Applied Soft Computing 27 (2015) 490–503

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Performance evaluation of automatically tuned continuous optimizers on different benchmark sets

Tianjun Liao [a,*], Daniel Molina [b], Thomas Stützle [c]

[a] State Key Laboratory of Complex System Simulation, Beijing Institute of System Engineering, Beijing, China
[b] Dept. of Computer Engineering, University of Cádiz, Cádiz, Spain
[c] IRIDIA, CoDE, Université Libre de Bruxelles (ULB), Brussels, Belgium

ARTICLE INFO

Article history:
Received 29 January 2014
Received in revised form 3 October 2014
Accepted 5 November 2014
Available online 22 November 2014

Keywords:
Continuous optimization
Benchmark sets
Evolutionary computation
Swarm intelligence
Automatic parameter tuning

ABSTRACT

The development of algorithms for tackling continuous optimization problems has been one of the most active research topics in soft computing in the last decades. It led to many high performing algorithms from areas such as evolutionary computation or swarm intelligence. These developments have been sidelined by an increasing effort of benchmarking algorithms using various benchmarking sets proposed by different researchers.

In this article, we explore the interaction between benchmark sets, algorithm tuning, and algorithm performance. To do so, we compare the performance of seven proven high-performing continuous optimizers on two different benchmark sets: the functions of the special session on real-parameter optimization from the IEEE 2005 Congress on Evolutionary Computation and the functions used for a recent special issue of the Soft Computing journal on large-scale optimization. While one conclusion of our experiments is that automatic algorithm tuning improves the performance of the tested continuous optimizers, our main conclusion is that the choice of the benchmark set has a much larger impact on the ranking of the compared optimizers. This latter conclusion is true whether one uses default or tuned parameter settings.

1. Introduction

Continuous optimization problems arise in many important design and control problems in areas such as engineering, telecommunications, or computer science. In many cases, the resulting problems have to be treated as black-box problems, where no explicit mathematical formulation of the problem is available and, therefore, derivatives may not be computed [9]. Often, such problems have multiple optima, correlated variables and non-linear objective functions, making their solution very hard. Thus, as an alternative to classical continuous optimization techniques, direct-search (or also called derivative-free) methods have emerged [9].

An important source of direct-search methods is the area of soft computing. Many techniques such as evolutionary computation [4], memetic algorithms [44], ant colony optimization [12], particle swarm optimization [27], differential evolution [52] or artificial bee colonies [25] have been at the core of recent advancements when developing optimization techniques. All these techniques have been applied to tackle (black-box) continuous optimization problems. Such algorithms have been developed starting more than four decades ago and thousands of articles have been published providing new algorithmic ideas or algorithm designs and analyses of such algorithms.

This proliferation of algorithmic techniques and variants thereof, has made it difficult to identify which algorithms contribute to define the state-of-the-art. To address some shortcomings in the evaluation of (nature-inspired) continuous optimizers, several benchmarking efforts have been undertaken. Early benchmarking efforts such as the first international contest on evolutionary optimization [5] in 1996 had some but little impact. Later efforts, however, have been followed much more strongly, possibly because the number of available algorithms has grown very much. A noteworthy example is the special session on real parameter optimization of the 2005 IEEE Congress on Evolutionary Computation (CEC'05) [53]. It proposed a set of 25 benchmark problems and identified the covariance matrix adaptation evolution strategy (CMA-ES) [16] with occasional restarts with increasing population size (IPOP-CMA-ES) [1] as the best performing algorithm on this CEC'05 benchmark set. The impact this special session

* Corresponding author. Tel.: +86 18302330650.
E-mail addresses: liaotianjun@gmail.com (T. Liao), daniel.molina@uca.es (D. Molina), stuetzle@ulb.ac.be (T. Stützle).

http://dx.doi.org/10.1016/j.asoc.2014.11.006
1568-4946/

had and still has is illustrated by the more than 750 citations the technical report defining the benchmark functions received in google scholar by December 2013. Other efforts include a special session on large scale global optimization held at the IEEE CEC'08 conference. The benchmark set of this special session was later extended in various successor competitions. In particular, the special issue on large scale continuous optimization problems of the Soft Computing journal [40] extended this benchmark set to comprise a total of 19 freely scalable benchmark functions; we refer to this benchmark set in the following as the SOCO benchmark set. In this special issue, a memetic algorithm that combines a local search algorithm with a differential evolution algorithm in the multiple offspring (MOS) framework [30] was the best-performing algorithm; we refer to this algorithm as MOS in what follows.

If one takes a closer look into the results on these two benchmark sets, an interesting difference arises. In fact, on the SOCO benchmark set, IPOP-CMA-ES, the winner of the CEC'05 benchmarking competition, was used as an algorithm to set a performance baseline and the final winner, MOS, performed significantly better than IPOP-CMA-ES. This result reminds of an observation made by Whitley et al. [56], p. 245:

> "*Test functions are commonly used to evaluate the effectiveness of different search algorithms. However, the results of evaluation are as dependent on the test problems as they are on the algorithms that are the subject of comparison.*"

Paraphrasing this statement in terms of the current benchmark sets, it would say that the choice of the benchmark set has a strong impact on the final ranking of the algorithms tested. Unfortunately, MOS was not tested on the CEC'05 benchmark set and it is therefore unclear whether it would perform better than IPOP-CMA-ES also on the CEC'05 benchmark set.

One goal of this article is to examine the impact the choice of the benchmark set has on the relative performance of current high performance continuous optimizers. We do so by choosing a number of algorithms that are derived from different search techniques and that have shown very good performance on at least one of the CEC'05 and SOCO benchmark sets. In particular, we include the already mentioned IPOP-CMA-ES [1] and MOS [30] algorithms as winners of the two benchmark competitions. In addition, we selected the following algorithms. MA-LSCh-CMA [42] is a memetic algorithm for continuous optimization based on local search chains; it was reported very high performance on the CEC'05 benchmark set. SaDE [48] is a self-adaptive differential evolution algorithm, which received the 2012 IEEE Transactions on Evolutionary Computation Outstanding Paper Award; IPSOLS [43] and IABCLS [3] are particle swarm optimization and artificial bee colony algorithms, which obtained the best performance in their respective fields on the SOCO benchmark set; UACOR [37] is a configurable framework of ant colony optimization for continuous domains that shows improved performance when compared to $IACO_R$ [33], which was the so far best performing ant colony optimization algorithm for continuous optimization on the above mentioned benchmark sets. Although some researchers have compared continuous optimizers from different metaheuristics [2,18,14,49], the aforementioned start-of-the-art continuous optimizers have not been evaluated together in the literature.

A second goal of this article is to examine the impact specific choices on the parameter settings have on the evaluation of continuous optimizers. Typically, parameter settings are defined by the algorithm designers based on preliminary experiments and considering a specific set of test problems. This may lead to implicit biases in the parameter tuning toward specific test functions or benchmark sets and an uneven effort in parameter tuning. Biases in algorithm tuning and uneven tuning effort are two problematic issues when comparing algorithms as one algorithm may seem superior to another one only because its parameters are more fine-tuned for a specific benchmark set. As an alternative, one may rely on automatically tuned parameter settings through the usage of automatic algorithm configuration techniques [7,23,24,39,46]. In fact, experience with these techniques has shown that often they are able to find improved parameter settings over manually tuned default settings and to make parameter tuning more efficient [7,23,22,24]. Therefore, here we compare the continuous optimizers using their default settings and based on parameters that are tuned by using an automatic algorithm configuration tool, irace [7,39], to give an unbiased comparison.

In a nutshell, the experimental results that we obtain indicate that (i) all algorithms may profit from the automatic algorithm configuration, (ii) the benchmark sets have a major impact on the ranking of the algorithms, and (iii) some algorithms exhibit poor scaling behavior with increasing dimension. We believe that these results also show that future benchmarking efforts need to consider larger and more systematically varied sets of benchmark problems and, in particular, the evaluation of the continuous optimizers on more benchmark problems stemming from real applications is desirable to steer future research efforts.

The article is organized as follows. Section 2 describes the seven continuous optimizers we test. Section 3 gives more background on automatic algorithm configuration and tuning. Next, Section 4 describes the benchmark sets and Section 5 the experimental setups. In Sections 6–8, we evaluate the optimizers using default and tuned parameters on the CEC'05 and SOCO benchmark sets. We end the article with a discussion of the obtained results and conclude in Section 9.

2. The tested continuous optimizers

In this section, we concisely introduce the seven continuous optimizers that we compare and justify their choice. Since these optimizers are well described in the literature, we refer to the original articles for their detailed description to keep the present article short.[1]

2.1. IPOP-CMA-ES

CMA-ES [16] is a (μ, λ)-evolution strategy that samples at each iteration new solutions based on a multivariate normal distribution. It adapts the covariance matrix of a normal search distribution during the execution to lead the sampling to the most promising part of the search space. Its design makes the algorithm invariant to the scaling of the evaluation function and to rotations of the variable's correlation. CMA-ES may be seen as a stochastic local search technique; IPOP-CMA-ES [1] provides an iterative restart scheme once CMA-ES is deemed to stagnate and at each restart increases the population size. It obtained the best performance among all the candidates in the special session on real parameter optimization of CEC'05 [53] and is since then seen as a state-of-the-art continuous optimizer.

Following [36], we consider seven parameters (a, b, c, d, e, f, g) for tuning. These parameter control some of the formulas that determine IPOP-CMA-ES's search behavior. In particular, these parameters are used in the following equations. The initial population size is $\lambda = 4 + \lfloor a \ln(D) \rfloor$, where D is the number of dimensions of the function to be optimized. The number of parent solutions is $\mu = \lfloor \lambda / b \rfloor$. The initial step-size is $\sigma^{(0)} = c(B - A)$, where $[A, B]^D$ is the initial search interval. The population size is multiplied by a

[1] The list of the parameters, their default and tuned values and their domains considered for tuning are given in Tables 2 and 3.

factor $ipop = d$ at each restart. Restarts occur if the stopping criterion is met, which uses the parameters $stopTolFunHist = 10^e$ (lower threshold on range of the improvement in the last generations), $stopTolFun = 10^f$ (lower threshold on all objective function values of the recent generation) and $stopTolX = 10^g$ (lower threshold on the standard deviation of the normal distribution).

2.2. MA-LSCh-CMA

Memetic algorithms (MAs) [44,28] often combine evolutionary algorithms with local search procedures to improve individuals. MA-LSCh-CMA [42] is a memetic algorithm for continuous optimization that combines (i) a real-coded steady state genetic algorithm (SSGA) [20] that was designed to promote high population diversity and (ii) CMA-ES [16] that was used as a local search procedure. MA-LSCh-CMA was reported to improve over IPOP-CMA-ES on the multi-model functions of CEC'05 benchmark set [42].

We consider eight parameters for tuning MA-LSCh-CMA. They are the population size p, the probability of mutating a chromosome *pmut*, the number of evaluations in each local search by CMA-ES *ils*, the ratio of the number of function evaluations allocated to SSGA and CMA-ES, r, a low threshold for the fitness improvement obtained by CMA-ES (10^{th}, where th is a parameter), and three parameters (a, b, c) that are the same as in CMA-ES (see Section 2.1).

2.3. SaDE

Different evolution (DE) [52] is an algorithm for continuous optimization, which includes three important parameters: population size p, scaling factor f and crossover rate cr. SaDE [48] uses a self-adaptive mechanism [13], that is, it uses feedback from the search process to update trial vector generation strategies and the associated parameters. This algorithm, for which very high performance was reported on the 10-dimensional functions of the CEC'05 benchmark set [48], won also the IEEE Transactions on Evolutionary Computation Outstanding Paper Award, illustrating its prominent role in DE.

Although SaDE adapts the parameters f and cr, the new parameters used to actually adapt f and cr are candidates for being fine-tuned. In particular, we consider five parameters for tuning SaDE. These are the population size p, the period of learning from previous experiences *lp*, the mean value (*fm*) and standard deviation (*fsd*) of the normal distribution to sample the value of f, and the standard deviation (*crsd*) of the normal distribution to sample the crossover rate cr.

2.4. MOS

MOS [29] is a multiple offspring sampling framework for dynamically hybridizing evolutionary algorithms. The framework dynamically evaluates the offspring techniques involved and adjusts their participation. A MOS-based memetic DE algorithm for continuous optimization (MOS) is proposed in [30]. In this hybrid, DE and a local search technique are executed in sequence by means of a high-level relay hybrid (HRH) [54]. The MOS algorithm obtained the best performance among the 13 algorithms published in the special issue related to the evaluation of algorithms on the SOCO benchmark set.

We consider five parameters for tuning MOS. They are the population size p, the scaling factor of DE, f, the crossover rate of DE, cr, a minimum participation ratio in the HRH approach *mpr* and the number of function evaluations for each call to the local search procedure, *fesls*.

2.5. IPSOLS

In a particle swarm optimization (PSO) algorithm, particles' velocities and positions are updated at each iteration to search promising solutions to the optimization problem at hand [26,27]. Over the recent years, many modifications to the original PSO has been proposed. IPSOLS [43] is a recent PSO algorithm that uses a time-varying, growing population size and combines the PSO algorithm with Powell's conjugate direction set method [47]. IPSOLS was tested on the SOCO benchmark set and in the competition it was the overall best performing PSO algorithm. Hence, we have selected it for the present paper.

The parameters that we have chosen for tuning are the same as described in the original paper [43] and we refer for further details to that paper. In summary, the parameter refer to the topology used (*Top*); *iw*, ϕ_1, and ϕ_2 are the inertia weight and the acceleration coefficients (personal and social), respectively; *initp* is the initial population size; *ftol* and *lsItr* are parameters for the stopping criterion of the local search related to a threshold of the improvement between two local search iterations and a maximum number of local search iterations; *lsF* and *stagItr* refer to a limit on the number of times local search can start from a same particle position and the number of iterations before restarting the algorithm.

2.6. IABCLS

The artificial bee colony (ABC) algorithm [25] was inspired by the foraging behavior of a honeybee swarm around their hive. The ABC algorithm involves three phases of foraging behavior which are conducted by the employed bees, onlookers and scouts, respectively. ABC algorithms essentially perform a local search that explores in parallel a number of solutions which is equal to the number of bees in the colony. IABCLS [3] is an ABC algorithm which uses an increasing colony size and an additional local search procedure. It is the first ABC algorithm that was evaluated on the SOCO benchmark set and it performed competitive to the algorithms published in SOCO. A recent comparison of various ABC algorithm variants confirmed IABCLS as a top performing ABC algorithm [35].

The parameters for tuning of IABCLS are *initp*, the initial population size; every g iterations one new individual is added to the population until a maximum population size *Fmax* is reached. If the population is increased, the new search point to be explored is biased by a factor 10^{rf}, where *rf* is a parameter, from a random location toward the location of the best solution found so far. If in the vicinity of a current solution for *limit* iterations no improved solution is found, it is replaced with a new random solution. The parameters *lsF*, *ftol*, and *lsItr* are analogous that what has been explained for IPSOLS. However, in the case of IABCLS the Mtsls1 [55] local search method is used.

2.7. UACOR

Ant colony optimization (ACO) was originally proposed for solving discrete optimization problems [11]. Recently, the adaption of ACO algorithms to continuous domains receives increasing attention [51,31,33]. These ACO algorithms generate a solution by sampling dimension by dimension variable values according to some probability distribution that is determined by an archive of high-quality solutions. UACOR [37,32] is a unified, configurable ant colony framework for continuous optimization that allows automatic algorithm configuration methods to instantiate the optimizer. In particular, the incremental ACO algorithm [33], which has shown to outperform earlier available ACO algorithms for continuous optimization, can be instantiated using a specific parameter setting from the UACOR framework. Other

high-performing instantiations of the UACOR framework have been generated and examined in [37].

The various parameters of UACOR refer to the following algorithmic components. *DefaultMode* refers to the type of solution construction (around any solution in the archive or only the best one—only in the latter case the parameter $EliteQ_{best}$ is relevant); parameters *initAS*, *IsIncrement*, and *GrowthIter* refer to population size and incremental colony size, respectively; parameters *Na* and *NaIsAS* refer to the choice of the number of ants in the solution construction and around which solutions are new one is constructed; Q_{best}, *WeightGsol*, q, and ξ are parameters defining the probability distributions in the solution construction; parameters *RmLocalWorse* and *SnewvsGsol* are used to bias the update of the solution archive in each iteration; parameters *LsType*, *LsIter*, and *LsFailures* define which local search is used (none, Powell, or Mtsls1) and are parameters for the stopping criterion of the local search; finally, parameters *RestartType*, *StagIter*, *StagThresh*, *Shakefactor*, and *RestartAS* refer to a possible partial algorithm restart if algorithm stagnation is detected.[2]

3. Automatic parameter tuning

In the literature, algorithm parameters are usually set based on the experience and preliminary experiments executed by an algorithm's designer(s). Sometimes, a "manual" tuning procedure, for example, through experimental design type analyses such as ANOVA or some other manual tuning approach is applied. For several of the algorithms we examine here, this was also the methodology followed by the original authors. In particular, the parameter values of IPOP-CMA-ES, MA-LSCh-CMA and SaDE were set by experience, while parameter values of MOS were derived from a manual tuning procedure. Differently, for the algorithms that have been developed by co-authors of the present paper, namely for IABCLS, IPSOLS and UACOR, a parameter tuning methodology based on the usage of automatic algorithm configuration techniques was employed. In fact, automatic algorithm configuration may examine the algorithm design space more carefully than what is possible by manual tuning and ultimately lead to a shift of paradigm in the design of heuristic algorithms [22].

Automatic algorithm configuration is also relevant in the context of algorithm comparisons due to a number of reasons. First, one may give an even tuning effort to all algorithms compared and avoid that the superiority of one algorithm over another is due to the fact that it has been tuned much more intensively than the other. Second, possible biases through the algorithm designers are avoided as the automatic tuning is done by a computer. Finally, the automatic algorithm configuration process often results in parameter settings that improve over the algorithm defaults [7,23]. Nevertheless, the choice of the training set of instances for the automatic configuration may play a crucial role. In the experimental section, we detail how we have taken care about these possible issues.

In this article, we employ Iterated F-Race [7], an automatic algorithm configuration method that is provided by the irace package [39]. F-race is a racing method that is used for selecting the best algorithm configuration from a set of given candidate algorithm configurations. F-race starts with an initial candidate set of algorithm configurations. All (surviving) candidate configurations are tested iteratively on a stream of problem instances. Once all algorithm configurations algorithms have seen one particular problem instance (this is one step of F-race), the algorithm configurations that perform statistically significantly worse than others are eliminated from the race; then, the next step of F-race is done, which consists in testing all surviving candidate configurations on the next problem instance. The statistical test applied to eliminate candidate algorithms is the F-test with its associated post hoc test [10]. Iterated F-race repeatedly generates sets of candidate configurations biased by the best candidate configurations of the previous iteration and applies F-race to determine the best overall algorithm configuration. For details on the iterated F-race procedure we refer to [7] and the publicly available irace package [39].

Note that in the context of tuning continuous optimizers, the fact that the statistical test in F-race relies on ranking the results of the algorithm configurations gives specific advantages as it eliminates the possibly high differences of the ranges of the objective function values across the various benchmark functions. Without ranking, few functions that show large objective function values would dominate the evaluation of algorithm performance.

4. Benchmark sets

We evaluate the seven continuous optimizers on the CEC'05 and the SOCO benchmark set. The two benchmark sets contain 25 and 19 functions, respectively. Common characteristics of both benchmark sets are that (i) they include few standard benchmark functions from the continuous optimization literature such as Sphere, Rosenbrock, Rastrigin, or Griewank; (ii) they randomly shift functions to avoid that algorithms exploit implicitly (or explicitly) the fact that for many benchmark functions optimal solutions are at the center of the search space; (iii) the functions in both benchmark sets are typically bound constrained; the only exception are two functions from the CEC'05 benchmark set (functions f_{cec7} and f_{cec25}), for which only an initialization range is given; (iv) they both contain hybrid functions that are composed of several, simpler functions. However, the hybrid functions of the CEC'05 benchmark appear to be harder that those of the SOCO benchmark set. A significant difference between the two benchmark sets is that 16 of the 25 CEC'05 functions are artificially rotated to induce stronger dependencies between (all) variables. This is one aspect that potentially makes the CEC'05 benchmark functions more difficult to solve than functions from the SOCO benchmark set. While one effect of rotation is to avoid that functions are separable, it should be remarked that also in the SOCO benchmark set 15 out of 19 functions are not separable, that is, they cannot be optimized by considering each dimension independently of the others. The properties of the CEC'05 and SOCO benchmark functions are listed in Table 1; a detailed description of each of the functions is given in the original description of the CEC'05 and SOCO benchmark functions [53,21].

The functions of both benchmark sets are freely scalable; however, in the CEC'05 benchmark set only rotation matrices for the functions of dimension 10, 30, and 50 are given so that these are also the dimensions that are usually used when evaluating algorithms on this benchmark set. Here, we follow this standard. The functions of the SOCO benchmark set were proposed for evaluation in dimensions up to 1000. However, as our focus here is not on examining explicitly the scaling behavior of algorithms, in this paper we restrict the evaluation to problems of dimension 10, 50, and 100. For each of the benchmark sets, we apply the termination conditions described in [53,21] for the CEC'05 and SOCO benchmark sets: the maximum number of function evaluations given as $10\,000 \times D$ for the CEC'05 functions, and $5000 \times D$ evaluations for the SOCO functions, respectively, where D is the dimensionality of a function. The error values of the objective functions at the termination criterion are recorded; the error value is $f(\boldsymbol{x}) - f(\boldsymbol{x}^*)$, where $\boldsymbol{x}$ is a candidate solution and $\boldsymbol{x}^*$ is the optimal solution (optimal solutions are known by the construction of the benchmark functions). Following the CEC'05 and SOCO benchmark definition, error values lower than 10^{-8} are approximated to 10^{-8} (10^{-8} is referred

[2] A detailed explanation of the UACOR framework would take several journal pages; we refer to [37], pp. 3–13, for the detailed description.

Table 1
Description of the SOCO and CEC'05 benchmark function suite. Given is the identifier of the function, the name, the range for the feasible range of variable values (an exception are functions f_{cec7} and f_{cec25}, where initialization ranges are specified), an indication whether the functions are uni- (U) or multi- (M) modal, whether they are separable (Y) or not (N), and whether they are artificially rotated (Y) or not (N).

ID	Name/description	Range $[X_{min}, X_{max}]^D$	Uni/Multimodal	Separable	Rotated
f_{soco1}	Shift.Sphere	$[-100,100]^D$	U	Y	N
f_{soco2}	Shift.Schwefel 2.21	$[-100,100]^D$	U	N	N
f_{soco3}	Shift.Rosenbrock	$[-100,100]^D$	M	N	N
f_{soco4}	Shift.Rastrigin	$[-5,5]^D$	M	Y	N
f_{soco5}	Shift.Griewank	$[-600,600]^D$	M	N	N
f_{soco6}	Shift.Ackley	$[-32,32]^D$	M	Y	N
f_{soco7}	Shift.Schwefel 2.22	$[-10,10]^D$	U	Y	N
f_{soco8}	Shift.Schwefel 1.2	$[-65.536,65.536]^D$	U	N	N
f_{soco9}	Shift.Extended f_{10}	$[-100,100]^D$	U	N	N
f_{soco10}	Shift.Bohachevsky	$[-15,15]^D$	U	N	N
f_{soco11}	Shift.Schaffer	$[-100,100]^D$	U	N	N
f_{soco12}	$f_{soco9} \oplus 0.25 f_{soco1}$	$[-100,100]^D$	M	N	N
f_{soco13}	$f_{soco9} \oplus 0.25 f_{soco3}$	$[-100,100]^D$	M	N	N
f_{soco14}	$f_{soco9} \oplus 0.25 f_{soco4}$	$[-5,5]^D$	M	N	N
f_{soco15}	$f_{soco10} \oplus 0.25 f_{soco7}$	$[-10,10]^D$	M	N	N
f_{soco16}	$f_{soco9} \oplus 0.5 f_{soco1}$	$[-100,100]^D$	M	N	N
f_{soco17}	$f_{soco9} \oplus 0.75 f_{soco3}$	$[-100,100]^D$	M	N	N
f_{soco18}	$f_{soco9} \oplus 0.75 f_{soco4}$	$[-5,5]^D$	M	N	N
f_{soco19}	$f_{soco10} \oplus 0.75 f_{soco7}$	$[-10,10]^D$	M	N	N
f_{cec1}	Shift.Sphere	$[-100,100]^D$	U	Y	N
f_{cec2}	Shift.Schwefel 1.2	$[-100,100]^D$	U	N	N
f_{cec3}	Shift.Ro.Elliptic	$[-100,100]^D$	U	N	Y
f_{cec4}	Shift.Schwefel 1.2 Noise	$[-100,100]^D$	U	N	N
f_{cec5}	Schwefel 2.6 Opt on Bound	$[-100,100]^D$	U	N	N
f_{cec6}	Shift.Rosenbrock	$[-100,100]^D$	M	N	N
f_{cec7}	Shift.Ro.Griewank No Bound	$[0,600]^D$[a]	M	N	Y
f_{cec8}	Shift.Ro.Ackley Opt on Bound	$[-32,32]^D$	M	N	Y
f_{cec9}	Shift.Rastrigin	$[-5,5]^D$	M	Y	N
f_{cec10}	Shift.Ro.Rastrigin	$[-5,5]^D$	M	N	Y
f_{cec11}	Shift.Ro.Weierstrass	$[-0.5,0.5]^D$	M	N	Y
f_{cec12}	Schwefel 2.13	$[-\pi,\pi]^D$	M	N	N
f_{cec13}	Griewank plus Rosenbrock	$[-3,1]^D$	M	N	N
f_{cec14}	Shift.Ro.Exp.Scaffer	$[-100,100]^D$	M	N	Y
f_{cec15}	Hybrid Composition	$[-5,5]^D$	M	N	N
f_{cec16}	Ro. Hybrid Composition	$[-5,5]^D$	M	N	Y
f_{cec17}	Ro. Hybrid Composition	$[-5,5]^D$	M	N	Y
f_{cec18}	Ro. Hybrid Composition	$[-5,5]^D$	M	N	Y
f_{cec19}	Ro. Hybrid Composition	$[-5,5]^D$	M	N	Y
f_{cec20}	Ro. Hybrid Composition	$[-5,5]^D$	M	N	Y
f_{cec21}	Ro. Hybrid Composition	$[-5,5]^D$	M	N	Y
f_{cec22}	Ro. Hybrid Composition	$[-5,5]^D$	M	N	Y
f_{cec23}	Ro. Hybrid Composition	$[-5,5]^D$	M	N	Y
f_{cec24}	Ro. Hybrid Composition	$[-5,5]^D$	M	N	Y
f_{cec25}	Ro. Hybrid Composition	$[2,5]^D$[a]	M	N	Y

[a] Initialization range without bound constraints. Its global optimum is outside of initialization range.

to as optimum threshold) for CEC'05 functions, and error values lower than 10^{-14} are approximated to 10^{-14} (10^{-14} is referred to as optimum threshold) for SOCO functions. The threshold levels can be seen as tolerances below which differences in objective values do not make (any) practical differences in case of real optimization problems; for benchmark problems, in the past they have also be used to avoid numerical issues.

Another significant benchmarking effort has been introduced by the Black-Box Optimization Benchmarking (BBOB) series of workshops that are held regularly at the GECCO conference since 2009 (with the only exception of 2011). This benchmark set comprises 25 functions that are scaled from 2 to 40 dimensions. It is accessible through the COCO (COmparing Continuous Optimisers, see http://coco.gforge.inria.fr/doku.php) platform, which provides extensive tools for the analysis of the tested algorithms. It shares with the CEC'05 benchmark set the fact that many functions are artificially rotated. Also more recently, benchmarking efforts continue. For example, the CEC'05 benchmarking effort was continued at CEC 2013, where 28 benchmark functions (with a large overlap to the CEC'05 benchmark set) have been proposed. We expect that the algorithms behave similarly on the CEC'05 and CEC'13 benchmark sets as they have considerable overlaps and their definition is based on the same principles.

5. Experimental setup

In this paper, we evaluate the optimizers following the experimental protocols of the CEC'05 and SOCO benchmark sets and run them on the dimensions detailed in the previous section. Each optimizer is run 25 times independently on each function. We consider the final mean error values over the 25 runs on each function. Since the range of the mean error values varies strongly from function to function, we rank the optimizers's results on each function (best rank is one and the worst rank is seven, as seven optimizers are compared), and then we compute each optimizer's mean rank over each benchmark. Better and worse performance corresponds to lower and higher mean rank values, respectively. The best, mean, median, and worst error values of each optimizer for each function are available as Supplementary material [34].

For parameter tuning, we follow the automatic tuning procedure used for tuning UACOR [37], and apply the same procedure to the other six continuous optimizers. As UACOR was developed

as a configurable framework, no default parameter settings were defined but the parameterization of UACOR was done directly by an automatic algorithm configuration method. Therefore, in the case of UACOR, the default and the tuned configurations are considered to coincide for what follows.

The tuning procedure we apply is the following. The performance measure for tuning is the objective function error value of each instance. The tuning budget is set to 5000 runs of the continuous optimizer undergoing tuning. The settings of Iterated F-Race are the default ones [39]. To cope with the problem of choosing an appropriate training set, each of the continuous optimizers is tuned twice. The first tuning uses as training set the 25 benchmark functions of 10 dimensions from the CEC'05 benchmark set. These functions are presented in a random order (to avoid biases as the functions tend to be ordered from easy to hard) to Iterated F-race and the number of function evaluations of each optimizer run is equal to $10\,000 \times D$ (where $D = 10$). The second tuning uses as training set the 19 benchmark functions of 10 dimensions from the SOCO benchmark set. The functions are again presented to Iterated F-race in a random order and the number of function evaluations of each run of a candidate configuration is equal to $5000 \times D$ ($D = 10$). The obtained configurations in the first and second tuning are identified by tcec and tsoco, respectively. The configurations are given in Tables 2 and 3.

Note that parameter settings for IPSOLS and IABCLS were obtained in the original papers by an automatic algorithm tuning using the SOCO benchmark functions as training set and that IPSOLS and IABCLS have not been evaluated on the CEC'05 benchmark set so far. The tuning procedure followed in the original papers was different from the one applied here[3] and therefore we re-tune IPSOLS and IABCLS with the tuning setup outlined above; the settings of IPSOLS and IABCLS from the original papers we consider here as "default".

We present the main results of our experiments in three stages. First, we evaluate default and tcec parameter settings on the CEC'05 benchmark set; second, we evaluate default and tsoco parameter settings on the SOCO benchmark; third, we evaluate tcec (tsoco) parameter settings on the SOCO (CEC'05) benchmark set, that is, on the respectively other set.

6. Evaluation on the CEC'05 benchmark set

In this section, we analyze the optimizers' performance using default and tcec parameter settings on the CEC'05 benchmark functions in dimensions 10, 30 and 50.

6.1. Effectiveness of automatic tuning

As a first test, we examine the performance improvement obtained by using tuned parameter settings with respect to default ones. Fig. 1 shows the comparison of the mean rank between default and tuned settings for each optimizer on the CEC'05 benchmark functions of dimensions 10, 30 and 50 (except UACOR, see above). Except for MA-LSCh-CMA on problems of dimension 50, the tuned parameter settings always result in a better ranking, that is, in lower mean error values on a majority of the benchmark functions than the default parameter setting. Note that IABCLS and IPSOLS have previously been tuned on the SOCO benchmark set and not on the CEC'05 benchmark set. Hence, for these two algorithms the results confirm that there are apparently differences between the SOCO and CEC'05 benchmark set and that tuning directly on the CEC'05 functions improves performance, while for the other algorithms the results confirm that often the tuned parameter settings improve over manually tuned settings.

6.2. Performance comparison of optimizers

Fig. 2 compares the mean rank values for the optimizers using default (left plots) and tuned (right plots) parameter settings on the CEC'05 benchmark functions of dimensions 10, 30 and 50 (from top to bottom). If a mean rank value in Fig. 2 is highlighted, this indicates statistically significantly worse performance of the corresponding optimizer than the best performing one according to Holm's multiple test at significance level $\alpha = 0.05$.

Let us focus first on the ranking of the optimizers under default parameter settings. For dimensions 10 and 30, IPOP-CMA-ES is the best ranking optimizer, while on dimension 50 it is MA-LSCh-CMA. MOS, the top performing algorithm on the SOCO benchmark set [40] was never previously tested on the CEC'05 benchmark set. Maybe surprisingly it is always statistically significantly worse than the best algorithm and for dimensions 30 and 50 the worst ranking algorithm among those compared. IPSOLS and IABCLS perform significantly worse than the best-performing optimizer in each dimension. SaDE ranks third and forth on dimensions 10 and 30 (the difference to the best not being statistically significant); on dimension 50 is performs statistically worse than the best algorithm. UACOR ranks second or third and is never statistically significantly worse than the best algorithm.

Considering tuned parameter settings, IPOP-CMA-ES now ranks best across all dimensions. Particularly striking is the behavior of SaDE, which ranks second on dimension 10 but it is the worst ranking algorithm on dimension 50, thus, exhibiting apparently poor scaling behavior. MA-LSCh-CMA ranks second on dimensions 30 and 50. While in dimension 10 IPSOLS is the only optimizer that performs statistically significantly worse than IPOP-CMA-ES, in dimensions 30 and 50, this is the case for IPSOLS, MOS, IABCLS and SaDE; using tuned parameter settings, MOS is not anymore the worst ranking of the seven algorithms, different from what had happened using default parameter settings.

Considering both, default and tuned parameter settings, we find that IPOP-CMA-ES, MA-LSCh-CMA and UACOR are the only three algorithms that never perform significantly worse than the best performing one in any dimension.

6.3. Additional experiments

When comparing tuned to default parameter settings, MA-LSCh-CMA was the only algorithm where the tuned settings in one dimension (dimension 50) performed worse than default settings (see Fig. 1). One reason may be that the tuning was done only on benchmark functions of dimension 10, thus, resulting possibly in poor scaling behavior with the number of dimensions. To examine this possibility, we re-tuned MA-LSCh-CMA on the 30-dimensional training problems (this tuned setup is called tcec-30D) and repeated the tests on dimensions 30 and 50. An analogous tuning has been done with SaDE as this algorithm suffered from a strong drop of performance with increasing problem dimension: the tuned SaDE ranked second for dimension 10 and worst for dimension 50. Fig. 3 shows that the parameter configurations obtained with the tcec-30D tuning setup improve over the default settings or the settings obtained by tuning on the 10-dimensional problems for both, MA-LSCh-CMA and SaDE. However,

[3] In particular, in the original papers a tuning budget of 50 000 runs of candidate configurations was considered and the functions were not presented to F-race one by one but in blocks covering the whole benchmark set.

Table 2
Parameters that have been considered for tuning. Given are the default values of the parameters and the range we considered for tuning. The last two columns are for the settings tuned on the CEC'05 (tcec) and SOCO (tsoco) benchmark sets, respectively. The type of parameter indicates whether the parameter is a categorical (c), an integer (i) or a real-valued (r) parameter.

IPOP-CMA-ES

Parameter (tuning)	Internal parameter	Default	Types	Range	Tuned	
					tcec	tsoco
a	Init pop size: $\lambda = 4 + \lfloor a \ln(D) \rfloor$	3	r	[1, 10]	7.315	9.742
b	Parent size: $\mu = \lfloor \lambda / b \rfloor$	2	r	[1, 5]	3.776	1.627
c	Init step size: $\sigma_0 = c \cdot (B - A)$	0.5	r	(0, 1)	0.8297	0.6256
d	IPOP factor: $ipop = d$	2	r	[1, 4]	2.030	3.560
e	$stopTolFun = 10^e$	−12	r	[-20, -6]	−8.104	−11.030
f	$stopTolFunHist = 10^f$	−20	r	[-20, -6]	−6.688	−13.270
g	$stopTolX = 10^g$	−12	r	[-20, -6]	−13.85	−13.36

MA-LSCh-CMA

Parameter (tuning)	Default	Types	Range	Tuned	
				tcec	tsoco
p	60	i	[30, 120]	86	66
$pmut$	0.125	r	(0, 0.2]	0.02710	0.03506
ils	500	i	[100, 700]	317	521
r	0.5	r	[0.2, 0.8]	0.6302	0.7338
th	−8	i	[−14, −1]	−11	−12
a	3	r	[1, 10]	2.152	3.563
b	2	r	[1, 5]	3.344	4.134
c	0.3	r	(0, 1)	0.1711	0.2886

SaDE

Parameter (tuning)	Default	Types	Range	Tuned	
				tcec	tsoco
p	50	i	[10, 100]	46	41
lp	50	i	[20, 80]	68	61
fm	0.5	r	[0, 1]	0.6322	0.6358
fsd	0.3	r	[0, 1]	0.8808	0.8177
$crsd$	0.1	r	[0, 1]	0.07112	0.8083

MOS

Parameter (tuning)	Default	Types	Range	Tuned	
				tcec	tsoco
p	15	i	[10, 100]	47	11
f	0.5	i	[0, 1]	0.9291	0.7198
cr	0.5	r	[0, 1]	0.5936	0.6723
mpr	0.05	r	[0.01, 0.1]	0.02747	0.07065
$fesls$	3000	r	[1000, 5000]	3495	4781

IPSOLS

Parameter (tuning)	Default	Types	Range	Tuned	
				tcec	tsoco
Top	FC	c	{FC, R}	FC	FC
iw	0	r	[0, 1]	0.4912	0.8091
ϕ_1	0.84	r	[0, 4]	1.968	0.3978
ϕ_2	0.85	r	[0, 4]	1.921	3.899
$initp$	1	i	[1, 100]	61	6
lsF	13	i	[1, 20]	12	3
$stagItr$	27	i	[2, 30]	16	24
g	1	i	[1,10]	2	6
$ftol$	−1	r	[−13, −1]	−1.485	−1.282
$lsItr$	87	i	[1, 100]	20	99

IABCLS

Parameter (tuning)	Default	Types	Range	Tuned	
				tcec	tsoco
$initp$	7	i	[2, 30]	20	6
g	9	i	[1, 10]	3	4
$Fmax$	44	i	[30, 100]	97	68
rf	−0.94	r	[−14, 0]	−1.327	−4.7370
$limit$	99	i	[10, 100]	52	76
lsF	9	i	[1, 20]	18	9
$ftol$	−5.609	r	[−13, −1]	−0.1455	−5.708
$lsItr$	82	i	[1, 100]	33	51

the improved performance for SaDE by tuning setting tcec-30D is not enough to improve its overall ranking when compared to the other algorithms: SaDE remains the fourth ranking algorithms on $D = 30$ and the worst ranking algorithm for $D = 50$ (see supplementary pages [34] for details).

7. Evaluation on the SOCO benchmark set

In this section, we evaluate the default parameter settings and the tsoco parameter settings on the SOCO benchmark set in 10, 50 and 100 dimensions.

Table 3
Parameters that have been considered for tuning. Given are the parameters and the continuous range we considered for tuning. The last two columns are for the settings on the CEC'05 (tcec) and the SOCO (tsoco) benchmark sets, respectively. The type of parameter indicates whether the parameter is a categorical (c), an integer (i) or a real-valued (r) parameter. Some settings are only significant for certain values of other settings; if they are not used, they are indicated by "∗".

Parameter (tuning)	Types	Range	Default[a]		Tuned[a]	
			tcec	tsoco	tcec	tsoco
DefaultMode	c	{T, F}	T	T	T	T
EliteQ$_{best}$	r	[0, 1]	*	*	*	*
InitAS	i	[20, 100]	66	48	66	48
IsIncrement	c	{T, F}	T	T	T	T
GrowthIter	i	[1, 30]	13	4	13	4
NaIsAS	c	{T, F}	T	F	T	F
Na	i	[2, 20]	*	16	*	16
Q_{best}	r	[0, 1]	0.5351	0.1895	0.5351	0.1895
WeightGsol	c	{T, F}	F	T	F	T
q	r	(0, 1)	*	0.2591	*	0.2591
ξ	r	(0, 1)	0.6945	0.6511	0.6945	0.6511
RmLocalWorse	c	{T, F}	T	F	T	F
SnewvsGsol	c	{T, F}	T	*	T	*
LsType	c	{F, Powell, Mtsls1}	Mtsls1	Mtsls1	Mtsls1	Mtsls1
LsIter	i	[1, 100]	28	84	28	84
LsFailures	i	[1, 20]	7	8	7	8
RestartType	c	{F, 1st, 2nd}	2nd	F	2nd	F
StagIter	r	[1, 1000]	11	*	11	*
StagThresh	r	[-15, 0]	-2.539	*	-2.539	*
Shakefactor	r	[-15, 0]	-0.02061	*	-0.02061	*
RestartAS	i	[2, 100]	10	*	10	*

[a] The default and tuned configurations of UACOR are considered coincident. The default configurations are derived from the automatic tuning procedures that are applied to tune other six optimizers.
* Parameter is not used in the specific configuration.

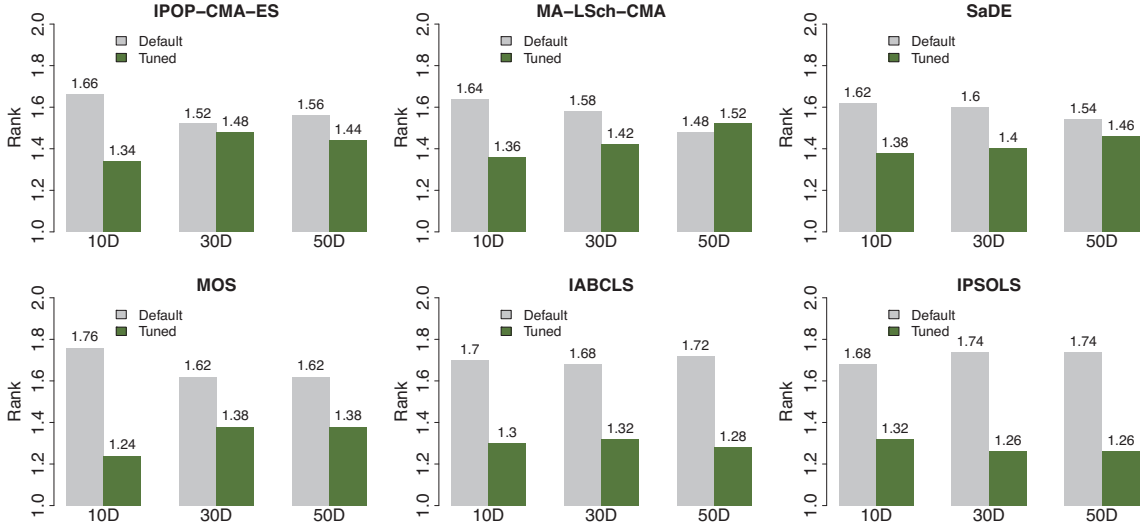

Fig. 1. Given are bar-plots that show the mean rank obtained by the default and the tuned parameter settings for the investigated optimizer on the CEC'05 benchmark set of dimensions 10, 30 and 50 respectively. The numbers on the top of the bars correspond to the mean rank values.

7.1. Effectiveness of automatic tuning

As before, we first examine the performance improvements obtained by tuned parameter settings with respect to default ones. Fig. 4 shows the comparison of the mean rank between default and tuned settings for IPOP-CMA-ES, MA-LSCh-CMA, SaDE, and MOS on the SOCO benchmark set of dimensions 10, 50 and 100. The tuned parameter settings improve in all cases over the default settings. Although SaDE uses a mechanism to self-adapt parameter settings during a run, the improvement given to SaDE through the offline tuning process is particularly striking.

As explained earlier, we use a different tuning setup for IPSOLS and IABCLS than the one used in the original papers. Fig. 5 compares the performance of the re-tuned algorithms to the settings from the original paper. While on the 10-dimensional functions, the performance of the "re-tuned" IPSOLS and IABCLS is worse than that with the original parameter settings, for 50 and 100 dimensions better results are obtained by the "re-tuned" settings. This may be an indication of over-tuning to the 10-dimensional functions in the original papers.

7.2. Performance comparison of optimizers

Fig. 6 compares the mean rank values for the optimizers using the default (left plots) and tuned (right plots) for the SOCO benchmark set of dimensions 10, 50 and 100 (from top to bottom). Again, a highlighted mean rank value in the plots indicates statistically significantly worse performance than the best optimizer according to Holm's test with $\alpha = 0.05$.

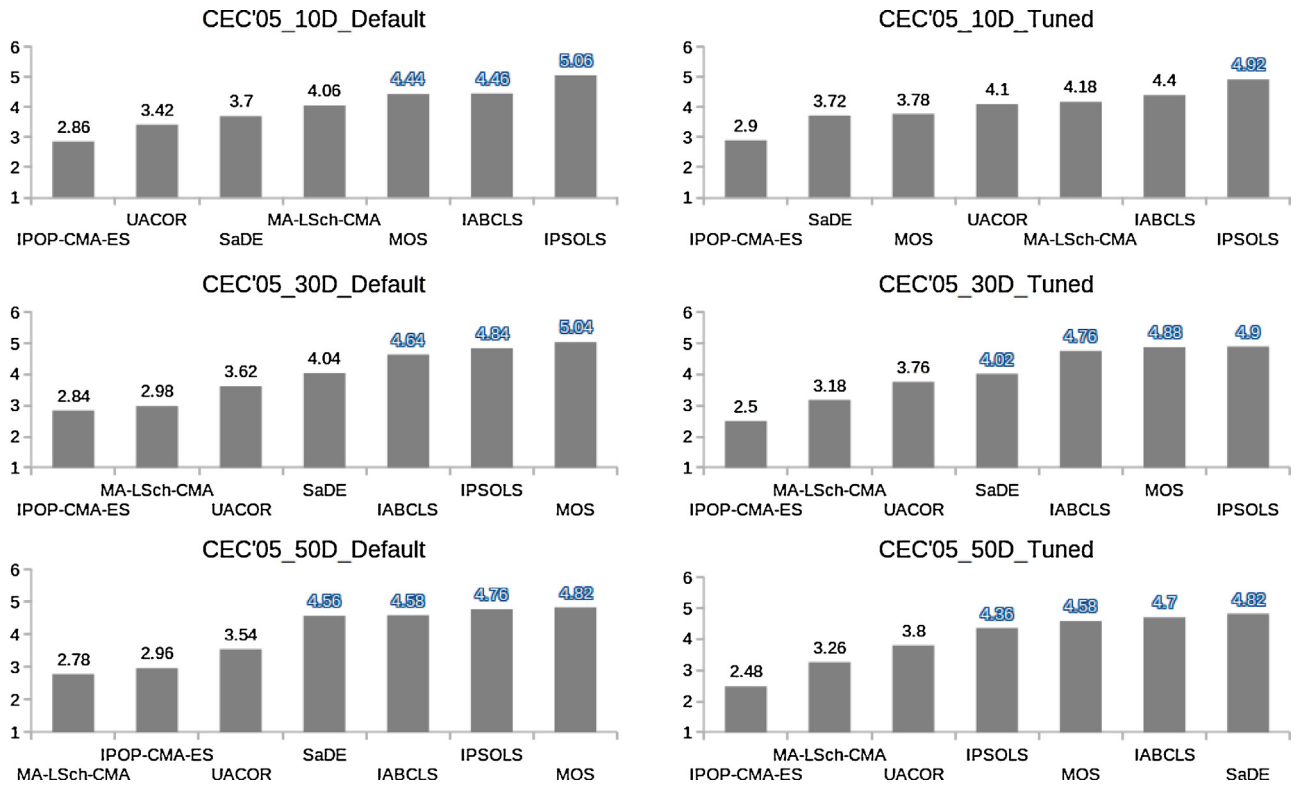

Fig. 2. Given are bar-plots with the mean rank values obtained by the default parameter settings of all optimizers (left plots) and by the tuned parameter settings (configuration tcec) of all the optimizers (right plots) on the CEC'05 benchmark set of dimensions 10, 30 and 50, respectively. Highlighted mean rank values indicate that its corresponding optimizer performs statistically significantly worse than the best-performing optimizer by the Holm's test at significance level $\alpha = 0.05$.

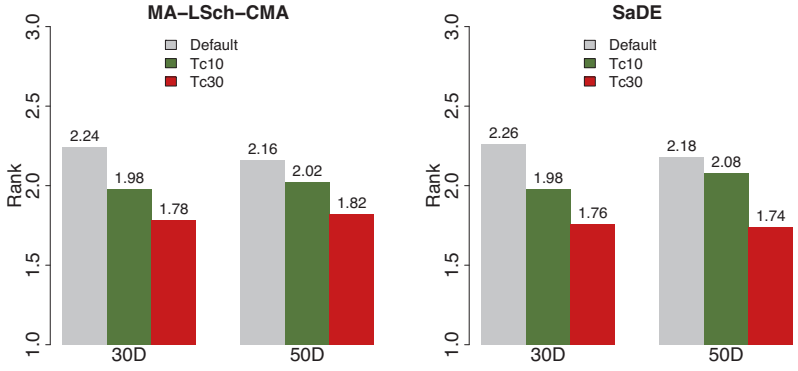

Fig. 3. Given are bar-plots with the mean rank values obtained by the default and tuned algorithm configurations (tuned on either 10- or 30-dimensional problems), indicated by Default, Tc10, Tc30, of MA-LSCh-CMA (left) and SaDE (right).

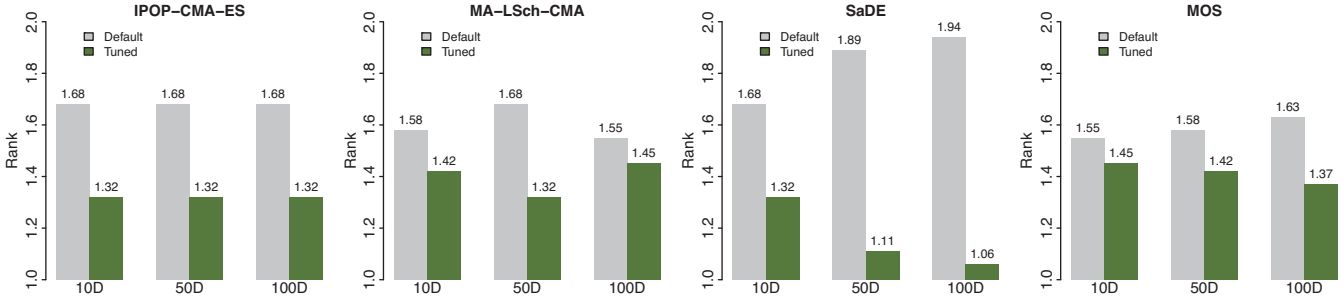

Fig. 4. Given are bar-plots with the mean rank values obtained by the default parameter settings and the parameter settings tsoco for each investigated optimizers on the SOCO benchmark set of dimensions 10, 50, and 100 respectively. The values labeled on the top of the bars correspond to the mean rank values.

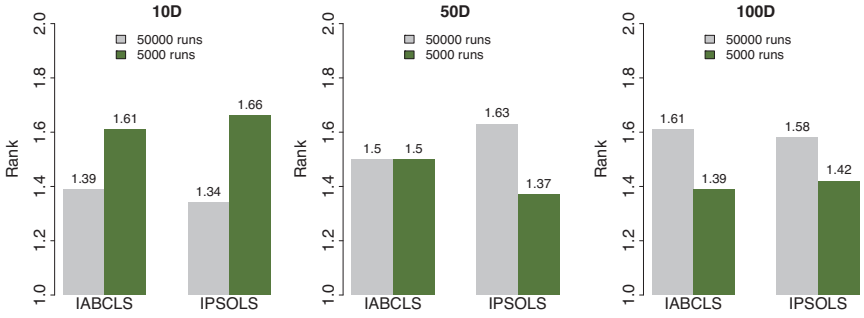

Fig. 5. Comparison between the "default" settings (indicated by "50000 runs", the tuning budget used in the original publication) and the re-tuned settings (indicated by 5000 runs, the tuning budget used in the present paper) of IABCLS and IPSOLS over dimensions 10, 50 and 100.

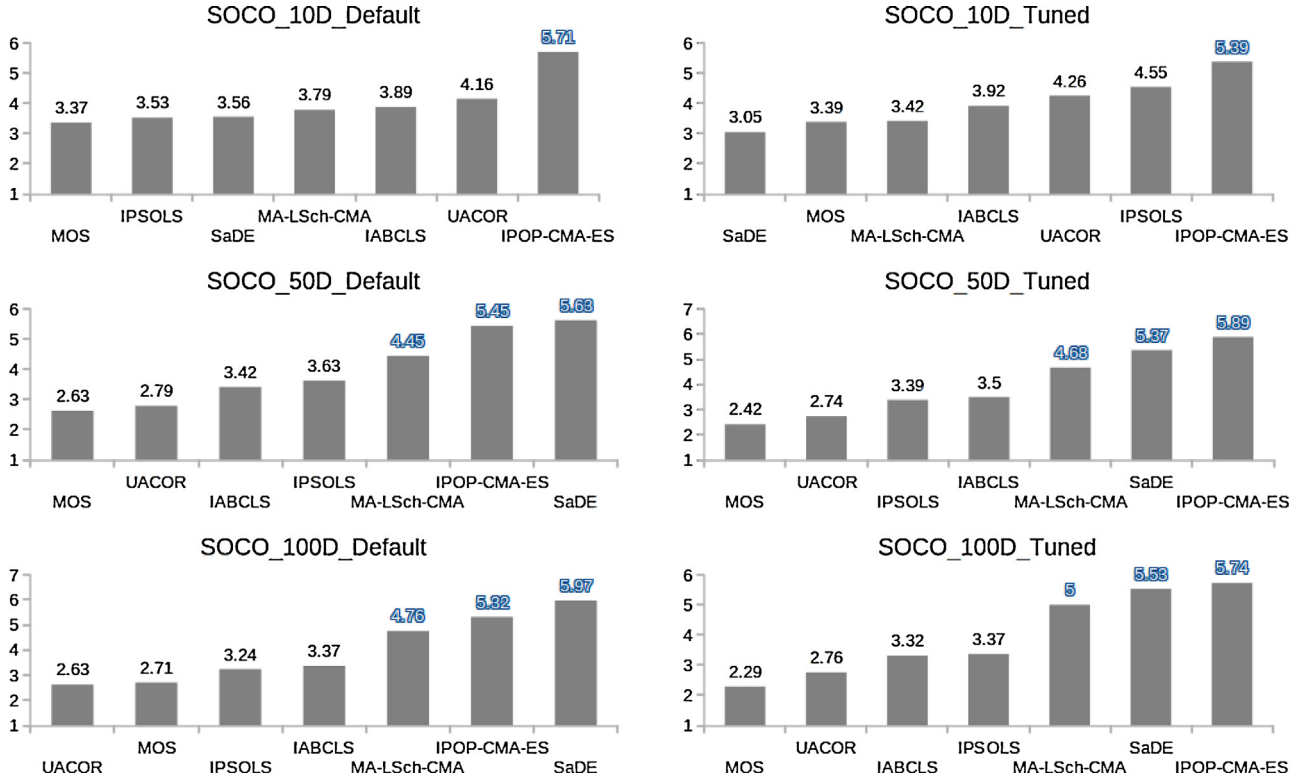

Fig. 6. Given are bar-plots with the mean rank values obtained by the default parameter settings of all optimizers (left plots) and by the tuned parameter settings (configuration tsoco) of all the optimizers (right plots) on the SOCO benchmark set of dimensions 10, 50 and 100, respectively. Highlighted mean rank values indicate that its corresponding optimizer performs statistically significantly worse than the best-performing optimizer by the Holm's test at significance level.

Considering the default optimizer configurations, the best ranking algorithms are MOS in dimensions 10 and 50 and UACOR in dimension 100. In 10 dimensions, IPOP-CMA-ES is the only optimizer that is statistically significantly worse than MOS, while in dimensions 50 and 100 this is the case for MA-LSCh-CMA, IPOP-CMA-ES and SaDE with respect to MOS or UACOR, respectively.

Considering tuned parameter settings, SaDE is the best ranking optimizer in 10 dimensions, while MOS becomes the best ranking algorithm in dimensions 50 and 100. IPOP-CMA-ES is the worst ranking algorithm across all dimensions, while it was the best ranking algorithm on the CEC'05 benchmark set. In dimension 10 it is the only algorithm that is statistically significantly worse than the best one, while in 50 and 100 dimensions this is also the case for SaDE and MA-LSCh-CMA (as in the case of default parameter settings). Considering SaDE, the poor scalability behavior to larger dimensions observed on the CEC'05 benchmark set is also evident on the SOCO benchmark set: the tuned version of SaDE is the best ranking algorithm in 10 dimensions but the worst in 50 and 100 dimensions.

Considering, MOS, UACOR, IABCLS and IPSOLS we observe that none of them is ranking statistically significantly worse than the best-performing optimizer in any of the comparisons on the SOCO benchmark functions.

8. Impact of tuning and testing on different benchmark sets

In the previous experiments, we tuned the optimizers on the 10-dimensional functions of one benchmark set and tested it on 10 and higher dimensional functions of the same benchmark set. When restricting to 10-dimensional functions, there is therefore some danger of over-tuning as the training and the test set are exactly the same. When comparing the algorithms on higher dimensional functions, one tests the generalization ability across different problem dimensions. One may wonder how one step further, a separation between the training set used for tuning and the test set impacts on the relative performance of the tuned algorithms.

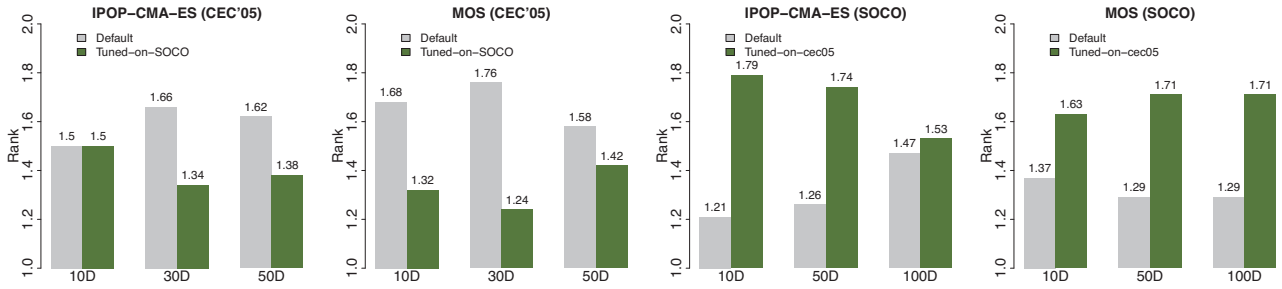

Fig. 7. Given are bar-plots with the mean rank values obtained by IPOP-CMA-ES and MOS on the CEC'05 and SOCO benchmark sets; left two plots: results when tuned on SOCO and tested on CEC'05; right two plots: results when tuned on CEC'05 and tested on SOCO.

As a first step, we applied IPOP-CMA-ES and MOS with the tcec (tsoco) parameter settings to the SOCO (CEC'05) benchmark set and compare to the default settings (results are given in Fig. 7). Interestingly, settings of IPOP-CMA-ES and MOS tuned on SOCO functions improve over the default parameter settings on CEC'05 functions. However, the opposite happens for both when tcec settings are tested on the SOCO functions: in that case, the default parameter settings perform better. This latter fact may be explained for MOS considering that MOS had been manually tuned by the algorithm designers for the SOCO benchmark competition. Nevertheless, tuning directly on the SOCO functions helped to improve MOS performance. Hence, tuned settings on the CEC'05 functions apparently generalize worse than those settings tuned on SOCO functions. Further research may be necessary to identify what actually makes a robust set of functions to obtain high generalization of the tuned parameter settings.

If we perform the cross-evaluation of all seven optimizers on the respectively other benchmark sets, we obtain the ranking given in Fig. 8: on the left side are given the CEC'05 benchmark results for tsoco settings and on the right are given the SOCO results for tcec settings. On the CEC'05 set, the overall ranking of the algorithms remains similar to that observed already in Section 6 with the noteworthy exception of UACOR, which now ranks poorly on the CEC'05 benchmark set.[4] On the SOCO benchmark set the ranking has changed more significantly and, in particular, IPSOLS is now the top ranking algorithm for dimensions 50 and 100. MOS looses its top-rank though it is not statistically significantly worse than the best algorithm; similarly MA-LSCh-CMA is now also not statistically significantly worse than the top performer. Still, IPOP-CMA-ES performs poorly on the SOCO functions and SaDE continues to show poor scaling behavior to larger dimensional functions.

9. Discussion and conclusions

We have tested seven high-performing optimizers considering "default" and automatically tuned parameter settings and compared the ranking of the mean errors of these optimizers on the CEC'05 and the SOCO benchmark sets. From a high-level perspective, one main result of the comparison is that the benchmark set has a considerable impact on the final ranking of the algorithms. While IPOP-CMA-ES was generally the best performing algorithm on the CEC'05 benchmark set, independent of whether one considers default or tuned parameter settings, on the SOCO benchmark set overall MOS was the best ranking algorithm. Concerning the other algorithms, MA-LSCh-CMA, which uses CMA-ES as a local search method and which was developed using as target benchmark the CEC'05 set, tends to follow the same trend as does IPOP-CMA-ES. IPSOLS and IABCLS, roughly speaking, follow the same trends as does MOS: relatively poor performance on the CEC'05 benchmark set but good performance on the SOCO benchmark set. Particular patterns are followed by SaDE and UACOR. SaDE appears to suffer from poor scaling behavior: while on the 10-dimensional functions it is a top performing algorithm, it is in all cases statistically significantly worse performing than the best algorithm on the test functions with 50 or 100 dimensions. UACOR is never statistically significantly worse than the best performing algorithm on any of the benchmark sets provided its parameter settings have been tuned on the same functions of the given benchmark set; an exception is only observed when tuning UACOR on the SOCO benchmark set but testing it on the CEC'05 one. The latter is possibly an effect of the larger tunability of UACOR, which is conceived as an algorithm framework rather than a stand-alone algorithm.

Hence, one main conclusion from these experiments is that the choice of the benchmark set on which the algorithms are evaluated determines to a large extent the ranking of the algorithms, the most striking example being the relative ranking of IPOP-CMA-ES and MOS on the two benchmark sets (best on one and worst on the other for CEC'05 and SOCO, respectively). The benchmark set has also a much larger influence on algorithm ranking than has the fact whether the algorithms are tuned or default parameter settings are taken (though tuning proved to be useful as in most cases improved performance over manual settings was obtained). Interestingly, whether an algorithm ranks among the better or the worse algorithms on a benchmark set is apparently determined in large part by the benchmark set that was used on which the algorithm has been evaluated first or that was used to evaluate several alternatives during an algorithm's design. Hence, in a sense, the benchmark sets introduce a clear bias toward one or another algorithm.

For example, one main difference between the benchmark sets is that, although in both benchmark sets most functions are not separable, in the CEC'05 benchmark set many functions are additionally artificially rotated to induce (stronger) variable correlations among all variables. Decisive for good performance on this benchmark set appears to be the ability of algorithms to deal with such rotations. An algorithm that was designed to have this property is CMA-ES, the local search algorithm underlying IPOP-CMA-ES: CMA-ES is known to be scale and rotation invariant [16].

How should one deal with these inherent biases? In fact, an open question is whether the SOCO or the CEC'05 benchmark set are the more appropriate benchmark sets. The answer probably is that none of the two is really representative for the multitude of possible functions or function characteristics that may arise in applications. In our opinion, more problems from real-world applications should be made available as test cases for direct-search methods. One

[4] The poor performance of UACOR is maybe an artifact of the fact that this algorithm is a flexible, configurable framework: this may help to fine-tune better the algorithm configurations to a specific benchmark set (or, say, application scenario), which may therefore not generalize that well to other scenarios (see also discussion in [50]). In a sense, this result may be due to the fact that the training set should be representative for the test set [6], which here is apparently not the case.

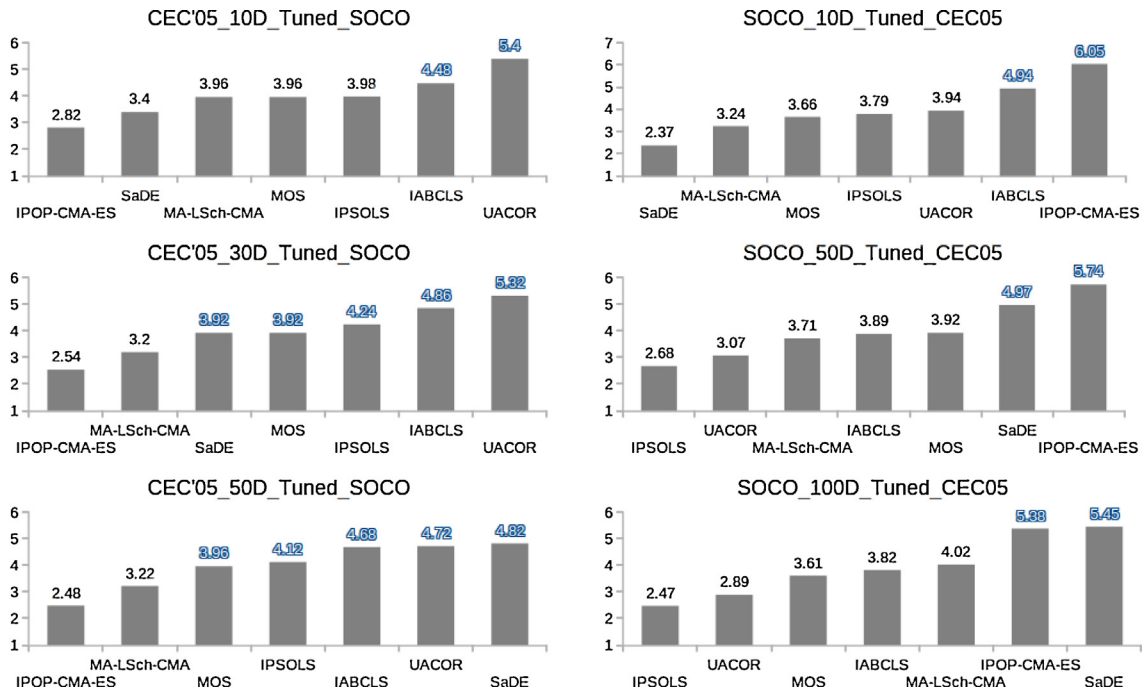

Fig. 8. Given are bar-plots with the mean rank values obtained by testing on the CEC'05 functions the parameter setting tuned on the SOCO functions (settings tsoco, left) and those obtained by testing on the SOCO functions the parameter settings tuned on the CEC'05 functions (settings tcec, right). The highlighted mean rank values indicate that the corresponding optimizer performs statistically significantly worse than the best-performing optimizer by the Holm's test at significance level $\alpha = 0.05$.

first step may be to collect such problems from published articles and to make them available through a dedicated benchmark problem website. This is certainly a time-consuming task both for the preparation of the benchmarks as well as for testing the continuous optimizers as the evaluation in real-world applications may be computationally expensive and may rely on platform specific simulations. However, it will help to avoid relying the algorithm development too strongly on artificial benchmark problems.

Artificial benchmark problems will certainly be used and be useful in future research efforts as they allow easier testing specific features of algorithm performance. Again, as a first step, we think that a systematic collection and classification of the available benchmark problems according to their specific function properties and known behavior of optimizers on these functions is needed. These benchmark problems ideally should be made available in different formats for easy integration into codes and collect problems from benchmark competitions such as SOCO, CEC'05 (and more recent editions CEC'13 and CEC'14), BBOB [17] but also of benchmark collections developed in the mathematical programming community such as CUTEr [15]. However, for this we believe that many more artificial benchmark problems with systematically varied features should complement the already available standard ones for experimental comparisons. For example, one may define benchmark functions with varying degrees of variable correlations among subsets of variables and varying conditioning numbers. In fact, rather hundreds or thousands of additional, new problems should be generated to extend the currently available benchmark problems. Such benchmark problems should enable studies that examine the relative performance of continuous optimizers in dependence of problem features, a good recent example being the study of Hansen et al. [19]. Such studies should lead to refined knowledge on the impact specific function features have on the relative performance of continuous optimizers. Such knowledge may also be useful to choose one over another optimizer if something is known about the high-level properties of the landscape of the function to be optimized. In this context, recent efforts in the landscape analysis of continuous optimization problems [41] and the usage of algorithm selection techniques [8,45] may provide important tools for further progress. In a sense, we think that the variety and the difficulty of continuous optimization problems will continue to raise interesting research questions that may engender further advances for their effective solution in the future.

On the algorithmic side, it may be useful to design solvers as configurable software frameworks that can be re-tuned for specific characteristics of functions. This direction may be useful when the optimization task is repetitive and functions with similar characteristics have to be repeatedly solved. The computational results of UACOR show that this may be one viable direction. In fact, such software frameworks should be designed in a way to include complementary algorithm components that are known to be able to deal effectively with specific function characteristics; the recent extension of UACOR to include CMA-ES as local search algorithm (which effectively deals with high variable correlations due to its rotation-invariance) [38], gives evidence for this fact.

Acknowledgments

This work was supported by the Meta-X project, by the Scientific Research Directorate of the French Community of Belgium, the COMEX project (P7/36) within the Inter-university Attraction Poles Programme of the Belgian Science Policy Office, and the project 71401167 by the National Natural Science Foundation of China (NSFC). Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a senior research associate, Tianjun Liao acknowledges a fellowship from the China Scholarship Council and a grant from NSFC project 71401167, and Daniel Molina acknowledges support from the research projects TIN2012-37930-C02-01 and P08-TIC-04173.

Appendix A. Supplementary Data

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.asoc.2014.11.006.

References

[1] A. Auger, N. Hansen, A restart CMA evolution strategy with increasing population size, in: Proceedings of the 2005 Congress on Evolutionary Computation (CEC 2005), IEEE Press, Piscataway, NJ, 2005, pp. 1769–1776.

[2] A. Auger, N. Hansen, J.M. Perez Zerpa, R. Ros, M. Schoenauer, Experimental comparisons of derivative free optimization algorithms, in: J. Vahrenhold (Ed.), Proceedings of the International Symposium on Experimental Algorithms, SEA'09, Lecture Notes in Computer Science, Springer, Heidelberg, Germany, 2009, pp. 3–15.

[3] D. Aydın, T. Liao, M.A. Montes de Oca, T. Stützle, Improving performance via population growth and local search: the case of the artificial bee colony algorithm. Technical Report TR/IRIDIA/2011-015, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

[4] T. Back, D.B. Fogel, Z. Michalewicz, Handbook of Evolutionary Computation, IOP Publishing, Bristol, UK, 1997.

[5] H. Bersini, M. Dorigo, S. Langerman, G. Seront, L.M. Gambardella, Results of the first international contest on evolutionary optimisation, in: T. Bäck, T. Fukuda, Z. Michalewicz (Eds.), Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96), IEEE Press, Piscataway, NJ, 1996, pp. 611–615.

[6] M. Birattari, Tuning metaheuristics: a machine learning perspective, in: Volume 197 of Studies in Computational Intelligence, Springer, Berlin/Heidelberg, Germany, 2009.

[7] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-race and iterated F-race: an overview, in: T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (Eds.), Experimental Methods for the Analysis of Optimization Algorithms, Springer, Berlin, Germany, 2010, pp. 311–336.

[8] B. Bischl, O. Mersmann, H. Trautmann, M. Preuss, Algorithm selection based on exploratory landscape analysis and cost-sensitive learning, in: T. Soule, J.H. Moore (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2012, ACM Press, New York, NY, 2012, pp. 313–320.

[9] A.R. Conn, K. Scheinberg, L.N. Vicente, Introduction to derivative-free optimization, in: MPS-SIAM Series on Optimization, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.

[10] W.J. Conover, Practical Nonparametric Statistics, third ed., John Wiley & Sons, New York, NY, 1999.

[11] M. Dorigo, Optimization, Learning and Natural Algorithms (in Italian), PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

[12] M. Dorigo, T. Stützle, Ant Colony Optimization, MIT Press, Cambridge, MA, 2004.

[13] A.E. Eiben, Z. Michalewicz, M. Schoenauer, J.E. Smith, Parameter control in evolutionary algorithms, in: F. Lobo, C.F. Lima, Z. Michalewicz (Eds.), Parameter Setting in Evolutionary Algorithms, Springer, Berlin, Germany, 2007, pp. 19–46.

[14] M. El-Abd, Performance assessment of foraging algorithms vs evolutionary algorithms, Inform. Sci. 182 (1) (2012) 243–263.

[15] N.I.M. Gould, D. Orban, P.L. Toint, CUTEr and SifDec: a constrained and unconstrained testing environment, revisited, ACM Trans. Math. Softw. 29 (2003) 373–394.

[16] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evol. Comput. 9 (2) (2001) 159–195.

[17] N. Hansen, A. Auger, S. Finck, R. Ros, Real-parameter black-box optimization benchmarking 2009: Experimental setup. Technical Report RR-6828, INRIA, France, 2009.

[18] N. Hansen, A. Auger, R. Ros, S. Finck, P. Pošík, Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009, in: Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO'10 (Companion), ACM, New York, NY, USA, 2010, pp. 1689–1696.

[19] N. Hansen, R. Ros, N. Mauny, M. Schoenauer, A. Auger, Impacts of invariance in search: when CMA-ES and PSO face ill-conditioned and non-separable problems, Appl. Soft Comput. 11 (8) (2011) 5755–5769.

[20] F. Herrera, M. Lozano, J. Verdegay, Tackling real-coded genetic algorithms: operators and tools for behavioural analysis, Artif. Intell. Rev. 12 (1998) 265–319.

[21] F. Herrera, M. Lozano, D. Molina, Test suite for the special issue of soft computing on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems, 2010 http://sci2s.ugr.es/eamhco/

[22] H.H. Hoos, Programming by optimization, Commun. ACM 55 (2) (2012) 70–80.

[23] F. Hutter, H.H. Hoos, K. Leyton-Brown, T. Stützle, ParamILS: an automatic algorithm configuration framework, J. Artif. Intell. Res. 36 (October) (2009) 267–306.

[24] F. Hutter, H.H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: C.A. Coello Coello (Ed.), Learning and Intelligent Optimization, 5th International Conference, LION 5, vol. 6683 of Lecture Notes in Computer Science, Springer, Heidelberg, Germany, 2011, pp. 507–523.

[25] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, J. Global Optim. 39 (2007) 459–471, 0925-5001.

[26] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, vol. 4, IEEE Press, Piscataway, NJ, USA, 1995, pp. 1942–1948.

[27] J. Kennedy, R.C. Eberhart, Y. Shi, Swarm Intelligence, Morgan Kaufmann Publishers, San Francisco, CA, 2001.

[28] N. Krasnogor, J. Smith, A tutorial for competent memetic algorithms: model, taxonomy, and design issues, IEEE Trans. Evol. Comput. 9 (5) (2005) 474–488.

[29] A. LaTorre, A framework for hybrid dynamic evolutionary algorithms: multiple offspring sampling (MOS), PhD thesis, Universidad Politécnica de Madrid, Spain, 2009.

[30] A. LaTorre, S. Muelas, J.-M. Peña, A MOS-based dynamic memetic differential evolution algorithm for continuous optimization: a scalability test, Soft Comput. 15 (11) (2011) 2187–2199.

[31] G. Leguizamón, C. Coello, An alternative ACOR algorithm for continuous optimization problems, in: M. Dorigo, M. Birattari, G.A. Di Caro, R. Doursat, A.P. Engelbrecht, D. Floreano, L.M. Gambardella, R. Groß, E. Sahin, T. Stützle, H. Sayama (Eds.), Proceedings of the Seventh International Conference on Swarm Intelligence, ANTS'10, vol. 6234 of Lecture Notes in Computer Science, Springer, Heidelberg, Germany, 2010, pp. 48–59.

[32] T. Liao, Population-based Heuristic Algorithms for Continuous and Mixed Discrete-Continuous Optimization Problem, PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 2013.

[33] T. Liao, M.A. Montes de Oca, D. Aydin, T. Stützle, M. Dorigo, An incremental ant colony algorithm with local search for continuous optimization, in: N. Krasnogor, P.L. Lanzi (Eds.), Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12–16, ACM Press, New York, NY, 2011, pp. 125–132.

[34] T. Liao, D. Molina, T. Stützle, Performance evaluation of automatically tuned continuous optimizers on different benchmark sets: Supplementary material, 2012 http://iridia.ulb.ac.be/supp/IridiaSupp2014-001/

[35] T. Liao, D. Aydin, T. Stützle, Artificial bee colonies for continuous optimization: experimental analysis and improvements, Swarm Intell. 7 (4) (2013) 327–356.

[36] T. Liao, M.A. Montes de Oca, T. Stützle, Computational results for an automatically tuned CMA-ES with increasing population size on the CEC'05 benchmark set, Soft Comput. 17 (6) (2013) 1031–1046.

[37] T. Liao, T. Stützle, M.A. Montes de Oca, M. Dorigo, A unified ant colony optimization algorithm for continuous optimization. Technical Report TR/IRIDIA/2013-002, IRIDIA, Université Libre de Bruxelles, Belgium, 2013.

[38] T. Liao, T. Stützle, M.A. Montes de Oca, M. Dorigo, A unified ant colony optimization algorithm for continuous optimization, Eur. J. Operat. Res. 234 (3) (2014) 597–609.

[39] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, M. Birattari, The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011, http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf.

[40] M. Lozano, D. Molina, F. Herrera, Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems, Soft Comput. 15 (2011) 2085–2087.

[41] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, Rudolph G., Exploratory landscape analysis, in: N. Krasnogor, P.L. Lanzi (Eds.), Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12–16, ACM Press, New York, NY, 2011, pp. 829–836.

[42] D. Molina, M. Lozano, C. García-Martínez, F. Herrera, Memetic algorithms for continuous optimisation based on local search chains, Evol. Comput. 18 (1) (2010) 27–63.

[43] M. Montes de Oca, D. Aydn, T. Stützle, An incremental particle swarm for large-scale continuous optimization problems: an example of tuning-in-the-loop (re)design of optimization algorithms, Soft Comput. 15 (2011) 2233–2255.

[44] P. Moscato, Memetic algorithms: a short introduction, in: New Ideas in Optimization, McGraw Hill, London, UK, 1999, pp. 219–234.

[45] M.A. Muñoz, M. Kirley, S.K. Halgamuge, A meta-learning prediction model of algorithm performance for continuous optimization problems, in: C.A.C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, M. Pavone (Eds.), Parallel Problem Solving from Nature XII, vol. 7491 of Lecture Notes in Computer Science, Heidelberg, Germany, 2012, pp. 226–235.

[46] V. Nannen, A.E. Eiben, Relevance estimation and value calibration of evolutionary algorithm parameters, in: M.M. Veloso (Ed.), Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07), AAAI Press, Menlo Park, CA, 2007, pp. 975–980.

[47] M. Powell, An efficient method for finding the minimum of a function of several variables without calculating derivatives, Comput. J. 7 (2) (1964) 155, 0010-4620.

[48] A. Qin, V. Huang, P. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Trans. Evol. Comput. 13 (2) (2009) 398–417.

[49] L.M. Rios, N.V. Sahinidis, Derivative-free optimization: a review of algorithms and comparison of software implementations, J. Global Optim. 56 (2013) 1247–1293.

[50] S.K. Smit, A.E. Eiben, Parameter tuning of evolutionary algorithms: generalist vs. specialist, in: C.D. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. Esparcia-Alcázar, C.K. Goh, J.J.M. Guervós, F. Neri, M. Preuss, J. Togelius, G.N. Yannakakis (Eds.), EvoApplications (1), vol. 6024 of Lecture Notes in Computer Science, Springer, Heidelberg, Germany, 2010, pp. 542–551.

[51] K. Socha, M. Dorigo, Ant colony optimization for continuous domains, Eur. J. Oper. Res. 185 (3) (2008) 1155–1173.

[52] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (4) (1997) 341–359.

[53] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical Report 2005005, Nanyang Technological University, 2005.
[54] E.-G. Talbi, A taxonomy of hybrid metaheuristics, J. Heuristics 8 (5) (2002) 541–564.
[55] L. Tseng, C. Chen, Multiple trajectory search for large scale global optimization, in: Proceeding of IEEE Congress on Evolutionary Computation, CEC, IEEE Press, Piscataway, NJ, 2008, pp. 3052–3059.
[56] L.D. Whitley, S. Rana, J. Dzubera, K.E. Mathias, Evaluating evolutionary algorithms, Artif. Intell. 85 (1996) 245–296.