Large Scale Optimization of Computationally Expensive Functions: an approach based on Parallel Cooperative Coevolution and Fitness Metamodeling

Ivanoe De Falco
Institute of High Performance Computing and Networking, National Research Council of Italy
Naples, Italy 80131
ivanoe.defalco@icar.cnr.it

Antonio Della Cioppa
Natural Computation Lab, DIEM, University of Salerno
Fisciano (SA), Italy 84084
adellacioppa@unisa.it

Giuseppe A. Trunfio
DADU, University of Sassari
Alghero (SS), Italy 07041
trunfio@uniss.it

ABSTRACT

In recent years, research on large scale global optimization (LSGO) provided metaheuristics able to effectively tackle real-valued objective functions depending on thousand of variables. Nevertheless, finding a suitable solution of LSGO problems often requires a significantly high number of fitness evaluations. Therefore, when the objective function is computationally expensive, metaheuristics-based solutions of LSGO problems can easily become infeasible or at least unattractive. In this paper, we address such an issue with a joint approach based on problem decomposition, fitness meta-modeling and parallel computing. We present a preliminary numerical investigation of the proposed methodology, which provided significant gains in terms of both exact evaluations of the objective functions and parallel speedup.

CCS CONCEPTS

•**Computing methodologies** → **Continuous space search; Randomized search;** *Shared memory algorithms; Heuristic function construction; Supervised learning by regression;*

KEYWORDS

Cooperative coevolution, large scale optimization, metamodeling

ACM Reference format:
Ivanoe De Falco, Antonio Della Cioppa, and Giuseppe A. Trunfio. 2017. Large Scale Optimization of Computationally Expensive Functions: an approach based on Parallel Cooperative Coevolution and Fitness Metamodeling. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3067695.3084214

1 INTRODUCTION

Over the past few years, there has been an increasing need of effective metaheuristics algorithms to tackle real-world problems involving the optimization of a high number of variables [1, 3, 4, 22]. For this reason, the field of research referred to as Large-scale global optimization (LSGO) is attracting an increasing number of researchers, according to the literature [9]. However, despite the progress made in the field, LSGO still requires a high number of objective function evaluations (e.g., from hundreds of thousands to millions) to provide a satisfactory near-optimal solution [21]. Therefore, for LSGO problems characterized by a computationally expensive objective function, as is the case of many real-world applications in engineering design, the use of metaheuristics proves often computationally infeasible. In spite of this, the issue of using metaheuristics for optimizing high-dimensional functions that are also computationally expensive did not attract enough research efforts.

A typical approach for dealing with expensive fitness evaluations consists of exploiting cheaper approximations of the objective function during the search (i.e., the so-called *surrogates* or *meta-models*) [8]. However, for high-dimensional problems also the quality of fitness surrogates is plagued by the curse of dimensionality, that is, it decreases rapidly as the number of involved variables increases.

To address the issue of LSGO with expensive objective functions, in this paper, we study the joint use of fitness surrogates and parallel computing in the context of a cooperative coevolutionary (CC) approach [17], in which the problem is decomposed into lower-dimensional *subcomponents*. The idea is that, thanks to the CC approach, the surrogates can operate within the lower dimensional search spaces originated from a decomposition of the original problem [5, 13]. This allows effectively approximating the exact fitness function at the obvious cost of building a higher number of surrogates (i.e., at least one for each subproblem originated by the CC decomposition), under the assumption that there is still a saving of computing time due to the high cost of the original fitness function. Moreover, the CC technique enables a natural data-parallel approach, given that the subcomponents can be evolved independently with only periodical exchange of information.

In the literature, accurate meta-modelling in conjunction with CC has been rarely used and the resulting framework has never been thoroughly investigated using LSGO problems [5, 13]. Nevertheless, quantifying the advantages provided by such a surrogate-assisted CC (SACC) is of some interest. For example, in case of *non-separable* problems [17], previous studies limited to some medium-sized test functions [14] showed that the uncertainties introduced by the approximate fitness can have contrasting effects. Another peculiar aspect of the CC technique is the periodical exchange of

GECCO '17 Companion, Berlin, Germany

978-1-4503-4939-0/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3067695.3084214

information between subcomponents, which should be more frequent in case of highly nonseparable problems. In case of a SACC, with meta-modelling at the subcomponent level, such interactions interfere with the accumulation of data on the fitness landscape that is required to train the surrogates.

To investigate the above issues, in the article we use a common meta-modelling approach, namely Radial Basis Function Network (RBFN) [26], within a CC optimizer based on Differential Evolution (DE) [20]. Moreover, we investigate the scalability of a data-parallel implementation for shared-memory architectures, based on OpenMP and C++. Our main aim is to quantify, using a standard LSGO test-bed, the gain of expensive function evaluations and computing time that can be achieved with the proposed SACC.

2 RELATED WORK

The joint use of the CC approach and fitness approximation is relatively rare in literature, especially when the application to LSGO problems is concerned. An earlier work based on CC optimization assisted by approximate models was conducted in 2001 by Nair and Keane in [10], where the authors applied a CC Genetic Algorithm to the design of a large space structure. However, such a study concerned a single test problem of relatively small size (i.e. 40 variables). Later, in [13], Ong and co-authors used a surrogate-assisted CC based on a real-coded GA and a RBFN [2] to approximate the exact fitness. The main aim of the study was to investigate the combined effect of non-separability and fitness meta-modelling. Also in this case, the approach was not applied to LSGO problems, since the investigation was based on only two objective functions, with 20 and 40 variables respectively. More recently, Goh and co-authors in [5] described a surrogate-assisted memetic CC algorithm for constrained optimization problems. In the proposed approach, a Local Search (LS) phase takes place within each subcomponent and for each individual by exploiting fitness surrogates built on the basis of the current archive of exact evaluations. Again, although the proposed algorithm has proved effective in some constraint-optimization benchmark test functions, it was not investigated in high-dimensional problems. A similar approach, implementing a CC with surrogate-assisted LS phase, was recently presented in [23]. The proposed CC optimizer was based on an adaptive DE (i.e., JADE [29]) and its effectiveness was investigated on a small set of 1000-dimensional problems. The results showed that in most cases the surrogate-assisted memetic approach can make the CC search more effective, although the gain in efficiency proved very sensitive to the algorithm's parameters. Another relevant study, including a specific investigation in LSGO, was illustrated in [6, 7], where the authors adopted a fitness inheritance (FI) strategy [19] in conjunction with a CC approach. In these cases, the results suggested no evidence of overall advantages provided by the FI strategy. The authors concluded that, likely, the adopted surrogate-assisted approach was not able to add any significant improvement to algorithms already including several sophisticated mechanisms that boost their performance.

Also the parallel implementations of CC algorithms, to be applied in the LSGO field, are relatively infrequent in literature. To our knowledge, the only relevant application was described in [16], where the author evaluated a master-slave CC optimizer on different hardware architectures and using a popular test suite with up to 1200 variables.

3 A SURROGATE-ASSISTED CC OPTIMIZER

According to the divide-and-conquer CC technique proposed by Potter and De Jong in [17], a d-dimensional continuous optimization problem is first decomposed into lower-dimensional easier-to-solve *subcomponents*. Then, a standard iterative algorithm is applied to separately evolve candidate solutions in each subcomponent, using fixed values for the variables not included within that particular subcomponent. Periodically, the fixed values are updated by the subcomponent they represent. More in details, a d-dimensional optimization problem is tackled using the CC approach by partitioning the d-dimensional set of search directions $G = \{1, 2, \ldots, d\}$ into k sets $G_1 \ldots G_k$, where each group G_i of directions defines a subcomponent. For example, a straightforward decomposition of the original d-dimensional search space consists of k subcomponents of the same dimension $d_k = d/k$, being the groups of directions associated to the subcomponents defined as:

$$G_i = \{(i-1) \times d_k + 1, \ldots, i \times d_k\} \quad (1)$$

Typically, the exchange of information between subcomponents is implemented through a common d-dimensional *context vector* $\mathbf{b}$, which is built using a representative individual provided by each subcomponent. Then, before its evaluation, each candidate solution is complemented through the appropriate elements of the context vector. In this study, the current best individual is used to represent a subcomponent in the context vector.

When applying a CC optimizer, an important aspect to be considered is *epistasis* [17, 25], which can negatively affect the convergence rate [17]. In practice, when interdependent variables are assigned to different subcomponents, it is more difficult to adjust correctly their values relying on a periodical access to the context vector, which represents only partially the convergence state of each subcomponent. To cope with the possible epistatic interaction between variables belonging to different subcomponents, a typical approach consists of using a dynamic grouping strategy called *Random Grouping* (RG) [27, 28], in which the directions of the original search space are periodically re-grouped in a random way to determine the CC subcomponents. In practice, the RG strategy increases the probability of having grouped together two dependent variables during the optimization process [12, 27]. The CC optimizer based on the RG strategy adopted in this study is outlined in Algorithm 1. The first step consists of randomly initializing both the population, composed of *numInd* individuals, and the context vector. The optimization is organized in *cycles* (lines 5-12 of Algorithm 1). During each cycle, first a new RG is executed by creating k groups of coordinates randomly drawn without replacement from the set $\{1, 2, \ldots, d\}$ (line 5). Then, the optimizers are activated in a round-robin fashion for the different subcomponents (lines 7-9). Before the next cycle, the context vector is updated using the current best individual of each sub-population (lines 11-12). Each optimizer is executed for *Ite* iterations at each cycle. The CC cycles terminate when the number of fitness evaluations reaches the value *maxFE*.

In this study, we adopted JADE [24, 29] as the evolutionary search algorithm within each subcomponent. JADE is an adaptive

variation of DE in which the parameters F and CR are evolved on the basis of their historical record of success. Moreover, JADE implements a mutation strategy called 'DE/current-to- $p$best', in which one of the vectors involved in every mutation is randomly selected among the $100p\%$ best individuals, being $p \in (0, 1]$ a further parameter. An outline of the modified JADE, able to exploit surrogate fitness, is shown in Algorithm 2. Because of the lack of space, we refer the reader to the original algorithm illustrated in [29] for a better understanding of basic DE and JADE functionalities. In the developed SACC, an approximate model of the objective function is built within each subcomponent in order to evaluate with a surrogate fitness most of the offspring produced in a CC cycle. In particular, we use a global meta-model trained with *online learning*, which consists of constructing an approximate model of the fitness landscape by using the data generated during the optimization process. As a consequence, at the beginning of each CC cycle a certain number of individuals have to be evaluated with the exact fitness in order to collect sufficient data for the model training. More in details, each subcomponent maintains an archive $\mathcal{A}$ of past evaluations $\langle \mathbf{x}, f(\mathbf{x}) \rangle$, where $\mathbf{x}$ is a member of the d_k-dimensional population and $f(\mathbf{x})$ is the corresponding value of the exact objective function. The purpose of $\mathcal{A}$ is to store a suitable number n_r of training patterns for building the fitness surrogate when required. At each activation of the subcomponent's optimizer, the archive of past evaluations is empty (see line 1 of Algorithm 2). Subsequently, every evaluation of an individual $\mathbf{x}$ with the original fitness function enriches $\mathcal{A}$ with a new pattern (lines 4 and 28). As shown in lines 21-23 of Algorithm 2, as soon as the archive reaches the minimum number of patterns, the fitness meta-model is built. Note that when $\mathcal{A}$ contains enough elements, a new RBFN is built at each JADE iteration with the aim of approximating at least the area of the fitness landscape corresponding to the current population. Then, a newly generated individual $\mathbf{x}$ is evaluated with the original fitness $f(\mathbf{x})$ if $|\mathcal{A}| < n_r$; otherwise, it is associated to an approximate fitness value $\hat{f}(\mathbf{x})$ (see lines 24-31 of Algorithm 2).

It is worth noting that, even after the meta-model has been built, we ensure that the exact fitness is assigned to the best individual in the offspring population (see lines 33-37 of Algorithm 2). This to reduce the risk of convergence misleading due to the differences between the surrogate and the real objective function [8].

The adopted meta-model is a RBFN, which can be seen as a special type of artificial neural network that uses radial basis functions in the activation layer [15] and is expressed as the following linear combination:

$$\hat{f}(\mathbf{x}) = \sum_{j=1}^{n_c} w_j \, h_j(\|\mathbf{x} - \mathbf{c}_j\|) \tag{2}$$

where $w_j \in \mathbb{R}$ are scalar weights, the n_c *centres* $\mathbf{c}_j \in \mathbb{R}^{d_k}$ are representative of the available data points in the archive $\mathcal{A}$ and the $h_j(\|\mathbf{x} - \mathbf{c}_j\|)$ are real-valued and radially symmetric functions, with $\|\cdot\|$ denoting the Euclidean norm. After some preliminary experiments with complex multi-modal problem, we selected quite general basis functions in the form:

$$h_j(\mathbf{x}) = \exp\left(-(\mathbf{x} - \mathbf{c}_j)^T \, \mathbf{R}_j \, (\mathbf{x} - \mathbf{c}_j)\right) \tag{3}$$

Algorithm 1: CCRG(f, d)

```
1  P ← initPopulation(d, numInd);
2  b ← initContextVector(P);
3  fitnessEvaluations ← 0;
4  while fitnessEvaluations < maxFE do
5      G = {G_1, ..., G_k} ← randomGrouping(d, k);
6      foreach G_i ∈ G do
7          P_i ← extractPopulation(P, G_i);
8          ⟨P_i, best_i, FE⟩ ← optimizer(f, P_i, b, G_i, Ite);
9          pop ← storePopulation(P_i, G_i);
10         fitnessEvaluations ← fitnessEvaluations + FE;
11     foreach G_i ∈ G do
12         b ← updateContextVector(best_i, G_i, b);
13 return f(b) and b;
```

where $\mathbf{R}_j^{-1} = diag(2\sigma_{1j}^2, \ldots, 2\sigma_{d_k j}^2)$. In practice, instead of using a global scaling parameter σ for all basis functions, as often found in the literature, we introduce a scaling parameter σ_{ij} for each basis function and direction as suggested in [18]. The adopted RBFN model depends on the vector of $n_c\,(1 + 2\,d_k)$ parameters $\mathbf{u} = \{\mathbf{w}, \boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_{n_c}, \mathbf{c}_1, \ldots, \mathbf{c}_{n_c}\}$, where $\mathbf{w}$ collects the weights w_i and each vector $\boldsymbol{\sigma}_j$ contains the scaling factors of the basis function centred in $\mathbf{c}_j$. As suggested in [18], we find a suitable value for $\mathbf{u}$ using a three-phase learning. In particular, we first select the number $n_c \leq n$ of centres. Then, a few iterations of a k-means clustering algorithm are used to partition the archive $\mathcal{A}$ into n_c clusters, whose centroids are the initial values of $\mathbf{c}_i$. Subsequently, the values σ_{ij} are simply initialized to the variances of cluster centres $\mathbf{c}_i$ in each direction, and the weights $\mathbf{w}$ are randomly initialized in $[-1, 1]$. In the third and final phase, we adjust all the parameters in $\mathbf{u}$ in order to minimize the error $E = \sum_{q=1}^{n}(y^{(q)} - \hat{f}(\mathbf{x}^{(q)}))^2/2$. This is done using a fixed number of a gradient descent procedure in which $\mathbf{u}$ is iteratively moved in the direction of the negative gradient $-\nabla E$ using a small learning rate η. According to some preliminary experiments, with the small-sized subcomponents used in this study (i.e., $d_k = 4$), a relatively small number of iterations of the above procedure are sufficient to find a suitable set of parameters for the RBFN fitness surrogate.

3.1 Parallel implementation

The parallel implementation was developed for multi-threaded execution, on shared-memory architectures, specifically exploiting the problem decomposition provided by the CC approach. In Algorithm 3 we outline the optimization procedure where, after the initialization of population and context vector, each thread executes in parallel the CC cycles within the available budget of exact fitness evaluations (lines 4-23). More in detail, in lines 7-10 a single thread performs the RG procedure by randomly reordering the search directions and distributing the groups of d_k variables among all the threads. Since the number of groups is not always a multiple of the number of threads, the function *computeIterations* tries to balance the computation by attributing a specific number of DE iterations to each thread (i.e., a thread with a greater number of subcomponents will execute less iterations on them). Note that this does not affect

Algorithm 2: optimizer(f, P, $\mathbf{b}$, G_i, $nIte$)

```
1  Empty archive 𝒜 of past evaluations;
2  for i ← 1 to |P| do
3      ϕ[i] ← f(P[i]);
4      Insert P[i] and ϕ[i] in 𝒜;
5  FE← |P|;
6  for g ← 1 to nIte do
7      surrogateTrained← false;
8      for i ← 1 to |P| do
9          Generate CR_i = N(μ_CR, 0.1) and F_i = Cauchy(μ_F, 0.1);
10         Randomly choose x_b from the 100p% best elements of P;
11         r_1 ← RandomInteger(1, |P|) with r_1 ≠ i;
12         r_2 ← RandomInteger(1, |P|) with r_2 ≠ r_1 ≠ i ;
13         ȳ_i ← P[i] + F_i · (x_b − P[i]) + F_i · (P[r_1] − P[r_2]) ;
14         j_rand ← RandomInteger (1, d_k);
15         for j ← 1 to d_k do
16             if j = j_rand or rand(0, 1) < CR_i then
17                 C[i][j] ← ȳ_i[j];
18             else
19                 C[i][j] ← P[i][j];
20     for i ← 1 to |C| do
21         if surrogateTrained=false and |𝒜| ≥ n_r then
22             f̂ ← trainSurrogateFitness(𝒜);
23             surrogateTrained← true;
24         if surrogateTrained=false then
25             γ[i] ← f(C[i]);
26             hasExactFitness[i] =true;
27             FE← FE +1;
28             Insert C[i] and γ[i] in 𝒜;
29         else
30             γ[i] ← f̂(C[i]);
31             hasExactFitness[i] =false;
32     j_b ← indexOfBestIndividual(γ);
33     while hasExactFitness[j_b] =false do
34         γ[j_b] ← f(C[j_b]);
35         Insert C[j_b] and γ[j_b] in 𝒜;
36         FE← FE +1;
37         hasExactFitness[j_b] =true;
38         j_b ← indexOfBestIndividual(γ);
39     m_CR ← m_F ← c_F ← c_CR ← 0;
40     for i ← 1 to |C| do
41         if γ[i] ≤ ϕ[i]) then
42             P[i] ← C[i]; ϕ[i] ← γ[i];
43             m_CR ← m_CR + CR_i;   m_F ← m_F + F_i^2;
44             c_CR ← c_CR + 1;   c_F ← c_F + F_i;
45     μ_CR ← (1 − c) · μ_CR + c · m_CR/c_CR;
46     μ_F ← (1 − c) · μ_F + c · m_F/c_F;
47 return P, j_b FE;
```

Algorithm 3: ParallelCCRG(f, d)

```
1  P ← initPopulation(d, numInd);
2  b ← initContextVector(P);
3  fitnessEvaluations ← 0;
4  do in parallel with nt threads
5      tid← getThreadId();
6      while fitnessEvaluations < maxFE do
7          single thread
8              𝒢 = {G_1, ..., G_k} ← randomGrouping(d, k);
9              G = {𝒢^(1), ..., 𝒢^(nt)} ← distributeGroups(𝒢, nt);
10             {Ite^(1), ..., Ite^(nt)} ← computeIterations(G, Ite);
11         barrier
12         FE^(tid) ← 0;
13         b^(tid) ← b;
14         foreach G_i ∈ 𝒢^(tid) do
15             P_i ← extractPopulation(P, G_i);
16             ⟨P_i, best_i, FE⟩ ← optimizer(f, P_i, b^(tid), G_i, Ite^(tid));
17             P ← storePopulation(P_i, G_i);
18             FE^(tid) ← FE^(tid)+FE;
19         atomic update
20             fitnessEvaluations ← fitnessEvaluations +FE^(tid);
21         foreach G_i ∈ 𝒢^(tid) do
22             b ← updateContextVector(best_i, G_i, b);
23         barrier
24 return f(b) and b;
```

the final result thanks to the cyclic reordering of search directions. At line 13, to improve data locality during the JADE iterations, each thread makes a local copy of the current context vector, which is accessed very frequently for evaluating the candidate solutions. At lines 14-18 each thread evolves its subcomponents and locally accumulates the number of exact fitness evaluations. The latter is then atomically added to a global variable at line 20. Subsequently, each thread updates the parts of the context vector corresponding to its subcomponents (line 22). Such an update can be executed without coordination between threads because each of them uses its local copy $\mathbf{b}^{(tid)}$ during the optimization. Moreover, the global vector $\mathbf{b}$ can be updated in parallel because the different threads operate on different elements of it. The barrier at line 23 ensures that all the threads finished their job before the start of the next cycle. This is necessary because, as mentioned above, each CC cycle begins with a random reordering and redistribution of search directions.

4 NUMERICAL RESULTS

4.1 Comparison between SACCDE and CCDE

To evaluate the effect of meta-modelling within subcomponents, we adopted the seven functions proposed for the CEC'08 special session on LSGO [21] listed in Table 1. In particular, all functions $f_1 - f_6$ have a global optimum point $\mathbf{x}^*$ which is shifted by a different amount in each dimension. Functions f_1 and f_2 are uni-modal while the remaining functions are multi-modal. Moreover, while f_1, f_4, and f_6 are separable functions, f_2, f_3, f_5, f_7, are non-separable. We conducted the numerical experiments on the above test functions using search spaces with 100, 500 and 1000 dimensions. The error value defined as $|f(\mathbf{x}) - f(\mathbf{x}^*)|$, where $f(\mathbf{x}^*)$ is the global optimum, was adopted as a performance metric for functions $f_1 - f_6$. Instead,

Table 1: Characteristics of the adopted benchmark test functions. The dimension d of the search space was set to 100, 500 and 1000. Note that test problems $f_1 - f_6$ are shifted with different values in every direction.

	Name	Domain	Characteristics
f_1	Sphere	$[-100, 100]^d$	Uni-modal, separable
f_2	Schwefel's 2.21	$[-100, 100]^d$	Uni-modal, non-separable
f_3	Rosenbrock's	$[-100, 100]^d$	Multi-modal, non-separable
f_4	Rastrigin's	$[-5, 5]^d$	Multi-modal, separable
f_5	Griewank's	$[-600, 600]^d$	Multi-modal, non-separable
f_6	Ackley's	$[-32, 32]^d$	Multi-modal, separable
f_7	FastFractal	$[-1, 1]^d$	Multi-modal, non-separable

for function f_7, the global optimum of which is usually unknown, the convergence was tested directly on the function value.

The two JADE parameters referred to as c and p in [29] were set to 0.1. Also, we used ten JADE iterations per cycle and subcomponent (i.e., $Ite = 10$ in Algorithm 1). However, as explained above, the number of JADE iterations per cycle can be slightly different for some subcomponent due to the need of better balancing the computational load between threads. We used a decomposition in subcomponents of size $d_k = 4$ with 25 individuals per subcomponent (e.g. the 1000-dimensional problem was tackled with a decomposition in 250 subcomponents, each with 25 vectors evolved though JADE). The minimum value n_r of the patterns needed for training the RBFN was set to the size of the population. Each training phase was carried out with 30 iterations of gradient descent with learning rate of 0.1.

In [21] the suggested number of exact function evaluations *maxFE* was set to $5000 \cdot d$. However, since we are interested in improving the convergence rate for computationally expensive objective functions, in this study the maximum number of fitness evaluations was set to the relatively low value of $1000 \cdot d$.

For each function, we collected some relevant statistics from the results of 50 independent runs. To improve the significance of the comparisons (e.g. to exclude the influence of a different initial population), the i-th runs of all algorithms were initialized by the same random seed, while i-th and j-th runs, with $i \neq j$, were initialized with different seeds. The results achieved with the developed SACCDE were compared with those given by the standard CCDE, obtained from the former by avoiding the use of metamodelling in fitness evaluation. When comparing SACCDE with CCDE on the single test functions, we carried out Mann-Whitney-Wilcoxon (MWW) with significance 0.05.

Table 2 shows the statistics on the achieved results for 100, 500 and 1000 variables respectively. When the average minimum is significantly lower, according to the MWW test, we have highlighted the results of the corresponding algorithm in the table. As can be seen, for functions f_1, f_2 and f_5-f_7, regardless of the number of variables, the surrogate-assisted approach SACCDE was always able to outperform CCDE. In the case of f_3, SACCDE provided better average results for 100 and 1000 variables and an equivalent average optimum for the 500-dimensional problem. Conversely, in the case of function f_4 the meta-modelling approach always worsened the optimization efficiency of the CCDE algorithm. A closer look at the statistics in Table 2 shows that in most cases the SACCDE algorithm was able to improve the results from one to

Table 2: Statistics of the results achieved on the 50 independent repetitions. For each function, the algorithm with the best average optimum is highlighted and the MWW p-value is shown.

	Alg.	Avg.	Std. Dev.	Best	Worst	p
			$d_k = 100$			
f_1	CCDE	6.8E-006	2.1E-006	2.9E-006	1.0E-005	0.00
	SACCDE	7.7E-013	2.4E-013	4.5E-013	1.0E-005	-
f_2	CCDE	1.2E+002	5.2E+000	1.1E+002	1.3E+002	0.00
	SACCDE	6.9E+001	8.3E+000	5.1E+001	1.3E+002	-
f_3	CCDE	6.8E+004	1.4E+005	3.7E+002	4.3E+005	0.00
	SACCDE	1.5E+004	4.0E+004	1.3E+002	4.3E+005	-
f_4	CCDE	1.4E+001	1.5E+000	1.0E+001	1.5E+001	-
	SACCDE	5.0E+001	1.0E+001	1.0E+001	7.0E+001	0.00
f_5	CCDE	3.5E-003	1.1E-002	1.7E-006	3.5E-002	0.00
	SACCDE	3.9E-005	7.9E-005	2.3E-013	2.0E-004	-
f_6	CCDE	5.4E-003	1.6E-003	2.6E-003	8.4E-003	0.00
	SACCDE	3.6E-010	1.1E-010	1.6E-010	8.4E-003	-
f_7	CCDE	-1.0E+003	7.3E+001	-1.1E+003	-9.5E+002	0.00
	SACCDE	-1.3E+003	6.0E+001	-1.4E+003	-9.5E+002	-
			$d_k = 500$			
f_1	CCDE	7.6E-006	1.3E-006	5.8E-006	9.9E-006	0.00
	SACCDE	6.2E-012	1.6E-012	3.7E-012	9.9E-006	-
f_2	CCDE	1.4E+002	3.3E+000	1.3E+002	1.4E+002	0.00
	SACCDE	1.2E+002	3.1E+000	1.1E+002	1.4E+002	-
f_3	CCDE	1.9E+004	4.2E+004	1.2E+003	1.4E+005	-
	SACCDE	1.6E+004	2.0E+004	8.5E+002	1.4E+005	0.50
f_4	CCDE	5.1E+001	5.8E+000	4.5E+001	6.0E+001	-
	SACCDE	4.0E+002	2.8E+001	4.5E+001	4.6E+002	0.00
f_5	CCDE	8.1E-007	1.2E-007	6.8E-007	1.1E-006	0.00
	SACCDE	2.2E-008	6.6E-008	1.3E-012	2.2E-007	-
f_6	CCDE	2.3E-003	1.7E-004	2.1E-003	2.7E-003	0.00
	SACCDE	8.0E-010	1.8E-010	4.9E-010	2.7E-003	-
f_7	CCDE	-4.9E+003	1.4E+002	-5.1E+003	-4.6E+003	0.00
	SACCDE	-6.0E+003	1.6E+002	-6.2E+003	-4.6E+003	-
			$d_k = 1000$			
f_1	CCDE	5.0E-005	5.1E-006	4.1E-005	5.8E-005	0.00
	SACCDE	1.3E-011	2.8E-012	7.6E-012	5.8E-005	-
f_2	CCDE	1.5E+002	9.8E-001	1.4E+002	1.5E+002	0.00
	SACCDE	1.3E+002	3.8E+000	1.3E+002	1.5E+002	-
f_3	CCDE	1.4E+004	4.0E+003	8.7E+003	2.1E+004	0.00
	SACCDE	3.4E+003	1.8E+003	2.0E+003	2.1E+004	-
f_4	CCDE	1.6E+002	8.8E+000	1.4E+002	1.7E+002	-
	SACCDE	6.2E+002	3.2E+001	1.4E+002	6.5E+002	0.00
f_5	CCDE	3.3E-006	2.8E-007	2.9E-006	3.7E-006	0.00
	SACCDE	3.4E-007	1.0E-006	4.1E-012	3.5E-006	-
f_6	CCDE	1.2E-002	2.1E-003	9.4E-003	1.7E-002	0.00
	SACCDE	9.3E-010	3.4E-010	4.2E-010	1.7E-002	-
f_7	CCDE	-9.3E+003	2.1E+002	-9.7E+003	-8.9E+003	0.00
	SACCDE	-1.2E+004	2.2E+002	-1.3E+004	-8.9E+003	-

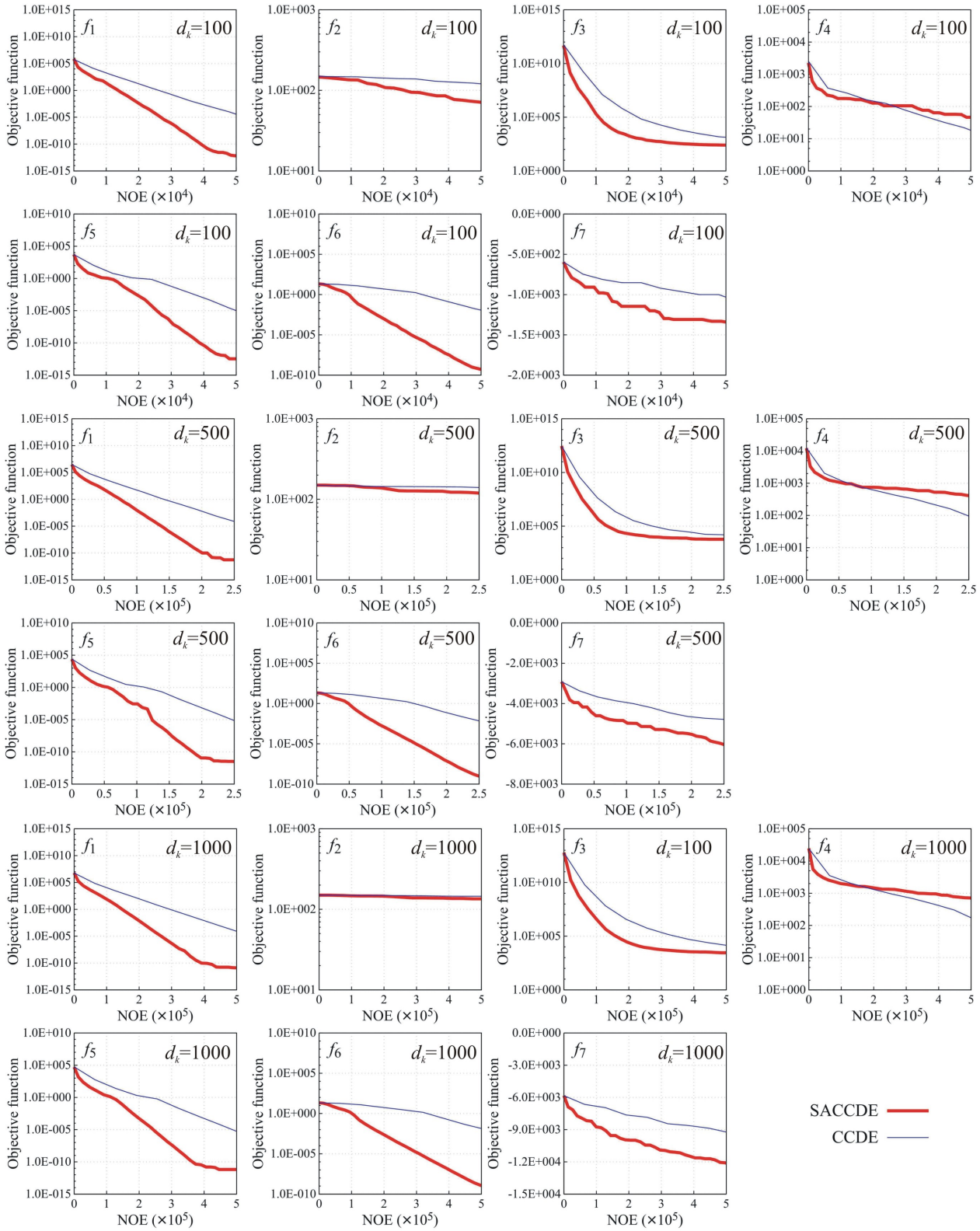

Figure 1: Median convergence plots obtained trough the different algorithms under comparison.

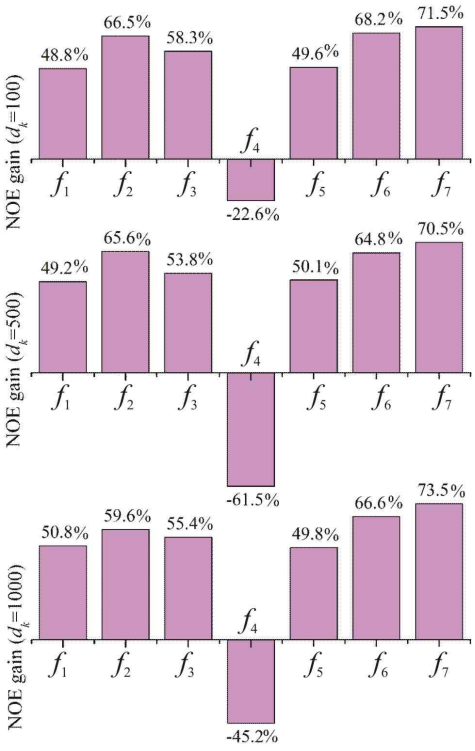

Figure 2: Gain of exact fitness evaluations provided by the SACCDE algorithm.

several orders of magnitude. This was the case, for example, of functions f_1, f_3 (1000 variables), f_5 and f_6. Instead, for f_2 and f_3 (100 variables) we observed only a small improvement, likely irrelevant for most practical purposes. The beneficial effect of meta-modelling in the case of f_7 was particularly evident in higher-dimensional problems, reaching about one order of magnitude in the case of 1000 variables. The loss of optimization efficiency due to the use of surrogate fitness in case of function f_4 was relatively small in the cases of 100 and 1000 variables and more pronounced for the 500-dimensional problem. In general, it seems that for problems that can be efficiently optimized using the CCDE algorithm, the use of surrogates within subcomponents could significantly improve the results (e.g., f_1, f_5 and f_6). In contrast, when the CCDE algorithm exhibited a poor optimization ability (e.g., f_2, f_4), the SACCDE variant could not improve significantly the results or even worsened the achieved optimum, as it was the case of f_4.

Some more insights can be gained from Figure 1, in which we show the median convergence plots. Interestingly, the main characteristics of the patterns exhibited by the different curves tend to persist regardless of the problem dimension. An exception is represented by the function f_2, for which the speed of convergence of SACCDE is significantly improved in the case of 100 variables and essentially unaltered for the problems with higher dimensionality. For f_1 and f_5 we observe a superior efficiency of the SACCDE optimization algorithm over almost the entire budget of exact function evaluations. However, especially for 1000 variables, the SACCDE convergence process entered in a stalling state in the final part of the process. Nevertheless, the achieved optimum had been already below $1.0E-10$, which can be considered suitable for most practical applications. Also in the case of f_3 the slope of the SACCDE was significantly superior to that of CCDE. However, such an improved efficiency was limited to almost half of the process and the estimated optima provided by SACCDE were only slightly better than the final best fitness of CCDE.

For function f_4, after an initial faster convergence, the SACCDE algorithm became less efficient than the simple CCDE, which led to a better result at the end of the process. The reason of such a result for function f_4 could be related to the particular fitness landscape originated by the Rastrigin's function and is consistent with previous studies in surrogate-assisted optimization [14]. In fact, function f_4 is multimodal and notoriously characterized by a huge number of local optima, which can easily lead to early convergence at false global optima of the surrogate model. It is worth noting that in [14] and in [23], the Rastrigin's function was effectively tackled using an approach based on surrogate-assisted local search rather than on a global metamodeling.

For both f_6 and f_7, as well as for f_2 with 100 variables, SACCDE provided a constantly superior speed of convergence and, likely, more fitness evaluations would have led to a greater superiority of result compared to CCDE.

In Figure 2 we show the average gain of exact fitness evaluations provided by the SACCDE algorithm. The percentages are computed as $100 \cdot (maxFE - m)/maxFE$, where $maxFE$ is the adopted budget of exact fitness evaluations and m is the number of exact fitness evaluations needed to the SACCDE algorithm for achieving the same result provided by CCDE. For example, in the case of the 1000-dimensional problem f_6 the gain was 66.6%, which means that SACCDE provided the final result of $1.2E-02$ attained by CCDE with a gain of 333000 exact fitness evaluations. Overall, the advantages provided by the fitness approximation within subcomponents can be relevant. Clearly, in Figure 2 the meaningfulness of the gains of fitness evaluations shown is greater when the CCDE algorithm proved effective in the optimization task.

4.2 Parallel efficiency

The parallel implementation was developed in C++ and OpenMP, which is an application programming interface for supporting the easy development shared-memory multiprocessing applications [11]. The program was compiled with Intel Compiler v. 16.0 and run under Linux on a workstation equipped with two Intel Xeon X5660 (2.80 GHz) 6-Core CPUs (12 cores in total). The results described in the following refer to the average run time obtained in ten repetitions of optimizations for the 1000-dimensional problems using $1.0E06$ exact fitness evaluations. It is worth noting that the used CPUs are endowed with Hyper-threading Intel's proprietary technology that associates two logical cores to each processor core physically present, by taking advantage of the superscalar architecture that allows multiple CPU instructions operating on separate data in parallel. For this reason, we studied the elapsed time of each optimization up to 24 threads.

In Figure 3 we plot, as a function of the number of threads, both the elapsed time (averaged on all test functions) and the achieved

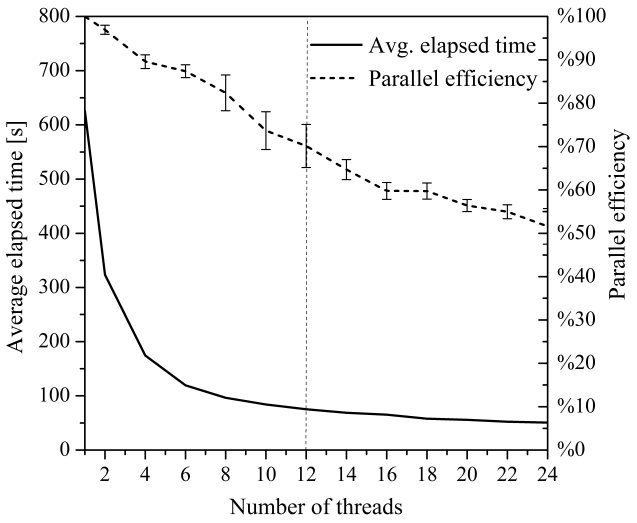

Figure 3: Average elapsed times and parallel efficiency as a function of the number of threads.

parallel efficiency (i.e. the ratio between single-thread and multi-thread execution divided by the number of threads). A single-thread optimization took 626.1 s on average, which could be reduced to the minimum value of 50.6 s using 24 threads. Note, however, that starting from about 12 threads the gain of computing time provided by the activation of more threads was modest and obviously due to the hyper-threading technology. Indeed, we observed an acceptable decline of parallel efficiency as the number of the involved threads increased. In particular, up to the number of available cores (i.e. 12), we achieved a satisfactory average value of efficiency always above 70%. For example, using 8 and 10 threads the values of parallel efficiency were 82% and 74% respectively.

5 CONCLUSIONS AND FUTURE WORK

The proposed approach combines meta-modeling and parallel computing for tackling computationally expensive LSGO problems. According to the results discussed in the present paper, the developed SACCDE algorithm could provide the same result of a standard CCDE optimizer with a gain of expensive fitness evaluations ranging from 50.8% to 73.5% for the 1000-dimensional problems. This, in case of objective functions involving significant computation, can represent a relevant economy of run time. Moreover, on a fairly standard workstation endowed with 12 CPU cores we achieved a speedup of about 12.4 over the single-threaded computation. Nevertheless, we discussed in the present study only a preliminary investigation, which should be enriched and refined with future work, especially by using a larger set of test functions and testing the scalability on more powerful parallel computers.

REFERENCES

[1] Ivan Blecic, Arnaldo Cecchini, and Giuseppe A. Trunfio. 2015. How much past to see the future: a computational study in calibrating urban cellular automata. *International Journal of Geographical Information Science* 29, 3 (2015), 349–374.

[2] Martin D. Buhmann. 2003. *Radial Basis Functions: Theory and Implementations.* Cambridge University Press.

[3] Tianyou Chai, Yaochu Jin, and Bernhard Sendhoff. 2013. Evolutionary complex engineering optimization: opportunities and challenges. *IEEE Computational Intelligence Magazine* 8, 3 (2013), 12–15.

[4] Shi Cheng, Yuhui Shi, Quande Qin, and Ruibin Bai. 2013. Swarm Intelligence in Big Data Analytics. In *Intelligent Data Engineering and Automated Learning - IDEAL 2013.* LNCS, Vol. 8206. Springer Berlin Heidelberg, 417–426.

[5] C.K. Goh, D. Lim, L. Ma, Y.S. Ong, and P.S. Dutta. 2011. A surrogate-assisted memetic co-evolutionary algorithm for expensive constrained optimization problems. In *Evolutionary Computation (CEC), 2011 IEEE Congress on.* 744–749.

[6] A. Hameed, D. Corne, D. Morgan, and A. Waldock. 2013. Large-scale optimization: Are co-operative co-evolution and fitness inheritance additive?. In *2013 13th UK Workshop on Computational Intelligence (UKCI).* 104–111.

[7] A. Hameed, A. Kononova, and D. Corne. 2015. Engineering Fitness Inheritance and Co-operative Evolution Into State-of-the-Art Optimizers. In *2015 IEEE Symposium Series on Computational Intelligence.* 1695–1702.

[8] Yaochu Jin. 2005. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Comput.* 9, 1 (2005), 3–12.

[9] Sedigheh Mahdavi, Mohammad Ebrahim Shiri, and Shahryar Rahnamayan. 2015. Metaheuristics in large-scale global continues optimization: A survey. *Inform. Sci.* 295, 0 (2015), 407 – 428.

[10] Prasanth B Nair and Andrew J Keane. 2001. Passive vibration suppression of flexible space structures via optimal geometric redesign. *AIAA journal* 39, 7 (2001), 1338–1346.

[11] Marco Oliverio, William Spataro, Donato D'Ambrosio, Rocco Rongo, Giuseppe Spingola, and Giuseppe A. Trunfio. 2011. OpenMP parallelization of the SCIARA Cellular Automata lava flow model: performance analysis on shared-memory computers. *Procedia Computer Science* 4 (2011), 271–280.

[12] Mohammad Nabi Omidvar, Xiaodong Li, Zhenyu Yang, and Xin Yao. 2010. Cooperative Co-evolution for large scale optimization through more frequent random grouping. In *IEEE Congress on Evolutionary Computation.* IEEE, 1–8.

[13] YewSoon Ong, A.J. Keane, and P.B. Nair. 2002. Surrogate-assisted coevolutionary search. In *Neural Information Processing, 2002. ICONIP '02. Proceedings of the 9th International Conference on*, Vol. 3. 1140–1145.

[14] Y. S. Ong, A.J. Keane, and P.B. Nair. 2002. Surrogate-Assisted Coevolutionary Search. In *9th International Conference on Neural Information Processing, Special Session on Trends in Global Optimization.* Singapore, 2195–2199.

[15] J. Park and I. W. Sandberg. 1991. Universal approximation using radial-basis-function networks. *Neural Comput.* 3, 2 (June 1991), 246–257.

[16] Konstantinos E. Parsopoulos. 2012. Parallel cooperative micro-particle swarm optimization: A master-slave model. *Appl. Soft Comput.* 12, 11 (2012), 3552–3579.

[17] Mitchell A. Potter and Kenneth A. De Jong. 1994. A Cooperative Coevolutionary Approach to Function Optimization. In *Parallel Problem Solving from Nature - PPSN III (LNCS)*, Vol. 866. Springer-Verlag, 249–257.

[18] Friedhelm Schwenker, Hans A. Kestler, and Gnther Palm. 2001. Three learning phases for radial-basis-function networks. *Neural Networks* 14, 4-5 (2001), 439 – 458.

[19] Robert E. Smith, B. A. Dike, and S. A. Stegmann. 1995. Fitness Inheritance in Genetic Algorithms. In *Proceedings of the 1995 ACM Symposium on Applied Computing (SAC '95).* ACM, New York, NY, USA, 345–350.

[20] Rainer Storn and Kenneth Price. 1997. Differential Evolution - A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization* 11, 4 (1997), 341–359.

[21] K. Tang, X. Yao, P. Suganthan, C. MacNish, Y. Chen, C. Chen, and Z. Yang. *Benchmark functions for the CEC' 2008 special session and competition on large scale global optimization.* Technical Report. NICAL, Department of Computer Science and Technology, University of Science and Technology of China, Hefei.

[22] SpencerAngus Thomas and Yaochu Jin. 2014. Reconstructing biological gene regulatory networks: where optimization meets big data. *Evolutionary Intelligence* 7, 1 (2014), 29–47.

[23] Giuseppe A. Trunfio. 2016. *Enhancing Cooperative Coevolution with Surrogate-Assisted Local Search.* Springer International Publishing, 63–90.

[24] Giuseppe A. Trunfio, Pawel Topa, and Jaroslaw Was. 2016. A new algorithm for adapting the configuration of subcomponents in large-scale optimization with cooperative coevolution. *Inf. Sci.* 372 (2016), 773–795.

[25] R. Paul Wiegand, William C. Liles, and Kenneth A. De Jong. 2002. *The Effects of Representational Bias on Collaboration Methods in Cooperative Coevolution.* Springer Berlin Heidelberg, Berlin, Heidelberg, 257–268.

[26] K. Won, T. Ray, and K. Tai. 2003. A framework for optimization using approximate functions. In *Proceedings of IEEE Congress on Evolutionary Computation.* 1077–1084.

[27] Zhenyu Yang, Ke Tang, and Xin Yao. 2008. Large scale evolutionary optimization using cooperative coevolution. *Inform. Sci.* 178, 15 (2008), 2985–2999.

[28] Zhenyu Yang, Ke Tang, and Xin Yao. 2008. Multilevel cooperative coevolution for large scale optimization. In *IEEE Congress on Evolutionary Computation* (2009-02-27). IEEE, 1663–1670.

[29] Jingqiao Zhang and A.C. Sanderson. 2009. JADE: Adaptive Differential Evolution With Optional External Archive. *IEEE Trans. Evol. Comput.* 13, 5 (Oct 2009), 945–958.