Compact Optimization Algorithms with Re-sampled Inheritance

Giovanni Iacca[1][0000−0001−9723−1830] and Fabio Caraffini[2][0000−0001−9199−7368]

[1] Department of Information Engineering and Computer Science, University of Trento, Povo 38123, Italy
giovanni.iacca@unitn.it
[2] Institute of Artificial Intelligence, School of Computer Science and Informatics, De Montfort University, Leicester LE1 9BH, UK
fabio.caraffini@dmu.ac.uk

Abstract. Compact optimization algorithms are a class of Estimation of Distribution Algorithms (EDAs) characterized by extremely limited memory requirements (hence they are called "compact"). As all EDAs, compact algorithms build and update a probabilistic model of the distribution of solutions within the search space, as opposed to population-based algorithms that instead make use of an explicit population of solutions. In addition to that, to keep their memory consumption low, compact algorithms purposely employ simple probabilistic models that can be described with a small number of parameters. Despite their simplicity, compact algorithms have shown good performances on a broad range of benchmark functions and real-world problems. However, compact algorithms also come with some drawbacks, i.e. they tend to premature convergence and show poorer performance on non-separable problems. To overcome these limitations, here we investigate a possible algorithmic scheme obtained by combining compact algorithms with a non-disruptive restart mechanism taken from the literature, named Re-Sampled Inheritance (RI). The resulting compact algorithms with RI are tested on the CEC 2014 benchmark functions. The numerical results show on the one hand that the use of RI consistently enhances the performances of compact algorithms, still keeping a limited usage of memory. On the other hand, our experiments show that among the tested algorithms, the best performance is obtained by compact Differential Evolution with RI.

Keywords: Compact Optimization · Differential Evolution · Bacterial Foraging Optimization · Particle Swarm Optimization · Genetic Algorithm

1 Introduction

Compact Optimization [1] is a branch of Computational Intelligence Optimization devoted to the study of optimization algorithms characterized by limited memory requirements. From an algorithmic point of view, compact algorithms belong to the family of the Estimation of Distribution Algorithms (EDAs) [2], i.e. algorithms that instead of evolving a population of solutions (as is typically done in population-based optimization algorithms, such as Evolutionary Algorithms and Swarm Intelligence algorithms), build and update a probabilistic model of the distribution of solutions within the search space. Depending on the specific probabilistic model (Gaussian, binomial, etc.), different EDAs can be implemented. In this regard, the specificity of compact algorithms is that they employ a separate distribution for each variable of the problem, and update it as long as the evolutionary process proceeds. Therefore, differently from population-based algorithms where at least n D-dimensional arrays need to be stored in memory (being n the population size and D the problem dimension), compact algorithms need to store only a much more compact "Probability Vector" ($\mathbf{PV}$) that describes the parameters of the probabilistic model. For instance, binary-encoded compact algorithms use as $\mathbf{PV}$ a single D-dimensional array $\mathbf{p} = [p_1, p2, \ldots, p_D]$. Each $p_i \in [0, 1]$, $i = 1, 2, \ldots, D$, represents the probability that the $i-th$ variable has value 1 (i.e., the relative frequency in a corresponding "virtual population" of N_p individuals, with N_p a parameter of the algorithm). Similarly, real-valued compact algorithms based on Gaussian distributions use as $\mathbf{PV}$ two D-dimensional arrays: an array of means $\boldsymbol{\mu} = [\mu_1, \mu_2, \ldots, \mu_n]$ and an array of variances $\boldsymbol{\sigma} = [\sigma_1, \sigma_2, \ldots, \sigma_n]$, describing a (normalized) Gaussian distribution of each variable in the search space.

In the past two decades, the Compact Optimization concept has been declined in a number of compact algorithms, sparkling from the seminal works by Harik et al. [3] and Corno et al. [4], who devised a similar algorithm dubbing it respectively "compact Genetic Algorithm" (cGA) and "Selfish Gene" (SG). The family of compact algorithms was then extended to include improved versions of cGA [5,6]), real-valued cGA (rcGA) [7], compact Differential Evolution (cDE) [8] and many of its variants [9,10,11,12,13,14,15,16,17,18], compact Particle Swarm Optimization (cPSO) [19], compact Bacterial Optimization (cBFO) [20], and, more recently, compact Teaching-Learning Based Optimization (cTLBO) [21,22], compact Artificial Bee Colony (cABC) [23,24], and compact Flower Pollination Algorithm (cFPA) [25].

Due to their limited usage of memory, compact algorithms are particularly suited for embedded devices, such as Wireless Sensor Networks motes, wearable devices, embedded controllers for robots and industrial plants, etc. Unsurprisingly, the literature abounds with successful examples of compact algorithms applications based on this kind of devices: for instance, cDE has been applied especially in control applications, such as real-time hardware-in-the-loop optimization of a control system for a permanent-magnet tubular linear synchronous motor [8], or real-time trajectory optimization of robotic arms [26,18] and Cartesian robots [17]. In [6], cGA was applied to micro-controller design, while cPSO was used for optimizing a power plant controller in [19]. In [27], cABC was used for topology control in Wireless Sensor Networks. The more recent cTLBO was instead applied to train Artificial Neural Networks in [22].

In this paper, we aim to push forward this research area by tackling the two main drawbacks of compact algorithms, i.e. their tendency to premature convergence (as they do not keep an actual population, they do not maintain explicitly diversity), and a poorer performance on non-separable problems (which is due to the fact that they process each variable separately). To overcome these limitations, we study the effect of a special case of restart named Re-Sampled Inheritance (RI) [28,29], which simply generates a random solution and then recombines it -by using an exponential crossover operator similarly to Differential Evolution- with the best solution detected so far by the compact algorithm. Previous works [30,29] have shown that the RI mechanism is a simple yet effective way to improve the performance of an optimization algorithm, as it allows to escape from local optima while preserving some of the information from the current best, thus guaranteeing a kind of inheritance and avoiding an excessively disruptive restart (compared to other restart mechanisms). Our hypothesis is then that on the one hand the re-sampling should allow compact algorithms to escape from local optima, on the other hand the inheritance mechanism -since it processes blocks of variables- should enable an overall performance improvement especially (but not only) on non-separable problems. Moreover, as the RI mechanism only needs to sample a random solution and recombine it with the current best, it does not require any additional memory with respect to a compact algorithm, therefore allowing to keep a low memory consumption.

To assess the effect of RI on different compact algorithms, we apply it separately to four (real-valued) compact algorithms taken from the literature, namely cDE (specifically, its 'light" version [11]), rcGA, cPSO, and cBFO, and perform extensive tests on the CEC 2014 benchmark [31] in 10, 50 and 100 dimensions.

The rest of this paper is organized as follows: Section 2 presents the background concepts on Compact Optimization. Section 3 describes the general algorithmic scheme which combines compact algorithms with Re-Sampled Inheritance. The numerical results are then presented and discussed in Section 4. Finally, Section 5 concludes this work.

2 Background

In the rest of this paper, we focus on real-valued compact algorithms as they have been shown to perform better than their binary-encoded counterparts [7]. In case of real values, the general structure of a compact algorithm is quite straightforward and can be described as follows. First of all, for each i-th variable a Gaussian Probability Distribution Function (PDF) is considered, truncated within the interval $[-1, 1]$, with mean μ_i and standard deviation σ_i taken from the Probability Vector $\mathbf{PV} = [\boldsymbol{\mu}, \boldsymbol{\sigma}]$. The height of the PDF is normalized in order to keep its area equal to 1.

At the beginning of the optimization process, for each design variable i, $\mu_i = 0$ (unless Re-Sampled Inheritance is used, see next section) and $\sigma_i = \lambda$, where λ is a large positive constant (e.g. $\lambda = 10$), such that it simulates a uniform distribution (thus exploring the search space). Subsequently, a starting individual, **elite**, is generated by sampling each i-th variable from the corresponding PDF. For more details about the sampling mechanism, see [7].

Then, the iterative process starts. At each step, depending on the specific compact algorithm, a candidate solution **x** is generated by sampling one or more individuals from the current **PV**. E.g., in rcGA [7], **x** is obtained by generating a single individual and recombining it with **elite** with binomial crossover. In cDE [8], **x** is obtained by generating a mutated individual (for instance sampling three individuals from **PV** and applying the rand/1 DE mutation), and then recombining it with the current **elite** by using either binomial or exponential crossover. Other compact algorithm paradigms, such as cPSO and cBFO, use the same mechanism for sampling new individuals, but apply different algorithmic operators inspired from the corresponding biological metaphor to generate a new candidate solution **x**. In all cases, the fitnesses of **elite** and **x** are compared and, according to the chosen elitism scheme (persistent or non-persistent, see [1]), **elite** is replaced by **x**. Furthermore, the fitness comparison is used to update the **PV**, i.e. it changes its μ and σ values by "moving" the Gaussian PDF towards the better solution and "shrinking" the PDF around it. Details for this update mechanism are given in [7]. This iterative process is executed until a certain stop condition is met. The pseudo-code of a general real-valued compact algorithm is given in Algorithm 1.

Algorithm 1: General structure of a compact algorithm

```
1  initialize PV = [$\boldsymbol{\mu}, \boldsymbol{\sigma}$];
2  generate elite by means of PV;
3  while stop condition is not met do
4      generate candidate solution x (according to the specific operators);
5      compare fitness of x and elite;
6      if elite replacement condition is true then
7          elite = x;
8      end if
9      update PV;
10 end while
11 return elite
```

3 Compact Optimization Algorithms with Re-Sampled Inheritance

The general scheme of a compact algorithm with Re-Sampled Inheritance is shown in Algorithm 2. The only difference w.r.t. the original compact algorithm shown in Algorithm 1 is the RI component, which enables the restart mechanism with inheritance, and is activated at the end of each execution of the compact algorithm (that is continued for a given % of the total budget).

The Re-Sampled Inheritance (see [28,29] for more details) first randomly generates a solution **x** from a uniform distribution within the given search space. Then, it recombines **x** with the current best solution $\mathbf{x_{best}}$ by applying the exponential crossover used in Differential Evolution. More specifically, a random initial index is selected in $[0, D)$, and the corresponding variable is copied from $\mathbf{x_{best}}$ into **x**. Then, as long as a (uniform) random number $rand(0,1)$ is less than or equal to Cr, the design variables from $\mathbf{x_{best}}$ are copied into the corresponding positions of $\mathbf{x}$, starting from the initial index. Cr, the crossover rate, is a parameter affecting the number of variables inherited from $\mathbf{x_{best}}$, and is set as in [11], i.e. $Cr = 1/\sqrt[D\alpha]{2}$, where $D\alpha$ is the expected number of variables that are copied from $\mathbf{x_{best}}$. As soon as $rand(0,1) > Cr$, the copy process is interrupted. The copy is handled as in a cyclic buffer, i.e. when the D-th variable is reached during the copy process the next to be copied is the first one. When the copy stops, the fitness of $\mathbf{x}$ is compared with that of $\mathbf{x_{best}}$. If the newly generated solution $\mathbf{x}$ outperforms the current best solution, the latter gets updated (i.e. $\mathbf{x_{best}} = \mathbf{x}$). The compact algorithm is then restarted after setting its mean value $\boldsymbol{\mu}$ (that is used in the Probability Vector $\mathbf{PV}$) equal to the new restarted point, i.e. $\boldsymbol{\mu} = \mathbf{x}$. This way, the new initial distribution is centered in a new point which, despite being randomly sampled, still contains some inheritance from the current best solution. At the end of each compact optimization routine, see Algorithm 1, an elite solution is returned and compared for replacement against the current best solution, as shown in Algorithm 2.

Algorithm 2: Compact algorithm with Re-Sampled Inheritance

```
1  generate a random solution $\mathbf{x}$ in the search space and set $\mathbf{x_{best}} = \mathbf{x}$;
2  while stop condition is not met do
3      // Compact algorithm
4      set $\boldsymbol{\mu} = \mathbf{x}$ and run compact algorithm as in Alg. 1 (for a % of the budget);
5      if elite is better than $\mathbf{x_{best}}$ then
6          $\mathbf{x_{best}} = \mathbf{elite}$;
7      end if
8      // Re-Sampled Inheritance
9      generate a random solution $\mathbf{x}$ (from a uniform distribution);
10     generate $i = round\,(D \cdot rand\,(0,1))$;
11     $\mathbf{x}[\mathbf{i}] = \mathbf{x_{best}}[\mathbf{i}]$;
12     $k = 1$;
13     while $rand\,(0,1) \leq Cr$ and $k < D$ do
14         $i = i + 1$;
15         if $i == D$ then
16             $i = 1$;
17         end if
18         $\mathbf{x}[\mathbf{i}] = \mathbf{x_{best}}[\mathbf{i}]$;
19         $k = k + 1$;
20     end while
21     if $\mathbf{x}$ is better than $\mathbf{x_{best}}$ then
22         $\mathbf{x_{best}} = \mathbf{x}$;
23     end if
24 end while
```

The rationale behind the RI mechanism is then to restart the algorithm from a partially random solution, i.e. a solution that is randomly generated but still inherits part of the variables from the current best. This way the restart is not entirely disruptive, but preserves at least a block of (an expected number of) $D\alpha$

variables. This partial inheritance allows the algorithm to keep some information from one restart and the next one, but also to escape from local optima. From this point of view, the RI mechanism shares some resemblance with the Iterated Local Search (ILS) methods [32], that try to apply a small perturbation to the best-so-far solution during restart (in fact, as small as possible to not disrupt it much, but as large as needed to allow the local search to converge to a different local optimum). However, ILS has been especially designed for (and applied to) combinatorial optimization, while here we focus on continuous optimization.

4 Numerical Results

In the following we present the numerical results obtained on the CEC 2014 benchmark [31]. This benchmark is composed of 30 functions, with different properties in terms of separability, ill-conditioning, and landscape structure. In particular, it is worth noting that except for f_8 (Shifted Rastrigin's Function) and f_{10} (Shifted Schwefel's Function), all CEC 2014 benchmark functions are non-separable. Therefore this benchmark is particularly suited for testing the performance of optimization algorithms on non-separable problems.

We considered the following four real-valued compact algorithms, with the parametrization proposed in their original papers:

- cDE "light" [11] with exponential crossover and parameters: $N_p = 300$, $F = 0.5$, and $\alpha_m = 0.25$;
- rcGA [7], with persistent elitism and parameters: $N_p = 300$;
- cPSO [19], with parameters: $N_p = 300$, $\phi_1 = 0.2$, $\phi_2 = 0.07$, $\phi_3 = 3.74$, $\gamma_1 = 1$, and $\gamma_2 = 1$;
- cBFO [20], with parameters: $N_p = 300$, $C_i = 0.1$, and $N_s = 4$.

As for the corresponding versions with RI (dubbed, respectively as RIcDE, RIrcGA, RIcPSO and RIcBFO), the same parametrization was kept for the compact optimization process while the RI component was parametrized with $\alpha = 0.05$ (such that only 5% of the variables are inherited, on average, from the current best). A number of fitness function calls equal to 25% of the total computational budget was assigned to execute the compact algorithm after each restart. It should be noted that these are the only two parameters of the RI mechanism and they were empirically set after having observed their effect in preliminary experiments.

Furthermore, in order to assess the effect of the RI mechanism w.r.t. a simple random restart without any form of inheritance, we included in our experimental setup also four variants of the same compact algorithms where the restart was applied, with the same period of the RI variants (25% of the total computational budget), by simply applying a uniform re-sampling of a new solution $\mathbf{x}$ within the search space, and restarting the compact algorithm by setting $\boldsymbol{\mu} = \mathbf{x}$. We dub these compact algorithms with random restart, respectively, as RecDE, RercGA, RecPSO and RecBFO.

Finally, to provide a baseline for all the compact algorithms with/without restart tested in this paper, we evaluated the performance of a simple Random Walk (RW) algorithm where at each step a new solution is generated by applying a uniform re-sampling within the search space. From our numerical results (see Table 5) it can be seen that its performance is -as expected- considerably worse than any of the compact algorithms considered in the experiments, thus highlighting that the "compact" logic is more than a mere random sampling and performs significantly better w.r.t. pure uniform random searches.

To assess the scalability of all the algorithms, we performed experiments in 10, 50 and 100 dimensions. Thus, the total experimental setup consists of 13 algorithms (4 compact algorithms, 4 RI variants, 4 variants with random restart, and RW) and $30 \times 3 = 90$ optimization problems (i.e. 30 functions each tested in three different dimensionalities). On each benchmark function, each algorithm was executed for 30 independent runs, to collect statistics on the fitness values obtained in each run at the end of the allotted computational budget. Each run was executed for a total budget of $5000 \times D$ function evaluations, being D the problem dimension.

In the following, for the sake of brevity we will show only a compact representation of the main experimental results[3]. For that, we will use the sequentially rejective Holm-Bonferroni procedure [33,34]. This procedure consists of the following: considering N_{TP} test problems (in our case, 90) and N_A optimization algorithms, the performance obtained by each algorithm on each problem is computed. This is measured as average of the best fitness values obtained by the algorithm on that problem over multiple (in our case, 30) independent runs, at the end of the computational budget (in our case, $5000 \times D$ function evaluations). Then, for each problem a score R_i is assigned to each algorithm, being N_A the score of the algorithm displaying the best performance (i.e., assuming minimization, the minimum average of the fitness values) on that problem, $N_A - 1$ the score of the second best, and so on. The algorithm displaying the worst performance scores 1. These scores are then averaged, for each algorithm, over the whole set of N_{TP} test problems. The algorithms are sorted on the basis of these average scores. Indicating with R_0 the rank of an algorithm taken as reference, and with R_j for $j = 1, \ldots, N_A - 1$ the rank of the remaining algorithms, the values z_j are calculated as:

$$z_j = \frac{R_j - R_0}{\sqrt{\frac{N_A(N_A+1)}{6N_{TP}}}}. \quad (1)$$

By means of the z_j values, the corresponding cumulative normal distribution values p_j are derived. These p_j values are then compared to the corresponding δ/j where δ is the confidence interval, set to 0.05: if $p_j > \delta/j$, the null-hypothesis (that the algorithm taken as reference has the same performance as the j-th algorithm) is accepted, otherwise is rejected as well as all the subsequent tests.

[3] Detailed numerical results are available at: `http://www.cse.dmu.ac.uk/~fcaraf00/NumericalResults/RICompactOptResults.pdf`.

Let us first consider, for each compact algorithm, how the corresponding algorithms with RI and random restart perform w.r.t. the original compact algorithm without restart. Tables 1-4 show, respectively, the results of the Holm-Bonferroni procedure (in this case with $N_A = 3$) on cDE, rcGA, cPSO and cBFO based algorithms. The tables display the ranks, z_j values, p_j values, and corresponding δ/j obtained by this procedure. In each case we considered as reference algorithm the corresponding algorithm with RI, whose rank is shown in parenthesis in each table caption. Moreover, we indicate in each table whether the null-hypothesis (that the algorithm taken as reference has the same performance as each other algorithm in the corresponding table row) is accepted or not.

Table 1. Holm-Bonferroni procedure (reference: RIcDE, Rank = 2.63e+00)

j	Optimizer	Rank	z_j	p_j	δ/j	Hypothesis
1	RecDE	2.33e+00	-2.85e+00	2.21e-03	5.00e-02	Rejected
2	cDE	1.03e+00	-1.52e+01	2.44e-52	2.50e-02	Rejected

Table 2. Holm-Bonferroni procedure (reference: RIrcGA, Rank = 2.53e+00)

j	Optimizer	Rank	z_j	p_j	δ/j	Hypothesis
1	RercGA	2.47e+00	-6.32e-01	2.64e-01	5.00e-02	Accepted
2	rcGA	1.00e+00	-1.45e+01	3.07e-48	2.50e-02	Rejected

Table 3. Holm-Bonferroni procedure (reference: RIcPSO, Rank = 1.99e+00)

j	Optimizer	Rank	z_j	p_j	δ/j	Hypothesis
1	RecPSO	2.01e+00	2.11e-01	5.83e-01	5.00e-02	Accepted
2	cPSO	2.00e+00	1.05e-01	5.42e-01	2.50e-02	Accepted

Table 4. Holm-Bonferroni procedure (reference: RIcBFO, Rank = 1.84e+00)

j	Optimizer	Rank	z_j	p_j	δ/j	Hypothesis
1	RecBFO	1.88e+00	3.16e-01	6.24e-01	5.00e-02	Accepted
2	cBFO	1.54e+00	-2.85e+00	2.21e-03	2.50e-02	Rejected

From these Holm-Bonferroni procedures, we can observe that, except for the case of cPSO (where, quite surprisingly, cPSO shows the same performance as the corresponding algorithms with RI and random restart) in all other cases the algorithms with RI score a better rank than their corresponding compact algorithms. It is also interesting to note that, while in the case of cDE RIcDE performs better also than RecDE, on the other compact algorithms it results that the RI variant is statistically equivalent to the variant with random restart (note that the null-hypothesis is accepted in those cases). This equivalence between RI and random restart in the case of rcGA, cPSO and cBFO might be due to parametrization used for RI (number of restarts and number of variables inherited from the current best), as well as the different algorithmic logics used by these algorithms compared to cDE. In general though, these observations demonstrate that the use of restarts, and, especially in the case of cDE, the use of RI is beneficial in terms of optimization performance.

Finally, we provide an overall comparison of all the 12 compact optimization algorithms, in addition to the Random Walk algorithm. The resulting Holm-Bonferroni procedure is reported in Table 5, where RIcDE is considered as reference algorithm (as it shows the highest rank) and $N_A = 13$. In this case, except for RecDE, all the hypotheses are sequentially rejected, meaning that when all the algorithms are considered together, RIcDE is statistically equivalent to RecDE (although it shows a slightly higher rank), but it shows a statistically better performance (on average, on the entire set of tested problems) than all other algorithms under study. As expected, the Random Walk algorithm performs worse than all other papers. Moreover, the rank shows that each compact algorithm with RI (or random restart) performs better (on average) than the corresponding compact algorithm algorithm, confirming the fact that the RI component is beneficial to all the compact algorithms considered in our experimentation.

Table 5. Holm-Bonferroni procedure (reference: RIcDE, Rank = 1.09e+01)

j	Optimizer	Rank	z_j	p_j	δ/j	Hypothesis
1	RecDE	1.06e+01	-6.41e-01	2.61e-01	5.00e-02	Accepted
2	RecBFO	8.56e+00	-4.38e+00	5.86e-06	2.50e-02	Rejected
3	RIcBFO	8.52e+00	-4.44e+00	4.40e-06	1.67e-02	Rejected
4	RIrcGA	8.38e+00	-4.71e+00	1.22e-06	1.25e-02	Rejected
5	RercGA	8.30e+00	-4.86e+00	5.93e-07	1.00e-02	Rejected
6	cBFO	8.07e+00	-5.29e+00	6.04e-08	8.33e-03	Rejected
7	cDE	8.03e+00	-5.35e+00	4.30e-08	7.14e-03	Rejected
8	rcGA	5.68e+00	-9.74e+00	1.05e-22	6.25e-03	Rejected
9	RecPSO	3.97e+00	-1.29e+01	1.73e-38	5.56e-03	Rejected
10	RIcPSO	3.97e+00	-1.29e+01	1.73e-38	5.56e-03	Rejected
11	cPSO	3.89e+00	-1.31e+01	2.61e-39	5.00e-03	Rejected
12	RW	1.43e+00	-1.76e+01	6.80e-70	4.55e-03	Rejected

5 Conclusions

In this paper we have presented an algorithmic scheme for solving continuous optimization problems on devices characterized by limited memory. The proposed scheme is based on a combination of a compact algorithm with a restart mechanism based on Re-Sampled Inheritance (RI). We tested this scheme on four different compact algorithms presented in the literature (namely: cDE, rcGA, cPSO, and cBFO) and performed numerical experiments on a broad range of benchmark functions in several dimensionalities. Our experiments show that the use of RI consistently enhances the performances of compact algorithms, still keeping a limited usage of memory. In addition to that, we noted that among the tested algorithms the best performance was obtained by cDE with Re-Sampled Inheritance.

In future works, we will further investigate the effect of the parametrization on the proposed compact algorithms with Re-Sampled Inheritance, focusing in particular on the influence of the number of restarts, as well as the number of variables inherited from the best individual at each restart. We will also investigate alternative inheritance mechanisms, for instance based on binomial crossover or exponential crossover on shuffled variables.

Acknowledgments. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 665347.

References

1. Neri, F., Iacca, G., Mininno, E.: Compact Optimization, pp. 337–364. Springer (2013)
2. Larrañaga, P., Lozano, J.A.: Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer Academic Publishers (2001)
3. Harik, G.R., Lobo, F.G., Goldberg, D.E.: The compact genetic algorithm. IEEE transactions on evolutionary computation 3(4), 287–297 (1999)
4. Corno, F., Reorda, M.S., Squillero, G.: The selfish gene algorithm: A new evolutionary optimization strategy. In: ACM Symposium on Applied Computing. pp. 349–355 (1998)
5. Ahn, C.W., Ramakrishna, R.S.: Elitism-based compact genetic algorithms. IEEE Transactions on Evolutionary Computation 7(4), 367–385 (2003)
6. Gallagher, J.C., Vigraham, S., Kramer, G.: A family of compact genetic algorithms for intrinsic evolvable hardware. IEEE Transactions on evolutionary computation 8(2), 111–126 (2004)
7. Mininno, E., Cupertino, F., Naso, D.: Real-valued compact genetic algorithms for embedded microcontroller optimization. IEEE Transactions on Evolutionary Computation 12(2), 203–219 (2008)
8. Mininno, E., Neri, F., Cupertino, F., Naso, D.: Compact differential evolution. IEEE Transactions on Evolutionary Computation 15(1), 32–54 (2011)
9. Iacca, G., Mallipeddi, R., Mininno, E., Neri, F., Suganthan, P.N.: Global supervision for compact differential evolution. In: IEEE Symposium on Differential Evolution. pp. 1–8 (2011)
10. Iacca, G., Mallipeddi, R., Mininno, E., Neri, F., Suganthan, P.N.: Super-fit and population size reduction in compact differential evolution. In: IEEE Workshop on Memetic Computing. pp. 1–8 (2011)
11. Iacca, G., Caraffini, F., Neri, F.: Compact differential evolution light: High performance despite limited memory requirement and modest computational overhead. Journal of Computer Science and Technology 27(5), 1056–1076 (2012)
12. Iacca, G., Mininno, E., Neri, F.: Composed compact differential evolution. Evolutionary Intelligence 4(1), 17–29 (2011)
13. Iacca, G., Neri, F., Mininno, E.: Opposition-based learning in compact differential evolution. In: Applications of Evolutionary Computation. pp. 264–273 (2011)
14. Iacca, G., Neri, F., Mininno, E.: Noise analysis compact differential evolution. International Journal of Systems Science 43(7), 1248–1267 (2012)
15. Jewajinda, Y.: Covariance matrix compact differential evolution for embedded intelligence. In: IEEE Region 10 Symposium. pp. 349–354 (2016)

16. Mallipeddi, R., Iacca, G., Suganthan, P.N., Neri, F., Mininno, E.: Ensemble strategies in compact differential evolution. In: IEEE Congress on Evolutionary Computation. pp. 1972–1977 (2011)
17. Neri, F.: Memetic compact differential evolution for cartesian robot control. IEEE Computational Intelligence Magazine 5(2), 54–65 (2010)
18. Neri, F., Iacca, G., Mininno, E.: Disturbed exploitation compact differential evolution for limited memory optimization problems. Information Sciences 181(12), 2469 – 2487 (2011)
19. Neri, F., Mininno, E., Iacca, G.: Compact particle swarm optimization. Information Sciences 239, 96 – 121 (2013)
20. Iacca, G., Neri, F., Mininno, E.: Compact bacterial foraging optimization. In: Swarm and Evolutionary Computation. pp. 84–92 (2012)
21. Yang, Z., Li, K., Guo, Y.: A new compact teaching-learning-based optimization method. In: International Conference on Intelligent Computing. pp. 717–726 (2014)
22. Yang, Z., Li, K., Guo, Y., Ma, H., Zheng, M.: Compact real-valued teaching-learning based optimization with the applications to neural network training. Knowledge-Based Systems (2018)
23. Banitalebi, A., Aziz, M.I.A., Bahar, A., Aziz, Z.A.: Enhanced compact artificial bee colony. Information Sciences 298, 491–511 (2015)
24. Dao, T.K., Chu, S.C., Shieh, C.S., Horng, M.F.: Compact artificial bee colony. In: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. pp. 96–105 (2014)
25. Dao, T.K., Pan, T.S., Nguyen, T.T., Chu, S.C., Pan, J.S.: A compact flower pollination algorithm optimization. In: International Conference on Computing Measurement Control and Sensor Network. pp. 76–79 (2016)
26. Iacca, G., Caraffini, F., Neri, F., Mininno, E.: Robot base disturbance optimization with compact differential evolution light. In: Applications of Evolutionary Computation. pp. 285–294 (2012)
27. Dao, T.K., Pan, T.S., Nguyen, T.T., Chu, S.C.: A compact artificial bee colony optimization for topology control scheme in wireless sensor networks. Journal of Information Hiding and Multimedia Signal Processing 6(2), 297–310 (2015)
28. Caraffini, F., Iacca, G., Neri, F., Picinali, L., Mininno, E.: A CMA-ES super-fit scheme for the re-sampled inheritance search. In: IEEE Congress on Evolutionary Computation. pp. 1123–1130 (2013)
29. Caraffini, F., Neri, F., Passow, B.N., Iacca, G.: Re-sampled inheritance search: high performance despite the simplicity. Soft Computing 17(12), 2235–2256 (2013)
30. Caraffini, F., Iacca, G., Yaman, A.: Improving (1+1) Covariance Matrix Adaptation Evolution Strategy: a simple yet efficient approach. In: International Global Optimization Workshop (to appear 2018)
31. Liang, J., Qu, B., Suganthan, P.: Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization. Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore (2013)
32. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search: Framework and applications. In: Handbook of metaheuristics, pp. 363–397. Springer (2010)
33. Garcia, S., Fernandez, A., Luengo, J., Herrera, F.: A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. Soft Computing 13(10), 959–977 (2008)
34. Holm, S.: A simple sequentially rejective multiple test procedure. Scandinavian Journal of Statistics 6(2), 65–70 (1979)