Hierarchical Cellular Genetic Algorithm

Stefan Janson[1], Enrique Alba[2],
Bernabé Dorronsoro[2], and Martin Middendorf[1]

[1] Department of Computer Science, University of Leipzig, Germany
{janson, middendorf}@informatik.uni-leipzig.de
[2] Department of Computer Science, University of Málaga, Spain
{eat, bernabe}@lcc.uma.es

Abstract. Cellular Genetic Algorithms (cGA) are spatially distributed Genetic Algorithms that, because of their high level of diversity, are superior to regular GAs on several optimization functions. Also, since these distributed algorithms only require communication between few closely arranged individuals, they are very suitable for a parallel implementation. We propose a new kind of cGA, called hierarchical cGA (H-cGA), where the population structure is augmented with a hierarchy according to the current fitness of the individuals. Better individuals are moved towards the center of the grid, so that high quality solutions are exploited quickly, while at the same time new solutions are provided by individuals at the outside that keep exploring the search space. This algorithmic variant is expected to increase the convergence speed of the cGA algorithm and maintain the diversity given by the distributed layout. We examine the effect of the introduced hierarchy by observing the variable takeover rates at different hierarchy levels and we compare the H-cGA to the cGA algorithm on a set of benchmark problems and show that the new approach performs promising.

1 Introduction

The cellular Genetic Algorithm (cGA) [1] is a kind of decentralized GA in which the population is arranged in a toroidal grid, usually of dimension 2. The characteristic feature of cGAs is a neighbourhood for each individual that is restricted to a certain subset of individuals in the immediate vicinity of its position. Individuals are allowed to interact only with other individuals belonging to their neighbourhood. This endows the cGA with useful properties for the optimization [2, 3] and also facilitates a parallel implementation because of its inherent parallel design. The cGA has already been successfully implemented on parallel platforms [1, 4] and in sequential computers [5].

While the distributed arrangement of the population of the cGA preserves a high level of diversity, it can delay the convergence speed of the algorithm because the cooperative search by the entire population is restricted. In order to increase the convergence speed of the cGA algorithm, we introduce a hierarchy into the grid population that orders the individuals according to their current

J. Gottlieb and G.R. Raidl (Eds.): EvoCOP 2006, LNCS 3906, pp. 111–122, 2006.

fitness. Such a population hierarchy has already been successfully applied to particle swarm optimization algorithms [6]. Here the convergence speed of the PSO algorithm was accelerated after using a hierarchical population when solving several continuous optimization problems.

In this work we examine the effect of introducing a hierarchy into a cellular Genetic Algorithm. Our new algorithm is called *Hierarchical Cellular Genetic Algorithm* (H-cGA). In the H-cGA a hierarchy is established within the population of a regular cGA by arranging individuals according to their fitness: the worse the fitness value of an individual is, the farther it is located from the center of the population. Thus, in this hierarchical population, the best individuals are placed in the center of the grid where they can mate with other high quality individuals. This way we hope to reach better solutions faster, while still preserving the diversity provided by a locally restricted parent selection and keeping the opportunity of an efficient parallel implementation.

The paper is structured as follows. In Section 2 we present our newly proposed algorithm, H-cGA, as well as a new selection operator we have designed for this algorithm. We provide a closer examination of the takeover behaviour of the algorithm in Section 3. Section 4 contains experiments on a benchmark of functions, where the performance of the H-cGA algorithm is compared with that of a canonical cGA. Finally, we give our conclusions and further research directions for our new approach.

2 The H-cGA Algorithm

In this section we present the hierarchical cGA algorithm. First we will briefly outline the procedure of a cellular GA. Then we describe the hierarchical ordering of the population that is introduced for the H-cGA algorithm and how this ordering is obtained. After that we introduce a new selection operator for this algorithm.

In a Cellular GA [1] the population is mapped onto a 2-dimensional toroidal grid of size $x \times y$, where an individual can only mate with individuals within a locally restricted neighbourhood. In each generation, an offspring is obtained for every individual, where one of the parents is the current individual and the other one is selected from its neighbourhood. Each offspring is mutated with a given probability, and replaces the current individual, if it has a better fitness value. The offspring (or the current individual, if better) is stored in a temporary auxiliary population, and this population replaces the whole current one after each generation. In the H-cGA algorithm the population is re-arranged after every generation with the hierarchical swap operation, as described in the following section. In Algorithm 1. we give a pseudocode of H-cGA.

2.1 Hierarchy

The hierarchy is imposed on the cellular population of the GA by defining a center at position $(x/2, y/2)$ and assigning hierarchy levels according to the

Algorithm 1. Pseudocode of H-cGA

```
1: proc Steps_Up(hcga)       //Algorithm parameters in 'hcga'
2: while not TerminationCondition() do
3:   for x ← 1 to WIDTH do
4:     for y ← 1 to HEIGHT do
5:       n_list←Compute_Neigh(cga,position(x,y));
6:       parent1←Individual_At(cga,position(x,y));
7:       parent2←Local_Select(n_list);
8:       Recombination(cga.Pc,n_list[parent1],n_list[parent2],aux_ind.chrom);
9:       Mutation(cga.Pm,aux_ind.chrom);
10:      aux_ind.fit←cga.Fit(Decode(aux_ind.chrom));
11:      Insert_New_Ind(position(x,y),aux_ind,cga,aux_pop);
12:    end for
13:  end for
14:  cga.pop←aux_pop;
15:  Swap_Operation(cga.pop);
16:  Update_Statistics(cga);
17: end while
18: end_proc Steps_Up;
```

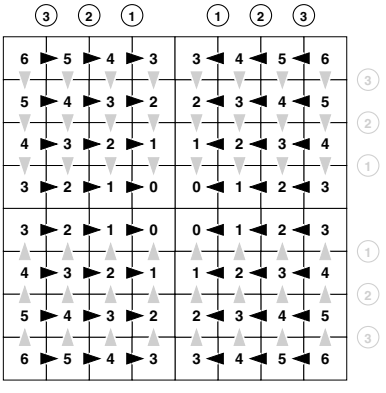

Fig. 1. The H-cGA and its different hierarchy levels

distance from the center. The center has level 0 and the level increases with increasing distance to the center (compare Fig. 1). The hierarchy is updated after each iteration of the cGA and individuals with high fitness are moved towards the center. Note, that the population topology is still toroidal when selecting parents.

In Fig. 1 we show how this swap operation is performed. It is applied between cells indicated by the arrows, in the order denoted by the numbers outside of the grid. The update of the hierarchy is performed alternatingly horizontally (black) and vertically (grey) by the swap operation. We assume an even number for the population dimensions x and y, so that the population can be uniquely divided

into left (upper) and right (lower) half, for the horizontal (vertical) swap. Note, that this implies that there are 4 individuals in the center of the population, i.e. on the highest level of the hierarchy.

In the following we describe the horizontal swap operation, the vertical swap is done accordingly. Each individual (i, j) in the left half compares itself with its left neighbor $(i-1, j)$ and if this one is better they swap their positions. These comparisons are performed starting from the center of the grid towards the outside, see Fig. 1. Thus, at first individuals in columns $(i, \frac{x}{2}-1)$ and $(i, \frac{x}{2}-2)$, for $i = 0, \ldots, y$, are compared. If the fitness value at position $(i, \frac{x}{2}-2)$ is better they swap positions. These pairwise comparisons are then continued towards the outside of the grid. Hence, an individual can advance only one level at a time but can drop several levels within one iteration.

2.2 Dissimilarity Selection

The proposed hierarchy promotes the recombination of good individuals within the population. In this respect, H-cGA is similar to a panmictic GA with a fitness-biased selection. In our H-cGA algorithm this selective recombination of the elite individuals of the population is already included in the hierarchy. Therefore, we examined a new selection operator that is not based on the relative fitness of the neighbouring individuals but instead considers the difference between the respective solution strings. As for the Binary Tournament (BT) selection, two neighbours are selected randomly, but in contrast to BT, where the better one is selected, the one that is more different from the focal individual is selected. All the considered problems are binary encoded, hence we use the Hamming-Distance for determining the dissimilarity.

The overall optimization progress of the algorithm is ensured by only replacing an individual if the newly generated individual, by crossover and mutation, is better than the previous one.

3 First Theoretical Results: Takeover Times

We are providing a closer examination of the properties of the proposed algorithm by studying the *takeover* time of the algorithm and comparing it to that of a canonical cGA. The takeover time is the time required until a single best individual has filled the entire population with copies of itself under the effects of selection only. First we are looking at a deterministic takeover process, where the best individual within the neighbourhood is always selected. Later we also consider BT selection and the newly proposed Dissimilarity selection. Initially all individuals get assigned random fitness values from [0:4094] and one individual gets the maximum fitness value of 4095. Then the selection-only algorithm is executed and the proportion of the entire population that holds the maximum fitness value at each iteration is recorded. The considered grid is of size 64×64 and hence the population consists of 4096 individuals.

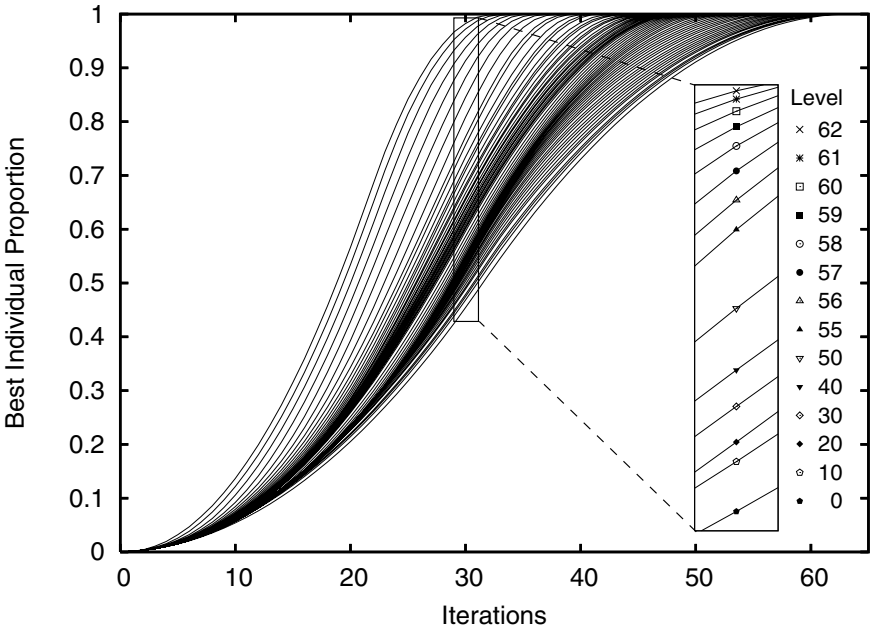

Fig. 2. Takeover curve for initially placing the best individual at different levels

In the H-cGA the different levels of the hierarchy influence the time required for takeover. In order to accurately determine this influence, we use the deterministic selection operator. The best individual is initially placed on each possible position on the grid and the takeover time for these 4096 different setups is measured. Then the results for all positions on a specific level of the hierarchy are averaged to obtain the takeover rate for introducing the best individual at this particular level. In Fig. 2 the obtained takeover curves are shown for introducing the best individual at different levels of the hierarchy. The slowest takeover rate is achieved when placing the best value at the center of the grid on level 0. This takeover rate is identical to the deterministic takeover for the regular cGA. The hierarchy level 62 consists of the 4 cells on the corners of the grid, where the fastest takeover rate is obtained. This increasing takeover speed with increasing hierarchy level is very regular, as can be seen in the detail display of iteration 30. The reason why having the best individual near the outside of the hierarchy accelerates takeover is, that, since for selection the topology is still toroidal, adjacent individuals on the opposite end of the grid also adopt the highest fitness value at the beginning of the run. Then the hierarchy swap operation moves the maximal value towards the center from several sides and therefore the actual takeover speed is increased.

We also measured the time required for takeover with BT and Dissimilarity selections. The best individual was placed at a random position and the experiments were repeated 100 times. For the experiments with Dissimilarity selection, the individuals are using a binary string that corresponds to the 12

bit representation of their current fitness value. Our results are shown in Table 1 (average number of iterations, standard deviation, minimum and maximum). As can be seen, initially placing the best individual at the center (H-cGA level 0) slows down the takeover compared to the regular cGA algorithm. This is because once the maximal value spreads it will be delayed until all of the central cells hold the best fitness value, otherwise it will be swapped towards the center again. In the two algorithms, the use of the BT selection induces a higher selection pressure than in the case of using the Dissimilarity selection. After applying the Kruskal-Wallis test (in some cases the data were not normally distributed) to the results in Table 1, we obtained that there exist statistically significant differences at a 95% confidence level.

Table 1. Takeover times for the algorithms with BT and Dissimilarity selections

Algorithm	Avg	Stddev	Min–Max
cGA BT	75.2	± 1.5	72.0– 80.0
cGA Dis	78.7	± 1.7	75.0– 83.0
H-cGA BT	71.0	± 6.3	48.0– 81.0
H-cGA Dis	79.6	± 3.8	71.0– 89.0
H-cGA BT level 0	81.5	± 1.5	78.0– 87.0
H-cGA BT level 62	46.3	± 2.1	42.0– 57.0

3.1 Fitting of the Takeover Curves

The takeover time decreases as the level at which the best individual is introduced increases, as shown in Fig. 2. In order to observe this distinction more precisely, we fitted the obtained takeover curves with a parameterized function, so that we can simply compare the respective fitting parameter.

The takeover time in Genetic Algorithms is usually fitted by a logistic growth curve (1) that models bounded population growth at a rate of a [7]. The size of the population at time t is given by $N(t)$ and the number of individuals at the beginning is $N(0)$. This growth curve is not applicable to the cGA, because instead it exhibits quadratic growth [8]. We used a simple formulation for the quadratic growth in the first half and the symmetrical saturation phase in the second half of the takeover (2). This quadratic growth curve is also controlled by a single parameter a.

$$N(t) = \frac{1}{1 + (\frac{1}{N(0)} - 1)\, e^{-at}} \tag{1}$$

$$N(t) = \begin{cases} a\, t^2 & \text{if } t < T/2, \\ -a\, (t - T)^2 + 1 & \text{otherwise.} \end{cases} \tag{2}$$

We fitted the parameter a to the takeover curves obtained for different levels of the hierarchy. This is done by minimizing the mean squared error (MSE)

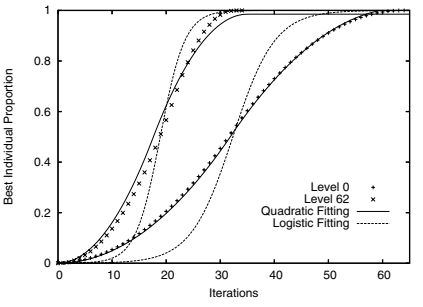

Fig. 3. Takeover curves fitted

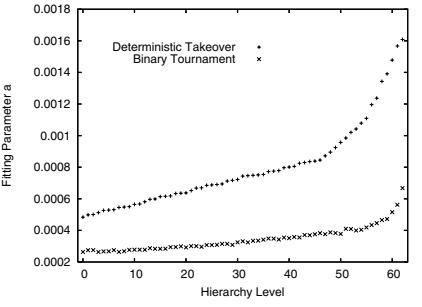

Fig. 4. The parameter a for different levels

between the observed data points and the parameterized curve. In Fig. 3 the deterministic takeover curves for introducing the best individual at level 0 and level 62 and the respective fitted curves are reported.

Similar to the takeover curves at different levels, the parameters for fitting these curves are also well ordered. In Fig. 4 we depict the parameter a returned by the fitting for the different levels. This is done for deterministic takeover and takeover with BT selection. The growth rates for the deterministic takeover are fully ordered and, with few exceptions, also for the BT takeover a increases as the level increases.

4 Computational Experiments

In this section, we present the results of our tests on a selected benchmark of problems. We briefly describe the problems composing the benchmark used in our testing in Section 4.1, and test the algorithms on those problems in Section 4.2.

4.1 Test Problems

For testing our algorithms, we have selected a benchmark of problems with many different features, such as multimodality, deceptiveness, use of constraints, or problem generators. The proposed problems are Onemax, the Massively Multimodal Deceptive Problem (MMDP), P-PEAKS, and the Minimum Tardy Task Problem (MTTP). We show in Table 2, for each of these problems, the fitness function to optimize, the chromosome length (size of the problem), and the value of this optimum.

Onemax. The objective of the Onemax [9] problem is to maximize the number of 1s in the binary string. We are considering strings of length 500 bits.

MMDP. The Massively Multimodal Deceptive Problem (MMDP) is a problem specifically designed for being difficult to solve by evolutionary algorithms [10]. It consists of k deceptive subproblems each of size 6 bits. These substrings are

Table 2. Benchmark of problems

Problem	Fitness function	n	Optimum
Onemax	$f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i$	500	500
MMDP	$f(\boldsymbol{s}) = \sum_{i=1}^{k} \text{fitness}(\boldsymbol{s_i});\ k = n/6$	240	40
P-PEAKS	$f(\boldsymbol{x}) = \frac{1}{n} \max_{1 \le i \le p} \{n - \text{HamDist}(\boldsymbol{x}, \text{Peak}_i)\}$	100	1.0
MTTP	$f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i \cdot w_i$	20 100 200	0.02429 0.005 0.0025

evaluated according to the number of ones in the substring as given in Fig. 5. It is easy to see that these subfunctions have two global obtima and a deceptive attractor in the middle point. We are using an instance of size $k = 40$ with optimum value 40.0. The number of local optima for this instance is very large (22^k), while there are only 2^k global solutions.

Unitation	Subfunction value
0	1.000000
1	0.000000
2	0.360384
3	0.640576
4	0.360384
5	0.000000
6	1.000000

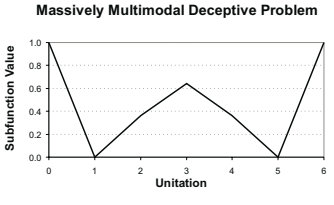

Fig. 5. Basic deceptive bipolar function ($\boldsymbol{s_i}$) for MMDP

P-PEAKS. In this multimodal problem generator [11] P peaks are generated and the fitness value is calculated as the distance to the nearest peak divided by the string length, having a maximum value of 1.0. We are using a test instance with 100 peaks of length 100.

MTTP. The Minimum Tardy Task Problem (MTTP) [12] is a task scheduling problem. Each task has a length (the time it takes), a deadline (before which it has to be finished), and a weight w_i. The objective is to maximize the weights of the scheduled tasks. We are considering here three instances of sizes 20, 100 and 200, taken from [13].

4.2 Results

In this section we present and analyze the results of our experiments. We compare the behaviour of H-cGA to the cGA algorithm. The same parametrization is used for the two algorithms (see Table 3), and both BT and Dissimilarity selection have been tested. We used a population of size 400 arranged in a grid of size

Table 3. Parametrization used in our algorithms

Population	20×20 individuals
Selection of Parents	itself + (BT or Dissimilarity)
Recombination	DPX, $p_c = 1.0$
Bit Mutation	Bit-flip, $p_m = 1/\#bits$
Replacement	Replace if Better
Stopping Criterion	Find Optimum or Reach 2500 Generations

Table 4. Results for the different test functions. Given are the success rate, average number of required steps to reach the optimum, standard deviation, maximum and minimum, and statistical significance of the results.

Problem	Algorithm	Success	Avg	Stddev	Min– Max	p-value
Onemax	cGA BT	100%	129.4	± 7.3	111.2–145.2	+
	cGA Dis	100%	140.7	± 8.1	121.6–161.2	
	H-cGA BT	100%	94.1	± 5.0	83.2–106.4	
	H-cGA Dis	100%	103.1	± 5.6	90.4–116.8	
MMDP	cGA BT	67%	202.4	± 154.7	120.8–859.2	+
	cGA Dis	97%	179.8	± 106.3	116.8–846.0	
	H-cGA BT	55%	102.6	± 76.1	68.8–652.8	
	H-cGA Dis	92%	122.3	± 111.7	73.2–837.6	
P-Peaks	cGA BT	100%	41.9	± 3.0	32.0– 48.4	+
	cGA Dis	100%	52.9	± 5.2	38.4– 66.0	
	H-cGA BT	100%	47.2	± 8.6	30.8– 71.2	
	H-cGA Dis	100%	81.1	± 17.1	45.2–130.8	
MTTP-20	cGA BT	100%	5.1	± 1.2	1.6– 8.0	+
	cGA Dis	100%	6.0	± 1.3	2.0– 9.2	
	H-cGA BT	100%	4.7	± 1.1	1.6– 7.2	
	H-cGA Dis	100%	5.5	± 1.2	2.8– 8.0	
MTTP-100	cGA BT	100%	162.2	± 29.3	101.6–241.6	+
	cGA Dis	100%	174.6	± 26.3	96.4–238.8	
	H-cGA BT	100%	138.3	± 35.4	62.0–245.6	
	H-cGA Dis	100%	132.4	± 26.2	64.0–186.8	
MTTP-200	cGA BT	100%	483.1	± 55.3	341.6–632.4	+
	cGA Dis	100%	481.0	± 71.6	258.8–634.8	
	H-cGA BT	100%	436.2	± 79.7	270.4–631.2	
	H-cGA Dis	100%	395.3	± 72.6	257.6–578.8	

20×20 with a Linear5 neighbourhood (the cell itself and its North, East, South and West neighbours are considered). In all our experiments, one parent is the center individual itself and the other parent is selected either by BT or Dissimilarity selection (it is ensured that the two parents are different). An individual is replaced only if the newly generated fitness value is better. The recombination

method used is the two point crossover (DPX), and the selected offspring is the one having the largest part of the best parent. The mutation and crossover probabilities are 1.0 and bit mutation is performed with probability $1/\#bits$ for genome string of length $\#bits$. In order to have statistical confidence, all the presented results are average over 100 runs, and the analysis of variance –ANOVA– statistical test (or Kruskal-Wallis if the data is not normally distributed) is applied to the results. We consider in this paper a 95% confidence level.

In Table 4 we present the results we have obtained for all the test problems. Specifically, we show the success rate (number of runs in which the optimum was found), and some measures on the number of evaluations made to find the optimum, such as the average value, the standard deviation, and the maximum and minimum values. The results of our statistical tests are in column p-values, where symbol '+' means that there exists statistically significant differences. The evaluated algorithms are cGA and H-cGA both with BT and Dissimilarity selections.

As can be seen in Table 4, both the cGA and the H-cGA algorithms were able to find the optimum value in each run for all the problems, with the exception of MMDP. For this problem, the cGA algorithm achieves slightly better success rates than H-cGA. Regarding the average number of evaluations required to reach the optimum, the hierarchical algorithm always outperforms cGA, except for the P-peaks problem. Hence, the use of a hierarchical population allows us to accelerate the convergence speed of the algorithm to the optimum, while it retains the interesting diversity management of the canonical cGA.

If we now compare the results of the algorithms when using the two different studied selection schemes, we notice that with the Dissimilarity selection the success rate for the MMDP problem can be increased for both the cGA and the H-cGA algorithm.

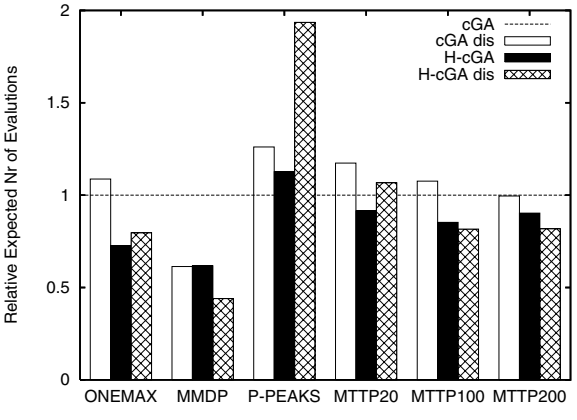

Fig. 6. Expected number of evaluations required to reach the optimum, relative to the steps required by cGA, for the different benchmark problems

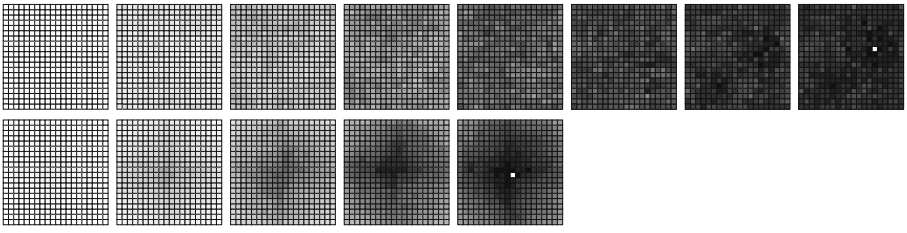

Fig. 7. Evolution of the population for the cGA (top) and the H-cGA (bottom)

In Fig. 6 we plot the expected number of evaluations, defined as the average number of evaluations divided by the success rate, required to find the optimum value for each problem. The displayed results are relative to the expected number of evaluations for the cGA.

The expected number of evaluations is increased when using the Dissimilarity selection compared to the equivalent algorithms with BT. For the cGA the Dissimilarity selection was able to reduce the number of required evaluations only for the MMDP problem. But for the H-cGA it proved to be useful also for the two large MTTP instances. In general, the expected number of evaluations is lower for the two studied versions of H-cGA.

Finally, in order to illustrate the effects of using a hierarchical population in the cGA, we show a sample run of the cGA (top) and the H-cGA (bottom) algorithm in Fig. 7. The pictures are snapshots of the population taken every 50 iterations for the MDDP problem until the optimum is found (iteration 383 for cGA and 233 for H-cGA). The darker an individual is coloured the higher its fitness is; the white cell in the last image contains the maximum fitness. As can be seen, the H-cGA algorithm quickly focuses on promising solutions, while at the same time different solutions of lower quality are kept at the outside of the hierarchy.

5 Conclusions and Future Work

In this work we have presented a new algorithm called hierarchical cGA, or H-cGA. We included the idea of establishing a hierarchy among the individuals of the population of a canonical cGA. With this hierarchical model we achieve different levels of the exploration/exploitation tradeoff of the algorithm in distinct zones of the population simultaneously. We studied these specific behaviours at different hierarchy levels by examining the respective takeover rates.

We have compared the H-cGA with two different selection methods to the equivalent cGAs and the hierarchical algorithm performed better on almost all test functions. The newly proposed Dissimilarity selection was not useful in all the scenarios, but for the H-cGA algorithm it improved the performance for both the MMDP and the MMTP functions. This selection promotes more diversity into the population but, as a consequence, the convergence is usually slower.

Future work will concentrate on other forms of hierarchies, with respect to the shape or to the criterion that allows ascending in levels. Furthermore, we will consider position-dependant algorithm changes, to emphasize different search strategies on different levels of the hierarchy.

References

1. Manderick, B., Spiessens, P.: Fine-grained parallel genetic algorithm. In Schaffer, J., ed.: Proceedings of the Third International Conference on Genetic Algorithms, Morgan-Kaufmann (1989) 428–433
2. Alba, E., Tomassini, M.: Parallelism and Evolutionary Algorithms. IEEE Trans. on Evolutionary Computation **6**(5) (2002) 443–462
3. Cant-Paz, E.: Efficient and Accurate Parallel Genetic Algorithms. 2nd edn. Volume 1 of Book Series on Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers (2000)
4. Alba, E.: Parallel Metaheuristics: A New Class of Algorithms. Wiley (2005)
5. Alba, E., Dorronsoro, B.: The exploration/exploitation tradeoff in dynamic cellular evolutionary algorithms. IEEE Trans. on Evolutionary Computation **9**(2) (2005) 126–142
6. Janson, S., Middendorf, M.: A hierarchical particle swarm optimizer and its adaptive variant. IEEE Systems, Man and Cybernetics - Part B **35**(6) (2005) 1272–1282
7. Goldberg, D., Deb., K. Foundations of Genetic Algorithms. In: A comparative analysis of selection scheme used in genetic algorithms. Morgan Kaufmann Publishers (1991) 69–93
8. Giacobini, M., Tomassini, M., Tettamanzi, A., Alba, E.: Synchronous and asynchronous cellular evolutionary algorithms for regular lattices. IEEE Transactions on Evolutionary Computation **9**(5) (2005) 489–505
9. Schaffer, J., Eshelman, L.: On crossover as an evolutionary viable strategy. In: 4th ICGA, Morgan Kaufmann (1991) 61–68
10. Goldberg, D., Deb, K., Horn, J.: Massively multimodality, deception and genetic algorithms. In: Proc. of the PPSN-2, North-Holland (1992) 37–46
11. Jong, K.D., Potter, M., Spears, W.: Using problem generators to explore the effects of epistasis. In: 7th ICGA, Morgan Kaufman (1997) 338–345
12. Stinson, D.: An Introduction to the Design and Analysis of Algorithms. The Charles Babbage Research Center, Winnipeg, Manitoba, Canada (1985 (second edition, 1987))
13. S.K., Bäck, T., Heitkötter, J.: An evolutionary approach to combinatorial optimization problems. In: Proceedings of the 22nd annual ACM computer science conference (CSC '94), New York, NY, USA, ACM Press (1994) 66–73