	SR	AES	SR	AES	SR	AES
30	1	1.00	7113.6	1.00	6359.0	1.00	6174.7	1.00	8032.3
	2	1.00	575925.1	1.00	1259930.9	0.98	1039215.7	1.00	984182.4
	3	1.00	42163.2	1.00	55774.1	1.00	57231.4	1.00	53642.9
40	4	1.00	16372.8	1.00	10595.5	1.00	11162.9	1.00	14443.2
	5	1.00	10794.2	1.00	8657.3	1.00	9383.0	1.00	11295.4
	6	0.72	2202740.0	0.74	2166332.1	0.74	1808885.2	0.70	2280305.8
50	7	1.00	12816.0	1.00	13766.4	1.00	13366.1	1.00	15194.9
	8	1.00	53925.1	1.00	68207.0	1.00	49337.3	1.00	85060.8
	9	0.96	1501134.0	1.00	1194707.5	1.00	1930878.7	1.00	1687541.8
100	10	0.64	2753109.0	0.50	3196460.2	0.54	3096293.3	0.56	3341808.0
	11	1.00	88473.6	1.00	102954.2	1.00	135037.4	1.00	129772.8
	12	1.00	222808.3	1.00	363876.5	1.00	262114.6	1.00	314110.1

6 and 10 can not be solved to optimality in all the runs by anyone of the proposed cGAs. But the success rate is in general very high in every algorithm, since the average value of SR for all the instances is always higher than 92.66% for all the algorithms. Moreover, the differences in the average SR of the cGAs for all the instances is very small because they all are between values 92.66% of Square and 94.83% of AF LS and AF NRS. We must remember that our goal is not to solve the problem to optimality, what it can be done for example including a local search operator (e.g. WSAT [8]) in the algorithm. Instead, we head in our study in a comparative analysis of algorithms.

Let us now analyze the efficiency (measured in terms of AES) of the studied cGAs. Among the synchronous proposed algorithms, the most efficient one seems to be Square (see Table 2), since it obtains the lowest values of AES in 6 out of the 12 tested instances. Rectang is also very efficient, because it is the best of the three algorithms in 5 out of the 12 instances. In terms of the average value for AES in all the instances, Square is also the best one, with differences of just 0.47% in the case of Narrow and 15.76% with Rectang. In contrast, Square is the algorithm with the lowest success rate.

As to the static asynchronous algorithms (see Table 3), we can see that LS has a lower computational cost (in terms of the AES value) than the other cGAs in 6 out of the 12 studied instances. After computing the average value of AES for all the instances, we see that LS obtains the best (smallest) value too, with a difference of 11% with FRS and NRS, and a 16% with respect to UC.

Tables 4, 5, and 6 summarize the results achieved for all the dynamic adaptive cGAs studied. If we compare the al-

Table 4: Results for the adaptive cGAs (based on the average fitness).

Inst.		AF LS		AF FRS		AF NRS		AF UC	
n	#	SR	AES	SR	AES	SR	AES	SR	AES
30	1	1.00	7807.7	1.00	9432.0	1.00	7021.4	1.00	8208.0
	2	1.00	1083277.4	1.00	607086.7	0.98	689930.4	0.98	1145969.6
	3	1.00	65085.1	1.00	50561.3	1.00	60215.0	1.00	75939.8
40	4	1.00	15431.0	1.00	15215.0	1.00	13219.2	1.00	14912.6
	5	1.00	10422.7	1.00	9642.2	1.00	9406.1	1.00	10514.9
	6	0.76	2845549.9	0.72	1932720.0	0.78	1899504.0	0.72	1684248.0
50	7	1.00	13020.5	1.00	14906.9	1.00	16490.9	1.00	13838.4
	8	1.00	81121.0	1.00	80824.3	1.00	70272.0	1.00	79856.6
	9	1.00	1235318.4	1.00	1012369.0	1.00	1620679.7	1.00	1318204.8
100	10	0.62	2701955.6	0.64	5134549.5	0.62	3653842.1	0.60	4064448.0
	11	1.00	137065.0	1.00	134231.0	1.00	107089.9	1.00	124655.0
	12	1.00	329385.6	1.00	312030.7	1.00	337331.5	1.00	400495.7

Table 5: Results for the adaptive cGAs (based on the population entropy).

Inst.		PH LS		PH FRS		PH NRS		PH UC	
n	#	SR	AES	SR	AES	SR	AES	SR	AES
30	1	1.00	6736.3	1.00	8046.7	1.00	8179.2	1.00	8769.6
	2	1.00	579680.6	0.98	1359127.8	0.98	856085.9	0.98	1012599.2
	3	1.00	65865.6	1.00	70447.7	1.00	55978.6	1.00	55713.6
40	4	1.00	13965.1	1.00	11946.2	1.00	16142.4	1.00	17426.9
	5	1.00	10123.2	1.00	9452.2	1.00	10624.3	1.00	11877.1
	6	0.84	1430502.9	0.76	2368523.4	0.74	1966362.8	0.76	1729242.9
50	7	1.00	11678.4	1.00	17487.4	1.00	15370.6	1.00	13475.5
	8	1.00	64152.0	1.00	79260.5	1.00	58596.5	1.00	52989.1
	9	1.00	1174190.4	0.96	1319100.0	0.98	1439991.2	0.96	1119216.0
100	10	0.46	2880651.1	0.60	3695520.0	0.58	4010136.8	0.46	1597855.3
	11	1.00	109903.7	1.00	151067.5	1.00	109486.1	1.00	177621.1
	12	1.00	345660.5	1.00	328564.8	1.00	363421.4	1.00	346294.1

Table 6: Results for the adaptive cGAs (based on the average fitness and the population entropy).

Inst.		AFPH LS		AFPH FRS		AFPH NRS		AFPH UC	
n	#	SR	AES	SR	AES	SR	AES	SR	AES
30	1	1.00	7597.4	1.00	7836.5	1.00	7701.1	1.00	8190.7
	2	0.98	891239.5	1.00	994469.8	1.00	823760.6	1.00	667382.4
	3	1.00	57081.6	1.00	53582.4	1.00	59616.0	1.00	51220.8
40	4	1.00	13720.3	1.00	14120.6	1.00	14112.0	1.00	14676.5
	5	1.00	10195.2	1.00	11111.0	1.00	9878.4	1.00	10661.8
	6	0.68	2503287.5	0.76	2383408.4	0.76	2293200.0	0.68	1847401.4
50	7	1.00	13291.2	1.00	13873.0	1.00	13671.4	1.00	15206.4
	8	1.00	59495.0	1.00	70652.2	1.00	59261.8	1.00	64368.0
	9	1.00	1275027.8	0.98	906215.5	0.98	1229081.1	1.00	1247382.7
100	10	0.58	2953301.0	0.58	3537757.2	0.56	2325558.9	0.48	2736672.0
	11	1.00	154742.4	1.00	79300.8	1.00	131385.6	1.00	167281.9
	12	1.00	458671.7	1.00	299335.7	1.00	342794.9	1.00	298022.4

gorithms in terms of the convergence speed (AES), we can see that algorithms using AF (based on the average fitness) have the best results in general, followed by the cGAs using AFPH and PH.

Since the 19 algorithmic approaches are independent, a statistic proof for k-independent samples is appropriate. Hence, we have made an analysis of variance of nonparametric classification, because the assumption on the normality and the homogeneity of the variances was impossible to verify. The Kruskal-Wallis ANOVA by Ranks test was used for computing the p-values for each instance. Under the null hypothesis (all samples come from populations with identical AES medians) we observed that the mean computational cost is not the same in all cases. Consequently this hypothesis was rejected. Moreover, the test values allowed us to distinguish two different groups, the first one exceeds widely the signification value ($\alpha = 0.05$) , and it is conformed by instances: 2, 3, 6, 8, 9, 10 and 12. Hence, the other group of instances

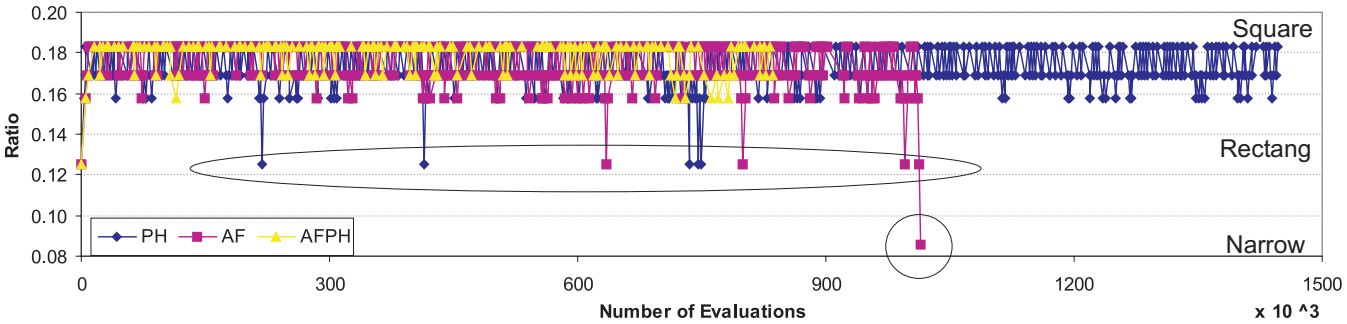

Figure 5: Typical evolution of the ratio (instance 9).

with statistically significant differences is composed by instances: 1, 4, 5, 7 and 11.

In order to summarize the results of our 19 cGAs used and get some useful conclusions, we present in Table 7 two rankings with the algorithms ordered from best to worst. These rankings have been made by summing up the position that algorithms hold in a sorted list (from best to worst) for each one of the 12 instances in terms of AES (when SR = 1.0) and SR (in the cases in which SR $\neq$ 1.0) respectively.

On the one hand, in the column SR = 1.0 of Table 7 we can see that the asynchronous cGAs using static grids execute, in general, a smaller number of fitness function evaluations (better efficiency) than the other studied algorithms. The exception is UC, since all the algorithms implementing the UC update policy are located at the bottom of the rank. Regarding the synchronous cGAs, we can see that the efficiency becomes worse as the ratio gets smaller (rectangular populations). It should be expected, since rectangular populations promote exploration, delaying the convergence of the algorithm.

On the other hand, in the column entitled SR $\neq$ 1.0 of Table 7 it can be seen that the adaptive algorithms, and particularly those using the AF adaptive criterion, obtain a higher success rate (more effectiveness) than the other algorithms used. Like in the case of the previously commented ranking (when SR = 1.0), the worst algorithms are, in general, those using the UC update policy. Finally, in contrast to the ranking made when SR = 1.0, the synchronous algorithms with rectangular populations are better suited than the square one in this case.

The studied synchronous algorithms are located, in general, close to or below the middle positions of the table in the two rankings. This means that asynchronous algorithms perform, in general, better than the synchronous ones both in terms of efficiency and efficacy. Moreover, the asynchronous cGAs implementing adaptive populations stand out as the best ones in terms of efficacy (specially those using the AF criterion), being also very well suited in terms of efficiency (they are located at the middle of the rank).

In order to better understanding the behavior of our adaptive cGAs, we plot in Figure 5 the evolution of the ratio fluctuation produced by AF LS, PH LS, and AFPH LS during a typical execution for a selected instance of the benchmark.

All the algorithms in Figure 5 have a clear trend towards promoting exploitation (square ratios over 0.18). The general trend to evolve towards the square shape is an expected scenario, because after an initial phase of exploration the search focuses towards population exploitation. We can see in Figure 5 the existence of brief and periodic visits to smaller ratios (below 0.13) with the goal of improving the exploration of the algorithm. This automatic and alternate shifting between exploitation and exploration in some problems is a noticeable feature of the self-guided mechanism we propose, because it shows how the algorithm decides by itself when to explore and when to exploit the population. The implication on this and other problems of this feature is quite important, as we state in [1].

Table 7: Ranking of the algorithms.

SR = 1.0			SR < 1.0		
Rank	Algorithm	Sum of Positions	Rank	Algorithm	Sum of Positions
1	NRS	46	1	AF NRS	6
2	LS	49	1	AF LS	6
3	FRS	67	3	AFPH FRS	11
4	PH LS	68	4	AF FRS	14
5	Square	79	4	AFPH NRS	14
6	AFPH NRS	80	4	PH FRS	14
7	AF NRS	82	7	PH LS	18
7	AFPH FRS	82	8	LS	19
7	AFPH LS	82	8	PH NRS	19
10	Rectangular	83	8	AF UC	19
11	AF FRS	91	11	Narrow	20
12	PH NRS	92	12	Rectangular	22
13	PH FRS	95	13	NRS	23
14	AFPH UC	101	14	FRS	25
15	PH UC	105	15	UC	26
16	AF LS	109	15	PH UC	26
17	Narrow	112	17	AFPH LS	27
18	UC	114	18	AFPH UC	33
19	AF UC	116	19	Square	35

5. CONCLUSIONS

We have studied in this work the behavior of 19 different cellular genetic algorithms on a well-known benchmark for the SAT problem. These algorithms use both synchronous and asynchronous update policies. We use cellular algorithms to this end because algorithms using decentralized populations are often able to find the optimal solution of complex instances for which other GAs (without a decentralized population) can not find it [2].

We can conclude from this work that the static asynchronous algorithms are more efficient (AES) than the other ones, since NRS, LS, and FRS are the top three cGAs of our ranking. Conversely, the asynchronous adaptive algorithms (specially those using the AF criterion) are the most effective (SR) for the tested benchmark, although we are finding additional support for our conclusions in other instances of the SAT problem. Hence, we can conclude that the dynamic grid changes favor the exploration/exploitation balance of the algorithm, helping it to avoid getting stuck in local optima in many cases. In general, the tested asynchronous cGAs perform better than the synchronous ones

for the studied benchmark, since these synchronous algorithms are located close to or below the middle positions in the two rankings made.

An important advantage for the studied adaptive algorithms is that it is not necessary to set the population shape to any *ad hoc* value, because a dynamical recalculation of which is the most appropriate one is performed during the search. This also helps in reducing the overhead to a minimum.

Regarding to the studied synchronous algorithms, we can conclude that Square and Rectang are the most efficient algorithms (with very similar results), while Square is the worst of the three ones in terms of efficacy. This fact agrees with the assumption that non-square grids are often more efficient than square ones.

As a future work, it may be interesting to try some other criteria for the adaptation of the shape of the population in order to improve our results (e.g., a criterion based on the standard deviation of the population), or to compare the results with those of synchronous cGAs using the adaptive criteria. Another direct future extension of our work is to add to our cGAs some optimization procedures (often problem-dependent techniques) used in the literature for this problem.

Acknowledgements

This work has been funded by MCYT and FEDER under contract TIC2002-04498-C05-02 (the TRACER project) `http://tracer.lcc.uma.es`.

6. REFERENCES

[1] E. Alba and B. Dorronsoro. The exploration/exploitation tradeoff in dynamic cellular evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 9(2):126–142, April 2005.

[2] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, October 2002.

[3] E. Alba and J. M. Troya. Cellular evolutionary algorithms: Evaluating the influence of ratio. In M. Schoenauer et al., editor, *PPSN IV*, volume 1917 of *LNCS*, pages 29–38. Springer-Verlag, 2000.

[4] T. Bäck, A. Eiben, and M. Vink. A superior evolutionary algorithm for 3-SAT. In *International Conference on Evolutionary Programming, in coop. with IEEE NN Council*, volume 1477 of *LNCS*. Springer-Verlag, 1998.

[5] S. Cook. The complexity of theorem-proving procedures. *In Proc. of the 3rd Anual ACM Symp. on the Theory of Computing*, pages 151–158, 1971.

[6] B. Dorronsoro, E. Alba, M. Giacobini, and M. Tomassini. The influence of grid shape and asynchronicity on cellular evolutionary algorithms. In *IEEE CEC04*, pages 2152–2158, Portland, Oregon, 2004. IEEE Press.

[7] A. Eiben and J. van der Hauw. Solving 3-SAT with adaptive genetic algorithms. In *IEEE CEC97*, pages 81–86. IEEE Press, 1997.

[8] G. Folino, C. Pizzuti, and G. Spezzano. Parallel hybrid method for SAT that couples genetic algorithms and local search. *IEEE Transactions on Evolutionary Computation*, 5:323–334, Aug. 2001.

[9] M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-completeness*. Freeman, San Francisco, CA, 1979.

[10] M. Giacobini, E. Alba, A. Tettamanzi, and M. Tomassini. Modeling selection intensity for toroidal cellular evolutionary algorithms. In K. Deb, editor, *GECCO04*, volume 3102 of *LNCS*, pages 1138–1149, Seattle, Washington, 2004. Springer Verlag, Berlin.

[11] M. Giacobini, E. Alba, and M. Tomassini. Selection intensity in asynchronous cellular evolutionary algorithms. In E. Cantú-Paz et al., editor, *GECCO03*, pages 955–966. Springer Verlag, Berlin, 2003.

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

[13] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions for SAT problems. In P. Rosenbloom and P. Szolovits, editors, *10th National Conf. on AI*, pages 459–465, California, 1992. AAAI Press.

[14] G. Rudolph. On takeover times in spatially structured populations: Array and ring. In K. K. Lai, O. Katai, M. Gen, and B. Lin, editors, *Proceedings of theSecondAsia-Pacific Conference on Genetic Algorithms and Applications(APGA '00)*, pages 144–151, Hong Kong, PR China, 2000. Global-Link Publishing Company.

[15] J. Sarma and K. D. Jong. An analysis of the effect of the neighborhood size and shape on local selection algorithms. In H. Voigt, W. Ebeling, I. Rechenberg, and H. Schwefel, editors, *PPSN IV*, volume 1141 of *LNCS*, pages 236–244. Springer, 1996.

[16] J. Sarma and K. D. Jong. An analysis of local selection algorithms in a spatially structured evolutionary algorithm. In T. Bäck, editor, *ICGA97*, pages 181–186. Morgan Kaufmann, 1997.

[17] B. Schönfisch and A. de Roos. Synchronous and asynchronous updating in cellular automata. *BioSystems*, 51:123–143, 1999.