Escaping Local Optima via Parallelization and Migration

Vincenzo Cutello, Angelo G. De Michele and Mario Pavone

Department of Mathematics and Computer Science,
University of Catania,
v.le A. Doria 6 – 95125 Catania, Italy
angelo.demichele@gmail.com
{cutello, mpavone}@dmi.unict.it

Abstract. We present a new nature-inspired algorithm, $mt - GA$, which is a parallelized version of a simple GA, where subpopulations evolve independently from each other and on different threads. The overall goal is to develop a population-based algorithm capable to escape from local optima. In doing so, we used complex trap functions, and we provide experimental answers to some crucial implementation decision problems. The obtained results show the robustness and efficiency of the proposed algorithm, even when compared to well-known state-of-the art optimization algorithms based on the clonal selection principle.

Keywords: Genetic algorithms, multi-threaded genetic algorithms, trap functions, toy problems, global optimization, optimization.

1 Introduction

Many real-world problems are hard to solve because the parameters that influence the problem structures and dynamics over time are unknown and often impossible to be analytically solved. On such problems Evolutionary Algorithms (EA) seem to be perform quite well, primarily when the solutions are not known *a priori* or they are nonlinear. However, the EA it must be designed in such way to prevent getting trapped into local optima.

One feature that plays a key role on this issue is the diversity of individuals introduced into the population. Population diversity strongly influences, as known, both the *exploration* of the search space, and the *exploitation* of the information gained during the evolutionary process. The aim of this work, therefore, is to develop a population-based algorithm capable to escape from local optima maintaining itself *blind* on the problem's domain, i.e. general purpose and not tailored to the any specific problem. In order to achieve our result, we focused on developing a Genetic Algorithm (GA) based on Multi-Threads, where subsets of individuals evolve on different threads. Moreover, our GA is also equipped with a migration operator which allows pairs of solutions (individuals) to migrate between threads. Migration allows our algorithm to perform a careful and deep search of the solution space. In what follows, we will denote our algorithm with $mt - GA$.

To check the ability of $mt - GA$ to escape local maxima, we used *Trap Functions,* well-known toy problem, used for understanding the dynamics and search's ability of

a generic evolutionary algorithm [10]. We designed two variants of $mt - GA$ (synchronous, and asynchronous threads), and the trap functions are used as testbed in order to analyze and determine which variant is more suitable for our aims. Moreover, what solutions to select for the migration, and in what place, is also subject of this study.

2 Multi-Population GA and migrations

There is a very vast literature on multi-population Genetic Algorithms and the associated concept of migration. A comprehensive analysis of it, is certainly way beyond the scope of our contribution. For sake of completeness, however, we will mention few of the obtained results, especially in relation to the key decision about migration.

The standard GA has a single population which tries to explore the entire search space. The Multi-population approach tries to divide the search space into several parts and then uses a number of small populations to search them separately. The approach can obviously be parallelized and so such separate searches may run either synchronously or asynchronously (we will talk about this in the next sections). If we allow the different populations to "communicate", a key concept is migration.

In [7], the author proposes the random immigrants approach, sociologically inspired by the flux of immigrants that between generations move from one place to another. Technically, some individuals of the current population are replace with random individuals, called random immigrants, from another population at every generation. The choice of individuals to be replaced is usually governed by two main strategies: replacing random individuals or replacing the worst ones. In many ways, random immigrants may act as "genetic mutations" and thus the ratio of number of random immigrants to the the population size, is usually set to a small value.

The effect of the policy used to select migrants and the individuals they replace on the selection pressure in parallel evolutionary algorithms (EAs) with multiple populations is investigated in [1]. In particular, four possible combinations of random and fitness-based emigration and replacement of existing individuals are considered.

In [11] the author investigates a hybrid memory and random immigrants scheme, called memory-based immigrants, and a hybrid elitism and random immigrants scheme, called elitism-based immigrants, for improving the performance of genetic algorithms in dynamic environments.

To underline the importance on modern technical issues of such techniques, we mention the work in [2], where multi population GA's with immigrants schemes are designed for the dynamic dhortest path routing problem in mobile networks.

3 The Trap Functions

The trap functions problem [4, 5] is a toy yet complex problem that simply takes as input the number of $1's$ of bit strings of length ℓ. The fitness function $f(x)$ is defined as a function $\widehat{f}(\cdot)$ of the number of 1-bits, $u(x)$, in the binary input string x

$$f(x) = \widehat{f}(u(x)) = \widehat{f}\left(\sum_{k=1}^{\ell} x_k\right) \tag{1}$$

The definition of the function $\widehat{f}$, which depends on few numerical parameters, gives rise to two different scenarios: *simple trap function* and *complex trap function*. Both scenarios are shown in figure 1.

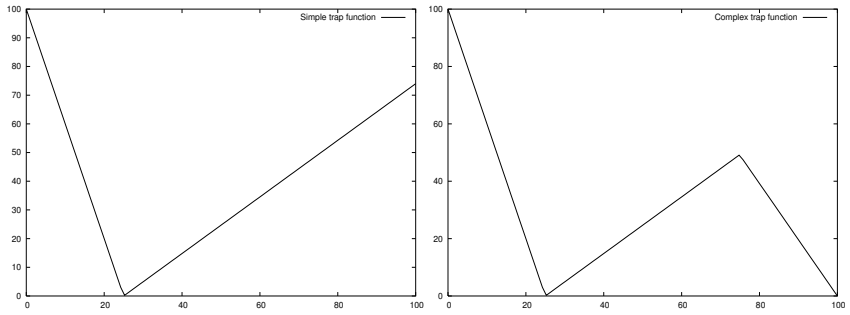

Fig. 1. Simple (left plot) and complex (right plot) trap functions.

The simple trap function is characterized by one global optimum (for a bit string of all 0's) and one local optimum (for a bit string of all 1's) that are the complement *bit-wise* of each other (see left plot in figure 1). Its formal definition is given by:

$$\widehat{f}(u(x)) = \begin{cases} \frac{a}{z}\left(z - u(x)\right), & \text{if } u(x) \leq z \\ \frac{b}{\ell - z}\left(u(x) - z\right), & \text{otherwise} \end{cases} \tag{2}$$

where the 3 parameters a, b, and z are such that [8] $z \approx (1/4)\ell$; $b = \ell - z - 1$; $1.5b \leq a \leq 2b$; a multiple of z.

The complex trap function, defined using 4 parameters, is instead more difficult to investigate because there are two directions where the algorithm may get trapped (see right plot of figure 1).

It is formally defined as:

$$\widehat{f}(u) = \begin{cases} \frac{a}{z_1}(z_1 - u(x)), & \text{if } u(x) \leq z_1 \\ \frac{b}{\ell - z_1}(u(x) - z_1), & \text{if } z_1 < u(x) \leq z_2 \\ \frac{b(z_2 - z_1)}{z_2}\left(1 - \frac{1}{z_1}(u(x) - z_2)\right) & \text{otherwise.} \end{cases} \tag{3}$$

If z_1, similarly to z in the case of simple trap function, verifies $z \approx (1/4)\ell$, the value of parameter $z_2 > z_1$ could be fixed as $z_2 = \ell - z_1$. We also note that if we fix $z_2 = \ell$ the complex trap function becomes a simple trap one. Summing up, for both kinds of trap functions there are many possible choices for the parameters a, b and z, (with $z_1 = z$ and $z_2 = \ell - z_1$ for the complex trap function) [8]. Some values are shown in table 1 and we will use them for our experiments.

The plots in figure 1 were produced using the following parameter values: $\ell = 100, z = 25, a = 100, b = 74$ for the simple trap function, and $\ell = 100, z_1 = 25, z_2 = 75, a = 100, b = 74$ for the complex trap function.

In the next sections, we will focus our discussions on complex trap functions.

Table 1. Parameter values used by simple and complex trap functions.

type	ℓ	z	a	b
I	10	3	12	6
II	20	5	20	14
III	50	10	80	39
IV	75	20	80	54
V	100	25	100	74

4 $mt - GA$: a Multi-Threaded Genetic Algorithm

We began our work with the implementation of a simple, standard GA.

Therefore, we used some classical operators for recombination, mutation, and selection mechanism, namely 2*–point crossover*, *bit flip*, and *roulette wheel selection*. Based on this simple algorithm, we studied and tested the impact, and improvements produced by its parallelization where subpopulations evolve separately each on a different thread. The motivation on parallelize a GA via threads is not only the speeding up the running times, but also, and primarily, because it provides a method where different processes are running in cooperation among them, each with specific tasks, and sharing gained information. To strengthen the cooperation produced by the parallelization of GA, we have designed a migration approach, where k individuals migrate from the i-th thread to another. In this way we give the opportunity to exchange among them the gained information. Moreover, such an approach guarantees sufficient introduction of diversity into each subpopulation, which, on the whole, helps $mt - GA$ in escaping from local optima, in according with the aims of this work. Algorithm 1 shows the pseudocode of the designed *Multi-Threaded Genetic Algorithm* ($mt - GA$).

As a classical nature-inspired algorithm, $mt - GA$ starts with a random creation of the initial population ($P^{(t=0)}$), where each chromosome is represented as a bit string of lenght ℓ. Afterwards, the population is divided in subpopulations, as many as the number n of threads, which will evolve independently from each other. The individuals for any subpopulation are selected in sequential order from the overall population ($P^{(t=0)}$); therefore, the i-th subpopulation ($P_i^{(t=0)}$), i.e. the i-th thread, will be assigned the individuals $\left(\left((i-1) \times \frac{popsize}{n}\right) + 1\right), \ldots, \left(i \times \frac{posize}{n}\right)$.

From this moment on, all the subpopulations will evolve in independent way. Thus, the description of the subsequent steps, given for the thread i and the population P_i is equivalent for all n threads.

At each timestep t, the fitness function value is computed for each individual. Such an evaluation will, obviously, increase the value of the global variable FFE so to force the termination of the Algorithm within a finite number of steps, as described later. To select individuals for offspring generation, we implemented the classical *Roulette Wheel Selection* mechanism; given a chromosome $x \in P_i^t$ with fitness $f(x)$, the probability p_x that it will be selected is given by

$$p_x = \frac{f(x)}{\sum_{y \in P_i^t} f(y)}.$$

Algorithm 1 Pseudo code of $mt - GA$

$t \leftarrow 0$
$FFE \leftarrow 0$
$P^t \leftarrow$ Create_Initial_Population
for $i = 1$ to n **do**
  $P_i^t \leftarrow$ Assign the individuals $\left((i-1)(\frac{popsize}{n}) + 1\right), \ldots, i(\frac{popsize}{n})$
end for
while $FFE < T_{max}$ **do**
  for all Thread i $(1 \leq i \leq n)$ **do**
    Compute_Fitness(P_i^t)
    $FFE \leftarrow FFE + \left(\frac{popsize}{n}\right)$
    Select $(\frac{popsize}{n})$ individuals via *Roulette Wheel Selection*
    Recombination for generating offsprings via $2-$*point Crossover* and *Bit_Flip Mutation*
    k individuals migrate to the j-th thread ($1 \leq j \leq n$, and $i \neq j$)
  end for
  $t \leftarrow t + 1$
end while

Once two individuals have been chosen for mating, we use the 2-*point crossover* operator to generate two offspring.

As last step, to each offspring is applied a mutation operator, which is basically a bit flip of the selected gene. The mutation is performed with a probability p^i, which accounts for the independent evolution of the population in the thread. $mt - GA$ can can therefore better explore the search space, as well as exploit more efficiently the gained information. In particular, some threads work more on the exploration of wide regions of the landscape; whilst others on the exploitation of the solutions found. Finally, after crossover and mutation operators, k individuals from each thread migrate to other threads. This helps $mt - GA$ in escaping from local optima by introducing diversity into the subpopulations, and brings to a thread information discovered by other threads.

It is obviously crucial to decide which individuals should be chosen from each subpopulation to be migrate, and in which thread they should migrate. Another important issue is synchronicity among threads. If they run synchronized, the cooperation among them is strengthened. If they run in an asynchronous manner, it is possible that one thread does not receive any migrants, because is not running in that time. Thus, we have also designed a third variant of $mt - GA$, which includes a *Birth operator*, which restores the right size of the given subpopulation introducing new elements randomly generated. The three variants of $mt - GA$ (*synchronous*, *asynchronous*, and *asynchronous with birth operator*), have been the subject of our study and are described in section 5.

The algorithm terminates its execution when the fitness function evaluation number (FFE) is great or equal to T_{max}, i.e. the maximum number of allowed objective function evaluations .

Table 2. The three variants of $mt - GA$ on complex trap functions using *Elitism* approach as preservation strategy.

migrants	$Trap$	SR	AES	$best$	$mean$	σ	SR	AES	$best$	$mean$	σ
Synchronous Variant											
		migration place: next thread					*migration place: random thread*				
$best2$	$C(I)$	100	953.04	12.0	12.0	0.0	100	832.16	12.0	12.0	0.0
$best2$	$C(II)$	87	45040.92	20.0	18.61	3.59	96	43607.31	20.0	19.57	2.09
$best2$	$C(III)$	24	92272.91	80.0	41.86	21.85	58	93684.59	80.0	59.59	24.43
$best2$	$C(IV)$	2	135315.5	80.0	35.57	6.7	0	0.0	34.36	34.36	0.05
$best2$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$best + random$	$C(I)$	100	905.19	12.0	12.0	0.0	100	1082.91	12.0	12.0	0.0
$best + random$	$C(II)$	92	43512.0	20.0	19.15	2.89	98	39015.87	20.0	19.79	1.49
$best + random$	$C(III)$	35	84307.17	80.0	47.01	24.21	54	103432.78	80.0	57.46	25.01
$best + random$	$C(IV)$	0	0.0	34.36	34.36	0.05	1	111436.0	80.0	34.85	4.54
$best + random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$random$	$C(I)$	100	954.94	12.0	12.0	0.0	100	1049.46	12.0	12.0	0.0
$random$	$C(II)$	98	31512.93	20.0	19.79	1.49	97	29823.65	20.0	19.75	1.54
$random$	$C(III)$	57	99018.06	80.0	58.21	25.09	57	103747.83	80.0	58.58	24.82
$random$	$C(IV)$	1	142402.0	80.0	35.24	6.12	0	0.0	34.36	34.36	0.05
$random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
Asynchronous Variant											
		migration place: next thread					*migration place: random thread*				
$best2$	$C(I)$	100	1731.06	12.0	12.0	0.0	100	1866.19	12.0	12.0	0.0
$best2$	$C(II)$	76	58889.68	20.0	17.44	4.56	88	58618.03	20.0	18.75	3.4
$best2$	$C(III)$	27	91646.11	80.0	42.98	22.52	30	107803.47	80.0	44.5	23.24
$best2$	$C(IV)$	2	154088.5	80.0	35.28	6.39	1	52527.0	80.0	34.82	4.54
$best2$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$best + random$	$C(I)$	100	1649.15	12.0	12.0	0.0	100	1484.57	12.0	12.0	0.0
$best + random$	$C(II)$	80	48261.8	20.0	17.87	4.27	90	60251.61	20.0	18.93	3.2
$best + random$	$C(III)$	33	87326.87	80.0	46.21	23.79	42	99014.62	80.0	50.57	25.05
$best + random$	$C(IV)$	0	0.0	34.36	34.36	0.05	0	0.0	34.36	34.36	0.05
$best + random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$random$	$C(I)$	100	1341.8	12.0	12.0	0.0	100	1665.9	12.0	12.0	0.0
$random$	$C(II)$	99	39384.28	20.0	19.89	1.06	99	47915.18	20.0	19.89	1.06
$random$	$C(III)$	36	112243.72	80.0	47.65	24.28	34	83118.65	80.0	46.93	24.11
$random$	$C(IV)$	1	118636.0	80.0	34.82	4.54	0	0.0	34.36	34.36	0.052
$random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
Asynchronous Variant with Birth Operator											
		migration place: next thread					*migration place: random thread*				
$best2$	$C(I)$	100	501.11	12.0	12.0	0.0	100	511.01	12.0	12.0	0.0
$best2$	$C(II)$	100	27610.78	20.0	20.0	0.0	100	26663.05	20.0	20.0	0.0
$best2$	$C(III)$	44	106628.73	80.0	51.98	25.07	40	95407.25	80.0	50.22	24.69
$best2$	$C(IV)$	0	0.0	34.36	34.36	0.05	0	0.0	34.36	34.36	0.05
$best2$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$best + random$	$C(I)$	100	519.82	12.0	12.0	0.0	100	496.47	12.0	12.0	0.0
$best + random$	$C(II)$	100	23491.53	20.0	20.0	0.0	100	25244.08	20.0	20.0	0.0
$best + random$	$C(III)$	37	111004.54	80.0	48.14	24.42	46	118486.35	80.0	52.65	25.25
$best + random$	$C(IV)$	1	136321.0	80.0	34.82	4.54	1	51949.0	80.0	34.82	4.54
$best + random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$random$	$C(I)$	100	710.88	12.0	12.0	0.0	100	440.12	12.0	12.0	0.0
$random$	$C(II)$	100	24147.84	20.0	20.0	0.0	100	22398.65	20.0	20.0	0.0
$random$	$C(III)$	39	119943.66	80.0	49.04	24.75	35	112969.88	80.0	47.84	24.21
$random$	$C(IV)$	0	0.0	34.36	34.36	0.05	1	191156.0	80.0	34.82	4.54
$random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05

5 Experimental Results

In this section we present the study conducted in order to understand the validity of the novelties introduced. In particular, we concentrated on answering the questions:

1. which individuals should be selected as migrants?
2. to which thread should they migrate?
3. which variant of $mt - GA$ shows the best performances?

As anticipated in previous section (sec. 1), the three variants of $mt - GA$ which were implemented are (1) synchronous, (2) asynchronous, and (3) asynchronous with birth.

For the migration thread, we implemented two different protocols: (i) migrate to the next thread (i.e. from i to $i + 1$), and (ii) migrate to a randomly chosen thread.

Table 3. The three variants of $mt-GA$ on complex trap functions using *Substitution* approach as preservation strategy.

migrants	$Trap$	SR	AES	$best$	$mean$	σ	SR	AES	$best$	$mean$	σ
Synchronous Variant											
		migration place: next thread					*migration place: random thread*				
$best2$	$C(I)$	100	1133.5	12.0	12.0	0.0	100	1208.66	12.0	12.0	0.0
$best2$	$C(II)$	87	14040.73	20.0	18.61	3.59	100	18460.4	20.0	20.0	0.0
$best2$	$C(III)$	39	62181.23	80.0	49.07	24.73	91	75528.59	80.0	75.81	13.72
$best2$	$C(IV)$	1	135149.0	80.0	34.82	4.54	5	155389.8	80.0	36.64	9.95
$best2$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$best + random$	$C(I)$	100	987.61	12.0	12.0	0.0	100	1085.56	12.0	12.0	0.0
$best + random$	$C(II)$	99	29244.40	20.0	19.89	1.06	100	28607.12	20.0	20.0	0.0
$best + random$	$C(III)$	56	108547.59	80.0	57.7	25.16	78	117031.12	80.0	69.56	20.21
$best + random$	$C(IV)$	3	146127.0	80.0	35.73	7.78	1	241915.0	80.0	34.82	4.54
$best + random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$random$	$C(I)$	100	1142.13	12.0	12.0	0.0	100	1105.79	12.0	12.0	0.0
$random$	$C(II)$	100	22248.69	20.0	20.0	0.0	100	35201.9	20.0	20.0	0.0
$random$	$C(III)$	63	117734.73	80.0	62.24	23.84	72	117650.89	80.0	66.11	22.37
$random$	$C(IV)$	1	147825.0	80.0	34.82	4.54	2	116774.0	80.0	35.28	6.39
$random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
Asynchronous Variant											
		migration place: next thread					*migration place: random thread*				
$best2$	$C(I)$	100	2137.36	12.0	12.0	0.0	100	1954.92	12.0	12.0	0.0
$best2$	$C(II)$	92	49366.47	20.0	19.15	2.89	97	53139.79	20.0	19.68	1.82
$best2$	$C(III)$	47	110869.3	80.0	53.98	25.10	68	116860.8	80.0	64.03	23.33
$best2$	$C(IV)$	2	173201.0	80.0	35.28	6.39	2	130577.0	80.0	35.28	6.39
$best2$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$best + random$	$C(I)$	100	1512.2	12.0	12.0	0.0	100	2236.88	12.0	12.0	0.0
$best + random$	$C(II)$	97	46749.69	20.0	19.68	1.82	97	44977.67	20.0	19.68	1.82
$best + random$	$C(III)$	42	100841.66	80.0	50.86	24.84	66	127740.34	80.0	63.81	23.35
$best + random$	$C(IV)$	0	0.0	34.36	34.36	0.05	2	157680.5	80.0	35.28	6.39
$best + random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$random$	$C(I)$	100	2223.74	12.0	12.0	0.0	100	1325.07	12.0	12.0	0.0
$random$	$C(II)$	98	42429.27	20.0	19.79	1.49	100	41378.41	20.0	20.0	0.0
$random$	$C(III)$	49	107010.27	80.0	54.54	25.31	50	111112.34	80.0	54.65	25.35
$random$	$C(IV)$	0	0.0	34.36	34.36	0.05	1	128098.0	80.0	34.92	4.63
$random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
Asynchronous Variant with Birth Operator											
		migration place: next thread					*migration place: random thread*				
$best2$	$C(I)$	100	652.39	12.0	12.0	0.0	100	472.79	12.0	12.0	0.0
$best2$	$C(II)$	100	27712.16	20.0	20.0	0.0	100	23326.76	20.0	20.0	0.0
$best2$	$C(III)$	38	116789.77	80.0	49.52	4.45	66	121492.37	80.0	63.39	23.33
$best2$	$C(IV)$	1	148662.0	80.0	34.82	4.54	0	0.0	34.36	34.36	0.05
$best2$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$best + random$	$C(I)$	100	777.71	12.0	12.0	0.0	100	811.5	12.0	12.0	0.0
$best + random$	$C(II)$	100	26608.98	20.0	20.0	0.0	100	25715.95	20.0	20.0	0.0
$best + random$	$C(III)$	39	128374.16	80.0	49.69	24.52	56	117110.44	80.0	58.93	24.60
$best + random$	$C(IV)$	1	123084.0	80.0	34.82	4.54	2	166874.0	80.0	35.28	6.39
$best + random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$random$	$C(I)$	100	742.36	12.0	12.0	0.0	100	633.79	12.0	12.0	0.0
$random$	$C(II)$	100	21931.74	20.0	20.0	0.0	100	24375.95	20.0	20.0	0.0
$random$	$C(III)$	56	117295.36	80.0	58.04	25.01	50	113501.22	80.0	54.71	25.3
$random$	$C(IV)$	0	0.0	36.0	34.38	0.17	1	119316.0	80.0	34.82	4.54
$random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05

To choose k individuals for migration, instead, we implemented the following protocols: (a) the best k; (b) the best $k/2$, and the remaining $k/2$ randomly chosen; and (c) k randomly chosen.

A feature that plays also a central role in $mt-GA$, as well as in every evolutionary algorithm, is how to generate the new population for the next iteration. This decision, of course, influences the search ability, and then the overall performances of a generic algorithm.

We implemented three different strategies: *elitism*; *substitution*, and *no preservation*. The first strategy always maintains the best individuals found so far; the second one, instead, replaces the worst offspring with the best of its parents; finally, in the last one we do not preserve any individual.

To make a robust analysis of our study we have tested $mt-GA$ only on the complex trap functions, being the most difficult ones, especially when increasing the size of the search space, and enough challenging to answer the above open questions. After several

Table 4. The three variants of $mt-GA$ on complex trap functions using *no preservation* approach as preservation strategy.

Synchronous Variant											
migrants	$Trap$	SR	AES	$best$	$mean$	σ	SR	AES	$best$	$mean$	σ
		migration place: next thread					*migration place: random thread*				
$best2$	$C(I)$	100	1451.98	12.0	12.0	0.0	100	1721.94	12.0	12.0	0.0
$best2$	$C(II)$	98	70712.0	20.0	19.88	0.89	95	62842.19	20.0	19.47	2.32
$best2$	$C(III)$	0	0.0	29.25	29.25	0.0	0	0.0	29.25	29.25	0.0
$best2$	$C(IV)$	0	0.0	34.36	34.36	0.05	0	0.0	34.36	34.36	0.05
$best2$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$best + random$	$C(I)$	100	1825.64	12.0	12.0	0.0	100	1923.4	12.0	12.0	0.0
$best + random$	$C(II)$	100	61141.63	20.0	20.0	0.0	100	45439.82	20.0	20.0	0.0
$best + random$	$C(III)$	0	0.0	40.0	29.49	1.53	0	0.0	29.25	29.25	0.0
$best + random$	$C(IV)$	0	0.0	34.36	34.36	0.05	0	0.0	34.36	34.36	0.05
$best + random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$random$	$C(I)$	100	1561.16	12.0	12.0	0.0	100	1759.7	12.0	12.0	0.0
$random$	$C(II)$	100	41329.28	20.0	20.0	0.0	100	36417.68	20.0	20.0	0.0
$random$	$C(III)$	14	202747.08	80.0	40.54	17.84	27	169550.38	80.0	44.72	22.62
$random$	$C(IV)$	0	0.0	34.36	34.36	0.05	0	0.0	76.0	34.92	4.35
$random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
Asynchronous Variant											
		migration place: next thread					*migration place: random thread*				
$best2$	$C(I)$	100	4520.26	12.0	12.0	0.0	100	3726.43	12.0	12.0	0.0
$best2$	$C(II)$	90	70742.43	20.0	19.05	2.92	93	70598.37	20.0	19.31	2.55
$best2$	$C(III)$	0	0.0	29.25	29.25	0.0	0	0.0	29.25	29.25	0.0
$best2$	$C(IV)$	0	0.0	34.36	34.36	0.05	0	0.0	34.36	34.36	0.05
$best2$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$best + random$	$C(I)$	100	4060.6	12.0	12.0	0.0	100	3626.43	12.0	12.0	0.0
$best + random$	$C(II)$	100	51491.81	20.0	20.0	0.0	100	42154.58	20.0	20.0	0.0
$best + random$	$C(III)$	10	187785.3	80.0	37.18	15.07	42	149380.88	80.0	52.31	24.29
$best + random$	$C(IV)$	0	0.0	34.36	34.36	0.05	0	0.0	36.0	34.4	0.23
$best + random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$random$	$C(I)$	100	3963.11	12.0	12.0	0.0	100	3542.22	12.0	12.0	0.0
$random$	$C(II)$	100	50164.49	20.0	20.0	0.0	100	39202.16	20.0	20.0	0.0
$random$	$C(III)$	18	176831.11	80.0	41.68	19.42	76	147446.23	80.0	69.48	19.62
$random$	$C(IV)$	0	0.0	34.36	34.36	0.05	4	163231.75	80.0	36.58	9.62
$random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
Asynchronous Variant with Birth Operator											
		migration place: next thread					*migration place: random thread*				
$best2$	$C(I)$	100	2269.42	12.0	12.0	0.0	100	2041.41	12.0	12.0	0.0
$best2$	$C(II)$	100	58625.36	20.0	20.0	0.0	100	40907.38	20.0	20.0	0.0
$best2$	$C(III)$	0	0.0	29.25	29.25	0.0	0	0.0	29.25	29.25	0.0
$best2$	$C(IV)$	0	0.0	34.36	34.36	0.05	0	0.0	34.36	34.36	0.05
$best2$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$best + random$	$C(I)$	100	2012.56	12.0	12.0	0.0	100	1948.07	12.0	12.0	0.0
$best + random$	$C(II)$	100	48335.62	20.0	0.0	0.0	100	34830.15	20.0	20.0	0.0
$best + random$	$C(III)$	7	150069.58	80.0	34.88	13.67	23	147841.39	80.0	41.4175	21.11
$best + random$	$C(IV)$	0	0.0	34.36	34.36	0.05	0	0.0	36.0	34.38	0.17
$best + random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05
$random$	$C(I)$	100	2355.47	12.0	12.0	0.0	100	1195.44	12.0	12.0	0.0
$random$	$C(II)$	100	43478.27	20.0	20.0	0.0	100	28972.43	20.0	20.0	0.0
$random$	$C(III)$	12	173612.67	80.0	37.22	16.86	57	151960.47	80.0	59.14	24.54
$random$	$C(IV)$	0	0.0	34.36	34.36	0.05	1	232352.0	80.0	35.51	6.49
$random$	$C(V)$	0	0.0	49.33	49.33	0.05	0	0.0	49.33	49.33	0.05

experiments, and a preliminary investigation on the best parameter tuning, we have fixed ($popsize = 40$) as population size; ($n = 4$) as number of threads; and ($k = 2$) as number of migrants from one thread to another. The low population size is due to both the parallelization and the developed migration strategy, thanks to which $mt-GA$ maintains a good diversity into the subpopulations, performing a proper exploration and exploitation of the landscape.

All the presented experiments were performed on 100 independent runs, and the maximum number of fitness function evaluations allowed has been fixed to 2.5×10^5.

In tables 2, 3, and 4 we show the results obtained by our study in order to understand the right answers to our open questions. For each experiment and each variant, we show the success rate (SR); the average number of fitness evaluations to solution (AES); best solution found ($best$); the mean value of the best fitness values for all runs ($mean$); and the relative standard deviation (σ).

In table 2 we show the results obtained by the three variants of $mt - GA$ when the best solutions are always maintained in the new population (elitism approach). By inspecting these results, $mt - GA$ obtains the best performances for both synchronous and asynchronous variants when the $k = 2$ migrants are randomly chosen and the migration place is the next thread; whilst, instead, the last variant (asynchronous with birth operator) works better when the migration place is randomly chosen, and the migrants are selected among both best and random. All in all, this last variant seems to produce better results in all experiments. This can be explained because the introduction of new chromosomes balances the choice to focus the evolution onto the best solutions found so far (feature of any elitism approach).

Table 3 presents the results of $mt - GA$ when the worst offspring x_1 is replaced with the best one of its parents x_2; of course this is done if $f(x_1) < f(x_2)$. Using this strategy for the generation of the new population, the synchronous variant produces the best overall performances on all experiments conducted, and with respect to all the studied variants. In particular the absolute best results are obtained when the 2 best individuals of each thread become migrants, and the place of migration is randomly selected. Choosing randomly the migration thread helps considerably $mt-GA$ in finding better solutions independently on how the migrants are chosen, because in this way the cooperation between the $n = 4$ threads is improved. Also in this table, for both asynchronous variants the best place where to migrate is randomly chosen; the migrants, instead, are selected randomly for the simple variant, whilst with the birth operator they are chosen by picking the best, and one randomly. Comparing only these two last variants on table 3, is possible to see how the use of the birth operator helps $mt - GA$ to achieve a higher success rate.

For the last experiments, showed in table 4, we obtain a different behavior on both $mt - GA$ variants. In particular, without any preservation of the best solutions during the construction of the new population, the synchronous variant shows its worst performances (random migrants, and random migration place); whilst the asynchronous ones achieve the best results in overall. The best selection for both asynchronous variants is given by random migrants and random migration threads. With respect to previous tables, these experiments are the only one where the simple variant outperforms the one with the birth operator. By inspecting all three tables together, it is possible to claim that the best performances, in order to achieve the aims of this work, are given by the synchronous variant selecting the best two individuals for migrating in a thread randomly chosen.

In order to better understand the robustness of the performances and the quality of the solutions produced by all $mt - GA$ variants, we have compared the designed algorithm with two well-known clonal selection algorithms: CLONALG [6], and $opt - IA$ [3, 4]). We underline the fact that today $opt - IA$ represents one of the best bio-inspired algorithms for optimization tasks [9]. The showed results for these two algorithms have been taken mainly from [5], and are showed in table 5.

By inspecting this table, it is possible to see how the three variants of $mt - GA$ outperforms all compared algorithms, achieving higher values of success rate (SR), except for the macro version of opt-IA on the traps C(IV) and C(V). From an overall point of view by inspecting the obtained results, it is possible to claim that $mt - GA$, as

Table 5. Comparisons between the three best variants of $mt-GA$ and two clonal selection algorithms: CLONALG [6, 5], and $opt-IA$ [3, 4].

Trap	SR	AES	SR	AES	SR	AES	SR	AES	SR	AES
	opt-IA [3], [5]						CLONALG$_1$			
	Inv		Macro		Inv+Macro		$\left(\frac{1}{\rho}\right) e^{(-f)}$		$e^{(-\rho * f)}$	
C(I)	100	371.15	100	737.78	100	388.42	100	272.5	100	251.3
C(II)	100	44079.57	100	27392.18	100	29271.68	100	17526.3	10	191852.7
C(III)	0	-	54	115908.61	24	149006.5	0	-	0	-
C(IV)	0	-	7	179593.29	2	154925	0	-	0	-
C(V)	0	-	2	353579	0	-	0	-	0	-
	Synchronous		Asynchronous		Asynchronous & Birth		CLONALG$_2$			
C(I)	100	1208.66	100	3542.22	100	1195.44	100	254.0	100	218.4
C(II)	100	18460.4	100	39202.16	100	28972.43	29	173992.6	24	172434.2
C(III)	91	75528.59	76	147446.23	57	151960.47	0	-	0	-
C(IV)	5	155389.8	4	163231.75	1	232352.0	0	-	0	-
C(V)	0	-	0	-	0	-	0	-	0	-

well as its three variants, are competitive on optimization tasks; are able to get out from local optima; and, finally, they prove to us that the implemented strategies are efficient, and robust.

6 Conclusions

The overall aim of this work is to develop a multi population-based algorithm capable to escape from local optima, which are the main reason why a generic optimization algorithm fails. In order to achieve the fixed aim, it is crucial to answer questions such as: (1) the best performance is obtained by running threads in a synchronous way, which strengthen the cooperation, or in asynchronous form? (2) which individuals should be selected as migrants? and (3) to what thread should they migrate? Our Algorithm, denoted $mt-GA$, addresses those questions and it contains a migration strategy which improves a the cooperation between the subpopulations, yet maintaining a sufficient amount of diversity.

Trap functions are a classical toy problems that represent a really useful tool in order to well understand the main features of a given EA, albeit they are not of immediate scientific interest. In particular, trap functions are mostly used for understanding the dynamics, and search's ability of a generic evolutionary algorithm. Although there exist two different scenarios (simple and complex), we focused our experiments only on complex trap functions, since they are sufficiently challenging to properly evaluate the goodness of $mt-GA$.

Many experiments were conducted on different complex trap functions, from which we tried to provide the right answer the above questions. We have tested three variants of $mt-GA$: (1) synchronous, (2) asynchronous, and (3) asynchronous with birth; this last variant is necessary when one thread does not receive migrants because it may not be running at that specific time. The migration place, and which migrants to select were problems studied as well.

By inspecting all performed experiments, we concluded that the best overall results are obtained by the synchronous variant, where the $k = 2$ migrants correspond

to the best two individuals, and the migration place is randomly chosen. The two asynchronous variants instead show a slightly lower performances to the synchronous one, and they both obtain the best performance when the migrants, and migration place are randomly chosen.

Finally, in order to properly evaluate the efficiency, and robustness of $mt-GA$, we compared the best results obtained by each variant with two well-known optimization algorithms based on the clonal selection principle, CLONALG [6] and $opt-IA$ [3, 4] (belonging to Artificial Immune Systems class). In particular, the latter represents today the state-of-the-art for global optimization tasks. From the comparisons, we can see that the three variants of $mt-GA$ are very competitive with the other algorithms. This proves the efficiency and robustness of $mt-GA$, and the high success rates achieved on the most of the instances confirms us that $mt-GA$ is really suitable for the fixed aim.

Acknowledgements: We wish to thank the anonymous referees for their very valuable comments.

References

1. Erick Cantú-Paz: "*Migration Policies, Selection Pressure, and Parallel Evolutionary Algorithms*", J. Heuristics, vol. 7, no. 4, pp. 311-334, 2001.
2. H. Cheng, S. Yang: Multi-population Genetic Algorithms with "*Immigrants Scheme for Dynamic Shortest Path Routing Problems in Mobile Ad Hoc Networks*", Applications of Evolutionary Computation Lecture Notes in Computer Science Vol. 6024, pp 562-571, 2010.
3. V. Cutello, G. Nicosia, M. Pavone: "*Exploring the Capability of Immune Algorithms: A Characterization of Hypermutation Operators*", 3rd International Conference on Artificial Immune Systems (ICARIS), LNCS 3239, pp. 263–276, 2004.
4. V. Cutello, G. Narzisi, G. Nicosia, M. Pavone, G. Sorace: "*How to Escape Traps Using Clonal Selection Algorithms*", 1st International Conference on Informatics in Control, Automation and Robotics (ICINCO), INSTICC Press, vol. 1, pp. 322–326, 2004.
5. V. Cutello, G. Narzisi, G. Nicosia, M. Pavone: "*Clonal Selection Algorithms: A Comparative Case Study using Effective Mutation Potentials*", 4th International Conference on Artificial Immune Systems (ICARIS), LNCS 3627, pp. 13–28, 2005.
6. L. N. De Castro, F. J. Von Zuben: "*Learning and Optimization using the Clonal Selection Principle*", IEEE Transaction on Evolutionary Computation, vol. 6, no. 3, pp. 239–251, 2002.
7. J.J. Grefenstette: "*Genetic algorithms for changing environments*", In: Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, pp. 137144, 1992.
8. S. Nijssen, T. Back: "*An analysis of the behavior of semplified evolutionary algorithms on trap functions*", IEEE Transaction on Evolutionary Computation, vol. 7, no. 1, pp. 11–22, 2003.
9. M. Pavone, G. Narzisi, G. Nicosia: "*Clonal Selection - An Immunological Algorithm for Global Optimization over Continuous Spaces*", Journal of Global Optimization, Vol. 53, No. 4, pp. 769–808, 2012.
10. A. Prugel-Bennett, A. Rogers: "*Modelling GA Dynamics*", Theoretical Aspects of Evolutionary Computing, pp. 59–86, 2001.
11. S. Yang: "*Genetic Algorithms with Memory- and Elitism-Based Immigrants in Dynamic Environments*", Evolutionary Computation, Vol. 16, No. 3 , pp. 385-416, 2008.