Parameter tuning versus adaptation: proof of principle study on differential evolution

By

Magnus Erik Hvass Pedersen, Andrew John Chipperfield

Hvass Laboratories Technical Report no. HL0802, 2008

Abstract

The efficacy of an optimization method often depends on the choosing of a number of control parameters. Practitioners have traditionally chosen these control parameters manually, often according to elaborate guidelines or in a trial-and-error manner, which is laborious and susceptible to human misconceptions of what causes an optimizer to perform well. In recent years many variants to original optimization methods have appeared, which seek to adapt the control parameters during optimization, so as to remedy the need for a practitioner to determine good control parameters for a problem at hand. Ironically however, these variants typically just introduce new and additional parameters that must be chosen by the practitioner. Despite this obvious paradox these optimizer variants are still considered state-of-the-art, because they do show performance improvement empirically. In this paper, such variants of the general purpose optimization method known as Differential Evolution (DE) are studied with the intent of determining if their schemes for adapting control parameters yield an actual performance advantage over the basic form of DE which keeps the control parameters fixed during optimization. To fairly compare the performance of these optimizer variants against each other, their control parameters are all tuned by an automated approach. This unveils their true performance capabilities, and the results show that the DE variants generally have comparable performance, and hence that adaptive parameter schemes do not appear to yield a general and consistent performance improvement, as previously believed.

Keywords: Numerical optimization, direct-search, stochastic, multi-agent, parameter tuning.

1 Introduction

The optimization method known as Differential Evolution (DE) was originally introduced by Storn and Price [1], and offers a way of optimizing a given problem without explicit knowledge of the gradient of that problem. This is particularly useful if the gradient is difficult or even impossible to derive.

Usually this kind of optimization method has a number of control parameters that allow for a practitioner to vary the optimizing behaviour and efficacy. Tuning these control parameters to a problem at hand is typically done manually by the practitioner in a trial-and-error fashion. Much research effort has been put into finding guidelines for choosing proper DE control parameters for different kinds of optimization problems, see for example the work by Storn et al. [2] [3], Liu and Lampinen [4], or the attempt at mathematically deriving

good DE parameters by Zaharie [5]. Other research has been devoted to eliminating the need for parameter tuning altogether, by perturbing or adapting the control parameters during optimization, which is frequently reported to yield an improvement over using fixed control parameters, see for example the work by Liu and Lampinen [6], Qin et al. [7] [8], and Brest et al. [9]. Ironically, however, the DE variants with so-called adaptive control parameters, devised to eliminate the need for parameter tuning, typically just introduce new and additional parameters that must be tuned.

This paper takes another approach, acknowledging the fact that DE itself is a complex adaptive system, and studies whether its control parameters can be tuned, so as to make it perform on par with the adaptive variants that are claimed to be superior. To unveil the core performance capabilities of all these DE variants, their control parameters must be tuned properly so the comparison is made fairly. Doing this without having to resort to manual calibration of parameters, the issue of finding the best choice of control parameters is considered an optimization problem in its own right, and hence solved by an overlaying optimization method. This is known in the literature as Meta-Optimization, Meta-Evolution, Super-Optimization, Automated Parameter Calibration, Hyper-Heuristics, etc. [10] [11] [12] [13] [14] [15] [16]. Our own approach to meta-optimization from [17] is used in this study as well, because it is both simple and efficient.

Using meta-optimization of DE control parameters, it is found that the basic DE with fixed parameters actually has performance comparable to the more complex and so-called adaptive DE variants when their control parameters are properly tuned. This comparability is in the sense that the basic DE sometimes performs better and sometimes worse than the variants with adaptive parameters, but on a whole the performance tendencies are quite similar. The study therefore disproves much of the "folklore" within this research field, both with regard to the guidelines for properly choosing control parameters, but also concerning the trend of developing ever more complex optimization methods. These findings are expected to extend to other optimization methods as well. For instance, it was already demonstrated in [17] that the optimization method known as Particle Swarm Optimization (PSO) and due to Kennedy et al. [18] [19], could actually be simplified without impairing its performance, if only its control parameters were properly tuned. And as DE is conceptually similar to the Genetic Algorithm (GA) by Holland [20], the same may hold for the GA as well as other evolutionary optimization methods.

The paper is organized as follows. Section 2 describes DE along with some adaptive variants that are considered state-of-the-art. Section 3 describes an optimization method especially suited for meta-optimization, and section 4 describes how to employ it as an overlaid meta-optimizer. The benchmark problems used in this study are described in section 5, and the experimental settings and results are given in section 6. Section 7 discusses these results and their implications for the research field, and conclusive remarks are given in section 8.

2 Differential Evolution

The population-based optimization method known as Differential Evolution (DE) is due to Storn and Price [1]. DE does not explicitly rely on the gradient of the problem it is optimizing, but works by having multiple agents collaborate in a direct-search manner, treating the optimization problem as a black-box which merely delivers a measure of fitness for candidate solutions. DE then creates new candidate solutions by combining the solutions of randomly chosen agents from its population, and accepting the new solutions in case of fitness improvement.

2.1 Mutation And Crossover

DE employs evolutionary operators that are dubbed crossover and mutation in its attempt to minimize some fitness function $f : \mathbb{R}^n \to \mathbb{R}$. The operators are typically applied in turn but have been combined for a more concise description in the following. This study uses the DE/rand/1/bin variant, because it is believed to be the best performing and hence the most popular of the basic DE variants [3] [21].

Let $\vec{y} \in \mathbb{R}^n$ be the new candidate solution for an agent whose currently best known solution is $\vec{x}$, and let $\vec{a}$, $\vec{b}$, and $\vec{c}$ be the solutions of distinct and randomly chosen agents, which must also be distinct from the agent $\vec{x}$ that is currently being updated. The elements of $\vec{y} = [y_1, \cdots, y_n]$ are then computed as follows:

$$y_i = \begin{cases} a_i + F \cdot (b_i - c_i) & , r_i < CR \vee i = R \\ x_i & , \text{else} \end{cases} \tag{1}$$

where $F \in [0, 2]$ is a user-adjustable control parameter called the differential weight, and the randomly chosen index $R \in \{1, \cdots, n\}$ ensures at least one element differs from that of the original agent: $y_R \neq x_R$. While the rest of the elements are either chosen from the original solution $\vec{x}$ or computed from combining other agents' solutions, according to the user-adjustable crossover probability $CR \in [0, 1]$, and with $r_i \sim U(0, 1)$ being a uniform random number drawn for each use. An additional user-adjustable control parameter of DE is the population size NP, that is, the number of agents in the DE population.

Once the new potential solution $\vec{y}$ has been computed, it will replace the agent's original solution $\vec{x}$ in the case of improvement to the fitness. The DE algorithm is shown in figure 1.

2.2 Perturbed Control Parameters

It has been recognized since the inception of DE that different choices of control parameters cause it to perform worse or better on particular problems, and that the selection of good parameters is a challenging art, see for example Storn et al. [2] [3], Liu and Lampinen [4], and Zaharie [5].

An attempt to remedy the need for a user to determine the best DE control parameters for a given optimization problem is to perturb the parameters during

optimization. Several schemes for perturbing the differential weight F have been proposed in the literature [3] [22], where the ones used by Storn himself are generally the simple Dither and Jitter schemes [21] [23]. In the Dither scheme a random differential weight F is picked on a per-agent basis:

$$F = F_l + r' \cdot (F_h - F_l) \tag{2}$$

so $r' \sim U(0,1)$ is a random number picked once for each agent being updated using Eq.(1). Another way of perturbing the differential weight is to use the Jitter scheme, also from [3] [23]:

$$F_i = F_{mid} \cdot (1 + \delta \cdot (r'_i - 0.5))$$

where $r'_i \sim U(0,1)$ is now being picked for each element of the vector being updated in Eq.(1). The parameter δ determines the scale of perturbation and will be eliminated shortly. Although these formulae appear to be distinct at first glance they are in fact equivalent, with the exception of Dither drawing a random differential weight once for each agent-vector to be updated, and Jitter drawing a random weight for each element of that vector. To see this let $F_l = F_{mid} \cdot (1 - \delta/2)$ and $F_u = F_{mid} \cdot (1 + \delta/2)$ in Eq.(2). A simpler and more mathematical description of Dither would therefore be:

$$F \sim U(F_l, F_u)$$

And for Jitter:

$$F_i \sim U(F_{mid} \cdot (1 - \delta/2), F_{mid} \cdot (1 + \delta/2))$$

Using a midpoint and a range instead of perturbation boundaries has the advantage of being independent from each other, where as the F_l and F_h boundaries must also satisfy $F_l < F_u$, which would make automatic tuning of these more difficult later in this study. A common notation will therefore be used here for both Dither and Jitter, having a midpoint F_{mid} and a range F_{range}. For Dither this means the differential weight is picked randomly as:

$$F \sim U(F_{mid} - F_{range}, F_{mid} + F_{range})$$

And for Jitter this would be:

$$F_i \sim U(F_{mid} - F_{range}, F_{mid} + F_{range})$$

meaning that F_i should be drawn once for every vector-element i in Eq.(1), as opposed to once for the entire vector in the Dither scheme. The midpoint and range of perturbation are indeed just other control parameters, where the midpoint can be anywhere between $F_{mid} \in [0, 2]$, and the perturbation range can be anywhere between $F_{range} \in [0, 3]$, which were chosen to allow for unusual values when F_{mid} and F_{range} will be automatically tuned later in this study. Note that this also allows for negative differential weights to occur. For Dither

this does not change much, because the negative weight is going to be multiplied with the difference between two randomly picked agents in Eq.(1), and these agents could just as well have been chosen in the reverse order, making the negative differential weight equivalent to its positive counterpart. For Jitter this is not the case however, because the computation of some vector-elements in Eq.(1) might use a positive differential weight and others a negative. The semantic meaning of this and how it relates to optimization performance is impossible to foresee, but the experiments below have shown this is not an issue.

The intention of perturbing the differential weight F was to remove the need for a user to select a proper value for F. But it has actually introduced two new control parameters which must be selected by the user, as hinted at above. These new parameters determine the boundaries within which the perturbation of F should occur. It seems to be the belief of researchers and practitioners, that replacing fixed control parameters with stochastic ones makes it easier for a user to tune the optimizer's behaviour, because the choosing of boundaries for the stochastic parameters affect optimization performance more leniently than choosing the actual parameters. Perhaps the belief is that sooner or later a good choice of control parameter will be chosen at random. But one has to remember that completely random sampling of anything, whether it be control parameters or solutions to the actual optimization problem at hand, statistically takes a great many samples before finding a good choice. Therefore one might question if perturbation of parameters such as done in the Dither and Jitter schemes has a real chance of finding good choices of control parameters, or whether such perturbation just slightly alters the underlying dynamic behaviour of the DE agents, without having a generally adaptive effect on the control parameters that would make DE perform well on mostly any given optimization problem.

2.3 Adaptive Control Parameters

Perturbing control parameters can be taken a step further by using the fitness improvement to guide selection of control parameters. This is thought not only to alleviate the need for a user to choose control parameters, matching them to the problem at hand, but also to better adapt these parameters during optimization, so as to better balance exploration versus exploitation of the search-space.

The Fuzzy Adaptive DE (FADE) by Liu and Lampinen [6] uses Fuzzy Logic to adapt the DE control parameters during an optimization run. Fuzzy logic, originally due to Zadeh [24], provides a means for logical reasoning with uncertainties. In FADE the fuzzy reasoning is used to alter DE control parameters according to observations of fitness improvements and population diversity, and is reported to outperform DE with fixed and hand-tuned control parameters, especially on a range of benchmark problems with higher dimensionalities.

Another example of an adaptive DE variant is the Self-adaptive DE (SaDE) due to Qin and Suganthan [7]. The SaDE variant perturbs the F parameter according to a normal distribution, and while the CR parameter is also randomly picked, its observed effect on fitness improvement influences how and when

this random picking occurs. Additionally SaDE uses several DE variants (e.g. DE/rand/1/bin and DE/current-to-best/1/bin) which are switched between in a stochastic manner, according to their observed ability to improve the fitness during optimization. Note that this is significantly more complicated than the original DE method, and will require a good deal more effort in implementation, which must be made up for by improved performance.

In this study the adaptive DE variant known as JDE will be used, which is due to Brest et al. [9]. This variant has been chosen here because its performance has been found to compare well against other so-called state-of-the-art, adaptive DE variants [9] [25]. It is presented by its authors as a Self-Adaptive DE as well, because it eliminates the need for a user to select the F and CR parameters; yet ironically introduces 8 new user-adjustable parameters to achieve this. But this tendency is common for adaptive DE variants, and indeed also for the comparatively simpler Dither and Jitter variants described above, which merely perturb the control parameters. The JDE variant works as follows. First assign start values to F and CR, call these F_{init} and CR_{init}, respectively. Then before computing the new potential solution of a DE agent using Eq.(1), first decide what parameters F and CR to use in that formula. With probability $\tau_F \in [0, 1]$ draw a new random $F \sim U(F_l, F_l + F_u)$, otherwise reuse F from previously, where each DE agent retains its own F parameter. Similarly each agent retains its own CR parameter, for which a new random value $CR \sim U(CR_l, CR_l+CR_u)$ is picked with probability $\tau_{CR} \in [0, 1]$, and otherwise the old CR value for that agent is reused. Whichever F and CR values are being used in the computation of Eq.(1), they will survive to the next iteration, or be discarded along with the agent's new potential solution $\vec{y}$, according to fitness improvement.

As noted previously, there are generally two reasons to introduce adaptive parameters. First is the belief that it remedies the need for a user to manually select parameters that yield good performance on a problem at hand. Second is the belief that different choices of parameters are needed at different stages of optimization so as to balance exploration and exploitation of the search-space. Otherwise the parameters could just as well be held fixed during optimization. Both these beliefs will be studied thoroughly in this paper, but it should be noted at this point that the second reason to have parameter adaption is perhaps a paradox. Biasing future control parameters towards values that have been observed to work well until now seems to be contradictory to the need to have different parameters at different stages of an optimization run. If anything, the future control parameters should be chosen to be dissimilar to the parameters that have previously worked well.

3 Local Unimodal Sampling

We introduced Local Unimodal Sampling (LUS) in [26], so named because it was designed to deplete unimodal optima, although it has been found to work well for harder optimization problems. The LUS method is used here as the overlaying meta-optimizer for finding the control parameters of DE and its variants in an

offline manner. LUS is often able to locate the optimum in comparatively few iterations, which is required due to the computational time needed for each iteration in meta-optimization.

For the sampling done by the LUS method, the new potential solution $\vec{y} \in \mathbb{R}^n$ is chosen randomly from the neighbourhood of the current solution $\vec{x}$ by adding a random vector $\vec{a}$. When the LUS method is used for meta-optimization these represent different choices of DE control parameters, meaning that $\vec{x} = [NP, CR, F]$ for DE/rand/1/bin with NP being the number of agents in the DE population, and CR and F being the control parameters from Eq.(1). The potential new choice of control parameters $\vec{y}$ is hence found from the current choice $\vec{x}$ as follows:

$$\vec{y} = \vec{x} + \vec{a}$$

where the vector $\vec{a}$ is picked randomly and uniformly from the hypercube bounded by $\pm\vec{d}$, that is:

$$\vec{a} \sim U\left(-\vec{d}, \vec{d}\right)$$

with $\vec{d}$ being the current search-range, initially chosen as the full range of the search-space and decreased during an optimization run as described next. The search-space in meta-optimization constitutes the valid choices of control parameters, and will be detailed later.

When a sample fails to improve the fitness, the search-range $\vec{d}$ is decreased for all dimensions simultaneously by multiplying with a factor q for each failure to improve the fitness:

$$\vec{d} \leftarrow q \cdot \vec{d}$$

with the decrease factor q being defined as:

$$q = \sqrt[n/\beta]{1/2} = 2^{-\beta/n} \tag{3}$$

Here $0 < \beta < 1$ causes slower decrease of the search-range, and $\beta > 1$ causes more rapid decrease. Note that applying this n times yields a search-range reduction of $q^n = 1/2^\beta$, and for $\beta = 1$ this would mean a halving of the search-range for all dimensions. For the experiments in this paper, a value of $\beta = 1/3$ is used as it has been found to yield good results on a broad range of problems. The algorithm for the LUS optimization method is shown in figure 2.

4 Meta-Optimization

There are a number of parameters controlling the behaviour and efficacy of DE. These control parameters are usually found by manual experimentation, which is time-consuming and susceptible to human misconceptions of the inner-working of the optimization method. The problem of finding the best choice of control parameters for a given optimization method, is considered here as an optimization problem in its own right and is termed Meta-Optimization. In other words, the idea is to have an optimization method act as an overlaying meta-optimizer, trying to find the best performing control parameters for another optimization

method, which in turn is used to optimize one or more actual problems. The overall concept is depicted in figure 3.

It is important to understand the difference between adaptation and meta-optimization (or tuning) of control parameters. Adaptation of control parameters occurs in an online manner during optimization, and hence requires the adaptive mechanism to be embedded in the optimizer's own algorithm. This complicates the implementation and makes it difficult to use the adaptive schemes for other optimization methods. Conversely, in meta-optimization the control parameters are tuned in an offline manner between optimization runs, and means the optimizer's original algorithm can be reused without change, apart from a new choice of control parameters. It also means that meta-optimization can be readily used on many different optimization methods, without the need for specializing their algorithms. Already, at the conceptual level, meta-optimization presents clear advantages over parameter adaptation.

Meta-optimization is reported to have been used by Mercer and Sampson [27] for finding optimal parameter settings of a Genetic Algorithm (GA). Another early example of meta-optimizing discrete parameters of a GA is due to Grefenstette [10], and results with meta-optimizing GA parameters are also reported by Keane [28]. Meta-optimizing both the control parameters and the GA operators was studied by Bäck [11]. But meta-optimization is very time-consuming, and it was not until recently that computers had enough processing power to conduct more realistic experiments in meta-optimization; see for example the recent experiments by Meissner et al. [13]. Limiting the time usage was sought by Ridge and Kudenko [29] by first screening and ranking how the indvidual control parameters affect performance and then focusing on the ones of seemingly greatest impact, and they also allowed for meta-optimization with regard to the conflicting measures of fitness results achieved and computational time used [30] [31]. More statistically oriented approaches to meta-optimization have also been pursued by François and Lavergne [32], Czarn et al. [33], and by Nannen et al. [14] [34]. Birattari [12] used what is known as a racing approach, in which several choices of control parameters are maintained in a pool that is being depleted iteratively whenever statistical evidence suggests that a choice of control parameters is inferior to the others in the pool. The approach to meta-optimization used here and first introduced in [17], is preferred because it works well for real-valued control parameters, is both simple and fast, and yields good results.

4.1 Rating Optimizer Performance

The crux of automatically finding good control parameters for an optimization method, is to define an appropriate performance measure that can be made the subject of meta-optimization. The performance measure must reflect how the optimization method is ultimately to be used, but at the same time allow for efficient meta-optimization. A typical way of performing optimization with DE is to let it run for some predetermined and seemingly adequate number of iterations. This means the optimization performance of DE with a given choice

of control parameters can be rated in terms of the fitness that can be obtained within this number of iterations.

4.2 Tuning For Multiple Problems

Another important factor of using DE in practice is that its performance must generalize well to other optimization problems. A way of achieving this would be to tune the control parameters to perform well on multiple problems simultaneously. This means that meta-optimizing the DE parameters now becomes a multi-objective optimization problem, which introduces new challenges.

The difficulty with optimizing multiple objectives simultaneously is that the objectives may be conflicting or contradictory. So when the meta-optimizer improves the DE performance on one problem, it may worsen its performance on another. A number of optimization methods have been introduced to deal with multi-objective optimization problems in general, see e.g. [35] [36] [37] [38]. But these techniques are not suited as the overlaid meta-optimizer because they are more complicated than what is desired.

Instead, one of the simplest approaches to multi-objective optimization is to combine the individual objectives into a single fitness measure by weighting their sum. It is not clear who is the original author of this technique, but it is mentioned in [39] as having been popular for many years. The weighted-sum approach also works well for meta-optimization, as it is sought to improve the overall performance. So instead of tuning the DE parameters to work well on just a single problem, the performance evaluation now consists of using DE with a given choice of control parameters on two or more actual problems, and adding the results to create the overall performance measure used to guide the meta-optimizer. This performance measure will also be referred to as the meta-fitness. This performance measure also has the advantage of being supported directly by the time-saving technique described next.

4.3 Preemptive Fitness Evaluation

Since DE is stochastic by nature, it will be likely that it gives a different result for each optimization run, which may interfere with the meta-optimizer's ability to accurately determine good control parameters. A simple way of lessening this stochastic noise is to perform a number of optimization runs in succession, and use the average of the fitnesses obtained to guide the meta-optimization. But repeating optimization runs also causes the computational time to grow undesirably. For ordinary optimization of noisy fitness functions, it has been proposed in the literature to schedule the number of repeats for when they are necessary to distinguish fitness values, see for example [40] [41].

A simpler way of saving computational time when repeating fitness evaluations in general, is to preemptively abort a fitness evaluation once the fitness becomes worse than that needed for the optimizer to accept the new candidate solution, and the fitness is known not to improve for the rest of the evaluation. This technique is termed Preemptive Fitness Evaluation and has been used by

researchers for decades, although its original author is difficult to establish as the technique is seldom mentioned in the literature.

Greedy optimization methods are generally compatible with preemptive fitness evaluation because they only move their optimizing agents in the case of strict improvement to the fitness. Take for example the LUS method from above, which works by choosing a random set of parameters in the vicinity of the current parameters, and accepting them only in the case of improvement to the meta-fitness. Therefore the meta-fitness evaluation of the new DE control parameters can be aborted as soon as it becomes known that it is actually worse than that of the current parameters; and provided it will not improve later during computation of the meta-fitness. This is generally ensured in meta-optimization by only using problems that have non-negative fitnesses (or by offsetting their fitnesses to become non-negative by adding an appropriate constant value), so the best performance of DE when optimizing these problems is also non-negative.

Preemptive fitness evaluation is generally applicable to fitness functions that are iteratively computed, and where the overhead of monitoring the progress and consequently aborting the fitness evaluation, does not cancel out the gain in computational time that arises from only evaluating a part of the fitness function. Since each meta-fitness evaluation is computationally expensive, time-savings can be expected there. Indeed, depending on the experimental settings, the time-saving resulting from the use of preemptive fitness evaluation in meta-optimization ranges from approximately 50% to 85%.

4.4 Meta-Optimization Algorithm

The overall algorithm for performing meta-optimization is shown in figure 4, and the algorithm for computing the meta-fitness measure is shown in figure 5. The preemptive fitness limit is denoted L and is the limit beyond which the meta-fitness evaluation can be aborted. This limit is passed as an argument by the overlaying LUS meta-optimizer, so that L is the meta-fitness of the currently best known choice of DE control parameters. Note that the performance on the optimization problems are weighted equally important in this algorithm. This will be discussed later.

5 Benchmark Problems

To test the optimization performance of DE variants using different choices of control parameters, a suite of twelve benchmark problems is used. It is necessary to use benchmark problems in this study because some of the brute-force experiments in parameter tuning, which compute the optimization performance of DE using all possible combinations for its control parameters (broken into discrete intervals), require a very large number of fitness evaluations, and the computational cost of using real-world problems would have been prohibitive.

The benchmark problems are widely used in the literature and taken from a larger suite collected by Yao et al. [42]. The reason these particular problems have been chosen is that they generalize to higher dimensionalities, and they all have non-negative fitness values (their global optimum have fitness zero). Recall that the latter is required to use preemptive fitness evaluation in the meta-optimization algorithm. The benchmark problems vary from uni-modal (one local optimum which is hence also the global optimum, e.g. the Sphere problem), to multi-modal (several local optima where the optimizer may get stuck, e.g. the Ackley problem), and from separable (the dimensions of the search-space are independent of each other in their influence on the fitness, e.g. the Sphere problem), to non-separable (the dimensions of the search-space may be intricately dependent on each other in their influence on the overall fitness, e.g. the Rosenbrock problem). The benchmark problems are shown in table 1 and their initialization and search-space boundaries are shown in table 2. The asymmetrical initialization ranges are chosen to further increase the difficulty of optimizing these benchmark problems.

6 Experimental Results

This section presents several studies to uncover whether there is an advantage to using control parameter adaptation in different situations. But first the experimental settings are described.

6.1 Optimization Settings

To lessen the effect of stochastic variation a number of optimization runs must be performed and their results averaged, so as to give a more truthful measure of the performance that can be expected from a single optimization run. Here 50 optimization runs of each benchmark problem are used, which have been found to be sufficient. For each of these optimization runs a number of iterations are executed, equal to 200 times the dimensionality of the benchmark problem in question. In these experiments all benchmark problems will have $n = 30$ dimensions, meaning that each optimization run will consist of $200 \cdot n = 200 \cdot 30 = 6000$ fitness evaluations. This is significantly less iterations than most researchers report results for in the literature (see e.g. [3] [6] [9] [25] which use up to 2000000 fitness evaluations per run), and although there are real-world optimization problems for which long optimization runs can be used (see [3] for some examples), there are also many real-world problems which require substantial computational time for each fitness evaluation, making such long optimization runs impossible. For example in Computational Fluid Dynamics (CFD) where a physical shape such as an aircraft wing or a ship-hull must be optimized, see e.g. Eres et al. [43]. Indeed, in some of our previous work only $20 \cdot n$ fitness evaluations could be used, due to the computational time required for the problem considered [17]. Although this study uses cheaply computed benchmark problems, it also contains some experiments in brute-force tuning

of DE control parameters, in which the DE performance for an entire grid of parameter combinations are computed, so that using very long DE runs would have caused those experiments to take about a year of computation time instead of a day. The choice of $200 \cdot n$ fitness evaluations per optimization run is therefore considered a good compromise, also giving a better chance of the results being applicable to a broader range of real-world problems. Should a practitioner need to use longer optimization runs, then the meta-optimization experiments can be repeated using those settings. It is expected that the findings concerning parameter tuning versus adaptation still hold true for very long optimization runs, although the actual DE parameters most suitable for such long runs, are likely to be somewhat different from the ones found under the experimental settings used here.

6.2 Statistical Significance

Since the benchmark experiments are stochastic their results ought to be compared in terms of statistical significance. The test used here is known as the one-tailed t-test for two independent samples, with a rejection threshold of $p > 0.05$, see for example Crow et al. [44]. Using this test, the DE variant with the best average result for a given benchmark problem is compared to the results of the other DE variants, and the best results are marked as statistically significant only if all the t-tests agree with this.

6.3 Meta-Optimization Settings

The experimental settings used for meta-optimization with regard to benchmark problems are as follows. Six meta-optimization runs are conducted with the LUS method as the overlaying meta-optimizer, trying to find the best performing DE control parameters. Each meta-optimization run has a number of iterations which equals 20 times the number of control parameters to be meta-optimized. So for DE/rand/1/bin which has three control parameters, 60 iterations will be performed for each meta-optimization run of the overlaying LUS method. For the JDE variant which has 9 control parameters, 180 iterations will be performed in each run of the overlaid meta-optimizer. In each of these iterations, the DE variant is in turn made to perform optimization runs on benchmark problems, to assess its performance with a given choice of control parameters, using the general optimization settings described in section 6.1 above. The performance on different benchmark problems are weighted equally, and since these problems have widely varying fitness ranges, it might have been desirable to weigh their influence on the tuning process differently. This has not been found to be a crucial issue, however, and further investigation into this is left as a topic for future research. These meta-optimization settings were found to be adequate in discovering control parameters that cause DE to perform well. The boundaries for the parameter search-spaces in all the meta-optimization experiments are shown in table 3. Note that we allow for smaller population sizes NP than usual [2] [3] [4], in part because these have been found to be adequate, but also

because the smaller a parameter search-space is, the faster the discovery of good parameter choices using meta-optimization.

For the JDE boundaries it should be noted, that if a given combination of parameters has $CR_l + CR_u > 1$ then CR_u is adjusted prior to evaluating the performance of JDE with these parameters, so as to ensure the sum is no more than one: $CR_u \leftarrow 1 - CR_l$, because the crossover probability CR must be between zero and one, and is picked as: $CR \sim U(CR_l, CR_l + CR_u)$.

6.4 Hand-Tuned Parameters

The purpose of this first study is to establish how the DE variants with default choices of parameters fare against each other. The control parameters in table 4 are standard in the literature [1] [2] [22] [9], and are presumably hand-tuned. Table 5 show the results of optimizing the benchmark problems using these parameters. It can be seen that JDE/rand/1/bin performs best when using these default hand-tuned parameters. The worst performing DE variants appear to be the ones using Dither and Jitter to perturb the F parameter. Recall however, that these benchmark problems have optimal fitness values of zero, and none of the results using hand-tuned parameters approach this, when given more realistic optimization run-lengths of 6000 fitness evaluations.

One of the original authors of DE has suggested using DE/rand/1/bin with $F = 0.85$, and $F_{mid} = 0.85$ for the Jitter variant [21], but this yielded slightly worse performance on these problems and optimization run-lengths.

6.5 Average Performance Capability

The purpose of the study in this section is to unveil the core performance capabilities of the DE variants, when their control parameters have all been properly tuned. The cost of doing such tuning will be ignored for now. In the experiments in this section, the DE parameters are tuned to perform well on all twelve benchmark problems on average, and the parameters can be seen in table 6. These meta-optimized parameters are interesting compared to the hand-tuned parameters from table 4. First is the favouring of small population sizes NP, which are actually only a fractional part of the dimensionality of the search-space, and is therefore in stark contrast to the literature which generally recommends $10 \cdot n$ for n-dimensional search-spaces, see e.g. [1] [2] [4] [22], although small populations have been reported to work well on certain problems [3]. The fact that smaller populations are preferred may at first sight not appear so strange, however, because only a modest number of optimization iterations are allowed, meaning that smaller populations will cause each agent to receive more iterations to refine its candidate solution. But later experiments show that in some cases large populations are favoured for similar optimization run-lengths, so it seems virtually impossible to make such general predictions. The second notable difference between hand-tuned and meta-optimized parameters is the crossover probability CR which is now held very low, while the hand-tuned parameters used high CR values. Third is the differential weight F, which is perturbed

much more for the Dither and Jitter variants, where the latter has a notably large perturbation range, F_{range}, which is otherwise recommended to be very low [22] [23]. The JDE parameters are generally difficult to interpret, in part because there are so many of them, but also because they seem even more intricately dependent on each other. It can however be noted that the crossover probability CR appears to be higher valued than for the other DE variants, and the differential weight F appears to be less perturbed than for the Dither and Jitter variants, albeit with roughly the same midpoint as the Jitter variant: $F_{mid} = (F_l + F_u)/2 \simeq 0.52$.

The results of using the DE variants with the meta-optimized parameters from table 6 are shown in table 7. The first thing to note, is how these relate to the results obtained using DE variants with hand-tuned parameters, as previously shown in table 5. As can be seen the performance of all DE variants improves greatly when their parameters are properly tuned. While there is no guarantee that these meta-optimized parameters are the very best that exist, they do incur great enough performance improvement to establish confidence in our meta-optimization technique being able to find at least near-optimal choices of control parameters.

Concerning the performance of DE when using fixed versus adaptive control parameters, the experimental results show that there appears to be an advantage of the Dither variant on four of the twelve benchmark problems, but this should only be taken as a rough indication as the statistical tests make assumptions on the distribution of results which may or may not hold. Also of interest is that the JDE variant sometimes achieves worse results than even the basic DE with its parameters held fixed during optimization. The reason JDE is claimed to have superiority over other DE variants [9] [25], seems therefore to lie in the very long optimization runs that are allowed in those studies. But as can be seen here, JDE appears to perform worse when allowed a more realistic number of iterations per optimization run. The conclusion of this study must be, that no DE variant appears to hold a general and consistent advantage over the others when their parameters are tuned for all the benchmark problems.

6.6 Specialization Ability

The purpose of the study in this section is to determine to what extent the DE variants can be tuned to perform well on a single problem at a time. In practice one would only do this if encountering premature convergence, because the additional optimization iterations used in tuning could otherwise just have been used to extend the ordinary optimization run. Furthermore, this study will tell us something about the versatility of the DE variants in terms of how much they can be specialized to a particular kind of optimization problem.

In the experiments above, the four DE variants were tuned for all benchmark problems simultaneously, and all DE variants alike were then found to have trouble optimizing particularly the Schwefel1-2 problem, but also the Rosenbrock problem. Tuning the parameters of the DE variants to just the Schwefel1-2 problem yields the parameters in table 8. It is interesting that the population

size NP has grown considerably for all these DE variants, when they must perform well on the Schwefel1-2 problem. Yet they are still not allowed more fitness evaluations during optimization, meaning that each agent now receives fewer optimization iterations with these larger populations. Another interesting thing to note is the crossover probability CR which for the Dither, Jitter, and JDE variants are kept close to their upper boundary of 1, where as the parameters tuned for all benchmark problems had CR values closer to their opposite boundary of 0, espcially for the Dither and Jitter variants. The differential weight F is perturbed heavily in the Dither variant (which is perturbation on a per-vector basis), while the Jitter variant (which is perturbation on a per-vector-element basis) uses only light perturbation, apparently corresponding to the advise given in the literature [3] [22] [23]. Tuning the parameters for the Rosenbrock problem yields the parameters in table 9. Here the population sizes are again very low, similar to the experiments in section 6.5, where the parameters were tuned to perform well on all benchmark problems on average. The remaining parameters for the Dither and Jitter variants also show similar tendencies to those in section 6.5, although with even heavier Jitter perturbation which is again contrary to the rule-of-thumb advise given in [3] [23]. The crossover probability for DE/rand/1/bin is also notably higher than it was when tuned for all benchmark problems simultaneously. The parameters for JDE are, as usual, difficult to interpret because their intrinsic meaning eludes human analysis. It can however be noted that they appear to compare roughly to the parameters of the simpler DE variants, namely in terms of a similar population size, and a similar crossover probability (ignoring the initial value CR_{init} which will probably not be held for long, due to the high probability τ_{CR} of perturbing it). The perturbed differential weight for JDE however, is centred around $F_{mid} = (F_l + F_u)/2 \simeq 0.87$, which is somewhat higher than for the other DE variants.

Using these parameters on the Schwefel1-2 and Rosenbrock problems, respectively, gives the results in table 10. Comparing these to the results in table 7 for parameters tuned on all benchmark problems, shows consistent improvement on the Schwefel1-2 problem. The Rosenbrock results are not so clear however. For instance, the basic DE/rand/1/bin and the Dither variant actually perform worse when compared to their results in table 7, where their control parameters were tuned for all benchmark problems. The reason for this might be found in the comparatively high standard deviations, which indicate significant performance volatility on that problem. The same volatility exists for the other DE variants as well, although they seem more capable of improving their performance when their parameters have been specifically tuned for this problem. In summary, the JDE variant performs best by far on the Schwefel1-2 problem, while the Dither and Jitter variants seem to perform best on the Rosenbrock problem, although the results are probably not statistically significant. Comparing the specialization ability of the Dither, Jitter, and JDE variants over using DE with fixed parameters, the DE variants with perturbed and adaptive parameters do appear to hold a slight advantage, especially on the Schwefel1-2 problem. This is perhaps ironic because these DE variants were designed specif-

ically to alleviate the need for tuning their parameters to specific problems in order for them to perform well. Whether these results hold true for a wider set of optimization problems, however, remains an open question.

6.7 Generalization Ability

This section studies the generalization ability of the four DE variants. That is, the ability to perform well on optimization problems for which the DE parameters were not specifically tuned. This is how most practitioners would perhaps prefer to use an optimizer – they would not have to retune the parameters for every optimization problem they encounter. It is therefore important to determine which DE variant is able to generalize best, especially because three of the variants were designed to generalize better by adjusting their parameters during optimization.

To test for generalization ability the DE parameters are tuned for just three out of the twelve benchmark problems, and their performance on all problems is then assessed, which is a simple form of cross-validation [45]. The problems used in the parameter tuning are: The Sphere problem because it is very simple and an optimizer ought to at least perform well on this problem, the Rastrigin problem because it is complex and the DE variants had varying success in optimizing it in the experiments above, and the Rosenbrock problem because it has proven to be one of the hardest to optimize. The parameters tuned to perform well on these three benchmark problems are found in table 11. Comparing these parameters to the ones in table 6 which were tuned for all twelve benchmark problems, reveals some interesting similarities. First that the population size NP is kept at 7 or 8, which is actually a little lower than they were in table 6, but with the same trend of having small populations. Second is the relationship between the crossover probability CR and differential weight F (also F_{mid} and F_{range}), while not identical to the relationships in table 6 they do show similar tendencies, namely that CR is low and F is high. The JDE parameters have seemingly less perturbation of CR and more of F, but also with somewhat similar tendencies to the parameters in table 6 where they were tuned for all benchmark problems and not just the three problems used here. These findings would suggest that the Sphere, Rastrigin, and Rosenbrock problems are possibly representative of the benchmark suite in terms of DE generalization ability, and hence that the DE parameters tuned for just these three problems might indeed work well on the other benchmark problems.

The results on all benchmark problems, using the DE variants with their parameters tuned for just three of the benchmark problems, are shown in table 12. While the Dither variant appears to hold a statistically significant advantage on three out of the twelve problems, it actually appears to perform worse on some of the other problems, notably Schwefel1-2 and Schwefel2-21. From this there does not appear to be any consistency in which one DE variant can be said to be superior over the others, and all the DE variants therefore appear to generalize roughly equally well, albeit slightly differently. This is a remarkable finding because the Dither, Jitter, and JDE variants were specifically designed to alleviate

the need for parameter tuning by perturbing or adapting their parameters, and thus believed to generalize well to problems for which their parameters were not specifically tuned. But the results here strongly suggest that DE variants with perturbed or adaptive parameters do not yield any such generalization advantage over using fixed parameters.

6.8 Rate of Convergence

The purpose of the study in this section is to assess whether some DE variants have an advantage in terms of the speed with which they approach the optimum. This is also known as their rate of convergence, and is important for optimization problems where sub-optimal solutions may be used before optimization has completely finished.

The parameters from table 11, which were tuned for just three benchmark problems, will be used here to show the rate of convergence for one problem the parameters were tuned for, and one problem for which the parameters were not tuned. The problems chosen are the Sphere and Griewank problems, because the four DE variants achieve roughly equivalent final results on those two problems as can be seen from table 12.

Figures 6 and 7 show the average fitness traces. Note that these are new experiments and the results might differ slightly from those in table 12. For the Sphere problem the DE variants have comparable rates of convergence, with exception of the Jitter variant which starts to perform worse than the others roughly in the middle of the optimization run, but slightly outperforms the other DE variants towards the end of the run. Other tests with the DE Jitter variant have been executed, and the erratic element midways through optimization persists in some form, but the end results vary somewhat in that it sometimes achieves better and sometimes worse end results than the other DE variants. The rate of convergence on the Griewank problem is perhaps even more interesting. Not only do the four DE variants have comparable performance, but the convergence rate is curiously curved yet the DE variants have similar curvatures. This suggests the underlying dynamic behaviour of these four DE variants might be quite similar on the Griewank problem. Although these were just two out of twelve benchmark problems, and one should be vary of concluding too much from only two examples, it does appear that perturbing and adaptation of DE parameters as done by the Dither, Jitter, and JDE variants, does not yield any general improvement in terms of the rate with which an optimum is approached.

6.9 Ease of Tuning

The purpose of the study in this section is to show which DE variant is the easiest to tune and how long it actually takes. Figure 8 shows the meta-optimization progress for all four DE variants, taken from the experiments in section 6.5. The meta-fitness measure is the internal measure used for guiding the meta-optimization progress as described in section 4. To normalize these measures, one would have to divide them by the number of repeated optimization runs,

as well as by the total number of benchmark problems used in each meta-fitness evaluation. In other words, one would have to divide these measures by $50 \cdot 12 = 600$ to obtain normalized measures that would correspond to the sum of the average fitness values in table 7. But such normalization is not required for just comparing the progress traces with each other.

Roughly after iteration 18 in figure 8 the JDE variant appears to be the hardest to tune, and it takes an additional 35 iterations to find parameters for the JDE variant that perform on par with the other DE variants. Note that the meta-fitness axis is logscaled, so what appears at first glance to be a minor difference in performance is in fact an exponential difference in performance. It appears from this chart that the basic DE/rand/1/bin and its Dither variant have the most stable and smooth performance improvements throughout their tuning, while the Jitter variant is perhaps a little easier to tune after iteration 17-18. These results are probably as expected, where the JDE is generally a little harder to meta-optimize because it has many more parameters that need tuning. That the extra parameters of the JDE do not incur an even greater penalty in tuning them, perhaps only speaks to the credit of the meta-optimization technique employed.

Table 13 shows the actual time usage for meta-optimizing the parameters of these four DE variants. The table shows the time usage for the experiments in section 6.5 which used all twelve benchmark problems, and for the experiments in section 6.7 which used only three benchmark problems. The experiments were conducted on an Intel Pentium M 1.5 GHz laptop computer using an implementation in the ANSI C programming language. The basic DE/rand/1/bin takes around 18 minutes to tune for the twelve benchmark problems, while it is estimated through extrapolation to take 54 minutes without the use of preemptive fitness evaluation, which means a saving of 2/3 of the computational time. The JDE variant takes almost twice the amount of time to meta-optimize, or approximately 35 minutes. This is also interesting because JDE actually has three times the number of parameters that need be tuned and the meta-optimizer hence performs three times the number of iterations (note how the JDE trace in figure 8 continues long after the others). But again, due to the use of preemptive fitness evaluation, the time usage is not tripled as one would have expected, but merely doubled. This again speaks to the credit of the meta-optimization technique employed. Table 13 also shows the time usage for tuning the parameter to perform well on just three benchmark problems instead of all twelve. While the same relational tendencies amongst the DE variants prevail, it is interesting to see that tuning the basic DE/rand/1/bin takes roughly 4 minutes. In conclusion, one can clearly see from table 13, that the basic DE/rand/1/bin is the fastest of these DE variants to be tuned, in terms of wall-clock time usage. We also feel certain that no researcher has ever spent so little time tuning parameters by hand, nor achieved optimization results from their tuning on par with those achieved here.

6.10 Parameter Study

The study in this section shows how different parameter combinations relate to an optimizer's performance. The parameter-space is traversed at even intervals to create a mesh, and the performance of the optimizer is computed at each mesh-point according to the definition of meta-fitness in figure 5 and the settings from section 6.3.

However, as the number of parameters goes up, the number of mesh points will have to be increased exponentially to allow an equally fine coverage of the parameter-space. This is in essence the Curse of Dimensionality, only for the parameter-space, a name coined by Bellman [46, preface p. ix]. The depiction of the mesh is also increasingly difficult when the optimizer has more than 2 control parameters. For both these reasons, the DE/rand/1/bin variant is being used in this study as it has the fewest parameters, namely only 3 control parameters. A total of 10648 parameter combinations were computed, distributed evenly over the parameter-space, meaning that each parameter is allowed to take on 22 different values, which is considered a sufficiently fine mesh.

Figure 9 shows how the performance of DE/rand/1/bin relates to the choice of population size NP, when also varying the other parameters. As can be seen, the performance peaks at $NP \simeq 10$ and worsens with increased population size. This finding is similar to that of section 6.5 where the parameters were meta-optimized. Figure 10 shows how the performance relates to the choice of crossover probability CR, which appears to be more lenient in its impact on the performance. In particular, any value $CR \in [0, 0.7]$ appears to be an acceptable choice, although there appears to be a slight advantage to choices on the lower side of $CR \simeq 0.1$, which also matches the findings in section 6.5. Figure 11 shows how the performance relates to the choice of differential weight F, which appears to yield best performance at $F \simeq 0.65$, with smaller values of F causing much worse performance, and larger values causing gradually worse performance. This also matches the meta-optimized parameters from section 6.5.

Since this brute-force approach to parameter tuning did locate parameter combinations which made DE/rand/1/bin perform well on the benchmark suite, one could ask if it would be possible to use such simple means of parameter tuning in general. The brute-force approach is not generally viable however, because it took almost 26 hours to compute the performance measures for all these parameter combinations, whereas this particular DE variant could be meta-optimized in roughly 18 minutes (see table 13), which is approximately 1% of the computational time of the brute-force approach. Add to this the Curse of Dimensionality which would incur an exponential increase in time-usage for optimizers with more parameters. Take for instance the JDE variant which has 9 parameters instead of just 3, for which it would require approximately 333743 years to compute a similarly fine-grained mesh, having a total of $22^9 \simeq 1.21 \cdot 10^{12}$ parameter combinations in the mesh. Yet the meta-optimizer from section 4 could tune the JDE parameters in less than 35 minutes, which is a miniscule fraction of the time required by the brute-force approach.

6.11 Tuning Process

This section depicts the process of parameter tuning. It was shown in section 6.10 that the performance of DE/rand/1/bin was less influenced by the CR parameter, and this section will therefore focus on the NP and F parameters of that DE variant. Figure 12 shows the meta-optimization process from the experiments conducted in section 6.5. The lines show how the LUS meta-optimizer replaces one choice of DE parameters with another if it improves performance. The dots show parameter combinations which were contemplated by the LUS meta-optimizer, but were discarded because they yielded worse performance. Although there are many failed moves compared to the number of successful moves taken by the LUS meta-optimizer, it should be noted that due to the use of preemptive fitness evaluation the computations of these worsened performance measures are aborted preemptively. Hence, it does not represent as great a computational expense as appears. Figure 12 generally shows the tuning effort is becoming centred around the best performing parameter combinations, whose good ranges were previously deduced from figures 9 and 11. This confirms visually that the employed meta-optimization approach works well.

6.12 Temporal Parameters

This section studies whether there is any need for changing parameters during an optimization run, or if they can just as well be held fixed for the entire duration. The DE/rand/1/bin variant is used as the basis of this experiment, because it has all its parameters fixed during optimization. The population size NP will remain fixed, because it would introduce semantic questions on which DE agents should be kept and which discarded when the population size was to decrease, and what should be done to new DE agents when the population size should increase. But changing parameters CR and F, say, halfway during an optimization run is straightforward. Making such parameters temporal is done by replacing F with F_1 and F_2, and CR with CR_1 and CR_2, where the parameters with index 1 are used in the first half of the optimization run, and the parameters with index 2 are used in the last half of the run. It is possible to make an arbitrarily long sequence of parameters this way, but merely halving the optimization run will have to suffice for this study.

The brute-force approach from section 6.10 is also used here, for discovering how the changing of parameters relate to optimization performance. A performance mesh is computed having a resolution of 7 choices for each parameter, hence consisting of $7^5 = 16807$ mesh-points total, as there are now 5 control parameters. This performance mesh took almost 41 hours to compute, and has only one third of the resolution for each parameter compared to the mesh computed in section 6.10. Although a mesh is needed for the study here, it is again evident that it is not a suitable way of performing parameter tuning in general.

Using the data from the performance mesh, figure 13 shows how the difference $CR_1 - CR_2$ relates to performance, thus showing whether there is an advantage to having dissimilar crossover probabilities for each half of an optimization

run. This is indeed the case, as figure 13 shows a performance advantage, when the crossover probability for the first half of the optimization run is somewhat lower than for the last half of the run (provided the other control parameters are also tuned properly). The reverse is true for the differential weights F_1 and F_2, whose relation to performance is shown in figure 14, from which it can be seen that a performance advantage exists, when the differential weight for the first half of an optimization run is somewhat higher than for the last half of the run.

It could be interesting to see how big this performance advantage really is. Since the performance mesh computed above is rather coarse, it will be better to employ meta-optimization to find the temporal parameters that causes DE to perform best. It should also be noted that even though we have just introduced another set of DE parameters for the latter half of its optimization run, there is no telling how this abrupt change of parameters will affect the dynamic behaviour of the DE agents. One could then suggest using linearly or otherwise smoothly changing parameters during an optimization run, but this will likely have an altogether different effect on the dynamic behaviour of the DE agents. An advantage of using the meta-optimization approach from section 4, is that we need not understand the dynamic behaviour of the optimizer whose parameters are being tuned. The temporal parameters found thus, are:

$$NP = 9$$

$$CR_1 = 0.040135,\ CR_2 = 0.576005$$

$$F_1 = 0.955493,\ F_2 = 0.320264$$

Using these temporal parameters when optimizing the benchmark problems results in table 14, which for comparison has reprinted the results from table 7 of DE/rand/1/bin with fixed parameters. Although this is arguably a somewhat coarse experiment, the results do perhaps indicate that there may not exist any general advantage to using parameters that are changed somehow during optimization, regardless of whether this parameter changing is done randomly, adaptively, or at certain intervals according to known good values.

6.13 The Joker — DE/best/1/bin/simple

Although the general aim of this paper is to show whether one should use parameter tuning or adaptation, or both, and not to make an exhaustive search for the universally best DE variant, the following study will further strengthen the argument that one must always properly tune the parameters of an optimization method before drawing conclusions about its performance. We have thus far used the DE/*rand*/1/bin family of optimizers due to its popularity. The DE/*best*/1/bin variant on the other hand, has been long out of favour with researchers and practitioners because it is believed to have inferior performance with tendencies for premature convergence [3] [21]. The DE/best/1/bin replaces

Eq.(1) with the following:

$$y_i = \begin{cases} g_i + F \cdot (a_i - b_i) & , r_i < CR \vee i = R \\ x_i & , \text{else} \end{cases}$$

where $\vec{g}$ is the population's best known solution until now. In the original version of this, the agents $\vec{a}$ and $\vec{b}$ are chosen to be different not only from each other, but also from the agent currently being processed, but it simplifies the implementation a good deal if they need not be distinct from the agent $\vec{x}$ currently being updated. In particular, first the index r_a for agent $\vec{a}$ is chosen randomly from $\{1, \cdots, NP\}$, and then the index r_b for the other agent is determined by:

$$r_b = \begin{cases} r_a + r'_b & , r_a + r'_b \leq NP \\ r_a + r'_b - NP & , \text{else} \end{cases}$$

where $r'_b \in \{1, \cdots, NP - 1\}$ is picked randomly. This DE variant may be called DE/best/1/bin/simple, nicknamed The Joker. Its parameters meta-optimized to perform well on all twelve benchmark problems are:

$$NP = 172,\ CR = 0.965609,\ F = 0.361520 \tag{4}$$

Comparing these parameters to the ones tuned for the DE/rand/1/bin variants in table 6, the first striking thing is the big change in population size from $NP \simeq 10$ to $NP = 172$. This clearly shows that one cannot make generalizations about which population sizes should be used merely on whether a small or large number of fitness evaluations are allowed during optimization. There is in fact an intricate and mysterious relationship between DE algorithm, its control parameters, the optimization problem at hand, optimization run-lengths allowed, and the performance achieved. The second notable thing with these tuned parameters is the almost reversal of magnitude in parameters CR and F. Whether these are general tendencies for the DE/best/1/bin versus the DE/rand/1/bin-varieties is left as a topic for future research.

The results of using these tuned parameters when optimizing the benchmark functions are shown in table 15. Comparing these results to the ones for the tuned DE/rand/1/bin variants in table 7 shows the DE/best/1/bin/simple variant to perform much better on the Schwefel1-2 and Rosenbrock problems, yet having equally worse performance on e.g. the Rastrigin and Penalized2 problems. Figure 15 depicts the progress of tuning the parameters of DE/rand/1/bin and DE/best/1/bin/simple, clearly showing the latter to be the easier to tune. These findings will not be studied further in this paper, as it is not our aim here to determine which DE variant is universally best. However, it does strengthen the point to be made, that one should always properly tune the parameters of an optimization method to get a clear indication of what performance the optimizer is really capable of.

7 Discussion

This section is a deeper discussion of the experimental results and their implication for the research field in general.

7.1 Hand-Tuning Versus Meta-Optimization

It is obvious from the experimental results that automated tuning of control parameters can greatly improve the performance of an optimizer over using just hand-tuned parameters. However, we still find reluctance amongst researchers to start using meta-optimization, in part because they object to the large number of additional optimization runs that must be executed in the tuning process. While it is true that one could instead use this computational time to merely extend the ordinary optimization runs, there are strong points to be made in favour of meta-optimization.

The control parameters must be tuned at least once, whether by hand or otherwise. This remains an inescapable fact, for no researcher could publish a new optimization method without providing good common choices of control parameters. Similarly no practitioner could solve an optimization problem without knowing these and would probably also try a number of parameter combinations for himself to see if they improve performance for the problem at hand. But tuning control parameters by hand is a laborious task, especially as the number of parameters increases, which causes an exponential increase in the number of possible parameter combinations. And because the intrinsic meaning of the control parameters and their influence on optimization performance largely remains a mystery, it is virtually impossible for a human to deduce which parameters will cause the optimizer to perform well, reducing such hand-tuning to little more than trial-and-error guesswork. In fact, promoting the use of hand-tuning over an automated approach, would be similar to promoting hand-optimization of an actual problem, which is of course just plain silly.

Furthermore, every practitioner must be familiar with premature convergence, in which the optimizer stagnates at some point and cannot find improved solutions in the search-space, seemingly no matter how long the optimizer is allowed to run. At such a time the practitioner is left with two choices: Try different control parameters, or try another optimizer altogether. But here meta-optimization may also be of assistance because it allows for automatic discovery of control parameters that alleviate premature convergence or at least defers it. It is correct that there is an expense involved in the tuning process, but it may be the only way of ever finding the global optimum for the problem at hand.

When meta-optimizing control parameters for use on a real-world problem, the additional optimization runs required might be prohibitive. This was indeed the case with our study in [17] where parameter tuning could only be done for the smallest problems due to the computational cost involved. It is therefore important that one uses an optimizer whose performance generalizes to similar problems, and DE appears to do this well. A suggestion is also to first tune the parameters using benchmark problems that are fast to compute, and then use the control parameters thus discovered on the real-world problem at hand. From our experience it seems important that one uses several benchmark problems in this tuning, and that the dimensionality and optimization run-lengths are similar to the real-world problem in question. But this requires further investigation

and is suggested as a topic for future research.

7.2 Parameter Tuning Versus Adaptation

Since adaptive parameters just introduce other, often additional, control parameters the above discussion on parameter tuning holds true for adaptive parameters as well. Parameter tuning should therefore always be employed at least once. But do adaptive parameters have an advantage over parameters that are held fixed during optimization? The experimental results suggest that there is no general advantage to having so-called adaptive parameters as they do not greatly alter the underlying dynamic behaviour of the optimizer. Rather they just change it slightly giving the optimizer a small advantage on some problems, but a disadvantage on other problems. The real performance improvement seems to stem from the changing of core aspects of the optimization algorithm itself. This brings about another interesting topic, namely whether one should aim for simple or complex optimization methods. The more complex an optimization method becomes, the harder it is to describe and implement. So if there is no general advantage to more complex optimization methods, then one should clearly strive to develop ever simpler optimizers.

The literature contains many examples of complex variants of optimizers, which were originally intended to be simple. The governing principle of Self-Organization and Emergence as it occurs in nature is that simple individuals cooperate and complex collective behaviour emerges. There is also increased difficulty in analytically justifying more complex optimizers because their correctness cannot be proven as one would prove, say, sorting algorithms correct by the use of invariants. Optimization methods which work by direct-search (that is, they do not use gradients to guide their search) can currently only be proven empirically. Some researchers have attempted making mathematical analyses, but they are usually oversimplified and bear no real significance. For instance, a convergence proof showing the global optimum is approached when the number of iterations approaches infinity, has no real worth because the exact same thing can be proven for completely random search. A good example of an optimizer which is probably more complex than it should be, is the JDE variant studied here. In their original description of JDE [9], the authors encode each agent's F and CR values in that agent's solution vector $\vec{x}$ by extending the vector accordingly. But since these F and CR values never themselves undergo the crossover and mutation computation in Eq.(1), this encoding is not only unnecessary it makes the presentation hard to comprehend. It would have been difficult to implement JDE from the description in [9] had its authors not supplied us with the actual source-code. Another good example of an optimizer that is made more complex than it probably should be is the Stochastic Genetic Algorithm (StGA) by Tu and Lu [47] which extends and tries to improve upon the simple Genetic Algorithm of Holland [20]. While StGA did show improvement over other optimizers on a suite of benchmark problems, it eventually turned out the StGA implementation had an error that made it strongly biased towards finding the global optima of the benchmark problems considered [48]. Again, these issues

arise from the fact that a direct-search optimizer cannot yet be proven correct by analytical means, and the more complex an optimization method becomes the harder it gets to describe the method clearly and hence make a correct implementation. We therefore propose a new focus for researchers. That is, to discover the core components that make direct-search optimizers work, and we believe our meta-optimization approach will prove valuable in that process because it is an easy and fast way of tuning the control parameters of an optimizer, to make it perform its best.

7.3 Generalization & Specialization Ability

It has been established that one should always properly tune the control parameters of an optimization method, in order to thoroughly assess the performance it is capable of, and that the best way to do parameter tuning is by an automated approach such as the one presented in section 4. But a question arises on how researchers should actually test their new optimizers in practice, to reveal and compare the core performance capability against that of other optimizers. Firstly we suggest using either a suite of benchmark functions with realistic optimization run-lengths, so there is a better chance of the results generalizing to real-world problems, or one could use actual real-world problems as case studies. Secondly we suggest tuning the optimizer's control parameters to unveil the optimizer's generalization and specialization abilities. The generalization ability is documented by tuning the control parameters to perform well on a few optimization problems and then by using these parameters on the remaining problems to see how well the performance generalizes to problems for which the parameters were not tuned. The specialization ability is documented by tuning the control parameters to perform well on a single optimization problem of interest, to see how well the optimizer can be made to perform on one problem at a time disregarding how expensive this would have to be for every optimization problem a practitioner would encounter. These experiments may also be combined when dealing with a group of seemingly related optimization problems, so that parameter tuning is done for one single optimization problem, and the parameters discovered thus would then be used on all the optimization problems in question. This would be useful for time-consuming optimization problems often encountered in the real-world.

7.4 No Free Lunch

The results of this study, showing that DE variants have comparable performance when using various adaptive schemes for their control parameters, seems to confirm the No Free Lunch (NFL) set of theorems by Wolpert and Macready [49]. The most frequently quoted of the NFL theorems states that no optimizer is universally better than another, when performance is compared over all possible optimization problems, and it is indeed striking how much the results in this study resemble that statement, albeit for a selection of only twelve benchmark problems. However, according to NFL one optimizer can actually be better

aligned with certain kinds of optimization problems and hence yield improved performance on these. The optimizer will then necessarily have to pay for this with degraded performance on other problems. The challenge for a practitioner is therefore to choose the optimizer best aligned with the problem at hand. What is remarkable about the study here in relation to NFL, is that it indicates the underlying alignment between optimizer and problem, does not appear to change much under the use of adaptive schemes for the optimizer's control parameters, as otherwise believed by large parts of the research community.

7.5 Recommendations for Future Research

Apart from the recommendations for future research that have already been made, a number of other topics should also be mentioned. One topic is the tuning of control parameters to perform well with regard to other measures than the one in figure 5 (which used the average performance obtained after a fixed number of optimization iterations.) Other useful performance measures to tune for could be the number of optimization iterations required to achieve a certain goal, or perhaps the rate of convergence as measured by the integral of the fitness progress. Another research topic of interest is the tuning of discrete control parameters, for instance to select between different combinations of sub-algorithms for the optimizer, say, the crossover, mutation, and selection operators of DE. Currently the LUS meta-optimizer only supports real-valued parameters, and research would be needed to find a meta-optimizer, that is equally efficient for use on discrete control parameters, as LUS is for real-valued parameters. Indeed, an important research topic is what meta-optimizer is generally best. If one is going to use the meta-optimizer often, then it makes sense to use the one that performs best at this task. Finding the best meta-optimizer is in effect Meta-Meta-Optimization, which is readily supported by the source-code used in these experiments, and which is made available to the public as described below. However, meta-meta-optimization is extraordinarily time-consuming and we have yet to present actual results for this. Another interesting research topic is the use of meta-optimization in the presence of multiple objectives, where the meta-fitness would therefore also be multi-objective. Other research topics would be the use of meta-optimization in the presence of constraints, and so on.

Beyond the issue of automatic tuning of control parameters, we feel the most conceptually interesting, but also an even more challenging area of future research, is the automatic discovery of an actual optimization algorithm. This could for instance be done by the use of Genetic Programming (GP) of Koza [50], that provides a means for optimizing programs or abstract data-structures instead of just numerical values. Some preliminary results have been reported in this area by Bengio et al. [51] and Radi and Poli [52], while the concept was in fact suggested by Schmidhuber over two decades ago [53]. Recently Fukunaga [16] showed that a simple GP could indeed evolve heuristics for solving combinatorial SAT problems rivalling state-of-the-art heuristics developed by human researchers. But much work remains to be done, especially to automatically discover heuristics for numerical optimization over real-valued search-spaces, and

also to produce as simple and graceful an approach as the one we have used here for automatic tuning of just the control parameters. Taking the idea of evolving an optimization algorithm one step further, it can be used in a bootstrapped manner to gradually improve itself. Such an idea for general problem solving is proposed in the most ambitious Gödel Machine, also by Schmidhuber [54] [55], although it is not yet at an operational stage.

8 Conclusion

This was a study on whether the control parameters for DE should be adapted during optimization, or if the parameters could just as well be held fixed, if they were merely tuned properly. To fairly compare the performance of DE variants against each other, their control parameters were all automatically tuned using a meta-optimization approach. This uncovered the core performance capability of the DE variants showing that there does not appear to be any general and consistent advantage of the DE variants with adaptive control parameters. The basic form of DE with its control parameters held fixed during optimization performs roughly as well in terms of the end results achieved. Similar performance characteristics were also prevalent with regard to the rate of convergence. In fact, the only situation in which the DE variants with adaptive control parameters appeared to have an advantage over using fixed parameters was in terms of their specialization ability. That is, if their control parameters were tuned for just one problem then their performance was better on that one problem. Whether this specialization ability exists on a wider set of optimization problems is not yet known. But it is ironic because those DE variants were devised with the intent of having better generalization ability and so they would not require tuning of their parameters for specific problems.

The general lesson to be learned from this study is that an optimizer's control parameters must always be properly tuned to assess its true performance capability. This holds even if some of the parameters are made adaptive, because then there are just other paramaters that need to be tuned such as the endpoints of the range of adaptation. The fastest and easiest way of parameter tuning is by an automated approach, such as our meta-optimization technique. Whether more complex optimizers with so-called adaptive control parameters hold a general and consistent performance advantage over simpler optimizers using fixed control parameters is dubious.

9 Acknowledgements

Dr. Rainer Storn (researcher at Rohde & Schwarz, Germany) is thanked for suggestions on what DE variants to be used in this study. Associate Professor Janez Brest (University of Maribor, Slovenia) is thanked for supplying source-code for the JDE variant.

10 Source-Code

Source-code implemented in the ANSI C programming language and used in the experiments in this paper, can be found in the SwarmOps library on the internet address: `http://www.Hvass-Labs.org/`

References

[1] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341 – 359, 1997.

[2] R. Storn. On the usage of differential evolution for function optimization. In *Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS)*, pages 519–523, Berkeley, CA, USA, 1996.

[3] K. Price, R. Storn, and J. Lampinen. *Differential Evolution – A Practical Approach to Global Optimization*. Springer, 2005.

[4] J. Liu and J. Lampinen. On setting the control parameter of the differential evolution method. In *Proceedings of the 8th International Conference on Soft Computing (MENDEL)*, pages 11–18, Brno, Czech Republic, 2002.

[5] D. Zaharie. Critical values for the control parameters of differential evolution algorithms. In *Proceedings of MENDEL 2002, 8th International Mendel Conference on Soft Computing*, pages 62–67, Bruno, 2002.

[6] J. Liu and J. Lampinen. A fuzzy adaptive differential evolution algorithm. *Soft Computing*, 9(6):448–462, 2005.

[7] A.K. Qin and P.N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *Proceedings of the IEEE congress on evolutionary computation (CEC)*, pages 1785–1791, 2005.

[8] A.K. Qin, V.L. Huang, and P.N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation, Accepted*, 2008.

[9] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer. Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark functions. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.

[10] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

[11] T. Bäck. Parallel optimization of evolutionary algorithms. In *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature (PPSN)*, pages 418–427, London, UK, 1994. Springer-Verlag.

[12] M. Birattari. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, Belgium, 2004.

[13] M. Meissner, M. Schmuker, and G. Schneider. Optimized particle swarm optimization (OPSO) and its application to artificial neural network training. *BMC Bioinformatics*, 7(125), 2006.

[14] V. Nannen and A.E. Eiben. A method for parameter calibration and relevance estimation in evolutionary algorithms. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 183–190, Seattle, USA, 2006.

[15] E.K. Burke, M.R. Hyde, G. Kendall, and J. Woodward. Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO)*, pages 1559–1565, London, England, 2007.

[16] A.S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, 16(1):21–61, 2008.

[17] M.E.H. Pedersen and A.J. Chipperfield. Simplifying particle swarm optimization. *Applied Soft Computing*, In press, 2009.

[18] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, Perth, Australia, 1995.

[19] Y. Shi and R.C. Eberhart. A modified particle swarm optimizer. In *Proceedings of 1998 IEEE International Conference on Evolutionary Computation*, pages 69–73, Anchorage, AK, USA, 1998.

[20] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[21] R. Storn. Private correspondance, 2008.

[22] R. Storn. Differential evolution research – trends and open questions. In U. K. Chakraborty, editor, *Advances in Differential Evolution*, chapter 1. Springer, 2008.

[23] R. Storn. Optimization of wireless communications applications using differential evolution. In *SDR Technical Conference*, Denver, 2007.

[24] L.A. Zadeh. Fuzzy logic and approximate reasoning. *Synthese*, 30:407–428, 1975.

[25] J. Brest, B. Bošković, S. Greiner, V. Žumer, and M.S. Maučec. Performance comparison of self-adaptive and adaptive differential evolution algorithms. *Soft Computing*, 11:617–629, 2007.

[26] M.E.H. Pedersen and A.J. Chipperfield. Local unimodal sampling. Technical Report HL0801, Hvass Laboratories, 2008.

[27] R.E. Mercer and J.R. Sampson. Adaptive search using a reproductive metaplan. *Kybernetes (The International Journal of Systems and Cybernetics)*, 7:215–228, 1978.

[28] A.J. Keane. Genetic algorithm optimization in multi-peak problems: studies in convergence and robustness. *Artificial Intelligence in Engineering*, 9:75–83, 1995.

[29] E. Ridge and D. Kudenko. Sequential experiment designs for screening and tuning parameters of stochastic heuristics. In *Proceedings of the Ninth International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 27–34, Reykjavik, Iceland, 2006.

[30] E. Ridge and D. Kudenko. Tuning the performance of the MMAS heuristic. In *Proceedings of Engineering Stochastic Local Search Algorithms (SLS)*, pages 46–60, Brussels, Belgium, 2007.

[31] E. Ridge. *Design of experiments for the tuning of optimisation algorithms*. PhD thesis, Department of Computer Science, University of York, United Kingdom, 2007.

[32] O. François and C. Lavergne. Design of evolutionary algorithms – a statistical perspective. *IEEE Transactions on Evolutionary Computation*, 5(2):129–148, 2001.

[33] A. Czarn, C. MacNish, K. Vijayan, B. Turlach, and R. Gupta. Statistical exploratory analysis of genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8(4):405–421, 2004.

[34] W.A. de Landgraaf, A.E. Eiben, and V. Nannen. Parameter calibration using meta-algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 71–78, Singapore, 2007.

[35] D.E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, 1989.

[36] J. Horn, N. Nafpliotis, and D.E. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE International Conference on Evolutionary Computation*, volume 1, pages 82–87, New Jersey, USA, 1994.

[37] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[38] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[39] C.M. Fonseca and P.J. Fleming. Multiobjective optimization. In T. Bäck, D.B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, pages C4.5:1 – 9. IOP Publishing and Oxford University Press, 1997.

[40] A.N. Aizawa and B.W. Wah. Scheduling of genetic algorithms in a noisy environment. *Evolutionary Computation*, 2(2):97–122, 1994.

[41] W.J. Gutjahr and G.C. Pflug. Simulated annealing for noisy cost functions. *Journal of Global Optimization*, 8(1):1–13, 1996.

[42] X. Yao, Y. Ling, and G. Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102, 1999.

[43] M.H. Eres, G.E. Pound, Z. Jiao, J.L. Wason, F. Xu, A.J. Keane, and S.J. Cox. Implementation and utilisation of a grid-enabled problem solving environment in matlab. *Future Generation Computer Systems*, 21(6):920–929, 2005.

[44] E.L. Crow, F.A. Davis, and M.W. Maxfield. *Statistical Manual*. Dover Publications, 1960.

[45] G.T. Toussaint. Bibliography on estimation of misclassification. *IEEE Transactions on Information Theory*, 20(4):472–479, 1974.

[46] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[47] Z. Tu and Y. Lu. A robust stochastic genetic algorithm (StGA) for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 8(5):456–470, 2004.

[48] Z. Tu and Y. Lu. Errata to "A robust stochastic genetic algorithm (StGA) for global numerical optimization". *IEEE Transactions on Evolutionary Computation, Accepted*, 2008.

[49] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67 – 82, 1997.

[50] J.R. Koza. *Genetic Programming : on the programming of computers by means of natural selection*. Bradford, MIT Press, 1992.

[51] S. Bengio, Y. Bengio, and J. Cloutier. Use of genetic programming for the search of a new learning rule for neural networks. In *International Conference on Evolutionary Computation*, pages 324–327, 1994.

[52] A. Radi and R. Poli. Discovering efficient learning rules for feedforward neural networks using genetic programming. In *Recent Advances in Intelligent Paradigms and Applications*, pages 133–159. Physica-Verlag GmbH, 2003.

[53] J. Schmidhuber. Evolutionary principles in self-referential learning – Diploma Thesis. *Technische Universität München*, 1987.

[54] J. Schmidhuber. Gödel machines: fully self-referential optimal universal problem solvers. In *Artificial General Intelligence*, pages 201–228. Springer Verlag, 2006.

[55] J. Schmidhuber. Gödel machines: towards a technical justification of consciousness. In *Adaptive Agents and Multi-Agent Systems III (LNCS 3394)*, pages 1–23. Springer Verlag, 2005.

- Initialize all the agents with random positions / solutions in the search-space.
- Until a termination criterion is met (e.g. a given number of fitness evaluations have been executed, observed fitness stagnates, or a fitness threshold is met), repeat:
  - For each agent $\vec{x}$ in the population do:
    - Pick three agents $\vec{a}$, $\vec{b}$, $\vec{c}$ at random, which must be distinct from each other as well as $\vec{x}$.
    - Pick a random index $R \in \{1, \cdots, n\}$, where the highest possible value n, is the dimensionality of the problem $f : \mathbb{R}^n \rightarrow \mathbb{R}$ to be minimized.
    - Compute the agent's potential new solution $\vec{y} = [y_1, \cdots, y_n]$, by iterating over each $i \in \{1, \cdots, n\}$ as follows:
      - Pick $r_i \sim U(0, 1)$
      - Compute the i'th element of the potentially new solution $\vec{y}$, using Eq.(1).
    - If $(f(\vec{y}) < f(\vec{x}))$ then update the agent's best known solution:

$$\vec{x} \leftarrow \vec{y}$$

Figure 1: DE algorithm.

Sphere	$f(\vec{x}) = \sum_{i=1}^{n} x_i^2$
Schwefel2-22	$f(\vec{x}) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$
Schwefel1-2	$f(\vec{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$
Schwefel2-21	$f(\vec{x}) = \max \{ \lvert x_i \rvert : \ i \in \{1, \cdots, n\} \}$
Rosenbrock	$f(\vec{x}) = \sum_{i=1}^{n-1} \left(100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right)$
Step	$f(\vec{x}) = \sum_{i=1}^{n} \left(\lfloor x_i + 0.5 \rfloor \right)^2$
QuarticNoise	$f(\vec{x}) = \sum_{i=1}^{n} (i \cdot x_i^4 + r_i), \ r_i \sim U(0,1)$
Rastrigin	$f(\vec{x}) = \sum_{i=1}^{n} \left(x_i^2 + 10 - 10 \cdot \cos(2\pi x_i) \right)$
Ackley	$f(\vec{x}) = e + 20 - 20 \cdot \exp\left(-0.2 \cdot \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2} \right) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i) \right)$
Griewank	$f(\vec{x}) = 1 + \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}} \right)$
Penalized1	$f(\vec{x}) = \frac{\pi}{n} \left(10 \cdot \sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 \cdot \left(1 + 10 \cdot \sin^2(\pi y_{i+1}) \right) + (y_n - 1)^2 \right) + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$ $y_i = 1 + (x_i + 1)/4$ $u(x_i, a, k, m) = \begin{cases} k(-x_i - a)^m & , x_i < -a \\ 0 & , -a \leq x_i \leq a \\ k(x_i - a)^m & , x_i > a \end{cases}$
Penalized2	$f(\vec{x}) = 0.1 \left(\sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2 \cdot \left(1 + \sin^2(3\pi x_{i+1}) \right) + (x_n - 1)^2 \cdot \left(1 + \sin^2(2\pi x_n) \right) \right) + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$, with $u(\cdot)$ from above.

Table 1: Benchmark problems used in this study.

- Initialize $\vec{x}$ to a random solution in the search-space:

$$\vec{x} \sim U\left(\vec{b}_{lo}, \vec{b}_{up}\right)$$

  Where $\vec{b}_{lo}$ and $\vec{b}_{up}$ are the search-space boundaries.

- Set the initial sampling range $\vec{d}$ to cover the entire search-space:

$$\vec{d} \leftarrow \vec{b}_{up} - \vec{b}_{lo}$$

- Until a termination criterion is met, repeat the following:
  - Pick a random vector $\vec{a} \sim U\left(-\vec{d}, \vec{d}\right)$
  - Add this to the current solution $\vec{x}$, to create the new potential solution $\vec{y}$:

$$\vec{y} = \vec{x} + \vec{a}$$

  - If $(f(\vec{y}) < f(\vec{x}))$ then update the solution:

$$\vec{x} \leftarrow \vec{y}$$

    Otherwise decrease the search-range by multiplication with the factor q from Eq.(3):

$$\vec{d} \leftarrow q \cdot \vec{d}$$

    Note that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is the meta-fitness algorithm from figure 5.

Figure 2: LUS algorithm.

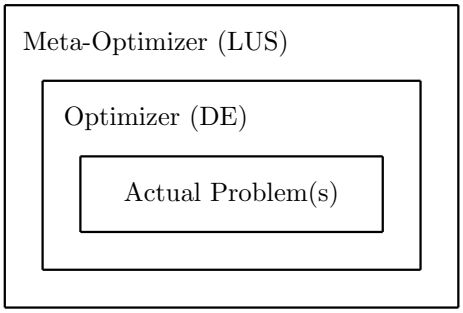

Figure 3: The concept of meta-optimization. The LUS optimization method is used as an overlaid meta-optimizer for finding good control parameters for DE, which in turn is used to optimize one or more actual problems.

- Initialize the LUS meta-optimizer with a random choice of DE control parameters.
- Perform a number of iterations of the LUS meta-optimizer, where each iteration consists of the following:
  - Determine a new choice of DE control parameters from the previously best known parameters, according to the optimization methodology of the LUS meta-optimizer.
  - Compute a performance measure for how good these new potential DE parameters fare when DE is used to optimize a number of problems. The algorithm for computing this meta-fitness measure is shown in figure 5.
  - Keep the new DE control parameters if the meta-fitness is an improvement, otherwise discard the parameters.

Figure 4: Overall algorithm for performing meta-optimization of DE control parameters.

Problem	Initialization	Boundaries
Sphere	$[50, 100]$	$[-100, 100]$
Schwefel2-22	$[5, 10]$	$[-10, 10]$
Schwefel1-2	$[50, 100]$	$[-100, 100]$
Schwefel2-21	$[50, 100]$	$[-100, 100]$
Rosenbrock	$[15, 30]$	$[-100, 100]$
Step	$[50, 100]$	$[-100, 100]$
QuarticNoise	$[0.64, 1.28]$	$[-1.28, 1.28]$
Rastrigin	$[2.56, 5.12]$	$[-5.12, 5.12]$
Ackley	$[15, 30]$	$[-30, 30]$
Griewank	$[300, 600]$	$[-600, 600]$
Penalized1	$[5, 50]$	$[-50, 50]$
Penalized2	$[5, 50]$	$[-50, 50]$

Table 2: Initialization ranges and search-space boundaries for the benchmark problems. These values are used for all dimensions.

- Initialize the problem-counter: $i \leftarrow 1$, and the fitness-sum: $s \leftarrow 0$
- While $(i \leq M)$ and $(s < L)$, do:
  - Initialize the run-counter: $j \leftarrow 1$
  - While $(j \leq N)$ and $(s < L)$, do:
    - Perform an optimization run on the i'th problem using DE with the given choice of control parameters.
    - Add the best fitness obtained in the run (call it $\bar{f}$) to the fitness-sum: $s \leftarrow s + \bar{f}$
    - Increment the run-counter: $j \leftarrow j + 1$
  - Increment the problem-counter: $i \leftarrow i + 1$
- Sort the actual problems descendingly according to their contributions to the overall fitness sum s. This will allow earlier preemptive abortion of the meta-fitness evaluation next time.
- Return s to the overlaying meta-optimizer as the meta-fitness value of DE with the given choice of control parameters.

Figure 5: Algorithm for performing a single meta-fitness evaluation in meta-optimization, for rating the performance of DE with a given choice of control parameters.

Basic	$NP \in \{4, \cdots, 200\}$	$CR \in [0, 1]$	$F \in [0, 2]$
Dither	$NP \in \{4, \cdots, 200\}$	$CR \in [0, 1]$	$F_{mid} \in [0, 2]$ $F_{range} \in [0, 3]$
Jitter	$NP \in \{4, \cdots, 200\}$	$CR \in [0, 1]$	$F_{mid} \in [0, 2]$ $F_{range} \in [0, 3]$
JDE	$NP \in \{4, \cdots, 200\}$	$CR_{init} \in [0, 1]$ $CR_l \in [0, 1]$ $CR_u \in [0, 1]$ $\tau_{CR} \in [0, 1]$	$F_{init} \in [0, 2]$ $F_l \in [0, 2]$ $F_u \in [0, 2]$ $\tau_F \in [0, 1]$

Table 3: Boundaries for the parameter search-spaces of DE/rand/1/bin variants as used in all the meta-optimization experiments.

Basic	$NP = 300$	$CR = 0.9$	$F = 0.5$
Dither	$NP = 300$	$CR = 0.9$	$F_{mid} = 0.75$ $F_{range} = 0.25$
Jitter	$NP = 300$	$CR = 0.9$	$F_{mid} = 0.5$ $F_{range} = 0.0005$
JDE	$NP = 100$	$CR_{init} = 0.9$ $CR_l = 0$ $CR_u = 1$ $\tau_{CR} = 0.1$	$F_{init} = 0.5$ $F_l = 0.1$ $F_u = 0.9$ $\tau_F = 0.1$

Table 4: Hand-tuned parameters for DE/rand/1/bin variants. Benchmark results for these parameters are found in table 5.

Problem	Basic	Dither	Jitter	JDE
Sphere	38621.4 (4548.21)	51852.9 (5586.94)	38016.2 (4017.8)	**6436.98 (1053.79)**
Schwefel2-22	113.81 (12.12)	2.18e+8 (5.1e+8)	1.5e+6 (3.18e+6)	**41.86 (5.23)**
Schwefel1-2	72882.7 (8989.52)	81794.7 (7984.69)	71020.1 (10460.3)	**51656.8 (8882.9)**
Schwefel2-21	78.9 (3.35)	83.63 (4.19)	78.72 (3.18)	**64.41 (5.25)**
Rosenbrock	1.17e+8 (2.49e+7)	1.27e+8 (2.69e+7)	1.09e+8 (2.98e+7)	**9.08e+6 (3.17e+6)**
Step	36961.7 (5420.24)	50946.4 (5148.09)	36811.8 (5068.24)	**6183.1 (1162.32)**
QuarticNoise	51.45 (10.89)	73.92 (14.46)	54.78 (10.12)	**17.24 (1.43)**
Rastrigin	348.03 (17.57)	383.34 (18.52)	347.85 (16.65)	**245.3 (15.92)**
Ackley	19.7 (0.37)	20.14 (0.16)	19.63 (0.35)	**18.39 (1.28)**
Griewank	347.13 (45.77)	464.52 (48.4)	350.73 (35.25)	**62.54 (10.37)**
Penalized1	6.1e+7 (2.03e+7)	1.55e+8 (4.43e+7)	5.89e+7 (1.85e+7)	**466079 (392499)**
Penalized2	1.34e+8 (3.81e+7)	3.11e+8 (7.89e+7)	1.34e+8 (3.72e+7)	**2.05e+6 (1.17e+6)**

Table 5: Results for DE/rand/1/bin variants when using the hand-tuned parameters from table 4. The problem dimensionalities are set to $n = 30$, and $n \cdot 200$ fitness evaluations are allowed per optimization run. Table shows the average fitness results obtained over 50 optimization runs, with the numbers in parentheses being the standard deviations. The statistically significant best results for each problem are printed in bold face.

Basic	$NP = 10$	$CR = 0.031855$	$F = 0.733094$
Dither	$NP = 7$	$CR = 0.021481$	$F_{mid} = 0.849680$ $F_{range} = 1.779813$
Jitter	$NP = 11$	$CR = 0.096154$	$F_{mid} = 0.503464$ $F_{range} = 0.954235$
JDE	$NP = 16$	$CR_{init} = 0.573335$ $CR_l = 0.128144$ $CR_u = 0.871238$ $\tau_{CR} = 0.705309$	$F_{init} = 0.500358$ $F_l = 0.419994$ $F_u = 0.621257$ $\tau_F = 0.573597$

Table 6: Parameters for DE/rand/1/bin variants tuned to perform well on all twelve benchmark problems on average. Benchmark results for these parameters are found in table 7.

Problem	Basic	Dither	Jitter	JDE
Sphere	3.56 (23.8)	1.03e-3 (1.22e-3)	0.14 (0.53)	206.99 (1399.43)
Schwefel2-22	0.08 (0.09)	**5.13e-3 (2.15e-3)**	0.02 (6.50e-3)	1.96 (2.37)
Schwefel1-2	21666.4 (4391.13)	36887 (7513.36)	26646.7 (4988.37)	20052.3 (5672.66)
Schwefel2-21	63.26 (2.57)	61.6 (2.71)	58.22 (3.46)	58.75 (5.82)
Rosenbrock	453.43 (203.38)	**306.52 (454.55)**	514.9 (708.73)	11519.2 (17512.5)
Step	23.12 (161.84)	0.14 (0.6)	23.76 (161.79)	3.06 (4.21)
QuarticNoise	15.29 (1.61)	14.78 (1.96)	13.85 (1.1)	**12.43 (1.15)**
Rastrigin	43.23 (14.81)	**33.57 (13.58)**	44.65 (13.21)	162.31 (16.36)
Ackley	11.94 (6.13)	13.03 (5.5)	16.46 (5.46)	19.83 (0.06)
Griewank	0.61 (2.29)	**0.04 (0.04)**	0.13 (0.08)	1.09 (0.1)
Penalized1	1.02e-7 (1.95e-7)	1.96e-9 (7.87e-9)	1.75e-11 (9.93e-11)	168.51 (821.97)
Penalized2	2.50e-3 (2.91e-3)	1.70e-3 (5.03e-3)	1.67e-3 (2.52e-3)	8.36 (8.86)

Table 7: Results for DE/rand/1/bin variants when using the parameters from table 6 that were meta-optimized for all benchmark problems. The problem dimensionalities are set to $n = 30$, and $n \cdot 200$ fitness evaluations are allowed per optimization run. Table shows the average fitness results obtained over 50 optimization runs, with the numbers in parentheses being the standard deviations. The statistically significant best results for each problem are printed in bold face.

Basic	$NP = 21$	$CR = 0.103123$	$F = 0.399898$
Dither	$NP = 48$	$CR = 0.998966$	$F_{mid} = 0.547226$ $F_{range} = 2.549470$
Jitter	$NP = 110$	$CR = 0.999320$	$F_{mid} = 0.508141$ $F_{range} = 0.006431$
JDE	$NP = 31$	$CR_{init} = 0.970131$ $CR_l = 0.976761$ $CR_u = 0.023239$ $\tau_{CR} = 0.501746$	$F_{init} = 1.728003$ $F_l = 0.321481$ $F_u = 1.379208$ $\tau_F = 0.093426$

Table 8: Parameters for DE/rand/1/bin variants tuned to perform well on just the Schwefel1-2 problem. Benchmark results for these parameters are found in table 10.

Basic	$NP = 9$	$CR = 0.248700$	$F = 0.725905$
Dither	$NP = 7$	$CR = 0.000276$	$F_{mid} = 0.688212$ $F_{range} = 1.945058$
Jitter	$NP = 8$	$CR = 0.023666$	$F_{mid} = 0.617744$ $F_{range} = 1.529568$
JDE	$NP = 8$	$CR_{init} = 0.677320$ $CR_l = 0.056926$ $CR_u = 0.019076$ $\tau_{CR} = 0.911053$	$F_{init} = 1.824508$ $F_l = 0.170150$ $F_u = 1.565926$ $\tau_F = 0.160525$

Table 9: Parameters for DE/rand/1/bin variants tuned to perform well on just the Rosenbrock problem. Benchmark results for these parameters are found in table 10.

Problem	Basic	Dither	Jitter	JDE
Schwefel1-2	17077.9 (4747.5)	5628.92 (2699.03)	8209.57 (3314.26)	**3219.09 (2337.94)**
Rosenbrock	533.55 (315.5)	351.77 (388.41)	320.51 (256.90)	384.95 (816.29)

Table 10: Specialization ability of DE/rand/1/bin variants on the Schwefel1-2 and Rosenbrock benchmark problems. Each DE variant is first meta-optimized to the problem in question, and the parameters thus discovered are then used to optimize the problem and the results are displayed. The problem dimensionalities are set to $n = 30$, and $n \cdot 200$ fitness evaluations are allowed per optimization run. Table shows the average fitness results obtained over 50 optimization runs, with the numbers in parentheses being the standard deviations. The statistically significant best results for each problem are printed in bold face.

Basic	$NP = 8$	$CR = 0.131305$	$F = 0.776182$
Dither	$NP = 7$	$CR = 0.001779$	$F_{mid} = 1.203716$ $F_{range} = 1.931654$
Jitter	$NP = 7$	$CR = 0.170015$	$F_{mid} = 0.788930$ $F_{range} = 0.598157$
JDE	$NP = 8$	$CR_{init} = 0.847650$ $CR_l = 0.104456$ $CR_u = 0.122205$ $\tau_{CR} = 0.875351$	$F_{init} = 0.453133$ $F_l = 0.247631$ $F_u = 1.548331$ $\tau_F = 0.659707$

Table 11: Parameters for DE/rand/1/bin variants tuned to perform well on the Sphere, Rastrigin, and Rosenbrock problems. All benchmark results for these parameters are found in table 12.

Problem	Basic	Dither	Jitter	JDE
Sphere	0.01 (0.02)	8.51e-3 (9.14e-3)	**2.98e-3 (5.51e-3)**	0.01 (0.02)
Schwefel2-22	0.62 (2.37)	**0.02 (0.01)**	1.64 (4.18)	1.17 (3.45)
Schwefel1-2	**28483.5 (5299.23)**	45693.2 (6551.02)	33009.7 (6637.83)	49706.9 (6636.66)
Schwefel2-21	54.05 (6.03)	65.94 (2.97)	55.08 (6.95)	53.02 (5.99)
Rosenbrock	1666.99 (8123.57)	377.38 (326.8)	518.79 (942.14)	576.19 (1392.01)
Step	0.3 (1.37)	0.02 (0.14)	1.68 (9.23)	1.74 (7.02)
QuarticNoise	13.86 (2.24)	14.44 (1.05)	14.93 (3.52)	**13.05 (1.46)**
Rastrigin	64.14 (13.09)	**25.88 (15.68)**	72.1 (19.74)	75.33 (15.7)
Ackley	18.15 (3.58)	**11.31 (4.99)**	17.28 (6.33)	19.49 (0.4)
Griewank	0.06 (0.07)	0.13 (0.07)	1.86 (12.64)	0.07 (0.07)
Penalized1	8.22e-11 (5.61e-10)	9.34e-6 (4.26e-5)	3.03e-12 (2.12e-11)	1198.06 (8386.4)
Penalized2	2.05e-3 (3.57e-3)	1.78e-3 (6.57e-3)	298.33 (2088.03)	0.13 (0.44)

Table 12: Generalization ability of DE/rand/1/bin variants with parameters meta-optimized for three benchmark problems (Sphere, Rosenbrock, and Rastrigin). The problem dimensionalities are set to $n = 30$, and $n \cdot 200$ fitness evaluations are allowed per optimization run. Table shows the average fitness results obtained over 50 optimization runs, with the numbers in parentheses being the standard deviations. The statistically significant best results for each problem are printed in bold face.

	Variant	Duration/Seconds
12 Bnch.	Basic	1073
	Dither	1412
	Jitter	1705
	JDE	2074
3 Bnch.	Basic	241
	Dither	251
	Jitter	434
	JDE	523

Table 13: Time usage for performing meta-optimization of DE/rand/1/bin variants using all twelve or only three benchmark problems.

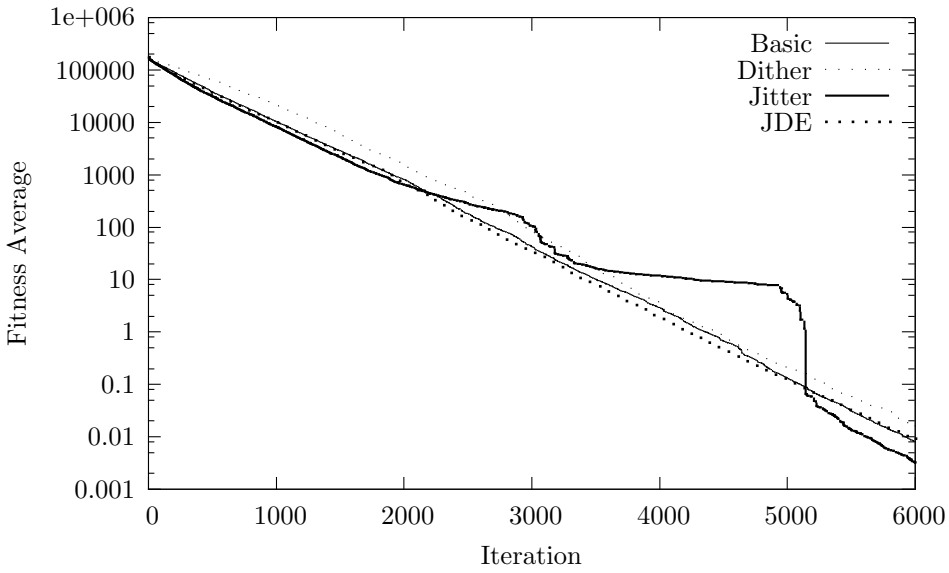

Figure 6: Optimization progress on the Sphere problem for DE/rand/1/bin variants, using the parameters from section 6.7 which were tuned for the Sphere, Rosenbrock, and Rastrigin problems. Fitness trace is averaged over 50 optimization runs, and fitness axis is log-scaled. Plot shows comparable progress for all but the DE Jitter variant, which is more erratic from half-way through optimization but finds slightly better solutions towards the end.

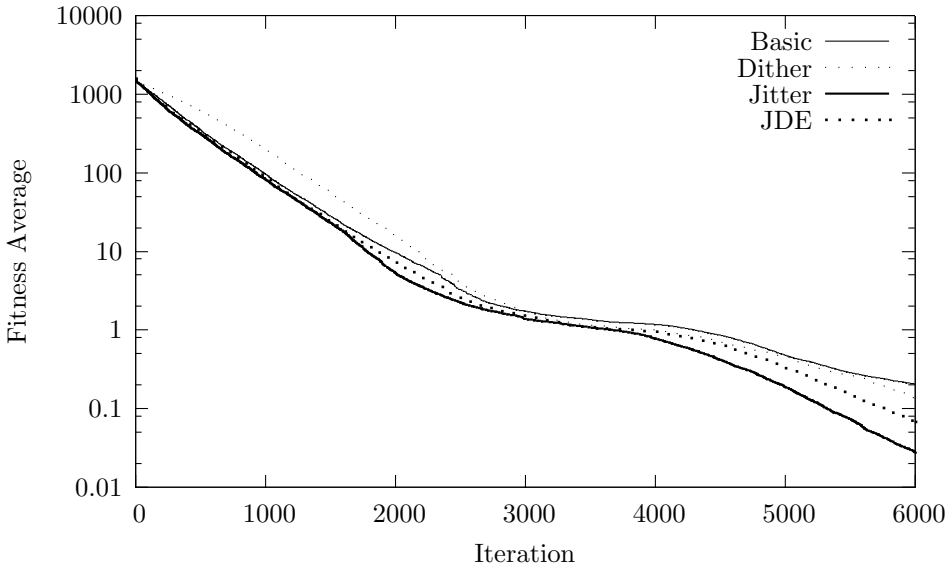

Figure 7: Optimization progress on the Griewank problem for DE/rand/1/bin variants, using the parameters from section 6.7 which were tuned for the Sphere, Rosenbrock, and Rastrigin problems. Fitness trace is averaged over 50 optimization runs, and fitness axis is log-scaled. Although there is a slight difference in the final results achieved by these DE variants, their progress is comparable and exhibits a clearly similar curvature.

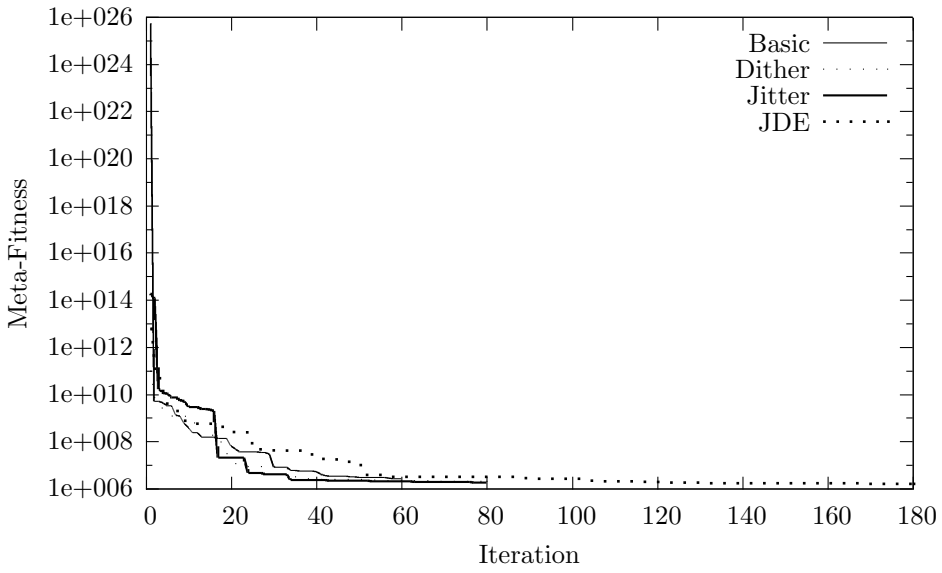

Figure 8: Meta-optimization progress for DE/rand/1/bin variants, showing how easy or hard they are to tune. Meta-fitness axis is log-scaled.

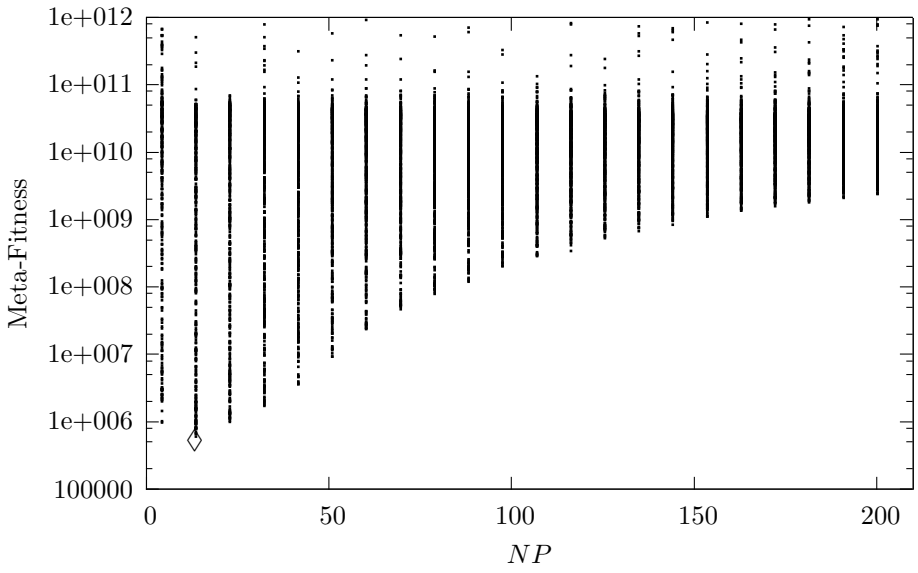

Figure 9: Performance of DE/rand/1/bin for different choices of NP when also varying parameters CR and F. Only entries with a meta-fitness below 1e+12 are shown, and the diamond indicates the best of these. Meta-fitness axis is log-scaled. Performance clearly worsens with increased population size NP.

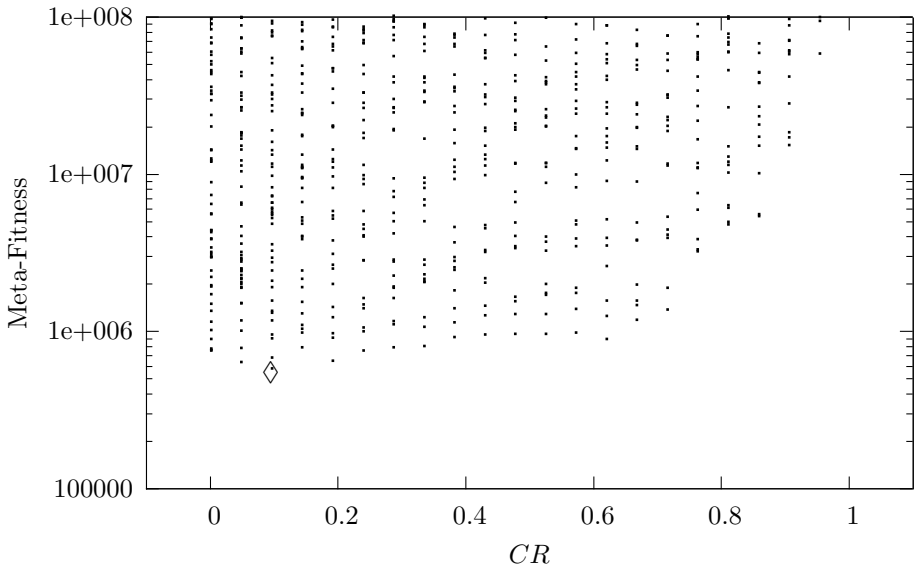

Figure 10: Performance of DE/rand/1/bin for different choices of CR when also varying parameters NP and F. Only entries with a meta-fitness below 1e+8 are shown, and the diamond indicates the best of these. Meta-fitness axis is log-scaled. Meta-fitness axis is log-scaled. Good choices of CR appear to be roughly in the range $[0, 0.7]$.

Problem	Temporal	Fixed
Sphere	35.13 (21.67)	**3.56 (23.8)**
Schwefel2-22	1.15 (0.26)	**0.08 (0.09)**
Schwefel1-2	**15985.4 (5159.88)**	21666.4 (4391.13)
Schwefel2-21	67 (2.82)	**63.26 (2.57)**
Rosenbrock	2991.78 (3321.16)	**453.43 (203.38)**
Step	23.8 (18.68)	23.12 (161.84)
QuarticNoise	**11.44 (0.83)**	15.29 (1.61)
Rastrigin	54.84 (9.31)	**43.23 (14.81)**
Ackley	17.99 (2.62)	**11.94 (6.13)**
Griewank	1.45 (0.67)	**0.61 (2.29)**
Penalized1	9.56e-4 (5.70e-3)	1.02e-7 (1.95e-7)
Penalized2	0.89 (0.52)	**2.50e-3 (2.91e-3)**

Table 14: Results for DE/rand/1/bin with temporal parameters meta-optimized for all benchmark problems. Results for fixed parameters are reprinted from table 7. The problem dimensionalities are set to $n = 30$, and $n \cdot 200$ fitness evaluations are allowed per optimization run. Table shows the average fitness results obtained over 50 optimization runs, with the numbers in parentheses being the standard deviations. The statistically significant best results for each problem are printed in bold face.

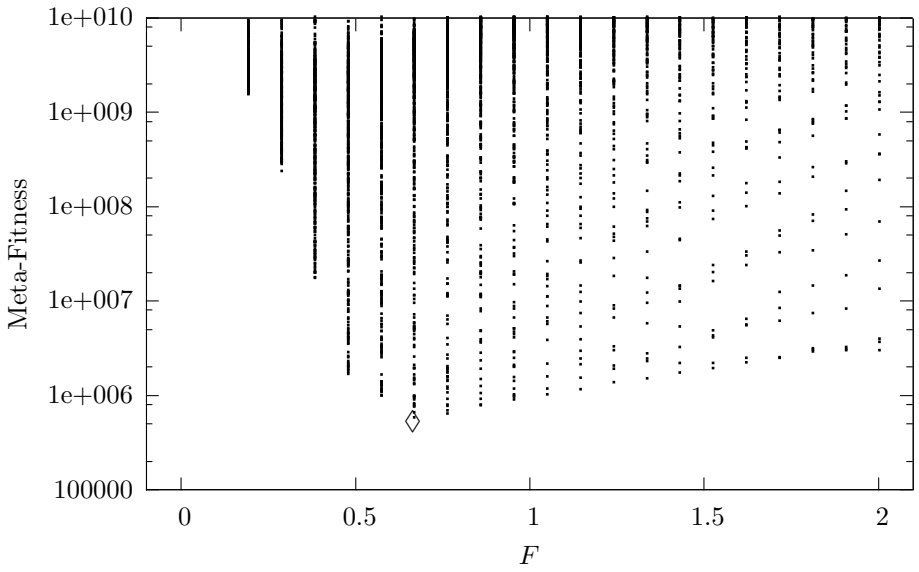

Figure 11: Performance of DE/rand/1/bin for different choices of F when also varying parameters NP and CR. Only entries with a meta-fitness below 1e+10 are shown, and the diamond indicates the best of these. Meta-fitness axis is log-scaled. Good choices of F appear to be in the range $[0.5, 1]$.

Problem	Result
Sphere	1.22e-3 (3.29e-3)
Schwefel2-22	117.84 (68.26)
Schwefel1-2	**2736.15 (2480.4)**
Schwefel2-21	57.25 (11.68)
Rosenbrock	**190.54 (190.84)**
Step	568.2 (1436)
QuarticNoise	27.97 (6.71)
Rastrigin	354.40 (62.84)
Ackley	19.75 (0.09)
Griewank	0.05 (0.06)
Penalized1	6.36e-20 (4.20e-19)
Penalized2	28.77 (10.03)

Table 15: Results for DE/best/1/bin/simple with parameters meta-optimized for all benchmark problems. The problem dimensionalities are set to $n = 30$, and $n \cdot 200$ fitness evaluations are allowed per optimization run. Table shows the average fitness results obtained over 50 optimization runs, with the numbers in parentheses being the standard deviations. Results printed in bold face are statistically significantly better than all the results in table 7.

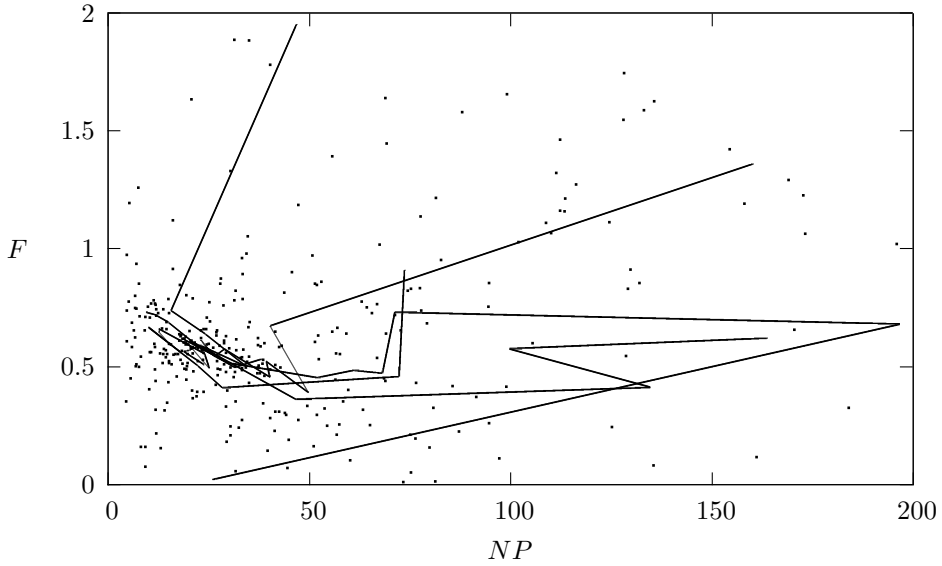

Figure 12: This plot demonstrates the process of meta-optimizing parameters NP and F for DE/rand/1/bin, by showing two things combined: 1) The lines show the successful DE parameter improvements of the LUS meta-optimizer for six different tuning runs. 2) The dots show the unsuccessful moves of these six meta-optimization runs, that is, the DE parameter combinations that were contemplated by the greedy LUS method, but were not accepted as they lead to worse DE performance. This demonstrates how the LUS meta-optimizer closes in on the most promising region of DE parameters (refer to figures 9 and 11 to see what those regions are).

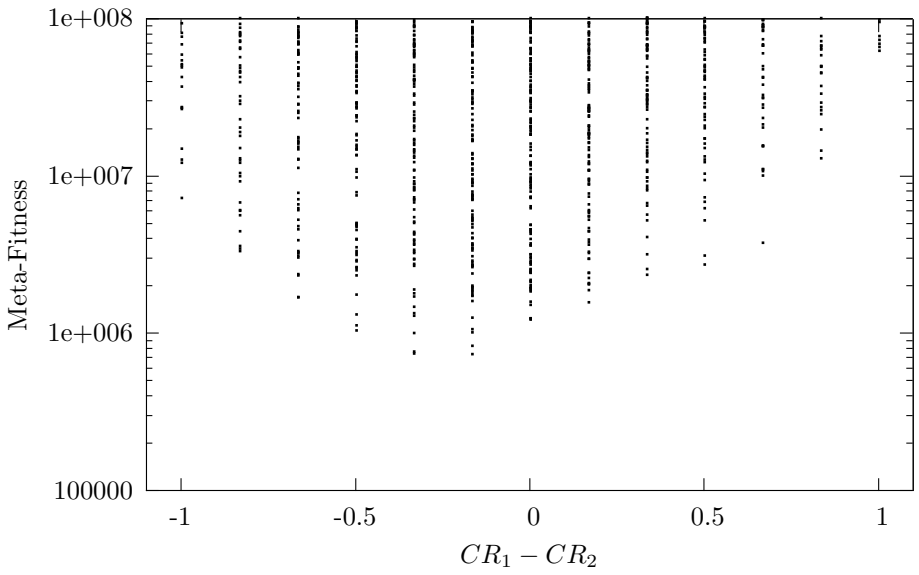

Figure 13: Performance of DE/rand/1/bin with temporal parameters. Plot shows the performance of choosing different values for CR_1 and CR_2, when also varying parameters NP, F_1 and F_2. Only entries with a meta-fitness below 1e+8 are shown. Meta-fitness axis is log-scaled. Best performance is achieved when the crossover probability for the first half of an optimization run (CR_1) is somewhat lower than for the last half of the run (CR_2).

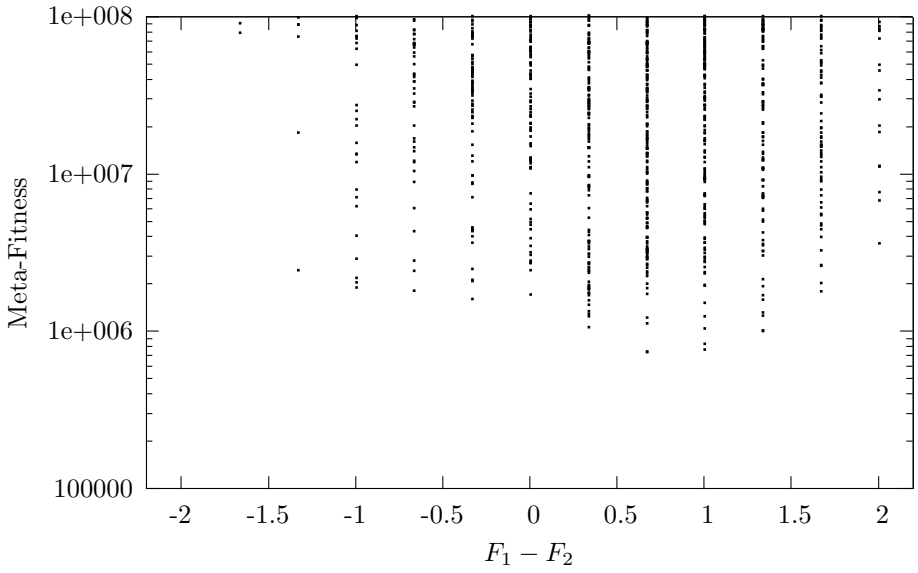

Figure 14: Performance of DE/rand/1/bin with temporal parameters. Plot shows the performance of choosing different values for F_1 and F_2, when also varying parameters NP, CR_1 and CR_2. Only entries with a meta-fitness below 1e+8 are shown. Meta-fitness axis is log-scaled. Best performance is achieved when the differential weight for the first half of an optimization run (F_1) is somewhat higher than for the last half of the run (F_2).

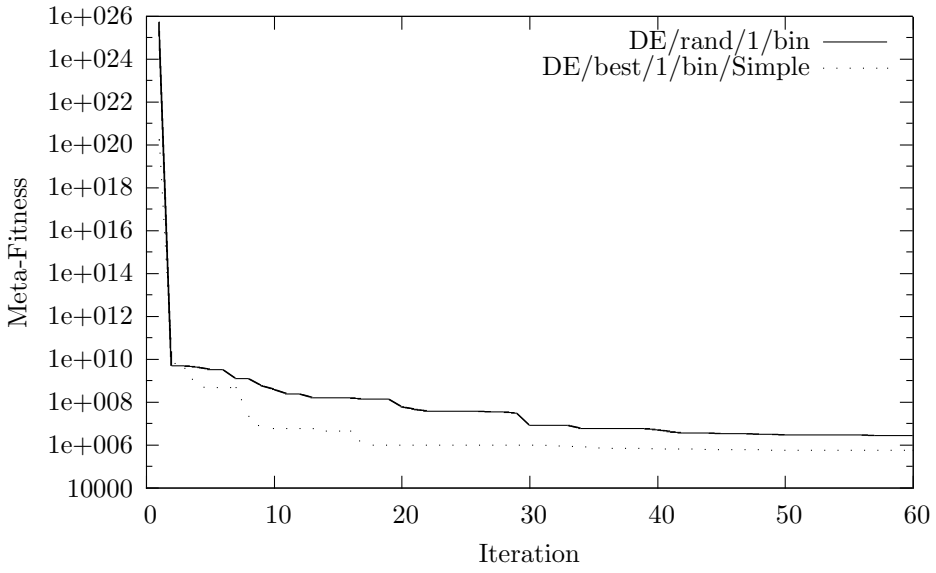

Figure 15: Meta-optimization progress for the DE/rand/1/bin and DE/best/1/bin/simple variants, showing the latter is clearly the easiest to tune. Meta-fitness axis is log-scaled.