Cellular Memetic Algorithms

Enrique Alba[1], **Bernabé Dorronsoro**[1], **and Hugo Alfonso**[2]
[1] Department of Computer Science, University of Málaga, Spain
[2]National University of La Pampa, General Pico, La Pampa, Argentina
E-mails: eat@lcc.uma.es; dorronsoro@uma.es; alfonsoh@ing.unlpam.edu.ar

ABSTRACT

This work is focussed on the development and analysis of a new class of algorithms, called cellular memetic algorithms (cMAs), which will be evaluated here on the satisfiability problem (SAT). For describing a cMA, we study the effects of adding specific knowledge of the problem to the fitness function, the crossover and mutation operators, and to the local search step in a canonical cellular genetic algorithm (cGA). Hence, the proposed cMAs are the result of including these hybridization techniques in different structural ways into a canonical cGA. We conclude that the performance of the cGA is largely improved by these hybrid extensions. The accuracy and efficiency of the resulting cMAs are even better than those of the best existing heuristics for SAT in many cases.

Keywords: Cellular evolutionary algorithms, memetic algorithms, SAT problem

1. INTRODUCTION

Evolutionary algorithms (EAs) are optimization techniques that work on a set (*population*) of potential solutions (*individuals*) by applying stochastic operators on them in order to search for an optimal solution. Most EAs use a single population (panmixia) of individuals and apply operators on them as a whole. In contrast, there exists also some tradition in using structured EAs, where the population is decentralized somehow. Among the many types of structured EAs, *distributed* and *cellular* algorithms are two popular optimization tools [1], [2] (see Fig. 1) . In many cases [3], these decentralized algorithms provide a better sampling of the search space, resulting in improved numerical behavior with respect to an equivalent algorithm in panmixia.

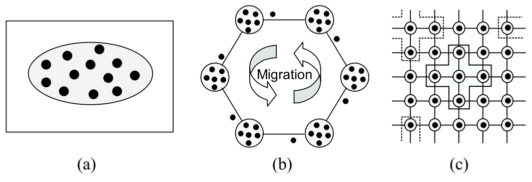

Fig. 1. Panmictic (a), distributed (b) and cellular (c) EAs.

Memetic Algorithms (MAs) are techniques combining features of different metaheuristics, such as population based algorithms (EAs) and local search methods, and also restart techniques and intensive hybridization with problem knowledge. The main characterization of MAs is the trade-off between exploration and exploitation of the search space that they perform. This work is focussed on cellular MAs (cMAs), a new class of algorithms we are introducing here, which are essentially cellular genetic algorithms [4], [5] (cGAs) wherein some knowledge of the problem is included in the recombination, mutation, local search, fitness function and/or the representation of the problem. Therefore, we will focus on cGAs in this paper, although the same methodology can be extended to other EAs. For testing our algorithms we have selected the satisfiability problem (SAT). This is a hard combinatorial problem, well-known in the literature, and having important practical applications, such as planning [6] and image interpretation [7], among others.

E. Alba and B. Dorronsoro acknowledge that this work has been partially funded by the Spanish MCYT and FEDER under contract TIN2005-08818-C04-01 (the OPLINK project).

In the field of evolutionary computation, the latest advances clearly show that EAs can yield good results for SAT when hybridizing them with additional techniques, e.g., adaptive fitness functions, problem-specific operators, or local optimization [8], [9], [10], [11], [12].

The motivation for this work is to study the behavior of different hybrid cGAs having generic or specific fitness functions and recombination, mutation, and local search operators. We compare them against the basic local search heuristics themselves, and versus two canonical cGAs (without local search). Additionally, two different ways of embedding local search have been analyzed: (1) a computationally light local search step applied to every individual, or (2) an in depth exploitation local search step applied with a low probability. This work is an extension to [13], including a preliminary study justifying the algorithms used, and a comparison of our results versus those of other well-known algorithms in the literature.

This paper is organized as follows. In Section 2 we define the SAT problem. Section 3 introduces the studied algorithms, including three simple heuristics (GRAD, Simulated Annealing –SA– and WSAT), a basic cGA, and several cMAs, which are the result of different combinations with the previous kinds of algorithms. Our results are summarized in Section 4, and the conclusions and future research directions are addressed in Section 5.

2. SATISFIABILITY PROBLEMS

The satisfiability problem (SAT) has received many attention from the scientific community since any NP problem can be translated into an equivalent SAT problem in polynomial time (Cook theorem) [14]; while the inverse transformation may not always exist in polynomial time. This 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 $\vec{x} = (x_1, x_2, \ldots, x_n)$ so 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:

$$f_{\text{SAT}}(\vec{x}) = c_1(\vec{x}) \wedge c_2(\vec{x}) \wedge \ldots \wedge c_{\text{m}}(\vec{x}) \ . \tag{1}$$

An instance of SAT, $\vec{x}$, is called satisfiable if $f_{\text{SAT}}(\vec{x}) = 1$, and unsatisfiable otherwise. A k-SAT instance is composed of

clauses with length k, and when $k \geq 3$ (the case we address in this work) the problem is NP-complete [15].
The fitness function we use here is the *stepwise adaptation of weights* (SAW) [11], described in Eq. 2. This adaptive function weights the values of the clauses with $w_i \in \mathbb{N}$ in order to give more importance to those clauses which are still not satisfied by the current best solution. These weights are adjusted dynamically according to the formula $w_i = w_i + 1 - c_i(\vec{x}^*)$, being $\vec{x}^*$ the current fittest individual.

$$f_{\mathrm{SAW}}(\vec{x}) = w_1 \cdot c_1(\vec{x}) + \ldots + w_{\mathrm{m}} \cdot c_{\mathrm{m}}(\vec{x}) \ . \qquad (2)$$

There exist other known adaptive fitness functions for SAT, such as the refining functions [16], but we do not include them in this study because "… the use of refining functions does not necessarily lead to improved performance when using the bit string representation", as it is concluded in [16]. Of course, the used SAW function has repeatedly shown to be much more suitable for SAT than just counting the number of unsatisfied clauses [11], [16].

3. CANONICAL AND ADVANCED COMPONENTS IN CELLULAR MAs

A detailed description of the algorithms used is given in the current section. Specifically, we study three simple heuristics for solving SAT, a basic cGA –called JCell–, and finally the proposed cMAs.

Three Basic Local Search Techniques for SAT

In this subsection, we present the three local search procedures (LS) used to illustrate how to construct a cMA and also used for solving SAT. Two of them were specifically designed for this problem: (i) a gradient algorithm (GRAD), based on the flip heuristic, and especially developed for this work by the authors, and (ii) the well-known WSAT algorithm. The third procedure included is Simulated Annealing (SA), a well-known metaheuristic.

GRAD: For this work, we have developed a new local search algorithm (called GRAD) for SAT. GRAD is an algorithm (based on the flip heuristic) that performs a gradient search in the space of solutions (see Algorithm 1). Basically, it consists in mutating the value of a variable according to the number of clauses it does not satisfy: the higher the number of unsatisfied clauses a variable belongs to, the higher the probability for mutating (flipping) it. As it can be seen in Algorithm 1, some noise is added to the search (with probability 0.2) in order to enhance its exploration capabilities. The main difference of GRAD with respect to the flip heuristic consists in that the latter flips a variable (v) in terms of the gain of that flip: $v = \{v_i / \max(\text{sat\_clauses}(\overline{v_i}) - \text{sat\_clauses}(v_i))\}$ ($i = 1$ to the number of variables), while in GRAD the flip is made to every variable (v) satisfying $v = \{v_i / \max(\text{unsat\_clauses}(v_i))\}$ with equal probability (independently of the gain). This difference makes GRAD computationally lighter than the flip heuristic.
The workings of GRAD are simple. The algorithm starts by randomly generating both the initial best solution and the first individual (lines 2 and 4, respectively). After that, until the final condition is met, the algorithm repeatedly generates a new individual (offspring) from the current one (parent), evaluates it, and replaces the best current solution with it if it is better (higher fitness value). The offspring is created by flipping the worst genes of the parent –those unsatisfying the largest number of clauses– with equal probability (lines 9 and 10). With a preset probability some noise (20%) is introduced in the search. In this case, the offspring is mutated by flipping the value of a randomly chosen variable of the parent unsatisfying one or more clauses (lines 6 and 7). Then, the search process is repeated for the offspring (lines 5 to 14). Every `MAX_STEPS` iterations the search is restarted –the current individual is randomly generated– (line 4). We have set this value to 10 times the number of variables. The algorithm stops (line 3) when the optimal or the best-known solution is found or after making 2 millions of fitness function evaluations.

Algorithm 1 Pseudocode of GRAD
```
 1: GRAD(problem)
 2: best_ind = New_Random_Ind();
 3: while ! Termination_Condition() do
 4:    ind = New_Random_Ind();
 5:    for steps ← 0 to MAX_STEPS do
 6:       if rand0to1() < prob_noise then
 7:          Flip(ind,Random_Variable_Unsatisfying_Any_Clause());
 8:       else
 9:          vars_to_flip[]=Vars_Unsatisfying_Max_Number_Of_Clauses();
10:          Flip_With_Equal_Probability(ind,vars_to_flip);
11:       end if
12:       Evaluate_Fitness(ind);
13:       best_ind = Best(ind,best_ind);
14:    end for
15: end while
```

WSAT: The WSAT algorithm [17] is a greedy heuristic specifically designed for SAT. Basically, it consists in repeatedly selecting a non satisfied clause (randomly) and flipping one of its variables (see Algorithm 2). There exist several methods for selecting this variable [18]. Among them, we have adopted the BEST strategy, which consists in flipping a variable of the clause with a given probability (prob_noise = 0.5), and otherwise flips the variable that minimizes the number of clauses that are true in the current state, but that would become false if the flip were made. After a number of steps (line 3), the search is "restarted" by replacing the current individual by a randomly generated one (line 4). Like in the case of GRAD, we "restart" every 10 times the number of variables steps, and the best found solution so far is always tracked.

Algorithm 2 Pseudocode of WSAT
```
 1: WSAT(problem)
 2: best_ind = New_Random_Ind();
 3: while ! Termination_Condition() do
 4:    ind = New_Random_Ind();
 5:    for steps ← 0 to MAX_STEPS do
 6:       clause = Random_Unsatisfied_Clause()
 7:       if rand0to1() < prob_noise then
 8:          Flip(ind,clause[randomInt(long_clause)]);
 9:       else
10:          for i ← 0 to long_clause do
11:             lost_clauses[i] = Broken_Clauses_After_Flip(i);
12:          end for
13:          Flip(ind, clause[Index_Of_Min_Value(lost_clauses)]);
14:       end if
15:       Evaluate_Fitness(ind);
16:       best_ind = Best(ind, best_ind);
17:    end for
18: end while
```

SA: Simulated Annealing [19] is probably one of the first metaheuristics with an explicit strategy to escape from local optima (see Algorithm 3 for a pseudocode). The core idea is to allow some movements resulting in solutions of worse quality in order to escape from local optima. For that, a parameter called temperature (`Temp`) is used, such that it

$$p(\text{Temp, offspr, ind}) = e^{\frac{(\text{Get\_Fit(offspr)} - \text{Get\_Fit(ind)}) \cdot 10^4}{\text{targetFitness} \cdot \text{Temp}}} \quad (3)$$

decreases during the execution (line 19) in order to reduce the probability for accepting movements with a loose in the quality of the solution (computed as shown in Eq. 3). Temp is initialized to a given upper bound T_{max} (line 5), and new individuals are computed while the current value of Temp is larger than a given threshold T_{min} (lines 6 to 20). If the new individual is better (higher fitness value) than the best so far one it is accepted as the new best one (lines 14 and 15) and, otherwise, it replaces the best one with a given probability (lines 16 and 17). After some experimental tests, we have set values to $T_{max} = 10$, $T_{min} = 1$, and $\text{coolingRate} = 0.8$.

Algorithm 3 Pseudocode of SA

```
1:  Simulated_Annealing(problem, $T_{max}$, coolingRate)
2:  ind = New_Random_Ind();
3:  best_ind = ind;
4:  while ! Termination_Condition() do
5:     Temp = $T_{max}$;
6:     while Temp $>$ $T_{min}$ do
7:        offspring = ind;              // generate an offspring
8:        for i $\leftarrow$ 0 to problem.num_vars do
9:           if rand0to1() $< 1/$problem.num_vars then
10:             Flip(offspring, i);
11:          end if
12:       end for
13:       Evaluate_Fitness(offspring);
14:       if Get_Fit(offspring) $>=$ Get_Fit(ind) then
15:          ind = offspring;
16:       else if rando0to1() $< p$(Temp, offspring, ind) then
17:          ind = offspring;
18:       end if
19:       Temp $*=$ coolingRate;
20:    end while
21:    best_ind = Best(offspring, best_ind);
22: end while
```

The Cellular GA

Cellular GAs are a class 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 Fig. 1.c). 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 at the level of each individual to improve on exploitation.

Algorithm 4 Pseudocode for a Canonical cGA

```
1:  JCell(cga)        //Algorithm parameters in 'cga'
2:  while ! Termination_Condition() do
3:     for individual $\leftarrow$ 1 to cga.popSize do
4:        n_list$\leftarrow$Get_Neighborhood(cga,position(individual));
5:        parents$\leftarrow$Selection(n_list);
6:        offspring$\leftarrow$Recombination(cga.$P_c$, parents);
7:        offspring$\leftarrow$Mutation(cga.$P_m$, offspring);
8:        Evaluate_Fitness(offspring);
9:        Insert(position(individual), offspring, cga, aux_pop);
10:    end for
11:    cga.pop$\leftarrow$aux_pop;
12: end while
```

In this section, a detailed description of JCell is presented (see a pseudocode in Algorithm 4). In JCell, the population is structured in a 2 dimensional toroidal grid, and a neighborhood of 5 individuals (NEWS) is defined on it (see Fig. 1.c). The algorithm iteratively considers as current each individual in the grid (line 3). An individual may only interact with individuals belonging to its neighborhood (line 4), so the parents are selected among the individuals of the neighborhood (line 5) with a given criterion. Recombination and mutation genetic operators are applied to the individuals in lines 6 and 7, with probabilities P_c and P_m, respectively. After that, the algorithm computes the fitness value of the offspring (line 8), and inserts it on the equivalent place of the current individual in the new (auxiliary) population (line 9) following a given replacement policy.

After applying this reproductive cycle to all the individuals of the population, the composed auxiliary population is assumed to be the new population for the next generation (line 11) — this is called synchronous update. This loop is repeated until a termination condition is met (line 2): usually to reach the optimum, to make a maximum number of fitness function evaluations, or a combination of they two.

Cellular Memetic Algorithms

Memetic algorithms are search algorithms in which some knowledge of the problem is used in one or more operators. The objective is to improve the behavior of the original algorithm. Not only local search, but also restart, structured and intensive search is commonly considered in MAs [20]. In this work, we implement some cellular memetic algorithms (cMAs), obtained by hybridizing JCell with different combinations of generic and specific recombination, mutation and local search operators (see Table 1), as well as an adaptive fitness function specifically designed for SAT, SAW (see Section 2). In Algorithm 5 we show the pseudocode for a canonical cMA. As it can be seen, the main difference between the pseudocodes of the canonical cMA and cGA is the local search step included in line 8 of the cMA.

Algorithm 5 Pseudocode for a Canonical cMA

```
1:  cMA(cma)        //Algorithm parameters in 'cma'
2:  while ! Termination_Condition() do
3:     for individual $\leftarrow$ 1 to cma.popSize do
4:        n_list$\leftarrow$Get_Neighborhood(cma,position(individual));
5:        parents$\leftarrow$Selection(n_list);
6:        offspring$\leftarrow$Recombination(cma.$P_c$, parents);
7:        offspring$\leftarrow$Mutation(cma.$P_m$, offspring);
8:        offspring$\leftarrow$Local_Search(cma.$P_{LS}$, offspring, {intensive|light});
9:        Evaluate_Fitness(offspring);
10:       Insert(position(individual), offspring, cma, aux_pop);
11:    end for
12:    cma.pop$\leftarrow$aux_pop;
13: end while
```

Table. 1. Operators used in JCell in this work for solving SAT.

Operator	Generic	Specific
Crossover	DPX	UCR
Mutation	BM	UCM
Local Search	SA	GRAD
		WSAT

We use two specific genetic operators for recombination and mutation which are called *Unsatisfied Clauses Recombination* (UCR) and *Unsatisfied Clauses Mutation* (UCM), respectively. The two operators focus on keeping constant the values of the variables satisfying all the clauses they belong to. Our UCR is exactly the same operator as the proposed C_B in [16], and UCM is the result of adding some noise to M_B (also proposed in [16]), as seen in Algorithm 6. We use two well-known generic recombination and mutation operators: two point recombination (DPX) and binary mutation (BM).

The three heuristics proposed hereinbefore have been adopted as LS operators in JCell. As it can be seen in Section 4, some different configurations of these local search methods have

Algorithm 6 Pseudocode for UCM

```
UCM(Indiv, Noise)
if rand0to1() $\leq$ Noise then
    Flip(Indiv, randomInt(Indiv.length));
else
    $\mathbf{M_B}$(Indiv);
end if
```

been studied. These configurations differ on the probability of applying the local search operator to the individuals and the intensity of the local search step. The idea is to regulate the overall computational effort to solve the problem in affordable times with commodity computers.

4. COMPUTATIONAL ANALYSIS

In this section we analyze the results for our tests over the 12 hard instances (from $n = 30$ to 100 variables) composing the suite 1 of the benchmark proposed in [10]. These instances belong to the SAT phase transition of difficulty, where hardest instances are located, since they verify that $m = 4.3 * n$ [21] (being m the number of clauses).

In the following subsections we present the results we obtained in the studies for this paper. All the algorithms have been tested in terms of efficiency –Average number of Evaluations to Solution (AES)– and efficacy –Success Rate (SR)– (see tables 2, 3, 5 and 6, and Fig. 2). The results have been obtained after making 50 independent runs of the algorithms for every instance. We have computed *p*-values by performing ANOVA tests on our results in order to assess their statistical significance. A 95% confidence level is considered, and statistical significant differences are shown with symbol '$+$' ('$\bullet$' means non-significance) in tables 3, 5 and 6.

The Effects of Structuring the Population and Using SAW

In this section we justify the election of both the structured (cellular) population and the use of the stepwise adaptation of weights function (SAW) in our cMAs. With that purpose, we study the behavior of JCell.DPX_BM (a cGA using SAW, and implementing generic recombination and mutation operators –DPX and BM, respectively–), compared to its generational version –non-structured population– (genGA), and JCell.DPX_BM using the most classic fitness function for SAT (JCell.DPX_BM_cl), i.e., counting the number of satisfied clauses by the potential solution.

Table. 2. The effects of structuring the population and using SAW. Average evaluations to solution (AES).

Inst.		genGA	JCell.DPX_BM_cl	JCell.DPX_BM
size (n)	#			
30	1	49801.0	**3415.7**	7438.0
30	2	1135843.2	—	**502208.4**
30	3	440886.2	**3024.0**	80029.4
40	4	855286.3	31932.0	**13829.8**
40	5	66193.9	**4861.4**	9391.7
40	6	1603008.0	—	**519868.8**
50	7	473839.4	14356.8	**13081.0**
50	8	1076077.4	**84088.8**	95379.8
50	9	1333872.0	—	**524164.1**
100	10	—	—	**601488.0**
100	11	—	223310.8	**165484.8**
100	12	—	**245232.0**	392871.8

Our results are given in Table 2, wherein we show for every instance, its size, its identifier (#), and the average number of evaluations needed to find the solution (AES) of the three algorithms (best values are **bolded**). Additionally, the number of runs in which the solution was found (success rate) is displayed in Fig. 2. After applying ANOVA tests to the results in Table 2, we obtained that there are statistically significant differences in all the instances. One can see that genGA is the algorithm reporting the worst AES results, and JCell.DPX_BM is worse than JCell.DPX_BM_cl with statistical significance only in 2 instances (numbers 1 and 5). In terms of SR (see Fig. 2), JCell.DPX_BM is better than the other two algorithms (higher efficacy) for the 12 instances. Moreover, only JCell.DPX_BM is able to find the optimal solution for all the instances. Hence, from these results we can conclude that JCell.DPX_BM, the algorithm with structured population and using SAW, is the best of the three compared ones both in terms of efficacy (SR) and efficiency (AES).

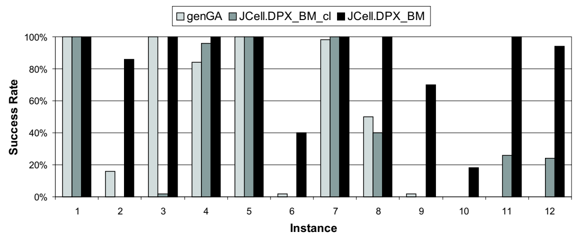

Fig. 2. The effects of structuring the population and using SAW. Success rate (SR).

From Table 2 and Fig. 2, it stands out that the tested cGA markedly improve its results when using the SAW fitness function. Because of that, all the algorithms studied in the following sections have been implemented using SAW. Although the authors are aware that the use of SAW does not necessarily improve the performance on other algorithms, the SAW fitness function has been implemented in GRAD, SA, and WSAT as also suggested in [13]. It is made in order to make fair comparisons with the cMAs, since these algorithms are hybridized with them, and the SAW fitness function is used.

Non Memetic Heuristics for SAT

In this section we study the behavior of GRAD, SA, and WSAT. Additionally, we test the behavior of two different cGAs without local search: the previously studied JCell.DPX_BM, and JCell.UCR_UCM, which is the result of hybridizing the former with recombination and mutation operators specifically designed for SAT (UCR and UCM). The parameters used are those of Table 4, but in this case $P_{\text{LS}} = 0.0$ (there is no LS). In Table 3 we show our results. The first issue we want to emphasize is that only WSAT is able to solve the problem in every run for all the instances. Conversely, in [9] WSAT only solved the problem in a 80% of the runs in the case of the largest instances (with $n = 100$), so we have a better implementation of WSAT here.

Comparing the three basic LS, it can be seen in Table 3 that SA obtains the worst results, both in terms of efficacy and efficiency (with statistical confidence, except for instance number 10). GRAD is similar to WSAT in efficacy, but needs a larger number of fitness function evaluations to find the solution (lower efficiency), except for instance 6 (significant values obtained in instances 3, 4, 7, 8, and 10 to 12). Hence, we can conclude that WSAT is the best of the three heuristics for the studied test-suite, followed by GRAD. We also conclude on the superiority of problem dependent algorithms versus generic patterns of search like SA.

Table. 3. Basic algorithms.

Inst.	GRAD		SA		WSAT		*p*-val	JCell.DPX_BM		JCell.UCR_UCM		*p*-val
	SR	AES	SR	AES	SR	AES		SR	AES	SR	AES	
1	1.00	203.56 ±219.73	1.00	685.22 ±844.74	1.00	**143.64** ±120.32	+	1.00	7438.04 ±3234.60	1.00	104100.48 ±121339.96	+
2	1.00	9681.06 ±9483.55	1.00	63346.60 ±93625.68	1.00	**8575.66** ±9244.08	+	0.86	502208.37 ±491034.68	0.10	697564.80 ±663741.62	•
3	1.00	8520.38 ±7724.11	1.00	16833.44 ±11002.84	1.00	**3984.34** ±4112.95	+	1.00	80029.44 ±54664.78	0.98	269282.94 ±223859.24	+
4	1.00	619.94 ±584.88	1.00	2173.62 ±2076.57	1.00	**199.56** ±193.56	+	1.00	13829.76 ±7801.37	0.10	1364688.00 ±500365.96	+
5	1.00	324.46 ±332.19	1.00	1202.86 ±1045.82	1.00	**103.66** ±88.02	+	1.00	9391.68 ±2478.37	1.00	249137.28 ±236218.19	+
6	1.00	**14368.98** ±13954.02	0.86	271701.47 ±418129.55	1.00	14621.04 ±18617.88	+	0.40	519868.80 ±552312.72	0.00	—	—
7	1.00	496.58 ±359.60	1.00	1614.76 ±1252.34	1.00	**200.84** ±154.81	+	1.00	13080.96 ±3346.94	0.10	1005494.40 ±721439.61	+
8	1.00	1761.74 ±1989.06	1.00	9512.84 ±10226.14	1.00	**793.38** ±870.94	+	1.00	95379.84 ±125768.68	0.00	—	—
9	1.00	82004.84 ±63217.93	1.00	201612.46 ±266218.97	1.00	**77696.42** ±75769.23	+	0.70	524164.11 ±432005.51	0.00	—	—
10	0.94	726522.51 ±525423.23	0.84	510006.12 ±419781.41	1.00	**189785.14** ±198738.78	+	0.18	601488.00 ±364655.49	0.00	—	—
11	1.00	5508.26 ±5940.96	1.00	18123.00 ±20635.35	1.00	**1501.74** ±1264.80	+	1.00	165484.80 ±190927.59	0.00	—	—
12	1.00	8920.38 ±9111.02	1.00	25539.84 ±22393.45	1.00	**1388.92** ±1308.27	+	0.94	392871.83 ±443791.69	0.00	—	—

With respect to the two cGAs, Table 3 states the opposite result: the cGA with generic operators outperforms the one including tailored mutation and recombination. This is clear since JCell.UCR_UCM reports a larger AES than JCell.DPX_BM (statistical confidence for all the instances, except 2). Moreover, JCell.UCR_UCM also performs a lower hit rate (SR) than JCell.DPX_BM in general. Moreover, JCell.UCR_UCM is not able to find the optimum in any of the 50 runs made for 6 out of the 12 instances. Probably, the reason for this poor behavior of JCell.UCR_UCM with respect to JCell.DPX_BM is that both UCR and UCM perform a too intensive exploitation of the search space, resulting in an important and fast loss of diversity in the population, thus making the algorithm to get stuck in local optima.

As a final conclusion, we can claim from Table 3 that the results of the two cGAs without explicit LS are always worse than those of the LS heuristics, both in terms of efficiency and efficacy. The best results of the table are those obtained by WSAT. Since we suspect that these results are too linked to the instances (specially to their "small" size) we will enlarge the test set at the end of next subsection with harder instances.

Cellular Memetic Algorithms

In this section we study the behavior of a large number of cMAs with different parameterizations. As it can be seen in Table 4 (wherein details of the cMAs are given) we hybridize the two simple cGAs of the previous section with three distinct local search methods (GRAD, SA, and WSAT). These local search methods have been applied in two different ways: (i) executing an intense local search step ($10 \times n$ fitness function evaluations) to a percentage of the individuals (called *intensive*), or (ii) applying a light local search step to all the individuals, consisting in making 20 fitness function evaluations (called *light*).

The results are shown in tables 5 and 6. Comparing them with those of the cGAs of Table 3, we can see that the behavior of the algorithm is generally improved both in efficiency and efficacy, specially in the cases of using GRAD and WSAT. Hence, the three local search methods used (generic and specific) help the algorithm for getting out from local optima. If we compare the studied cMAs in terms of the way the local search method is applied (intensive or light), we conclude that the intensive case always obtains better results than the other when hybridizing the algorithm with one specific heuristic (either GRAD or WSAT). Conversely, when using SA it is not always true, since SA applied in an intensive way only outperforms the other case in 9 out of the 24 tests. Hence, the cGAs hybridized with specialized LS have a better performance than the one using SA (generic). All these comparisons are statistically significant in 58 out of the 65 cases in which all the cMAs obtained the solution in almost 100% of the runs. As an interesting exception, we want to remark the good behavior of JCell.UCR_UCM+SA for instances 11 and 12 with respect to JCell.UCR_UCM hybridized with GRAD and WSAT, since the two latter cMAs are not able to find the optimal solution in any run. The reason is probably a too high intensification of GRAD and WSAT performed on the population (remind that they are still merged with UCR and UCM), guiding the algorithm towards a local optimum quickly.

We now proceed to compare the best algorithm of Table 3, WSAT, with the best one of tables 5 and 6, JCell.UCR_UCM_i+WSAT. These two algorithms are the best out of all the studied ones in terms of efficiency and efficacy. The two algorithms find the optimal solution in the 100% of the runs (SR=1.0 for every instance), but JCell.UCR_UCM_i+WSAT obtains worse (higher) results than WSAT in terms of AES (with statistically significant differences). Although we expected a hard comparison against the best algorithm in literature (WSAT), it was not the case since we got similar accuracy and only slightly worse efficiency in our tests. Since we suspected this holds only in the smaller instances we decided to test these two algorithms with larger instances in order to check if the cMA is able to outperform the state of the art WSAT in harder problems. For that, we have selected the 50 instances of 150 variables from the suite 2 of the same benchmark [10] studied before. The results with the larger instances show that JCell.UCR_UCM_i+WSAT solved the problem at least once (of 50 executions) in 26 out of the 50 instances composing

Table. 4. General parameterization for the studied cMAs.

	JCell.DPX_BM+ {GRAD,SA,WSAT}	JCell.DPX_BM_i+ {GRAD,SA,WSAT}	JCell.UCR_UCM+ {GRAD,SA,WSAT}	JCell.UCR_UCM_i+ {GRAD,SA,WSAT}
Local Search	Light $P_{LS} = 1.0$	Intensive $P_{LS} = 1.0/\text{popsize}$	Light $P_{LS} = 1.0$	Intensive $P_{LS} = 1.0/\text{popsize}$
Mutation	Bit-flip ($P_{bf} = 1/n$), $P_m = 1.0$		UCM, $P_m = 1.0$	
Crossover	DPX, $P_c = 1.0$		UCR, P_c	
Pop. Size	144 Individuals			
Selection	Itself + Binary Tournament			
Replacement	Replace if Better			
Stop Condition	Find a solution or achieve 100.000 generations			

Table. 5. Results for the proposed hybridizations to JCell.DPX_BM.

Inst.	JCell.DPX_BM						JCell.DPX_BM_i					
	+ GRAD		+ SA		+ WSAT		+ GRAD		+ SA		+ WSAT	
	SR	AES	SR	AES	SR	AES	SR	AES	SR	AES	SR	AES
1	1.00	3054.2 ±392.2	1.00	29474.5 ±583.4	1.00	2966.1 ±19.2	1.00	1072.8 ±1112.6	1.00	9649.6 ±25809.9	1.00	**569.9** ±302.8
2	1.00	33598.7 ±51766.6	1.00	195397.6 ±295646.3	1.00	32730.4 ±49353.2	1.00	50886.2 ±44167.7	0.90	559464.3 ±437996.2	1.00	**30885.5** ±22768.8
3	1.00	14761.2 ±24935.1	1.00	33005.4 ±6306.3	1.00	**4104.5** ±3325.1	1.00	20385.8 ±20115.7	1.00	255902.5 ±275734.7	1.00	9418.4 ±10239.6
4	1.00	5018.6 ±2397.8	1.00	31618.8 ±152.9	1.00	3972.9 ±1343.8	1.00	2573.4 ±2497.7	1.00	49310.9 ±64714.9	1.00	**794.7** ±693.7
5	1.00	3575.6 ±1131.5	1.00	31052.9 ±282.7	1.00	3008.3 ±8.4	1.00	1586.0 ±1757.9	1.00	13354.0 ±36668.5	1.00	**628.6** ±437.9
6	0.96	181863.6 ±343020.8	0.96	434235.9 ±519011.4	1.00	81966.1 ±114950.0	1.00	94046.4 ±114105.9	0.72	654160.4 ±476411.6	1.00	**41619.4** ±47466.8
7	1.00	5945.8 ±2416.8	1.00	33621.6 ±7313.2	1.00	4822.6 ±1364.9	1.00	2342.6 ±2972.9	1.00	37446.4 ±70165.5	1.00	**850.8** ±527.5
8	1.00	14930.8 ±7644.5	1.00	47688.6 ±15925.1	1.00	7138.3 ±3957.5	1.00	5164.5 ±5786.7	1.00	195816.2 ±155018.9	1.00	**2097.6** ±1886.8
9	0.80	787149.2 ±528237.4	0.50	720491.5 ±597642.6	1.00	600993.9 ±443475.3	0.82	963177.2 ±585320.7	0.34	883967.7 ±633307.9	1.00	**187814.5** ±148264.1
10	0.06	797880.3 ±824831.9	0.04	1209394.0 ±90058.5	0.06	1189559.7 ±374193.7	0.04	1302489.0 ±346149.9	0.10	1363627.4 ±368403.3	0.80	**792051.2** ±491548.4
11	1.00	58591.3 ±18897.3	1.00	1039910.2 ±205127.9	1.00	35571.0 ±9243.6	1.00	12539.8 ±10851.1	1.00	357207.9 ±422288.9	1.00	**2466.3** ±1846.4
12	0.96	70324.9 ±32808.8	0.98	1051351.2 ±174510.4	1.00	45950.2 ±19870.7	1.00	20018.2 ±19674.3	0.98	409492.6 ±425872.3	1.00	**3196.9** ±2938.3

Table. 6. Results for the proposed hybridizations to JCell.UCR_UCM.

Inst.	JCell.UCR_UCM						JCell.UCR_UCM_i					
	+ GRAD		+ SA		+ WSAT		+ GRAD		+ SA		+ WSAT	
	SR	AES	SR	AES	SR	AES	SR	AES	SR	AES	SR	AES
1	1.00	2981.2 ±18.9	1.00	29253.2 ±474.8	1.00	2953.1 ±27.8	1.00	1239.1 ±1467.1	1.00	21710.7 ±46248.4	1.00	**748.8** ±404.1
2	1.00	20294.7 ±23868.0	1.00	187088.9 ±281719.9	1.00	**14879.6** ±18766.3	1.00	58842.3 ±62944.9	1.00	686104.1 ±527621.8	1.00	31457.6 ±33033.8
3	1.00	4048.0 ±2832.1	1.00	41269.5 ±58635.5	1.00	**3641.2** ±1861.2	1.00	25086.8 ±24428.4	1.00	280148.0 ±217802.8	1.00	13614.9 ±13134.6
4	1.00	7853.8 ±9207.1	1.00	31527.8 ±187.2	1.00	3472.5 ±1773.7	1.00	2299.6 ±2937.4	1.00	63190.8 ±110063.6	1.00	**779.4** ±408.9
5	1.00	3466.3 ±1781.8	1.00	30893.9 ±246.8	1.00	2976.1 ±19.9	1.00	1193.1 ±1198.5	1.00	18722.7 ±56165.8	1.00	**624.7** ±369.2
6	1.00	379489.9 ±351593.1	1.00	274977.7 ±389332.4	1.00	162737.1 ±180706.5	1.00	86780.6 ±71185.9	0.94	849405.5 ±584901.9	1.00	**57997.9** ±48455.4
7	1.00	7335.1 ±7980.3	1.00	31715.6 ±134.0	1.00	3532.0 ±1807.9	1.00	1639.8 ±2297.5	1.00	96672.3 ±158359.2	1.00	**678.2** ±507.7
8	1.00	82967.7 ±76765.2	1.00	46418.0 ±15867.9	1.00	27090.5 ±30079.4	1.00	6747.4 ±8070.6	1.00	291700.2 ±225526.0	1.00	**1694.4** ±1619.9
9	0.42	1089600.1 ±642627.4	0.64	1365366.8 ±559506.2	0.56	694014.5 ±548185.0	0.92	566331.3 ±476381.3	0.48	1155717.8 ±529793.2	1.00	**305306.2** ±323215.9
10	0.00	—	0.00	—	0.00	—	0.76	885961.2 ±630092.4	0.16	1099241.9 ±768918.5	1.00	**425377.6** ±415069.5
11	0.00	—	0.64	1743364.38 ±190880.9	0.00	—	1.00	10560.4 ±11327.4	0.90	695508.1 ±855309.5	1.00	**2980.8** ±3334.6
12	0.00	—	0.40	1778928.5 ±200497.9	0.00	—	1.00	16623.6 ±18137.9	0.94	504324.6 ±770231.7	1.00	**3949.3** ±4646.0

the benchmark, while WSAT found the solution for the same 26 instances and 4 more ones (the optimum was found only once in these 4 instances). Hence, WSAT is able to find the optimum in a larger number of instances, with an average hit rate of 38.24%, which is quite close to 36.52%, the value obtained by JCell.UCR_UCM_i+WSAT.

Moreover, the average solution found for this benchmark (the optimal solution is 645 for all the instances) is 644.20 for JCell.UCR_UCM_i+WSAT and 643.00 for WSAT, so the cMA is more accurate than WSAT for this set of instances. Finally, if we compute the average AES for the instances solved (at least once) by the two algorithms we can see that the cMA (AES = 364383.67) is in this case more efficient than WSAT (AES = 372162.36). Hence, as we suspected, the cMA outperforms WSAT for this set of larger and more difficult problems. In fact, all these results represent the new state of the art since "our" WSAT is better than the one reported in the literature.

Comparison Against Other Results in the Literature

In this section we compare some of our results with those of the algorithms studied in [9], which were tested with the same benchmark we used in this work. The comparison is shown in Table 7, where only the efficacy of the algorithms is shown because the efficiency is measured in [9] as the number of bit flips to solution (AFS) instead of the number of function evaluations (AES).

Table. 7. Efficacy (SR) of different algorithms evaluated on SAT.

Algorithm	n = 30	n = 40	n = 50	n = 100
SAWEA	1.00	0.93	0.85	0.72
RFEA2	1.00	1.00	1.00	0.99
RFEA2+	1.00	1.00	1.00	0.97
FlipGA	1.00	1.00	1.00	0.87
ASAP	1.00	1.00	1.00	1.00
WSAT [9]	1.00	1.00	1.00	0.80
GRAD	1.00	1.00	1.00	0.98
WSAT	1.00	1.00	1.00	1.00
JCell.DPX_BM_i+WSAT	1.00	1.00	1.00	0.93
JCell.UCR_UCM_i+WSAT	1.00	1.00	1.00	1.00

As it can be seen in Table 7, only ASAP [9], our implementation of WSAT, and JCell.UCR_UCM_i+WSAT are able to find the solution in every run for all the instances. However, most of the algorithms of the table have a very high efficacy for the tested benchmark, usually with hit rates over 90%.

The difference in the behavior of our WSAT and that studied in [9] are both the noise probability and the termination condition. On the one hand, we use a noise probability of 0.5, while Gottlieb et al. do not specify in [9] the value they use. On the other hand, our algorithm finishes when 2 million function evaluations are made, while the termination condition in the algorithm of Gottlieb et al. is to reach 300000 flips. Although the number of evaluations made by our algorithm is higher than the number of flips in our implementation of WSAT (since we use SAW, the best stored individual has to be re-evaluated) the obtained SR values when setting the termination condition to 300000 evaluations (less than 300000 flips should be made) are 1.0, 1.0, 0.99, and 0.95 for the groups of instances with n = 30, 40, 50, and 100, respectively. These values are still higher (larger efficacy) than those obtained by Gottlieb el al.

5. CONCLUSIONS AND FURTHER WORK

In this work we have proposed several ways of creating cMAs by analyzing the behavior of adding 3 LS methods, 2 basic cGAs and 12 cMAs on the 3-SAT problem. These cMAs are the result of hybridizing the two cGAs with the 3 LS, applied with different parameterizations reinforcing diversification or intensification. Two LS, WSAT and GRAD (this one specially developed in this work), are specifically designed for SAT, while SA is a generic one.

We have seen that the results of the proposed basic cGAs (without local search) are far from those obtained by the three studied LS methods. After hybridizing these basic cGAs with a local search step, the resulting cMAs substantially improved the behavior of the original cGAs. Thus, the hybridization step helps the cMAs to avoid the local optima in which the simple cGAs get stuck. For smaller instances, the best of the tested cMAs (JCell.UCR_UCM_i+WSAT) is as accurate as the best reported algorithm (WSAT) but slightly less efficient.

After these results, we studied the behavior of WSAT and JCell.UCR_UCM_i+WSAT (the two best resulting algorithms) with a harder set of larger instances. The results confirm our suspicions, since the cMA is more accurate and efficient than WSAT for these harder instances. However, our results contrast with those of Gottlieb et al., who concluded in [9] that "A preliminary experimental investigation of EAs for constraint satisfaction problems using both an adaptive fitness function (based on f_{SAW}) and local search indicates that this combination is not beneficial". We found that this claim does not hold for our structured algorithms solving large instances. As a future works, it would be interesting to test some different parameterizations on the cMAs in order to improve the obtained results. Additionally, the hybridization of improved models of cGAs (asynchronous, adaptive, ...) could lead us to better results. Finally, an interesting work should be testing the algorithms with larger instances of the problem.

REFERENCES

[1] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms*, vol. 1 of *Book Series on GAs and EC*, Kluwer, 2000.

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

[3] E. Alba and J.M. Troya, "Improving Flexibility and Efficiency by Adding Parallelism to Genetic Algorithms," *Statistics and Computing*, vol. 12, no. 2, pp. 91–114, 2002.

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

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

[6] H.A. Kautz and B. Selman, "Planning as satisfiability," in *European Conf. on Artificial Intelligence*, 1992, pp. 359–363.

[7] R. Reiter and A. Mackworth, "A logical framework for depiction and image interpretation," *Artifitial Intelligence*, vol. 41(3), pp. 123–155, 1989.

[8] K.A. De Jong and W.M. Spears, "Using genetic algorithm to solve NP-complete problems," in *3rd ICGA*, James D. Schaffer, Ed. 1989, pp. 124–132, Morgan Kaufmann.

[9] J. Gottlieb, E. Marchiori, and C. Rossi, "Evolutionary algorithms for the satisfiability problem," *Evolutionary Computation*, vol. 10, no. 2, pp. 35–50, Spring 2002.

[10] T. Bäck, A.E. Eiben, and M.E. Vink, "A superior evolutionary algorithm for 3-SAT," in *7th Conf. on Evolutionary Programming*. 1998, Vol. 1477 of LNCS, pp. 125–136, Springer.

[11] A.E. Eiben and J.K. van der Hauw, "Solving 3-SAT with adaptive genetic algorithms," in *IEEE CEC97*, 1997, pp. 81–86.

[12] G. Folino, C. Pizzuti, and G. Spezzano, "Combining cellular genetic algorithms and local search for solving satisfiability problems," in *IEEE Int. Conf. Tools with AI*, 1998, pp. 192–198.

[13] E. Alba, B. Dorronsoro, and H. Alfonso, "Cellular memetic algorithms evaluated on SAT," in *CACIC*, 2005, vol. CD 1.

[14] S.A. Cook, "The complexity of theorem-proving procedures," *ACM Symp. on the Theory of Computing*, pp. 151–158, 1971.

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

[16] J. Gottlieb and N. Voss, "Representations, fitness functions and genetic operators for the satisfiability problem," in *Artificial Evolution*. 1998, LNCS, pp. 55–68, Springer.

[17] B. Selman, H. Kautz, and B. Cohen, "Noise strategies for improving local search," in *22th Nat. Conf. on Artificial Intelligence*, California, 1994, pp. 337–343, AAAI Press.

[18] D. McAllester, B. Selman, and H. Kautz, "Evidence for invariants in local search," in *14th Nat. Conf. on Artificial Intelligence*, Providence, RI, 1997, pp. 321–326.

[19] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 4598, pp. 671–680, 1983.

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

[21] D.G. Mitchell, B. Selman, and H.J. Levesque, "Hard and easy distributions for SAT problems," in *10th Nat. Conf. on Artificial Intelligence*, California, 1992, pp. 459–465.