2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

A Simple Cellular Genetic Algorithm for Continuous Optimization

Bernabé Dorronsoro, and Enrique Alba

Abstract— **Cellular genetic algorithms (cGAs) are a kind of genetic algorithm (GA) –population based heuristic– with a structured population so that individuals can only interact with their neighbors. The existence of small overlapped neighborhoods in this decentralized population provides both diversity and exploration, while the exploitation of the search space is strengthened inside each neighborhood. This balance between exploration and exploitation makes cGAs naturally suitable for solving complex problems. In this paper we tackle the minimization of a number of problems (both academic and from the real world) with a real-coded cGA, called JCell. The results show that JCell improves the compared algorithms for a number of the studied problems, thus increasing the overall performance with respect to other complex heterogeneous distributed GAs, belonging to the state-of-the-art in continuous optimization.**

I. Introduction

Cellular Genetic Algorithms (cGAs) [1] are a kind of decentralized Genetic Algorithms (GAs) in which the population is structured in such way that the interaction of the individuals composing it is limited to a certain subset of individuals in the immediate vicinity of their location. Usually, the population is arranged in a 2-D toroidal mesh in cGAs, and individuals are allowed to interact only with their nearby neighbors (see Figure 1). This endows the cGA with useful properties for the optimization [2] and also facilitates a parallel implementation because of its inherent parallel design [3], [4].

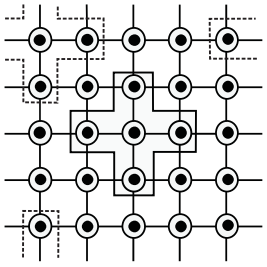

Fig. 1. Structure of the population in a cGA

In recent years, cGAs have been successfully applied to very complex (NP-hard) combinatorial optimization problems such as the Vehicle Routing (VRP) and the Satisfiability (SAT) problems [5], [6], becoming part of the state-of-the-art algorithms for those problems. Additionally, really competitive results have been also reported by cGAs for other problems belonging to a number of different fields such as logistics, telecommunications, scheduling, or academic, and holding very different features such as epistasis, multimodality, deceptiveness, parameter identification, constraints, or even multi-objective optimization [7], [8].

In [9] a first novel approach for tackling problems in the continuous search space with cGAs was made. In that work, a cGA using an arithmetic crossover (AX) and uniform mutation (UM) was applied to three real-encoded problems (Rastrigin, Ackley and the Fractal function) with promising results. The motivation of this work is to deep in the study of the behavior of cGAs when solving real coded problems. For that, we have selected a large benchmark of problems commonly used in the literature, having larger sizes (and thus higher complexity) than those studied in [9], and compare the results versus those of some of the best approaches in the literature at our known. One of the main contributions of this paper is that we highly improve the results reported in [9]; moreover, our simple cGA (called JCell), which implements two well-known recombination and crossover operators, reports highly competitive results, or even better in several cases, than the compared (complex) algorithms, belonging to the state-of-the-art in continuous optimization.

The structure of the paper is the following. In Section II we describe in detail the functioning of our cGA (called JCell). Section III presents the benchmark of problems we study in the current work. The results obtained, together with the parameterization of the algorithm, are presented in Section IV, and they are compared versus other algorithms belonging to the state-of-the-art for these same problems. Finally, we give our main conclusions and some future research lines in Section V.

II. Canonical Cellular Genetic Algorithm

In this section, we include a pseudocode of the canonical cGA search model and an explanation of the algorithm. In Algorithm 1 we can find a high-level description of a canonical cGA for a 2-D grid of size `HEIGHT`×`WIDTH` and for formally simultaneous update of all the cells (individuals). As it can be seen, a cGA starts with the cells 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 from the individual's neighborhood (including the individual itself), applying the genetic operators to them, and finally replacing the individual following a given replacement policy. Cells can be updated *synchronously* or *asynchronously* [9]. 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. In this work we focuss on synchronous cGAs. It can be seen in Algorithm 1 how the current individuals or the newly generated ones are

Enrique Alba and Bernabé Dorronsoro are with the Department of Languages and Computer Science, University of Málaga, Málaga, Spain (emails: bernabe,eat@lcc.uma.es).

0-7803-9487-9/06/$20.00/©2006 IEEE

placed in a in a new auxiliary population following a given replacement policy. After applying the reproductive cycle to all the individuals, the current population is replaced by the auxiliary one for the next generation. A *time step* (or generation) is defined as the updating of all the cells in the grid, which corresponds to one iteration of the `while` loop in Algorithm 1.

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;
```

III. Problems

We present in this section the *test suite* we use for evaluating our algorithm. We have chosen the same set of problems studied in [10], in order to easily compare our algorithms with some of the state-of-the-art in this field. The benchmark is composed of six classical and well known academic functions (presented in Section III-A), plus three additional problems taken from the real world (described in Section III-B). We consider that this benchmark is wide and diversified enough for sustaining our claims when evaluating the studied algorithmic approaches.

A. Test Functions

As stated before, six academic problems for continuous optimization are presented in this section. They are *Griewangk's* function (f_{Gri}) [11], *generalized Rastrigin's* function (f_{Ras}) [12], [13], *generalized Rosenbrock's* function (f_{Ros}) [14], *Schwefel's problem 1.2* (f_{Sch}) [15], the *Sphere* model (f_{Sph}) [14], [15], and *expansion of* f_{10} (ef_{10}) [16]. The selected set of minimization problems [10] incorporate problems characterized by many different features (linear and nonlinear, unimodal and multimodal, scalability, convexity, etc.), and allows us to make possible our comparison purposes with other algorithms in the literature. As in the case of [10], the considered dimension of the search space is 10 for $f_{\text{ef}_{10}}$ and 25 for the rest of the problems. All the details on these functions are given in Table I.

B. Real World Problems

Like in the case of [10], we include in our studies three real world problems for better assessing our conclusions. These problems are the *frequency modulation sound parameter identification problem* (f_{fms}) [17], *systems of linear equations* (f_{Sle}) [18] and a *polynomial fitting problem* (f_{Cheb}) [19].

1) Frequency modulation sound parameter identification problem: This problem is defined as determining the 6 real parameters $\vec{x} = (a_1, w_1, a_2, w_2, a_3, w_3)$ of the frequency modulated sound model given in (1) for approximating it to the sound wave given in (2) (where $\theta = 2 \cdot \pi/100$). The parameters are defined in the range $[-6.4, +6.35]$.

$$y(t) = a_1 \cdot \sin(\omega_1 \cdot t \cdot \theta + a_2 \cdot \sin(\omega_2 \cdot t \cdot \theta + a_3 \cdot \sin(\omega_3 \cdot t \cdot \theta))) \ , \quad (1)$$

$$y_0(t) = 1.0 \cdot \sin(5.0 \cdot t \cdot \theta - 1.5 \cdot \sin(4.8 \cdot t \cdot \theta + 2.0 \cdot \sin(4.9 \cdot t \cdot \theta))) \ . \quad (2)$$

The goal is to minimize the sum of square errors given by (3). This problem is a highly complex multimodal function having strong epistasis, with optimum value 0.0.

$$f_{\text{fms}}(\vec{x}) = \sum_{t=0}^{100} \Big(y(t) - y_0(t) \Big)^2 \ . \quad (3)$$

2) Systems of linear equations: This problem consists in finding the values for the vector $\vec{x}$ such that $A\vec{x} = \vec{b}$, with:

$$A = \begin{pmatrix} 5,4,5,2,9,5,4,2,3,1 \\ 9,7,1,1,7,2,2,6,6,9 \\ 3,1,8,6,9,7,4,2,1,6 \\ 8,3,7,3,7,5,3,9,9,5 \\ 9,5,1,6,3,4,2,3,3,9 \\ 1,2,3,1,7,6,6,3,3,3 \\ 1,5,7,8,1,4,7,8,4,8 \\ 9,3,8,6,3,4,7,1,8,1 \\ 8,2,8,5,3,8,7,2,7,5 \\ 2,1,2,2,9,8,7,4,4,1 \end{pmatrix} ; \ \vec{b} = \begin{pmatrix} 40 \\ 50 \\ 47 \\ 59 \\ 45 \\ 35 \\ 53 \\ 50 \\ 55 \\ 40 \end{pmatrix} \quad (4)$$

The evaluation function we minimize in our experiments is shown in (5). This function has the optimal solution $f_{\text{sle}}(\vec{x^*}) = 0.0$, and the values of the ten variables of the problem range into the interval $[-9.0, 11.0]$.

$$f_{\text{Sle}}(\vec{x}) = \left| \sum_{i=1}^{n} \sum_{j=1}^{n} (a_{ij} \cdot x_j) - b_i \right| \ . \quad (5)$$

3) Polynomial fitting problem: For this problem, the objective is to find the coefficients of the following polynomial in z:

$$P(z) = \sum_{j=0}^{2k} c_j z^j, \ \ k \in \mathbb{Z}^+ \quad (6)$$

such that $\forall z^j \in [-1, +1]$, $P(z) \in [-1, +1]$, $P(+1.2) \geq T_{2k}(+1.2)$, and $P(-1.2) \geq T_{2k}(-1.2)$, where $T_{2k}(z)$ is a f_{Cheb} polynomial of degree $2k$.

The solution to the polynomial fitting problem consists of the coefficients of $T_{2k}(z)$. This polynomial oscillates between -1 and +1. Outside this region, the polynomial rises steeply in the direction of high positive ordinate values. This problem has its roots in electronic filter design, and it challenges an optimization procedure by forcing it to find parameter values with grossly different magnitudes, something that is very common in industrial systems. The f_{Cheb} polynomial employed here is:

$$T_8(z) = 1 - 32z^2 + 160z^4 - 256z^6 + 128z^8 \ . \quad (7)$$

TABLE I
BENCHMARK OF ACADEMIC FUNCTIONS

Problem	Fitness function	Size (n)	Optimum	Variable value range
Griewangk	$f_{\text{Gri}}(\vec{x}) = \frac{1}{d}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$ $d = 4000$	25	$f_{\text{Gri}}(x^*) = 0.0$	$-600.0 \leq x_i \leq 600.0$
Rastrigin	$f_{\text{Ras}}(\vec{x}) = a \cdot n + \sum_{i=1}^{n} x_i^2 - a \cdot \cos(\omega \cdot x_i)$ $a = 10,\ \omega = 2\pi$	25	$f_{\text{Ras}}(x^*) = 0.0$	$-5.12 \leq x_i \leq 5.12$
Rosenbrock	$f_{\text{Ros}}(\vec{x}) = \sum_{i=1}^{n-1} (100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	25	$f_{\text{Ros}}(x^*) = 0.0$	$-5.12 \leq x_i \leq 5.12$
Schwefel	$f_{\text{Sch}}(\vec{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2$	25	$f_{\text{Sch}}(x^*) = 0.0$	$-65.536 \leq x_i \leq 65.536$
Sphere	$f_{\text{Sph}}(\vec{x}) = \sum_{i=1}^{n} x_i^2$	25	$f_{\text{Sph}}(x^*) = 0.0$	$-5.12 \leq x_i \leq 5.12$
$\mathbf{ef_{10}}$	$f_{\text{ef}_{10}}(\vec{x}) = f_{10}(x_1, x_2) + \ldots + f_{10}(x_{i-1}, x_i) + \ldots + f_{10}(x_n, x_1)$ $f_{10}(x, y) = (x^2 + y^2)^{0.25} \cdot [\sin^2(50 \cdot (x^2 + y^2)^{0.1}) + 1]$	10	$f_{\text{ef}_{10}}(x^*) = 0.0$	$-100.0 < x_i \leq 100.0$

It is a nine-parameter problem ($\vec{x} = [x_1, \ldots, x_9]$). A small correction is needed in order to transform the constraints of this problem into an objective function to be minimized, called f_{Cheb} (see [10] for all the details). Each parameter (coefficient) is in the range [-5.12,+5.12]. The objective function value of the optimum is $f_{\text{Cheb}}(\vec{x^*}) = 0.0$.

IV. EXPERIMENTATION

In this section we present the results of solving all the previously presented problems with JCell (our cGA) and compare them with those of other state-of-the-art algorithms. We use in this work the same parameterization for all the problems. As it can be seen in Table II, we use a population of 144 individuals, arranged in a 2-D grid of 12×12 cells. The neighborhood used is the so called NEWS (or linear5), and contains the considered individual –at position (x,y)– plus those located at its north, east, west, and south positions (as shown in Figure 1). The two parents are selected by using *binary tournament*, and they are forced to be different.

Offsprings are obtained by applying the *Blend Recombination Operator*, BLX-α [20] (with $\alpha = 0.5$), to two selected parents. The offspring $\vec{z}$ generated by BLX-α is composed of genes z_i randomly (uniformly) chosen from the interval $[\min(x_i, y_i) - I \cdot \alpha, \max(x_i, y_i) + I \cdot \alpha]$, where $I = \max(x_i, y_i) - \min(x_i, y_i)$, being x_i and y_i are the *i*th genes of the two parents $\vec{x}$ and $\vec{y}$, respectively. This recombination is accomplished with probability $p_c = 1.0$, and we apply to the resulting offspring the *Non-Uniform Mutation* operator, considered one of the most suitable for real-coded GAs [21]. This mutation operator is applied to the alleles of the offsprings with probability $p_m = 1/(2 \cdot n)$, where n is the size of the problem (length of the chromosome), and mutates them using a non uniform distribution, whom step size decreases as the execution advances. Thus, it makes an uniform search in the initial space (exploration stage) and very locally at a larger stage, favoring local exploitation. Finally, we use an elitist replacement policy (*Replace if Better*), in which the offspring replaces the current individual for the population of the next generation only if it is better (lower fitness value). The algorithm stops after making 800000 evaluations, the same limit used in [10].

TABLE II
PARAMETERIZATION

Population Size	144 individuals (12×12)
Neighborhood	NEWS
Selection of Parents	BT + BT
Recombination	BLX-α; $\alpha = 0.5$
Probability of Recombination	$p_c = 1.0$
Mutation	Non Uniform Mutation
Probability of Mutation	$p_m = 1/(2 \cdot n)$
Replacement	Replace if Better
Termination Condition	800000 fitness evaluations

For the experiments in this paper, 50 independent runs were made for every problem and algorithm. In order to obtain statistical significance in our comparisons we perform ANOVA or Kruskal-Wallis tests on the results (depending on whether the data are normally distributed or not, respectively). The normality of the data distribution is checked by using the Kolmogorov-Smirnov test. Matlab(c) was used for this statistical significance study. We have included this kind of analysis in response to the increasing necessity of mathematical formalisms in present metaheuristic studies.

A. Tuning the Algorithm

We proceed in this section to adjust the values for some parameters of the cGA in order to improve the behavior of the algorithm as possible. The base parameterization is that described in Table II (algorithm JCell), and we test the algorithm with two different mutation probabilities ($p_m = 2/n$ and $p_m = 1/n$), two different values for α ($\alpha = 0.25$ and $\alpha = 0.75$), and two recombination probabilities ($p_c = 0.5$ and $p_c = 0.75$). The best values obtained for every problem with the 7 cGAs are shown in Table III. If the global optimum has been located throughout some runs, we will present the percentage of runs in which this happens (in this case, a '%' symbol appears just after the figure in the table). Additionally, the average obtained results, together with the typical deviations, are given in Table IV. The best values in the two tables for each problem are **bolded**.

Regarding Table III, we can see that JCell (using the parameterization of Table II) is the most accurate of the tested algorithms since it obtains the best results for 6 out of the

TABLE III

COMPUTATIONAL RESULTS. BEST SOLUTIONS

Algorithms	f_{fms}	f_{Gri}	f_{Ras}	f_{Ros}	f_{Sch}	f_{Sph}	$f_{\text{ef}_{10}}$	f_{Sle}	f_{Cheb}
JCell	4.47E-27	**90%**	**80%**	1.331E0	**9.283E-161**	**1.718E-158**	**90%**	**26%**	1.284E3
$p_m = \frac{2}{n}$	2.946E-27	**78%**	**72%**	1.366E1	1.566E-36	8.136E-37	4.38E-17	**50%**	1.249E3
$p_m = \frac{1}{n}$	9.139E-27	**80%**	**68%**	9.873E-2	1.63E-101	6.891E-98	2.198E-56	**30%**	1.264E3
α = 0.25	1.377E-4	3.035E-4	7.317E-5	1.746E1	2.306E-5	6.437E-8	1.575E-8	**66%**	1.292E3
α = 0.75	**3.511E-36**	**48%**	**80%**	**4.538E-3**	1.32E-100	4.446E-98	1.907E-71	**42%**	**1.166E3**
p_c =0.50	4.456E-7	**2%**	2.923E-7	9.917E-1	8.515E-30	6.342E-23	1.335E-6	**14%**	1.225E3
p_c =0.75	1.25E-18	**60%**	**40%**	1.385E0	4.507E-134	1.298E-130	6.595E-80	**50%**	1.284E3

TABLE IV

COMPUTATIONAL RESULTS. AVERAGE SOLUTIONS

Algorithms	f_{fms}	f_{Gri}	f_{Ras}	f_{Ros}	f_{Sch}	f_{Sph}	$f_{\text{ef}_{10}}$	f_{Sle}	f_{Cheb}
JCell	9.058E0	**2.381E-3**	2.734E-5	1.781E1	**6.58E-158**	**1.495E-154**	**1.368E-3**	4.316E-10	1.392E3
	$\pm 6.654E0$	$\pm 4.75E-3$	$\pm 9.299E-6$	$\pm 1.089E1$	$\pm 2.175E-157$	$\pm 5.474E-154$	$\pm 5.804E-3$	$\pm 2.364E-9$	$\pm 4.799E1$
$p_m = \frac{2}{n}$	**2.106E0**	3.038E-3	9.112E-5	2.368E1	7.517E-35	2.165E-34	1.141E-2	1.572E-5	1.414E3
	$\pm 4.566E0$	$\pm 6.123E-3$	$\pm 2.822E-4$	$\pm 1.381E1$	$\pm 1.12E-34$	$\pm 5.125E-34$	$\pm 2.965E-2$	$\pm 1.112E-4$	$\pm 6.136E1$
$p_m = \frac{1}{n}$	4.068E0	3.532E-3	**7.312E-6**	2.139E1	2.302E-98	1.223E-95	4.826E-3	4.836E-10	1.401E3
	$\pm 5.787E0$	$\pm 5.401E-3$	$\pm 2.97E-5$	$\pm 1.355E1$	$\pm 9.126E-98$	$\pm 4.25E-95$	$\pm 1.153E-2$	$\pm 3.419E-9$	$\pm 5.839E1$
α = 0.25	7.658E0	1.536E-2	4.916E-4	2.272E1	9.668E-2	2.094E-6	9.683E-2	2.383E-5	1.423E3
	$\pm 6.491E0$	$\pm 1.366E-2$	$\pm 5.044E-4$	$\pm 7.447E0$	$\pm 9.666E-2$	$\pm 1.721E-6$	$\pm 1.052E-1$	$\pm 1.685E-4$	$\pm 5.706E1$
α = 0.75	7.746E0	8.778E-3	3.168E-5	**1.126E1**	2.869E-97	4.871E-95	3.85E-3	**1.776E-14**	**1.382E3**
	$\pm 7E0$	$\pm 1.037E-2$	$\pm 9.898E-5$	$\pm 1.353E1$	$\pm 8.308E-97$	$\pm 2.682E-94$	$\pm 9.981E-3$	$\pm 1.682E-14$	$\pm 6.646E1$
p_c = 0.50	7.976E0	8.961E-3	1.005E-4	3.299E1	1.844E-3	2.734E-8	2.96E-1	4.465E-5	1.391E3
	$\pm 7.007E0$	$\pm 1.28E-2$	$\pm 1.34E-4$	$\pm 2.674E1$	$\pm 1.073E-2$	$\pm 8.178E-8$	$\pm 4.956E-1$	$\pm 2.957E-4$	$\pm 6.151E1$
p_c =0.75	9.405E0	6.808E-3	2.848E-5	2.672E1	2.157E-10	2.645E-14	3.375E-2	1.487E-7	1.385E3
	$\pm 7.461E0$	$\pm 9.7E-3$	$\pm 8.529E-5$	$\pm 2.065E1$	$\pm 1.525E-9$	$\pm 1.87E-13$	$\pm 5.807E-2$	$\pm 8.85E-7$	$\pm 4.686E1$
p-values	+	+	+	+	+	+	+	+	+

TABLE V

COMPUTATIONAL RESULTS. AVERAGE EXECUTION TIMES (SECONDS)

Algorithms	f_{fms}	f_{Gri}	f_{Ras}	f_{Ros}	f_{Sch}	f_{Sph}	$f_{\text{ef}_{10}}$	f_{Sle}	f_{Cheb}
JCell	$\mathbf{132.79}_{\pm \mathbf{2.90}}$	$\mathbf{7.39}_{\pm \mathbf{7.70}}$	$\mathbf{9.63}_{\pm \mathbf{5.46}}$	$24.87_{\pm 0.22}$	$\mathbf{22.43}_{\pm \mathbf{0.20}}$	$18.31_{\pm 0.18}$	$24.93_{\pm 0.37}$	$10.41_{\pm 5.63}$	$540.96_{\pm 1.05}$
$p_m = \frac{2}{n}$	$140.17_{\pm 1.69}$	$15.97_{\pm 4.45}$	$19.08_{\pm 2.65}$	$\mathbf{24.69}_{\pm \mathbf{0.34}}$	$23.49_{\pm 0.16}$	$19.26_{\pm 0.16}$	$27.16_{\pm 0.13}$	$7.35_{\pm 4.14}$	$\mathbf{484.21}_{\pm \mathbf{0.82}}$
$p_m = \frac{1}{n}$	$137.74_{\pm 2.48}$	$9.87_{\pm 7.67}$	$11.94_{\pm 5.18}$	$29.61_{\pm 0.42}$	$26.97_{\pm 0.12}$	$17.70_{\pm 0.11}$	$27.13_{\pm 0.52}$	$7.70_{\pm 3.68}$	$565.67_{\pm 1.99}$
α = 0.25	$136.50_{\pm 2.71}$	$20.80_{\pm 0.18}$	$19.72_{\pm 0.11}$	$29.03_{\pm 0.21}$	$26.68_{\pm 0.25}$	$17.20_{\pm 0.14}$	$26.70_{\pm 0.25}$	$\mathbf{5.65}_{\pm \mathbf{4.40}}$	$564.54_{\pm 1.78}$
α = 0.75	$135.68_{\pm 3.58}$	$11.84_{\pm 8.13}$	$10.60_{\pm 6.06}$	$29.73_{\pm 0.54}$	$25.84_{\pm 0.14}$	$17.47_{\pm 0.19}$	$26.40_{\pm 0.07}$	$6.88_{\pm 3.96}$	$564.40_{\pm 1.89}$
p_c = 0.50	$135.57_{\pm 3.35}$	$17.86_{\pm 2.75}$	$17.52_{\pm 0.10}$	$27.03_{\pm 0.41}$	$23.47_{\pm 0.11}$	$\mathbf{14.97}_{\pm \mathbf{0.11}}$	$\mathbf{23.72}_{\pm \mathbf{0.21}}$	$7.40_{\pm 2.41}$	$562.64_{\pm 1.33}$
p_c =0.75	$139.05_{\pm 2.97}$	$11.11_{\pm 8.16}$	$14.63_{\pm 5.40}$	$28.24_{\pm 0.47}$	$25.14_{\pm 0.14}$	$16.21_{\pm 0.22}$	$24.22_{\pm 0.64}$	$7.07_{\pm 4.50}$	$563.36_{\pm 2.34}$
p-values	+	+	+	+	+	+	+	+	+

9 tested problems. Additionally, JCell also obtains the best overall behavior in terms of the average fitness values found in the 50 runs (4 out of the 9 problems), as it can be seen in Table IV. In the other 5 problems, the results reported by JCell have the same order of magnitude than the best value of the compared cGAs, with the exception of f_{Sle}. There exists statisticall confidence in the results of the compared cGAs for all the problems.

From tables III and IV we can see that changing the probabilities of recombination and mutation, and the value of α from the parameterization of the algorithm proposed in Table II does not lead us to improve the behavior of the algorithm, in general. The most distinguishable improvements are those of f_{Ros} and f_{Sle} when using $\alpha = 0.75$, and f_{fms} when increasing the mutation probability to $2/n$. From these three cases, only f_{fms} with $p_m = 2/n$ is better than JCell with statistical confidence. Hence, we conclude that JCell is the best one of the proposed algorithms and thus we select it for our comparisons with other approaches in the literature in the next section.

In order to make a deeper study of the proposed algorithms, we also include a comparison on the average run times (in seconds) performed by the algorithms for the proposed problems in Table V (the best –lower– result for each problem is bolded). As it can be seen, there exist statistically significant differences for every problem in the reported values. Thus, JCell is the fastest of the compared algorithms for 4 out of the nine problems (statistical differences were found for f_{fms}, f_{Gri}, and f_{Sch}), while there exist algorithms faster than JCell with statistical significance in only 3 out of all the studied problems (f_{Sph}, $f_{\text{ef}_{10}}$, and f_{Sle}). The most important differences in the values reported in Table V are due to the distinct hit rates obtained, since the algorithm stops before reaching 800000 evaluations when the optimum is found. As an example, JCell is between 2 and 3 times faster than the algorithm with $\alpha = 0.25$ when the optimal solution is found by the two algorithms, what is contradictory to the results of Table V, reporting 10.41 seconds for JCell and 5.65 seconds for the other algorithm (with $\alpha = 0.25$) in average. The reason is that the latter finds the optimum in 66% of the runs while the hit rate of the former is only 26% (see Table III).

B. Comparison with Other Algorithms

In this section we compare JCell with other algorithms belonging to the state-of-the-art in continuous optimization.

TABLE VI

COMPARISON WITH OTHER APPROACHES: ACADEMIC PROBLEMS

Algorithms	f_{Gri}		f_{Ras}		f_{Ros}		f_{Sch}		f_{Sph}		$f_{\mathrm{ef}_{10}}$	
	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best
JCell	**2.4E-3**	**90%**	2.7E-5	80%	1.8E1	1.3E0	**6.6E-158**	**9.3E-161**	**1.5E-154**	**1.7E-158**	1.4E-3	**90%**
GD-FCB	2E-2	4E-11	4E-11	9E-12	**9E0**	**3E-5**	4E0	6E-1	2E-13	4E-14	2E-3	8E-4
GD-BLX	5E-3	60.0%	**0E0**	**100%**	2E1	2E1	2E-6	7E-8	8E-53	8E-58	**6E-36**	5E-48
GD-EFR	7E-3	53.3%	**0E0**	**100%**	2E1	1E1	4E-6	2E-7	4E-47	8E-51	9E-35	6E-47

TABLE VII

COMPARISON WITH OTHER APPROACHES: REAL WORLD PROBLEMS

Algorithms	f_{fms}		f_{Sle}		f_{Cheb}	
	Avg.	Best	Avg.	Best	Avg.	Best
JCell	9.1E0	4.5E-27	**4.3E-10**	**26%**	1.4E3	1.3E3
GD-FCB	1E1	7E-26	4E1	3E0	**2E2**	4E1
GD-BLX	**3E0**	**66.7%**	3E1	2E0	**2E2**	**1E1**
GD-EFR	6E0	43.3%	2E1	9E-1	**2E2**	**1E1**

These algorithms are the gradual distributed real-coded GAs proposed in the work of Herrera and Lozano [10]. The problems studied in that work are the same of our benchmark, as well as the dimension of the search space used for each problem. Additionally, the termination condition of these compared algorithms is the same used in this work: achieving 800000 fitness function evaluations.

The three compared algorithms [10] are heterogeneous distributed real-coded GAs differing in the recombination operator used. They are based on an hypercube topology with three dimensions and there is a population of 20 individual in each vertex of the cube. Each subpopulation implements a different parameterization of the crossover operator such that those populations in the front side of the cube are devoted to exploration, while exploitation is enhanced in the crossover operators of the rear side populations. With a given frequency, there exist migrations of individuals between populations belonging to the same sides of the cube. The three heterogeneous distributed GAs presented in [10] are GD-FCB, which use the *Fuzzy Connectives Based Crossover Operator* (FCB) [22], GD-BLX, implementing the *Blend Crossover Operator* (BLX-α) [20], and GD-EFR, using the *Extended Fuzzy Recombination Operator* (EFR) [23]. The reader is referred to [10] for a complete description of this algorithmic model.

The results of all these algorithms, together with the results of JCell are given in tables VI (for the academic problems) and VII for the three problems taken from the real world.

JCell outperforms the other approaches in 3 out of the 6 academic functions of Table VI, with the important differences of more than 150 orders of magnitude for f_{Sch} and f_{Sph} (in average and best solutions) with respect to the best values reported by the other algorithms. Moreover, in the case of $f_{\mathrm{ef}_{10}}$, JCell finds the optimal solution in the 90% of the runs while the other three approaches are not able to find the optimal solution in any run (the best approach is 5E-48 for GD-BLX). The bad average value reported by JCell for this problem is because it gets stuck in local optima (value about 10^{-2}) in 10% of the runs. Additionally, we do not consider important the existing differences between JCell, GD-BLX, and GD-EFR for f_{Ras} since JCell finds the local optimum in 80% of the runs (versus 100% for GD-BLX and GD-FER). Finally, the average solution found by JCell for f_{Ros} is almost in the same order of magnitude than that of the best algorithm, GD-FCB.

Regarding the proposed problems based on the real world (see Table VII), JCell outperforms the other algorithms for f_{Sle}, obtaining an average result 11 orders of magnitude better than the compared algorithms. Moreover, only JCell is able to find the optimal solution for this problem, and it finds it in the 26% of the runs. In the case of f_{fms}, the average result thrown by all the algorithms is in the same order of magnitude, and the best solution found by JCell is close to the optimum (4.5E-27), although GD-BLX and GD-EFR found the optimal solution in the 66.7% and 43.3% of the runs, respectively. Finally, f_{Cheb} is the unique problem for which JCell is the worst of the compared algorithms both in terms of best and average solutions found. However, the differences between JCell and the other algorithms is only one order of magnitude in the average of the solutions found.

Finally, we can summarize these results concluding that JCell clearly outperforms the three heterogenous distributred approaches in 4 out of the 9 tested instances, improving the current state-of-the-art, and it is very close to the results of the best algorithms in the other problems. Maybe, the exception is f_{Cheb}, for which JCell reports the worst results of the compared algorithms, although the differences are only of one order of magnitude.

At this point, we would like to emphasize the simplicity of our approach versus the three distributed ones. We are using in this work a simple canonical cGA for solving the problems, while the compared algorithms are heterogeneous distributed GAs composed of 8 different subpopulations having each of them distinct parameterizations, with the consequent increment in the parameters needed by the algorithms. Moreover, our approach uses exactly the same recombination and mutation operators implemented in GD-BLX, so the difference in the performance of the two algorithms is intrinsic to the algorithmic model. Hence, the exploration/exploitation tradeoff accomplished by the canonical

cGA on the population, achieved by simply introducing the concept of neighborhood in a structured grid of individuals (for an extended work in this topic the reader is referred to [7]), reports similar results, and in several cases even better, with respect to the complex hypercube model that Herrera and Lozano propose in [10].

In figures 2 and 3 we plot the evolution of the average fitness value of the individuals in the population during typical executions of JCell for all the studied problems. Note that this average fitness value (ordinate axis) is in logarithmic scale. For the functions plotted in Figure 2, it can be seen in general a fast approximation to the best solution of the problem during the execution, meaning a progressive convergence of the population. Among these problems, it stands out the difficulty of f_{fms}, which reports the slowest convergence. After increasing the maximum number of allowed evaluations highly enough, we have checked that this convergence is maintained until the optimum is found in most runs for the functions in Figure 2. The exception is f_{fms}, which falls into local optima from which the algorithm can not escape in some runs.

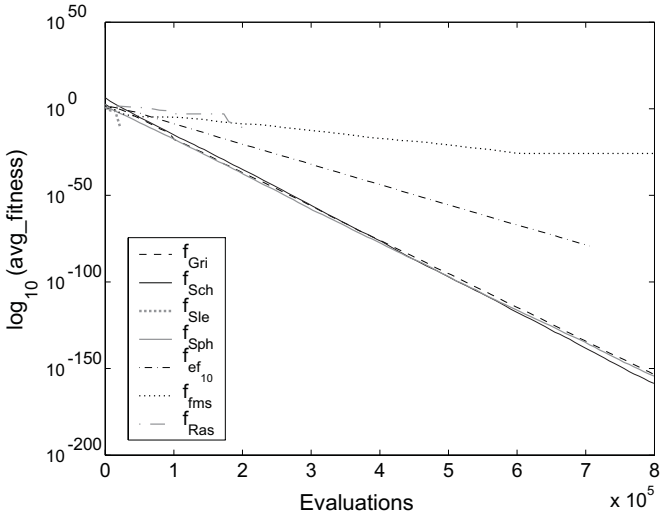

Fig. 2. Evolution of the average fitness value for f_{Gri}, f_{Sch}, f_{Sle}, f_{Sph}, $f_{\mathrm{ef}_{10}}$, f_{fms} and f_{Ras}

The most difficult problems for JCell among those studied in this work are f_{Cheb}, and f_{Ros}. The two problems are plotted in Figure 3. It can be seen how the algorithm experiments a really fast convergence at the begining of the search (notice that the average fitness value is plotted in logarithmic scale) but, after that, it gets stuck in local optima, the population diversity is quickly lost, and thus there are no more important improvings in the solutions.

V. Conclusions and Further work

We have proposed in this paper a new approach (called JCell) for solving problems in a continuous search space. Our proposal consists of a simple canonical cGA using two well-known recombination and mutation operators in the field of real-coded GAs.

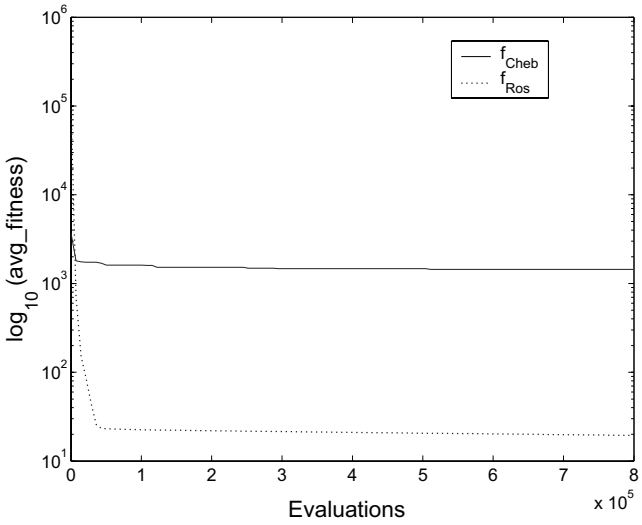

Fig. 3. Evolution of the average fitness value for f_{Cheb}, and f_{Ros}

The results reported in this first approach are really promising, since JCell outperformed other existing complex heterogeneous distributed algorithms belonging to the state-of-the-art. Additionally, one of these algorithms (GD-BLX) implements exactly the same recombination and mutation operators we use in JCell, what enhances the exploration/exploitation tradeoff that cGAs perform into the population simply by using the concept of neighborhood. In the case of the three compared heterogeneous GAs, this tradeoff between exploration and exploitation is accomplished by distributing the population in small subpopulations with different parameterizations in a hypercube, and restricting the interchange of individuals (migration) among subpopulations located in the same side of the hypercube. Looking the reported results, we can conclude that the complex hypercube model of the three compared algorithms does not perform a better exploration/exploitation tradeoff than JCell, which is based on concepts much simpler than the hypercube model and needs a lower number of parameters to be set (each subpopulation in the hypercube implements one different parameterization).

As further works, we propose to include the different techniques proposed by Alba and Dorronsoro in [7] for improving the exploration/exploitation tradeoff in cGAs. This will hopefully lead us to improve the current results. Additionally, it should be interesting to make a deeper study on the behavior of JCell for continuous optimization by testing some more different recombination and mutation operators, as well as hybridizing it with some local search method.

Acknowledgments

The authors acknowledge that this work has been partially funded by the Spanish MEC and FEDER under contract TIN2005-08818-C04-01 (the OPLINK project: http://oplink.lcc.uma.es).

REFERENCES

[1] B. Manderick and P. Spiessens, "Fine-grained parallel genetic algorithm," in *Proceedings of the Third International Conference on Genetic Algorithms*, J. Schaffer, Ed. Morgan-Kaufmann, 1989, pp. 428–433.

[2] M. Tomassini, *Spatially Structured Evolutionary Algorithms*, ser. Artificial Evolution in Space and Time Series: Natural Computing Series. Springer, 2005.

[3] E. Alba, *Parallel Metaheuristics: A New Class of Algorithms*, E. Alba, Ed. Wiley, 2005.

[4] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms*, 2nd ed., ser. Book Series on Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, 2000, vol. 1.

[5] E. Alba and B. Dorronsoro, "Solving the vehicle routing problem by using cellular genetic algorithms," in *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2004*, ser. LNCS, J. Gottlieb and G. R. Raidl, Eds., vol. 3004. Coimbra, Portugal: Springer Verlag, 5-7 April 2004, pp. 11–20.

[6] E. Alba, B. Dorronsoro, and H. Alfonso, "Advanced models of cellular genetic algorithms evaluated on SAT," in *Genetic and Evolutionary Computation COnference (GECCO)*, Washington, D.C. USA, June 2005, pp. 1123–1130.

[7] E. Alba and B. Dorronsoro, "The exploration/exploitation tradeoff in dynamic cellular evolutionary algorithms," *IEEE Trans. on Evolutionary Computation*, vol. 9, no. 2, pp. 126–142, April 2005.

[8] E. Alba, B. Dorronsoro, F. Luna, A. Nebro, P. Bouvry, and L. Hogie, "A cellular multi-objective genetic algorithm for optimal broadcasting strategy in metropolitan manets," *Computer Communications*, 2006, (to appear).

[9] E. Alba, B. Dorronsoro, M. Giacobini, and M. Tomasini, "Decentralized cellular evolutionary algorithms," in *Handbook of Bioinspired Algorithms and Applications*. CRC Press, 2005.

[10] F. Herrera and M. Lozano, "Gradual distributed real-coded genetic algorithms," *IEEE Trans. on Evolutionary Computation*, vol. 4, no. 1, pp. 43–63, April 200.

[11] A. O. Griewangk, "Generalized descent of global optimization," *J. Optim. Theory Appl.*, vol. 34, pp. 11–39, 1981.

[12] T. Bäck, "Self-adaptation in genetic algorithms," in *Proceedings of the 1st European Conference on Artificial Life*, F. J. Varela and P. Bourgine, Eds. Cambridge, MA: MIT Press, 1992, pp. 263–271.

[13] A. Töorn and Ž. Antanas, *Global Optimization*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer, 1989, vol. 350.

[14] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, University of Michigan, 1975, Ann Arbor.

[15] H.-P. Schwefel, *Numerical Optimization of Computer Models*. Chichester, England: Wiley, 1981.

[16] D. Whitley, R. Beveridge, C. Graves, and K. Mathias, "Test driving three 1995 genetic algorithms: New test functions and geometric matching," *Journal of Heuristics*, vol. 1, pp. 77–104, 1995.

[17] S. Tsutsui and Y. Fujimoto, "Forking genetic algorithm with blocking and shrinking modes," in *Proceedings of the 5th International Conference on Genetic Algorithms*, L. Eshelman, Ed. San Mateo, CA: Morgan Kaufmann, 1993, pp. 206–213.

[18] L. J. Eshelman, K. E. Mathias, and J. D. Schaffer, "Convergence controlled variation," in *Foundations of Genetic Algorithms 4*, R. Belew and M. Vose, Eds. San Mateo, CA: Morgan Kaufmann, 1989, pp. 203–224.

[19] R. Storn and K. Price, "Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces," Int. Comput. Sci. Inst., Berkeley, CA, Tech. Rep. TR-95-012, 1995.

[20] L. Eshelman and J. Schaffer, *Foundation of Genetic Algorithms 2*. San Mateo: Morgan Kaufmann, 1993, ch. RealCoded Genetic Algorithms and IntervalSchemata, pp. 187–202.

[21] F. Herrera, M. Lozano, and J. L. Verdegay, "Tackling real-coded genetic algorithms: Operators and tools for the behavioral analysis," *Artificial Intelligence Reviews*, vol. 12, no. 4, pp. 265–319, 1998.

[22] F. Herrera, E. Herrera-Viedma, M. Lozano, and J. Verdegay, "Fuzzy tools to improve genetic algorithms," in *Proc. Second European Congress on Intelligent Techniques and Soft Computing*, 1994, pp. 1532–1539.

[23] H.-M. Voigt, H. Mühlenbein, and D. Cetković, "Fuzzy recombination for the breeder genetic algorithm," in *Proceedings of the 6th International Conferenfe on Genetic Algorithms*, L. Eshelman, Ed., 1995, pp. 104–111.