Parallel Heterogeneous Genetic Algorithms for Continuous Optimization

Enrique Alba Francisco Luna Antonio J. Nebro
Departamento de Lenguajes y Ciencias de la Computación
E.T.S. Ingeniería Informática
Campus de Teatinos, 29071 Málaga (España)
{eat,flv,antonio}@lcc.uma.es

Abstract

In this paper we address the physical parallelization of a very efficient genetic algorithm (GA) known as gradual distributed real-coded GA (GD-RCGA). This search model naturally provides a set of eight sub-populations residing in a cube topology having two faces for promoting exploration and exploitation. The resulting technique has been shown to yield very accurate results on continuous optimization by using crossover operators tuned to exploit and explore the space inside each sub-population. Here, we encompass the first actual parallelization of the technique, and get deeper into the importance of running a synchronous versus an asynchronous version of the basic GD-RCGA model. Our results indicate that this model maintains a very high level of accuracy for continuous optimization when run in parallel, as well as we show the similarities between the sync and async versions. Finally, we show that async parallelization is really more scalable than the sync one, suggesting future research lines for WAN execution and new models of search based in the two-faced cube of the original model.

1 Introduction

The goal of this paper is to extend the existing work dealing with a new model for optimization in continuous domains with genetic algorithms (GAs). GAs are stochastic search techniques that iteratively improve a set of tentative solutions (population of individuals) by applying crossover operators (merging two or more parents to yield one or more offsprings) and mutations of their contents (random alterations of the problem variables).

The gradual distributed real-coded GA (GD-RCGA) model of search [8] is a kind of distributed technique that runs eight populations concurrently in a cubic topology with sparse migrations of individuals among them. Distributed algorithms are a subclass of decentralized evolutionary algorithms [1] aimed at reducing the convergence to local optima, promoting diversity, and finding alternative solutions to the same problem. The GD-RCGA model is suitable for the optimization of continuous functions, because it includes in the basic improvement loop of the algorithm the utilization of crossover operators for float genes (variables), engineered with fuzzy logic technology to deal with the traditional "fuzzy" GA concepts of exploration and exploitation.

The whole configuration of GD-RCGA is a cube with eight populations, four of them targeted to improve the exploration of the algorithm and another four ones aimed at exploiting the neighborhood of the best solutions. It performs sparse migrations of individuals inside and outside these two sets of four sub-algorithms (islands).

There exist some studies on GD-RCGA in the literature. However, although the algorithm exhibits a straightforward parallelization, only sequential implementations exist; in them, a concurrent execution of the islands is simulated at hand on a monoprocessor. The contribution of this work is, first, to provide a parallel implementation that really runs in a cluster of machines. This should show an improvement in terms of wall-clock time, and therefore should enlarge the complexity of the tasks solved by the algorithm. Additionally, we are interested in investigating the advantages that could outcome from an asynchronous design, instead of the synchronous search that the basic GD-RCGA suggests.

The paper is organized as follows. Section 2 presents the background to understand the GD-RCGA. In Section 3, we briefly introduce the problems contained in our benchmark. We then proceed to present the parameterization used here, and to analyze the results from a numerical and run time point of view in Section 4. Finally, we summarize the conclusions and discuss several lines for future research in Section 5.

2 Gradual Distributed Real-Coded Genetic Algorithms

In this section we describe the basic behavior of the *gradual distributed real-coded genetic algorithm* (GD-RCGA) [8], and explain how we have parallelized it as a set of concurrent objects using the JACO environment [12] in Java.

2.1 GD-RCGA

The present availability of crossover operators for real-coded genetic algorithms (RCGAs) allows the possibility of including in the same algorithm different exploration or exploitation degrees, which leads to the design of heterogeneous distributed RCGAs based on this kind of operators [7]. GD-RCGA is included into such class of heterogeneous algorithms, since it applies a different crossover operator in each of its component sub-populations. Fig. 1 contains a graphic outline of the algorithm.

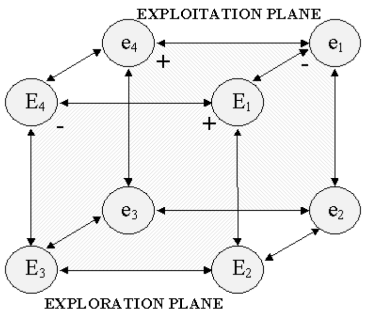

Figure 1. Structure of a GD-RCGA.

The distribution scheme of GD-RCGA is based on a hypercube topology with three dimensions. There are two important faces in this hypercube that have to be considered:

- The *front side* is devoted to exploration. It is made up of four sub-populations $E_1, \cdots, E_4$, in which several exploratory crossovers are applied.
- The *rear side* promotes exploitation. It is composed of sub-populations $e_1, \ldots\ , e_4$ that apply exploitative crossover operators.

One feature of GD-RCGA is the use of an *elitist strategy* [10] in the sub-populations, an important factor that may have some undesired influence on rapid convergence. However, this is necessary in order to solve complex problems because the best individual could disappear due to crossover or mutation.

The discussed structure is a parallel multi-resolution method using several crossover operators which allow to achieve simultaneously a diversified search (reliability), and an effective local tuning (accuracy). Furthermore, sub-populations are adequately connected for exploiting the multi-resolution in a *gradual* way, since the migrations between sub-populations belonging to different categories may induce the refinement/expansion of the best emerging zones.

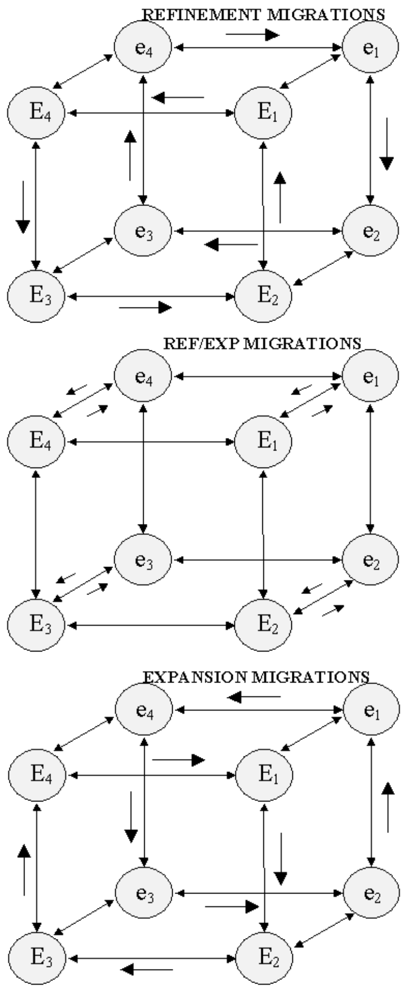

Figure 2. Three types of migration in a GD-RCGA.

Let us explain the migration schema and selection mechanism used in GD-RCGA for establishing a correct coordination between refinement and expansion.

2.1.1 Migration Schema

Distributed genetic algorithms behavior is strongly determined by the migration mechanism's action [4, 6]. We use an emigration model where copies of migrants are sent only towards immediate neighbors along a dimension of the hy-

	Exploitation				Exploration			
	e_4	e_3	e_2	e_1	E_1	E_2	E_3	E_4
Crossover	+	$\longleftarrow$		-	-	$\longrightarrow$		+
η_{min}	0.8	0.7	0.6	0.5	0.3	0.2	0.1	0.0

Table 1. Crossover exploration/exploitation degrees and η_{min} values for each sub-population.

percube, and each subsequent migration takes place along a different dimension of the hypercube. Particularly, the best element of each population is sent towards the corresponding sub-population every 160 iterations, as shown in Fig. 2. The sequence of application is, first, the refinement migrations; second, the refinement/expansion migrations; third, the expansion migrations; and finally, the sequence starts again. The place of an emigrant is always taken by the incoming individual.

This migration schema keeps a global elitist strategy, since the best element of all sub-populations is never lost, although it is moved from one sub-population to another.

2.1.2 Selection Mechanism

Here, we use the same selection mechanism as in [8]. *Linear ranking selection* is used because its pressure can be easily adjusted. In linear ranking selection, the individuals are sorted in order of raw fitness, and then the selection probability, p_s, of each chromosome C_i is computed according to its rank $rank(C_i)$, with $rank(C_{best}) = 1$, by using the following non-increasing assignment function:

$$p_s(C_i) = \frac{1}{N} \cdot \left(\eta_{max} - (\eta_{max} - \eta_{min}) \cdot \frac{rank(C_i)-1}{N-1}\right)$$

where N is the population size, and $\eta_{min} \in [0, 1]$ specifies the expected number of copies for the worst chromosome (the best one has $\eta_{max} = 2 - \eta_{min}$ expected copies). The selection pressure of linear ranking is determined by η_{min}. If η_{min} is low, high pressure is achieved, whereas if it is high, the pressure is low.

Linear ranking is combined with *stochastic universal sampling* [3]. This procedure guarantees that the number of copies of any chromosome is bounded by the floor and ceiling of its expected number of copies.

We have assigned a different selection pressure degree to every sub-population of the GD-RCGAs, by using the η_{min} values shown in Table 1.

We have implemented a GD-RCGA endowed with *fuzzy connectives-based* crossover operators, called GD-FCB [8]. FCB-crossover operators have different exploration or exploitation degrees which allow us to produce gradual effects by configuring them correctly.

2.2 GD-RCGA Parallelization using JACO

Two parallelizations of the GD-RCGA have been carried out by using the JACO runtime system [12]: a *Synchronous GD-RCGA* and a *Asynchronous GD-RCGA*. Both, the runtime system and the parallelizations, are discussed below.

2.2.1 The JACO Runtime System

JACO (JAva-based Concurrent Object system) is a runtime system implemented in Java. With JACO we can use Java to fast writing parallel programs according to a concurrent object model (high abstraction). This model considers a concurrent object as an active entity, with an internal state and a public interface. Objects can have also synchronization constraints, which disable some operations when they are not allowed.

2.2.2 Parallelization of GD-RCGA

Two versions of GD-RCGA have been implemented: *Synchronous GD-RCGA* and *Asynchronous GD-RCGA*. In the first one, every sub-population, for each migration phase, sends its best individual and it waits for another one coming from the corresponding neighbor. In the async mode, this consideration is not taken into account; thus, any individual stored in its buffer is valid to be included in the population at any time.

3 Problems

In this section we present the benchmark used to test our algorithms. We have analyzed the results of minimization experiments on six test functions and three real-word problems in order to better sustain our claims. We selected the same benchmark that [8] because it is very complete and for comparison purposes. They are described in sub-sections 3.1 and 3.2, respectively.

3.1 Test Functions

We have considered six classical and well-known test functions [8]: *sphere* model (f_{Sph}), *generalized Rosenbrock's* function (f_{Ros}), *Schwefel's problem* (f_{Sch}), *generalized Rastrigin's* function (f_{Ras}), *Griewangk's* function

f_{Sph}	$f_{Sph}(\overrightarrow{x}) = \sum_{i=1}^{n} x_i^2$ $-5.12 \leq x_i \leq 5.12$ $f_{Sph}(x^*) = 0$
f_{Ros}	$f_{Ros}(\overrightarrow{x}) = \sum_{i=1}^{n-1}(100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$ $-5.12 \leq x_i \leq 5.12$ $f_{Ros}(x^*) = 0$
f_{Sch}	$f_{Sch}(\overrightarrow{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2$ $-65.536 \leq x_i \leq 65.536$ $f_{Sch}(x^*) = 0$
f_{Ras}	$f_{Ras}(\overrightarrow{x}) = a \cdot n + \sum_{i=1}^{n}(x_i^2 - a \cdot \cos(\omega \cdot x_i))$ $a = 10, \omega = 2\pi$ $-5.12 \leq x_i \leq 5.12$ $f_{Ras}(x^*) = 0$
f_{Gri}	$f_{Gri}(\overrightarrow{x}) = 1 + \frac{1}{d}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$ $d = 4000$ $-600.0 \leq x_i \leq 600.0$ $f_{Gri}(x^*) = 0$
ef_{10}	$ef_{10}(\overrightarrow{x}) = f_{10}(x_1, x_2) + \cdots + f_{10}(x_{i-1}, x_i) + \cdots + f_{10}(x_n, x_1)$ $f_{10}(x, y) = (x^2 + y^2)^{0.25} \cdot [\sin^2(50 \cdot (x^2 + y^2)^{0.1}) + 1]$ $x, y \in (-100, 100]$ $f_{ef_{10}}(x^*) = 0$

Figure 3. Test functions.

(f_{Gri}), and *expansion of f10* (ef_{10}). Fig. 3 shows their formulation. The dimension of the search space is 10 for ef_{10} and 25 for the remaining test functions. Each one has its particular features:

- f_{Sph} is a continuous, strictly convex, and unimodal function.
- f_{Ros} is a continuous, nonseparable (nonlinear interactions among variables), and unimodal function, with the optimum located in a steep parabolic valley with a flat bottom (i.e., hard progress to the optimum).
- f_{Sch} is a continuous and unimodal function. Its difficulty concerns the fact that searching along the coordinate axes only gives a poor rate of convergence because the gradient of f_{Sch} is not oriented along the axes. It presents similar difficulties to f_{Ros}, but its valley is much narrower.
- f_{Ras} is a scalable, continuous, and multimodal function, which is made from f_{Sph} by modulating it with $a \cdot \cos(\omega \cdot x_i)$.
- f_{Gri} is a continuous and multimodal function. This function is difficult to optimize because it is nonseparable.
- f_{10} is a function that has nonlinear interactions between two variables. Its expanded version ef_{10} is built in such a way that it induces nonlinear interactions across multiple variables. It is nonseparable as well.

3.2 Real-World Problems

We have chosen the following three real-word problems: *systems of linear equations* [5], *frequency modulation sounds parameter identification problem* [14], and *polynomial fitting problem* [13]. They all are described below.

3.2.1 Systems of Linear Equations

The problem may be stated as solving for the elements of a vector X, given the matrix A and the vector B in the expression $A \cdot X = B$. The evaluation function used for these experiments is

$$f_{sle}(x_1, \cdots, x_n) = \left|\sum_{i=1}^{n}\sum_{j=1}^{n}(a_{ij} \cdot x_j) - b_j\right|.$$

Clearly, the best value for this objective function is $f_{sle}(x^*) = 0$. Furthermore, the range for parameters is $[-9.0, 11.0]$. Inter-parameter linkage (i.e., nonlinearity) is easily controlled in systems of linear equations, their nonlinearity does not deteriorate as increasing the number of parameters used, and they have proven to be quite difficult. We have considered a ten-parameter problem instance. Its matrices are included in Fig. 4.

$$\begin{vmatrix} 5 & 4 & 5 & 2 & 9 & 5 & 4 & 2 & 3 & 1 \\ 9 & 7 & 1 & 1 & 7 & 2 & 2 & 6 & 6 & 9 \\ 3 & 1 & 8 & 6 & 9 & 7 & 4 & 2 & 1 & 6 \\ 8 & 3 & 7 & 3 & 7 & 5 & 3 & 9 & 9 & 5 \\ 9 & 5 & 1 & 6 & 3 & 4 & 2 & 3 & 3 & 9 \\ 1 & 2 & 3 & 1 & 7 & 6 & 6 & 3 & 3 & 3 \\ 1 & 5 & 7 & 8 & 1 & 4 & 7 & 8 & 4 & 8 \\ 9 & 3 & 8 & 6 & 3 & 4 & 7 & 1 & 8 & 1 \\ 8 & 2 & 8 & 5 & 3 & 8 & 7 & 2 & 7 & 5 \\ 2 & 1 & 2 & 2 & 9 & 8 & 7 & 4 & 4 & 1 \end{vmatrix} \begin{vmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{vmatrix} = \begin{vmatrix} 40 \\ 50 \\ 47 \\ 59 \\ 45 \\ 35 \\ 53 \\ 50 \\ 55 \\ 40 \end{vmatrix}$$

Figure 4. Matrices of the linear equations problem instance.

3.2.2 Frequency Modulation Sounds Parameter Identification Problem

The problem is to specify six parameters a_1, ω_1, a_2, ω_2, a_3, ω_3 of the frequency modulation sound model represented by

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

with $\theta = (2 \cdot \pi/100)$. The fitness function is defined as the summation of square errors between the evolved data and the model data, as follows:

$$f_{fms}(a_1, \omega_1, a_2, \omega_2, a_3, \omega_3) = \sum_{t=0}^{100} (y(t) - y_0(t))^2$$

where the model data are given by the following equation:

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

Each parameter in the range $[-6.4, 6.35]$. This is a highly complex multimodal problem having strong epistasis, with minimum value $f_{fms}(x^*) = 0$.

3.2.3 Polynomial Fitting Problem

This problem lies in finding the coefficients of the following polynomial in z:

$$f(z) = \sum_{j=0}^{2k} c_j \times x^j,\ k > 0 \text{ is integer}$$

$$f(z) \in [-1, 1], \text{ for } z \in [-1, 1], \text{ and}$$
$$f(1.2) \geq T_{2k}(1.2) \text{ and } f(-1.2) \geq T_{2k}(-1.2)$$

where $T_{2k}(z)$ is a Chebyshev polynomial of degree $2k$.

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

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

It is a nine-parameter problem. A small correction is needed in order to transform the constraints of this problem into an objective function to be minimized, called f_{Cheb} (see [8] for the details). We consider that $C = (c_0, \cdots, c_8)$ is the solution to be evaluated, and $f_C(z) = \sum_{j=0}^{8} c_j \times z^j$.

Each parameter (coefficient) is in the range $[-5.12, 5.12]$. The objective function value of the optimum in $f_{Cheb}(C^*) = 0$.

4 Experiments

In this section, we present the parameters used for the GD-RCGA (Sub-section 4.1), and then discuss the experimental results obtained (Sub-section 4.2).

4.1 GD-RCGA Parameters

All our parallel GD-RCGA's use 20 individuals per subpopulation. The probability of mutation (p_m) is 0.125, and the crossover probability is 0.6. Moreover, $\lambda = 0.35$ for M-crossover of the FCB mechanism.

The mutation operator applied is *nonuniform* mutation [11]. This operator needs two parameters: b (set to value 5), which determines the degree of dependency on the number of iterations [9]; and T, which is the maximum number of generations. The original GD-RCGA work imposed a predefined number of iterations, but we cannot do the same because we want to measure the time to find equivalent solutions between the sync/async versions and with respect to the original work. Thus, we have calculated a maximum number of iterations for every problem (Table 2), and we have defined our goal as to reach the fitness value appearing in this table (that corresponds to the average of the best fitness function found in the reference work [8]).

Problem	MaxIt	TF	Problem	MaxIt	TF	Problem	MaxIt	TF
f_{Sph}	15000	2e-13	f_{Ros}	60000	9e0	f_{Sch}	5000	4e0
f_{Ras}	30000	4e-11	f_{Gri}	5000	2e-2	ef_{10}	15000	2e-3
f_{sle}	5000	4e1	f_{fms}	5000	1e1	f_{Cheb}	100000	2e2

Table 2. Maximum number of iterations (MaxIt) and target fitness (TF).

4.2 Results

Let us proceed with the analysis of the results. We show in Table 3 the execution time of synchronous and asynchronous versions of GD-RCGA for the nine problems. We consider first the one processor case. We can observe that the synchronous algorithm has produced a faster execution than the asynchronous one, and in fact we can notice that there exists statistical confidence for this claim (see the "+" symbols meaning significance of t-test) for six problems. However, we report similar times for the three complex instances. Although this is somewhat surprising, we can check in the right part of Table 3 that sync and async differences vanish when running the eight sub-algorithms in eight CPUs.

These results can be explained because of the very fast optimization achieved for the classical test functions, in which fitness evaluation is extremely fast and then residual times (setting up processes, delays in communications, etc.) dominate the whole execution time.

Then, we conclude that the run times provided by the parallel GD-RCGA model are relatively independent from the synchronization mechanism, because of its multi-migration scheme. However, we do report a great improvement in the efficiency of the asynchronous models with respect the synchronous ones. We include in the last column of Table 3 the `Improve` value as the difference between the efficiency of async and sync executions. One can notice that all but two values are positive and even very large numbers, meaning that the efficiency (and thus scalability and quality of the parallelization) is really higher in the asynchronous case. For f_{sle} the efficiency remains almost the same, and f_{Cheb} is an exception since there exist a large variance in the time and evaluations to find a solution.

Now, we turn to the analysis of the number of evaluations, i.e., the numerical effort to solve the problems with the sync and async algorithms. Overall, it seems that the two versions need a similar effort to solve all the optimization tasks, what is an expected result since all the machines have a similar computational power. There is an exception with the f_{Gri} function, in which the sync model is always more efficient numerically. For the rest, either they are similar ("-" symbol) or we find one of them more efficient than another depending on the number of CPUs. Hence, we cannot conclude nothing about the superiority of any of they two.

5 Conclusions and Future Work

In this paper we include a first study on parallelizing a sequential algorithm called GD-RCGA. The motivation for analyzing its parallel execution is that the algorithm has a extremely high accuracy for optimization problems coming from the continuous domain in mathematics. Since the algorithm performs a search based in the separate execution of eight sub-populations with migrations in a cube, it has been readily direct its physical parallelization.

Then, we have focused on the time and numerical efficiency. We have performed all the analysis under the assumption that our parallel versions must reach the same average solution quality as the one reported by the basic reference work. With this goal in mind, we have solved nine problems.

If we analyze the 1 CPU case, the sync model is faster; sync and async are quite similar when we shift to use 8 CPUs. There exist three exceptions (runtime point of view) and one exception (effort point of view) in which the sync model is faster than the async one, both on 1 and on 8 CPUs. We must conduct further research to confirm whether these few exceptions are due to the parallelization (JACO system) or it is an intrinsic property of the problems.

What remains a clear conclusion is that the asynchronous parallelization can provide a larger efficiency for all the problems, which confirm other existing results like [2]. The only exception is polynomial fitting, but we believe this is so because of the high variance in the results with the GD-RCGA model.

As a future work, we will construct the parallel models directly on Java, and possibly on C++. Also, we plan to introduce a restart technique, and a new modified model of search to improve the results on even more complex problems.

Acknowledgments

This work has been partially funded by the Ministry of Science and Technology and FEDER under contract TIC2002-04498-C05-02 (the TRACER project).

time	1 CPU			8 CPUs			
	Sync	Async	p-value	Sync	Async	p-value	Improve
f_{Sph}	51826	60221	+	9115	7890	+	24.3%
f_{Ros}	28173	111638	+	4797	8150	-	97.8%
f_{Sch}	9670	12952	+	1729	1956	-	12.9%
f_{Ras}	111367	121567	+	16867	16073	-	12.0%
f_{Gri}	10344	17533	+	1879	2339	+	24.8%
ef_{10}	54215	62840	+	8982	8710	+	14.7%
f_{sle}	1123	1104	-	563	566	-	-0.5%
f_{fms}	8894	10353	-	1714	1612	-	15.4%
f_{Cheb}	8863	8935	-	1430	11436	-	-67.7%

Table 3. Execution times (in ms) of the synchronous and asynchronous GD-RCGA.

evals	1 CPU				8 CPUs			
	Sync	Async	ratio	p-value	Sync	Async	ratio	p-value
f_{Sph}	215505.1	219571.5	1.02	-	233929.1	212543.7	0.90	+
f_{Ros}	110389.5	389569.4	3.53	+	114053.1	211710.7	1.86	-
f_{Sch}	33933.5	41857.7	1.23	+	33965.8	41046.1	1.21	-
f_{Ras}	444465.8	423820.1	0.95	+	432104.0	429567.4	0.99	-
f_{Gri}	36526.1	55806.3	1.53	+	38614.1	53480.6	1.39	+
ef_{10}	226262.1	229478.1	1.01	-	238077.1	233348.5	0.98	+
f_{sle}	176.2	176.3	1.00	-	176.0	176.9	1.01	-
f_{fms}	12759.5	14391.7	1.13	-	15728.9	15444.2	0.98	-
f_{Cheb}	6227.8	6059.3	0.97	-	5551.1	65007.7	11.71	-

Table 4. Number of evaluations of synchronous and asynchronous GD-RCGA.

References

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

[2] E. Alba and J. M. Troya. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *FGCS*, 17:451–465, January 2001.

[3] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In J. J. Grefenstette, editor, *Proc. of 2nd Int. Conf. Genetic Algorithms Appl.*, pages 14–21, Hillsdale, NJ, 1987. Lawrence Erlbaum.

[4] E. Cantú-Paz. A summary of research on parallel genetic algorithms. Technical Report 95007, Univ. Illinois, Urbana-Champaign, Illinois GA Laboratory, 1995.

[5] L. J. Eshelman, K. E. Mathias, and J. D. Schaffer. Convergence controlled variation. In R. Belew and M. Vose, editors, *Foundations of Genetic Algorithms 4*, pages 203–224, San Mateo, CA, 1997. Morgan Kaufmann.

[6] D. E. Goldberg, H. Kargupta, J. Horn, and E. Cantú-Paz. Critical deme size for serial and parallel genetic algorithms. Technical Report 95002, Univ. Illinois, Urbana-Campaign, Illinois Genetic Algorithms Laboratory, 1995.

[7] F. Herrera and M. Lozano. Heterogeneous distributed genetic algorithms based on the crossover operator. In *2nd IEE/IEEE Int. Conf. Genetic Algorithms Eng. Syst.: Innovations Appl.*, pages 203–208, 1997.

[8] F. Herrera and M. Lozano. Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(1):43–63, 2000.

[9] F. Herrera, M. Lozano, and J. L. Verdegay. Fuzzy connective based crossover operators to model genetic algorithms population diversity. Technical Report DECSAI-95110, University of Granada, 18071 Granada, Spain, March 1995.

[10] K. A. D. Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, Univ. Michigan, Ann Arbor, 1975.

[11] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, Germany, 1992.

[12] A. J. Nebro, E. Alba, F. Luna, and J. M. Troya. .NET as a platform for implementing concurrent objects. In B. Monien and R. Feldmann, editors, *8th Euro-Par*, pages 125–130, Paderborn, GE, August 2002. Springer-Verlag.

[13] R. Storn and K. Price. Differential evolution – a simple efficient adaptive scheme for global optimization over continuous spaces. Technical Report 95-012, Int. Compt. Sci. Inst., Berkeley, CA, 1995.

[14] S. Tsutsui and Y. Fujimoto. Forking genetic algorithm with blocking and shrinking modes. In S. Forrest, editor, *Proc. 5th Int. Conf. Genetic Algorithms*, pages 206–213, San Mateo, CA, 1993. Morgan Kaufmann.