AOAB – Automated Optimization Algorithm Benchmarking

Thomas Weise, Li Niu, and Ke Tang
Nature Inspired Computation and Applications Laboratory (NICAL)
University of Science and Technology of China (USTC)
Hefei 230027, Anhui, China
tweise@ustc.edu.cn · newly@mail.ustc.edu.cn · ketang@ustc.edu.cn

ABSTRACT

In this paper we present AOAB, the Automated Optimization Algorithm Benchmarking system. AOAB can be used to automatically conduct experiments with numerical optimization algorithms by applying them to different benchmarks with different parameter settings. Based on the results, AOAB can automatically perform comparisons between different algorithms and settings. It can aid the researcher to identify trends for good parameter settings and to find which algorithms are suitable for which type of problem.

We introduce the system structure of AOAB (the server and the graphical client interface), define the way in which optimizers and benchmark functions can be implemented for the use in AOAB, and conduct an illustrative example experiment with our system: a comparison between Random Search and two Hill Climbers.

Categories and Subject Descriptors

F.2.1 [**ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY**]: Numerical Algorithms and Problems; G.1.6 [**NUMERICAL ANALYSIS**]: Optimization—*Global optimization*

General Terms

Measurement, Performance

Keywords

AOAB, Automated Benchmarking, Optimization Algorithm, Distribution, Single-Objective Optimization, Numerical Optimization

1. Introduction

One of the most fundamental principles in our world is the search for optimal states. Many optimization tasks can be expressed as single-objective, continuous, numerical optimization problems with an n dimensional, real-valued search space. For solving such problems, today we can choose between various optimization algorithms, such as Ant Colony Optimization [9, 21], Differential Evolution [33], Downhill Simplex search [28], Estimation of Distribution Algorithms [29], Evolutionary Programming [11, 39], Evolution Strategies [14, 30], Extremal Optimization [3, 8], Genetic Algorithms [18], Hill Climbing [31], Memetic Algorithms [27], Particle Swarm Optimization [10], Random Optimization [13], Random Sampling [1, 5], Simulated Annealing [20], and Tabu Search [2, 12] – just to name a few [37]. Better yet, for each of these algorithms, there exist many variants (sometimes more than thirty), each coming with difference, characteristic parameters. In practice, it is often not clear which algorithm is the most suitable for a given problem. Even for theoretical benchmark problems, only sparse comparison data is available.

Usually, when a new algorithm or an extension of an existing one is developed, the involved researchers will test their new creation in order to proof its "raison d'être". This is often done by applying it to some selected optimization problems and by comparing its performance with some established algorithms. This procedure is cumbersome for several reasons: *(1)* The implementation of the optimization algorithm has to be extended with logging capabilities. *(2)* It then must be executed for each function in the benchmark several times in order to obtain statistically reliable results. *(3)* This procedure is often repeated for multiple possible configurations of the algorithm and, *(4)* in case the data needed for comparison is not available, also for the algorithms against which it competes. *(5)* The next step is then to read in the log files and to perform said comparison. *(6)* It is a bad practice to just check which algorithm has the better average performance. Instead, statistical significance tests [32] should be applied (which is more complicated) and, *(7)* in the case that multiple configurations were tested, trends in terms of the influence of the parameters should be checked as well. *(8)* Furthermore, diagrams and graphics have to be created from the logged data. *(9)* After all of this has been done, the researcher should take care of the long-term storage of her results, since most often they are needed one or two years later for reference.

It is clear that this procedure is time consuming, error-prone, cumbersome, and – basically – has nothing to do with algorithm design. Since the later is considered to be a much more interesting activity, thorough benchmarking un-

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

fortunately is often neglected in favor of parameter/feature tuning, fiddling, and minimalistic tests.[1]

In this paper, we propose and discuss the realization of a system which relieves the researcher from most of the tasks involved in testing her algorithm, AOAB – the Automated Optimization Algorithm Benchmarking system. It aims to support sustainable and thorough algorithm testing and comparing. Specifically, AOAB addresses the following issues:

1. It automates logging and makes it completely transparent.
2. It requires the researcher to deal with *almost no* code additional to the plain algorithm implementation, no libraries or additional files.
3. It automates running experiment series for different configuration parameters of an optimization algorithm and/or comparisons between different algorithms.
4. It automates the experiment evaluation, trend analysis, statistical algorithm comparisons, and drawing of graphics.
5. It provides a convenient way for long-term data storage.
6. It provides a graphical user interface which allows uploading new algorithms and benchmark functions to the system, to configure new benchmarks, and to start and to evaluate experiments.[2]
7. The system is inherently distributed and able to make use of an arbitrary number of computers for performing the experiments. The distribution of computation is transparent to the user and requires no interaction.
8. The system is able to deal with different programming languages and platforms (`Java`, `C++`, `Matlab`, ...) and can easily be extended for new languages and platforms.

The remainder of this paper is structured as follows. In Section 2 we give an overview on work related to our AOAB project whose system structure is discussed in detail in Section 3. Implementers of optimization algorithms and benchmark functions have to adhere to the simple criteria outlined in Section 4 in order to utilize AOAB. Section 5 shortly discusses some initial experiments and results obtained with our system. We conclude the paper in Section 6 with a summary and a discussion of the planned future work.

2. Related Work

The question "Which optimization algorithms is best?" has been answered by Wolpert and Macready [38]: None. The performance of all optimization algorithms is the same when averaged over all possible optimization problems. Therefore, the question has to be refined to "Which of the optimization algorithm is good for a certain class of problems?".

Tackling such questions on a theoretical basis is complicated and not always possible. Benchmarking, i. e., applying the algorithms to several clearly specified example problems, becomes the technique of choice when comparing optimization method. In the area of numerical optimization, the two most notable events related to this topic are the Black Box Optimization Benchmarking (BBOB) workshop [16, 17] located at GECCO and the regularly reoccurring competitions [19, 22, 23, 34–36] at the IEEE CEC.

In the latter, the benchmark functions are defined in a document and implementations are provided for several programming languages. In the 2010 challenge [36], for example, the goal is to find algorithms suitable for large-scale numerical optimization. A set of twenty benchmark functions comprising separable, non-separable, and m-separable functions is provided as benchmark along with implementations in `Matlab` and `Java`. The contestants in the competition themselves are responsible for logging and evaluating their data according to predefined, fixed criteria.

BBOB, on the other hand, provides a more comprehensive framework for benchmarking: COmparing Continuous Optimisers (COCO) [16, 17]. It again includes the benchmark functions, but already provides logging and evaluation utilities, thus reducing the development overhead for users. Many concepts of COCO are similar to AOAB. There are, however, also striking differences. AOAB, for instance, targets experimental series (currently in full factorial design [4, 40]) for also analyzing the influence of the parameter settings of the optimization algorithms. The goal of AOAB is to provide comprehensive data for not only comparing optimizers but for analyzing their behavior. Also, AOAB is not bound to a certain benchmark structure. While COCO requires a certain effort to change the benchmark, in AOAB a benchmark is simply a set of functions and arbitrary new benchmarks can be defined and new functions can be uploaded at any time. Besides helping researchers to compare and analyze their algorithms, AOAB features a second use case: students who are delving into the matter of optimization algorithms may use it as an easy framework to implement them and in order to explore their behavior.

A very general tool for testing and benchmarking optimization is the Optimization Algorithm Toolkit (OAT) implemented in `Java` by Brownlee [6, 7] in 2007. Whereas OAT supports arbitrary problem domains, it also requires the researcher to delve much deeper into the system itself, mainly due to its generality and the integration into a graphical user interface. Unlike AOAB, OAT cannot support different programming languages and does not allow for automatic factorial experiments in terms of optimization algorithms and their parameters. It is designed to be used on a single computer only and does not support distribution of the work load. In terms of continuous optimization algorithm benchmarking, the highest supported dimension of the search space seems to be 2.

3. System Structure

The structure of our system is illustrated in Figure 1. It consists of a client with a graphical user interface, the AOAB server, the database (solely accessed by the server), and an arbitrary number of worker machines.

3.1 The AOAB Client and User-provided Data

The client with its graphical user interface allows the user to upload new algorithm and objective function implementations to the AOAB server. In both cases, the user first selects the source files and then specifies for which programming language and platform they are intended. Algorithms as well as objective functions can be tagged with

[1] The first author admits that he suffered from this disease in his first years as PhD student.

[2] This feature makes it also suitable for teaching purposes.

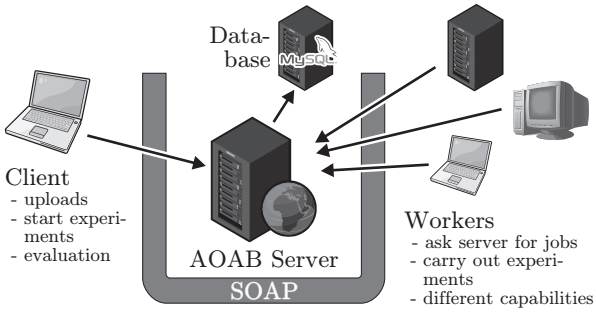

Figure 1: The structure of the AOAB system.

parameters. An implementation of a certain Genetic Algorithm, for instance, may be characterized with the parameters `populationSize` (positive integer), `crossoverRate`, and `mutationRate` (floating point numbers). In case of uploading an objective function, parameters defining the limits of the search space (`minX`, `maxX`) as well as its `dimension` may be declared along with their default values.

With the GUI the user can furthermore compose new benchmarks consisting of a selection of objective functions, the maximum number of function evaluations to be used in the optimization process, and possible settings for their parameter values. In a benchmark definition, parameters of the functions (for instance `dimension`) may be overridden. A function once uploaded may be part of multiple different benchmarks. The Sphere function, for instance, is used in [16] with `dimension` $\in \{2, 3, 5, 10, 20, 40\}$ and in [39], `dimension` $= 30$, amongst others.

To each benchmark-to-function relation, an *interesting point set* (IPS) is furthermore assigned. An IPS consists of definitions of interesting values and interesting function evaluations (IFEs), combining both the vertical-cut and horizontal-cut views discussed in [16]. The former are objective values which, when surpassed, should lead to the collection of one data point by the logging code. The IFEs, on the other hand, are function evaluation numbers which, too, should lead to the collection of a point when exhausted by the optimizer. The explicit IPS definition allows collecting data according to the (often more or less arbitrarily chosen) criteria used by the original benchmark designers. Yet, we would recommend having the IPS contain around 1000 entries in order to collect enough data for being able to sufficiently characterize the behavior of an optimization algorithm. Due to the automatic removal of redundant points, space in the database is preserved.

For starting an experiment, the user selects an algorithm implementation and a benchmark.[3] Additionally, she may specify value sets for each parameter of the algorithm or benchmark. For the aforementioned GA example, she may be interested in carrying out experiments with `populationSize` $\in \{50, 100, 200\}$ and `mutationRate` $\in \{0.01, 0.05\}$. Maybe she is less interested in the crossover rate and, by not specifying a setting for it, simply uses the default value. She now can upload this configuration to the AOAB server which is then to deal with all tasks involved in carrying out the experiments.

After the experiment has been finished, the user can evaluate the results. Like in the COCO/BBOB scenario [16, 17], a pdf document is produced containing various evaluations and diagrams. The difference is that we aim to perform an automated statistical comparison between different algorithms and configurations. If the user chooses the `populationSize`s and `mutationRate`s as in the GA example, this may result in six different configurations.[4] The goal of our system is to discover which configuration is the best as well as trends (such as, hypothetically, "large values of `populationSize` are good" or "the results get worse with rising `dimension`"), but also *when* these trends set in. One algorithm, for instance, may initially quickly detect a good area in the search space but then prematurely converge. Another one may proceed much slower but finally find better solutions. With the collected data, it is possible to make statements regarding such issues automatically. We will provide a plugin architecture allowing adding, removing, and selecting evaluation modules in the GUI. This architecture will allow us to, for instance, include the evaluation modules provided by COCO [16, 17] into AOAB.

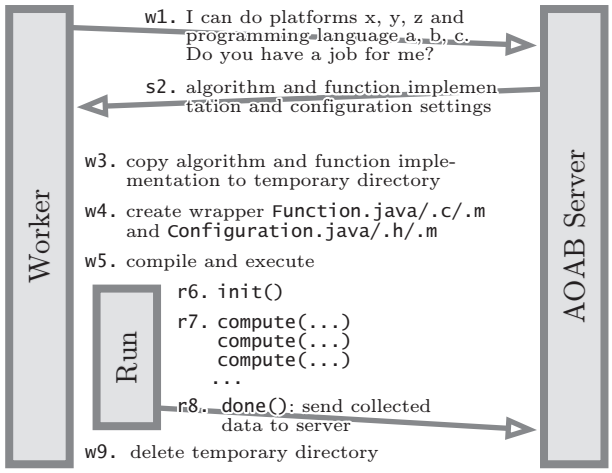

Figure 2: The AOAB way of experiment execution. Prefixes: `s` $\widehat{=}$ step takes place at server, `w` $\widehat{=}$ worker client, `r` $\widehat{=}$ experiment process

3.2 The AOAB Workers

An AOAB worker is a process which occupies exactly one virtual CPU on a computer and is able to request and perform jobs from the AOAB server. It simply runs the procedure illustrated in Figure 2 in a loop. Every worker is characterized by its capabilities. A worker running on a `Windows` machine may, for instance, have detected a `Matlab` and a `Visual C++` installation on startup while failing to find the `Java JDK` needed to compile `Java` programs. As a consequence, it is able to process `Matlab` and `C/Visual C++` jobs, but none which require `Java` or depend on a `Linux` platform.

[3]Notice that algorithm/function upload, benchmark specification, experiment starting, and evaluation may all be performed by different users.

[4]Currently, only full factorial designs [4, 40] are supported. Later, support for other methods such as fractional factorial designs [26] will be added.

A job request (w1 in Figure 2) sent to the AOAB job service contains these capabilities and is either (s2) answered negatively or with a job. A job is defined by all parameter settings of the run to be executed and two zip archives containing the algorithm and benchmark function sources. The worker will then copy the source files into a temporary directory (w3) and create an additional file representing the configuration. In the C programming language, this file would be a header configuration.h containing one #define for each parameter setting (such as #define populationSize 100, #define selectionAlgorithm 0, ...).

Furthermore, the benchmark functions are not used as uploaded but wrapped into code which performs in-memory logging. The worker then compiles the sources to a program (w5) which it subsequently executes. This process is the running the optimization algorithm extended with logging code. When it finishes, it sends all the collected data as a SOAP message to the server. The worker can now delete the temporary directory (w9) and request the next job.

3.3 The AOAB Server and System Robustness

The AOAB server is the center of the system and provides its functionality as web services via a SOAP [25] interface. It is responsible for storing and retrieving experimental results, optimization algorithm, and objective function implementations. It determines which runs have to be performed when an experiment is started by inferring the configurations, the platform and programming language from the parameter settings and the selected algorithm implementation. It furthermore manages a job queue for these runs for the workers.

One interesting feature of all components of AOAB (except for the database, obviously) is that they are stateless. The AOAB server, for instance, keeps all information regarding the experiments and the job queue in the database. If a worker requests a job, the priority of the returned job in the queue will be decreased, but the job itself is not removed from the queue until a corresponding result has been received and successfully stored in the database. Hence, if either the AOAB server or the worker crash (or both), no experimental data or experiment job will be lost. Instead, the worst that can happen is that the job is not executed until all other jobs in the queue have been processed. Furthermore, the AOAB server does not keep track of the available workers or job assignments. Instead, a worker ready to process a job will request one. Therefore, the number of workers can arbitrarily change at any given time, be it by new computers becoming available or crashes. If an uploaded algorithm implementation is faulty, the server may detect that the number of workers involved in the related jobs is suspiciously increasing. Since the AOAB server also does not keep track of connected clients, the same robustness is provided in this area as well.

In our opinion, such an application structure is extremely useful for larger experiments for two reasons: On one hand, it allows the user to utilize all resources in a lab or cluster whenever they become (and for as long as they stay) available. On the other hand, it is often the case that one computer or run fails for an arbitrary reason which, in many systems, would force the user to deal with the experiment-running code or logging data in order to isolate and restart the run [16]. With our system structure, such things become unnecessary and are handled by the system automatically.

```
1  public class Sphere {
2
3    public static double compute(double[] x) {
4      int i;
5      double s, t;
6
7      s = 0d;
8      for (i = Configuration.dimension; (--i) >= 0; ) {
9        t = x[i];
10       s += (t * t);
11     }
12     return s;
13   }
14 }
```

Listing 1: The implementation of the sphere function in Java for AOAB.

4. Source Formats

The core idea of our experimenting approach is that the AOAB workers assemble the programs to be executed on the machine in which they run. This involves placing the sources of the optimization algorithm and the benchmark function to be used in the current run into one temporary directory and to automatically generate some additional source code for logging and providing the configuration parameters. This method has the advantage that issues like data representation, different numeric representations, and processor optimization on different computers are automatically solved. The overhead caused by compiling the programs is relevant only for optimization problems with low dimensions and/or low numbers of maximum allowed function evaluations. In order to be compatible with this concept, algorithm and function implementers have to adhere to a few simple constraints. These restrictions are discussed in the following two sections on basis of examples in the Java language which, however, easily carry over to other languages such as C, C#, Matlab, or Python.

4.1 Implementing Benchmark Functions

Listing 1 contains an example implementation of the simple sphere function in Java for AOAB. In this case, there are two restrictions: *(a)* The main class must contain a function called compute returning a double and taking an array of double as parameter (see line 3 in Listing 1). *(b)* The implementation must obey to the parameters defined by the user in the GUI when uploading it. These will be made available in a auto-generated class called Configuration. A typical example for such a parameter is dimension (see line 8).

4.2 Implementing Optimization Algorithms

The restrictions imposed on implementing algorithms are similar. Listing 2 provides an example implementation of the random sampling [1, 5] algorithm in Java. Here, the following constraints have to be considered: *(a)* The objective function is assumed to exist in a class called Function and has exactly the character mentioned in Section 4.1 (see line 21 of Listing 2). *(b)* Additionally, this class has two more functions init and done which have to be invoked before and after the optimization process, respectively (lines 9 and 23). *(c)* All configuration values such as, for example, the populationSize of an EA, but also the boundaries and dimension of the search space (minX, maxX, dimension), the maximum number of allowed function evaluations, maxFEs (lines 12 to 15), reside as constants

```
1  public class RandomSampling {
2
3    public static void main(String[] params) {
4      final double[] d;
5      int i, j;
6      inal Random r;
7      double v, f;
8
9      Function.init();
10
11     r = new Random();
12     v = (Configuration.maxX - Configuration.minX);
13     d = new double[Configuration.dimension];
14
15     for(i = Configuration.maxFEs; (--i) > 0; ) {
16       for(j = d.length; (--j) >= 0; ) {
17         d[j] = (Configuration.minX +
18                     (v * r.nextDouble()));
19       }
20       //random sampling [1, 5] ignores the fitness
21       f = Function.compute(d);
22     }
23     Function.done();
24   }
25 }
```

Listing 2: The implementation of a random sampling [1, 5] algorithm in `Java` for AOAB.

in the class `Configuration`. Since the algorithm uploader who specifies the parameters of the algorithm implementation usually is also the implementer, adhering to the rules regarding the use of parameters should be natural.

5. Initial Experiments

Major parts of AOAB are still in implementation stage. However, we are already able to conduct experiments and evaluations automatically. In the following we will show some proof-of-concept tests. It should be noted that these experiments were merely performed in order to provide illustrative examples for the capabilities of our system and we did not intend to provide competition benchmark or comparison data of optimization algorithms.

5.1 Setup

We compared three different algorithms: plain random sampling (RS) [1, 5] and two non-adaptive hill climbers (HC-G, HC-C). The random sampling algorithm RS uses all function evaluations (FEs) granted to it to create (independent) random points in the search space via uniform sampling. It remembers the best candidate solution discovered and returns it as result when all FEs are exhausted. A hill climber starts at a random point p in the search space and randomly creates neighbors q around this point. As soon as it discovers a better solution ($f(q) < f(p)$), it transcends to it ($p \leftarrow q$). For sampling the neighborhood, our hill climbers therefore add (independent) random values to each element of the p. In case of HC-G, the Gaussian distribution with standard deviation 1 is used. HC-C generates Cauchy-distributed random numbers via inverse transformation sampling of uniformly distributed random numbers (see Fig. 3.m).

AOAB automatically applied `C++` implementations of these algorithms to the first eight functions from the benchmark used in the 2010 Competition on Large-Scale Global Optimization [36] with `dimension` $\in \{200, 300, 400\}$ and `maxFEs` $= 3\,000\,000$. As prescribed in [36], 25 runs were executed per configuration. Here we present an excerpt of the results. Besides tables with statistics about the performance of optimizers, AOAB can produce elaborate graphical output. In Section 5, we present some of the diagrams AOAB created as evaluation of the experiment[5].

5.2 Performance Diagrams

In [36], the first benchmark function (f_1) is a shifted ellipse. The graphics Fig. 3.a to Fig. 3.c illustrate the performance of the different algorithms when applied to f_1 in terms of best objective value discovered until a certain function evaluation (both axes are logarithmically scaled). During the experiments, AOAB has recorded data according to the interesting point set. For a specific FE, it can thus compute the median, minimum, maximum, and the quantiles (5%, 25%, 75%, and 95%) of the performance of these independent runs. We choose such a representation over an arithmetic mean $\pm 2 * stddev$ plot since we assume that the performances of the runs may not be normally distributed. Fig. 3.d to Fig. 3.f sketch the same information for the fourth benchmark function (f_4) from [36], a rotated and shifted version of the elliptic function. Here, only the FE axis is logarithmically scaled.

Such graphics already provide some initial insight into the behavior of optimization algorithms. Here, it becomes for instance clear that even trivial hill climbers already behave very differently from random sampling. The RS algorithm initially finds some improvements and later needs exponentially increasing time to discover better solutions. The hill climbers, on the other hand, seem to need some time to discover the "right direction" towards the optimum of the (unimodal) objective function and then are able to improve very quickly for some time. Since the algorithms are non-adaptive and sample the neighborhood with constant variance/scale, this improvement ends at some point in time and converges to the RS behavior.

5.3 Statistical Test Result Diagrams

An interesting question in optimization research is not only *whether* a certain algorithm is better than another one, but also *when* it becomes better (or worse, in the opposite case). With graphics of the types used in Fig. 3.g to Fig. 3.l, we want to show how AOAB can help to find answers to this question as well. In most research papers, the authors compare the average results of their algorithms with the average results of some other algorithm in order to determine which is better. However, such a comparison is more or less useless. If we assume that the objective values are continuous and the algorithms do not end up in similar local optima, there will always be differences in the mean performance. Whether these differences are *significant* or not, however, is not a priori clear. It is thus necessary compare the results using statistical tests. The best practice may be the application of non-parametric tests since these make the least assumptions about the distribution of the run performance. AOAB here uses the Mann-Whitney U-test Mann and Whitney [24], Siegel and Castellan Jr. [32] in a two-tailed version with a significance level of 2%.

However, AOAB does not only apply the test at the end of the experiment, but at each recorded time step. This means that we can determine when and for how long a

[5]The graphics have been slightly beautified/zoomed for presentation in this paper and due to the development/non-beautified state of the system.

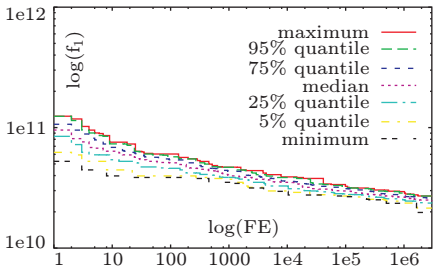

Fig. 3.a: Performance of RS on f_1, `dimension` = 200.

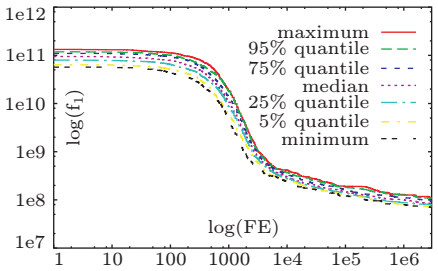

Fig. 3.b: Performance of HC-G on f_1, `dimension` = 200.

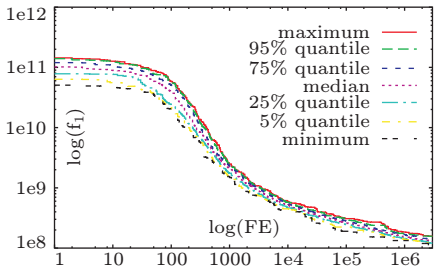

Fig. 3.c: Performance of HC-C on f_1, `dimension` = 200.

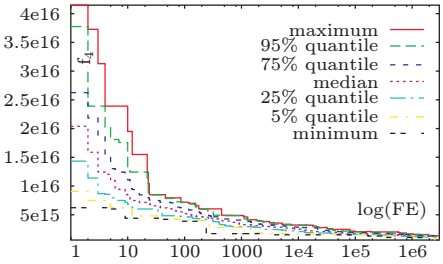

Fig. 3.d: Performance of RS on f_4, `dimension` = 400.

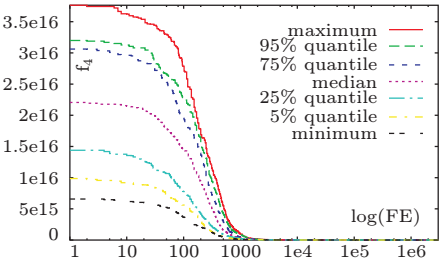

Fig. 3.e: Performance of HC-G on f_4, `dimension` = 400.

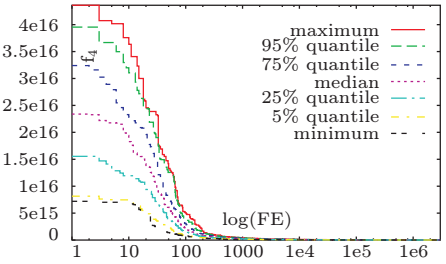

Fig. 3.f: Performance of HC-C on f_4, `dimension` = 400.

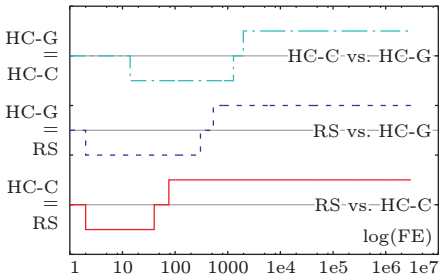

Fig. 3.g: Comparison of the optimizers on f_7, `dimension` = 200.

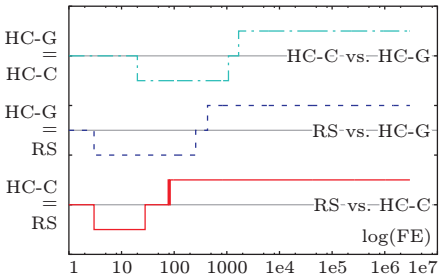

Fig. 3.h: Comparison of the optimizers on f_7, `dimension` = 300.

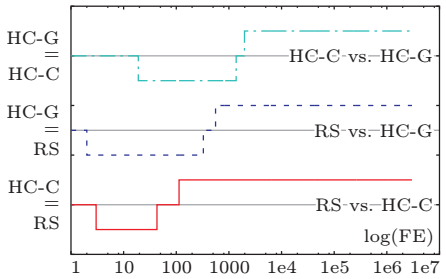

Fig. 3.i: Comparison of the optimizers on f_7, `dimension` = 400.

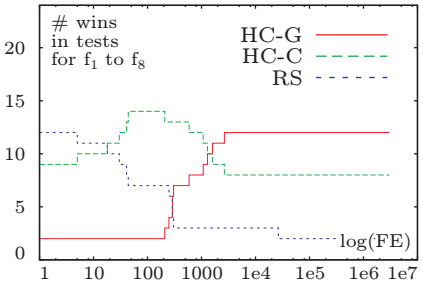

Fig. 3.j: The total test wins in all functions `dimension` = 200.

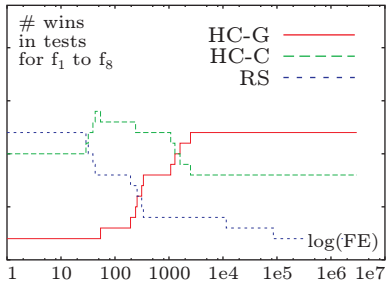

Fig. 3.k: The total test wins in all functions `dimension` = 300.

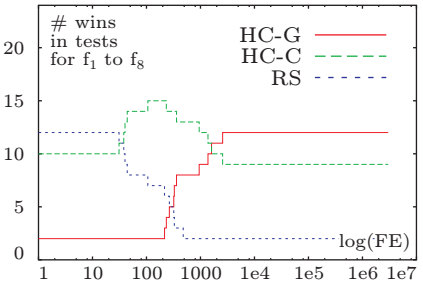

Fig. 3.l: The total test wins in all functions `dimension` = 400.

Histogram of samples from the two random number generators used in the two hill climbing algorithms:

The interval (-7,7) has uniformly been divided into 1000 baskets and each generator has been sampled 1e8 times. The y-axis illustrates the fraction of samples which fell into the baskets. HC-G uses the Gaussian random numbers R_G~N(0,1). In HC-C, Cauchy-distributed random R_C numbers are used, which have been obtained by applying inverse transformation sampling to uniformly distributed random numbers u, i.e., by computing tan(πu).

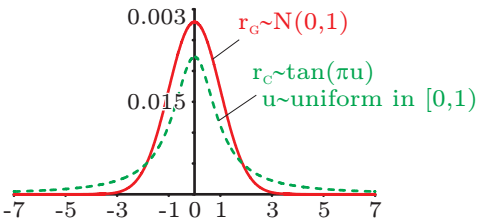

Fig. 3.m: The different distributions used in the hill climbers: R_G in HC-G and R_C in HC-C.

Figure 3: Some graphics produced by AOAB (and manually post-processed and rearranged for presentation in this paper).

certain algorithm is *significantly* better than another one and when we cannot say that both behave differently without making an error with high probability. In Fig. 3.g to Fig. 3.i, we provide the pair-wise comparison results between the three algorithms over the function evaluations for `dimension` $\in \{200, 300, 400\}$ for f_7 (the Single-group Shifted m-dimensional Schwefel's Problem 1.2 combined with the Sphere function [36]). All three diagrams exhibit the same behavior: in the comparison between HC-C and RS (red line), initially there is no significant difference. For the next approximate 70 FEs, the random sampling finds improvements more quickly than the hill climber. When its fast convergence phase is reached, HC-C then outperforms the random search significantly. The relation between HC-G and RS looks quite similar.

More interesting is the comparison between HC-G and HC-C: After neither of the two being better than the other in the startup phase, the Cauchy-distributed neighborhood sampling picks up pace faster and for some time, outperforms HC-G significantly. However, for all settings of the parameter `dimension`, after around 1000 FEs the situation turns around and HC-G suddenly finds the better solutions. For this behavior, we believe the reason is as follows: The Cauchy distribution applied in our algorithm has a heavier tail than the Gaussian distribution used (see Fig. 3.m). The latter is thus more likely to sample points closer to its center which, presumably, leads to shorter search steps. The HC-G algorithm therefore likely needs more search steps when approaching the optimum. However, once a certain proximity to the optimum is reached, the higher probability of HC-G to sample the close neighborhood allows it to find improvements for a longer time than HC-C can. These assumptions are mainly based on the fact that both algorithms are non-adaptive. Sampling parallel to the axes in [15] may play a role in the initially higher speed of HC-C as well.

In Fig. 3.j to Fig. 3.l, we joined all information given by all tests in the different benchmark functions for the FE steps to *score* values. For every comparison an optimizer significantly wins in a step, it receives 1 point. In the eight functions, it could therefore score at most 16 points. The total score sum will always be between 0 and 24 for each step. A high score indicates better performance. The three comparison graphs in our example illustrate that in the beginning, none of our two hill climbers beats random sampling. During the first 1000 steps, HC-C seems to be the algorithm of choice. If more FEs are available, HC-G is the most promising approach in average in the benchmark.

With AOAB, such conclusions can not only be drawn by comparing algorithms, but also by comparing different parameter settings for the same algorithm. If the distribution used for sampling was a parameter choice in a hill climber, for example, the experiment we performed would indicate that switching between Cauchy and Gaussian distributions may be a good idea at some point in time (in an otherwise non-adaptive algorithm, that is).

6. Conclusion and Future Work

In this paper, we presented the architecture of an Automated Optimization Algorithm Benchmarking tool called AOAB. We outlined its ability of running efficient and distributed experimental series. It can not only be used for comparing algorithms, but also for gaining insight into the effects of different parameter settings. Our system requires a minimum learning curve and basically supports arbitrary programming languages. We therefore think that it is an ideal tool for

1. optimization algorithm developers who want to test their approaches as well as for
2. students learning about optimization.

In Section 5, we provide some results produced by AOAB for an initial proof-of-concept experiment and showed how our system can support in drawing conclusions about the performance of different optimizers.

Although the basic building blocks of AOAB are implemented, some of the functionality is still missing and the development of the graphical user interface is still in its early stages. During the next six to twelve months, we aim at providing a ready-to-use version of the system along with an installer. From there on, we plan to use it as aid for teaching courses on global optimization.

We will also implement all the optimization algorithms mentioned in Section 1 for AOAB. Based on the resulting algorithm repository, we will conduct a study on how the behavior of different optimizers changes when the scale of the problems increases. By doing so, we hope to complement existing theoretical work with practical experimental results and, at the same time, open new perspectives in understanding of optimization methods in a way similar to Section 5.3.

Another goal we strive to attain with AOAB in the middle-term future is to federate research on optimization algorithms: A researcher may use her own AOAB server for her experiments. Additionally, a research group may also maintain a central AOAB server. If a researcher has finished a series of experiments and analysis and found an interesting algorithm, she may press a button in her local GUI to upload the algorithm and experiments from her local machine to the groups AOAB server. From this server, in turn, the data may be forwarded to the AOAB server hosted at our institute. Of course, this should also work the other way around, i. e., it should be possible to download benchmarks, functions, and algorithm implementations from a central server to the locally running instances. With such a federation, it would be easy to aggregate and keep track on a giant pool of performance data and to provide automatically updated websites featuring e.g. top-ten lists of algorithms for certain benchmarks.

7. REFERENCES

[1] A. Auger and R. Ros. Benchmarking the Pure Random Search on the BBOB-2009 Testbed. In *11th Annual Conference – Companion on Genetic and Evolutionary Computation Conference (GECCO'09)*, pages 2479–2484, New York, USA, 2009. ACM. 10.1145/1570256.1570347.

[2] R. Battiti and G. Tecchiolli. The Continuous Reactive Tabu Search: Blending Combinatorial Optimization and Stochastic Search for Global Optimization. *Annals of Operations Research*, 63(2):151–188, 1996.

[3] S. Boettcher and A. G. Percus. Extremal Optimization: Methods Derived from Co-Evolution. In *Genetic and Evolutionary Computation Conference (GECCO'99)*, pages 825–832. Morgan Kaufmann Publishers Inc., 1999.

[4] G. E. P. Box, J. S. Hunter, and W. G. Hunter. *Statistics for Experimenters: Design, Innovation, and Discovery.* John Wiley & Sons Ltd., New York, USA, 1978.

[5] S. H. Brooks. A Discussion of Random Methods for Seeking Maxima. *Operations Research*, 6(2):244–251, 1958. 10.1287/opre.6.2.244.

[6] J. Brownlee. OAT HowTo: High-Level Domain, Problem, and Algorithm Implementation. Technical Report 20071218A, Swinburne University of Technology, Melbourne, Australia, 2007.

[7] J. Brownlee. OAT: The Optimization Algorithm Toolkit. Technical Report 20071220A, Complex Intelligent Systems Laboratory, Swinburne University of Technology, Melbourne, Australia, 2007.

[8] F. L. de Sousa and F. M. Ramos. Function Optimization using Extremal Dynamics. In *4th International Conference on Inverse Problems in Engineering: Theory and Practice – Inverse Problems in Engineering: Theory and Practice (ICIPE'02)*, New York, USA, 2002. UEF. 10.1080/089161502753341870.

[9] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 26(1):29–41, 1996. 10.1109/3477.484436.

[10] R. C. Eberhart and J. Kennedy. A New Optimizer Using Particle Swarm Theory. In *Sixth International Symposium on Micro Machine and Human Science (MHS'95)*, pages 39–43, Piscataway, NJ, USA, 1995. IEEE Computer Society. 10.1109/MHS.1995.494249.

[11] L. J. Fogel. *On the Organization of Intellect*. PhD thesis, University of California (UCLA), LA, CA, USA, 1964.

[12] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers & Operations Research*, 13(5):533–549, 1986. 10.1016/0305-0548(86)90048-1.

[13] L. S. Gurin and L. A. Rastrigin. Convergence of the Random Search Method in the Presence of Noise. *Automation and Remote Control*, 26:1505–1511, 1965.

[14] N. Hansen and A. Ostermeier. Convergence Properties of Evolution Strategies with the Derandomized Covariance Matrix Adaption: The $(\mu/\mu_I, \lambda)$-CMA-ES. In *5th European Congress on Intelligent Techniques and Soft Computing (EUFIT'97)*, volume 1, pages 650–654, Aachen, Germany, 1997. ELITE Foundation.

[15] N. Hansen, F. Gemperle, A. Auger, and P. Koumoutsakos. When Do Heavy-Tail Distributions Help? In *Proceedings of 9th International Conference on Parallel Problem Solving from Nature (PPSN IX)*, volume 4193/2006 of *Lecture Notes in Computer Science (LNCS)*, pages 62–71, Berlin, Germany, 2006. Springer-Verlag GmbH.

[16] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-Parameter Black-Box Optimization Benchmarking 2009: Experimental Setup. Rapports de Recherche RR-6828, INRIA, 2009.

[17] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-Parameter Black-Box Optimization Benchmarking 2010: Experimental Setup. Rapports de Recherche 7215, INRIA, 2010.

[18] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, Ann Arbor, MI, USA, 1975.

[19] V. L. Huang, A. K. Qin, K. Deb, E. Zitzler, P. N. Suganthan, J. J. Liang, M. Preuß, and S. Huband. Problem Definitions for Performance Assessment of Multi-objective Optimization Algorithms, Special Session on Constrained Real-Parameter Optimization. Technical report, Nanyang Technological University (NTU), Singapore, 2007.

[20] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation. Part I, Graph Partitioning. *Operations Research*, 37(6), 1989. 10.1287/opre.37.6.865.

[21] P. Korošec and J. Šilc. Real-Parameter Optimization Using Stigmergy. In *Second International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2006)*, Informacijska Družba (Information Society), pages 73–84, Ljubljana, Slovenia, 2006. Jožef Stefan Institute.

[22] C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H.-G. Beyer, and P. N. Suganthan. Benchmark Generator for CEC'2009 Competition on Dynamic Optimization. Technical report, University of Leicester, Department of Computer Science, Leicester, UK, 2008.

[23] J. J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. N. Suganthan, C. A. Coello Coello, and K. Deb. Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-Parameter Optimization. Technical report, Nanyang Technological University (NTU), Singapore, 2006.

[24] H. B. Mann and D. R. Whitney. On a Test of whether One of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947. 10.1214/aoms/1177730491.

[25] N. Mitra and Y. Lafon. *SOAP Version 1.2 Part 0: Primer (Second Edition)*. MIT, ERCIM, Keio University, 2007.

[26] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons Ltd., New York, USA, 1991.

[27] P. Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report C3P 826, Caltech Concurrent Computation Program, California Institute of Technology (Caltech), Pasadena, CA, USA, 1989.

[28] J. A. Nelder and R. A. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4): 308–313, 1965. 10.1093/comjnl/7.4.308.

[29] M. Pelikan, D. E. Goldberg, and F. G. Lobo. A Survey of Optimization by Building and Using Probabilistic Models. IlliGAL Report 99018, Illinois Genetic Algorithms Laboratory (IlliGAL), University of Illinois at Urbana-Champaign, Urbana, IL, USA, 1999.

[30] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Technische Universität Berlin, Berlin, Germany, 1971/1973.

[31] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (AIMA)*. Prentice Hall International Inc., Upper Saddle River, NJ, USA, 2nd edition, 2002.

[32] S. Siegel and N. J. Castellan Jr. *Nonparametric Statistics for The Behavioral Sciences*. Humanities/Social Sciences/ Languages. McGraw-Hill, New York, USA, 1956/1988.

[33] R. M. Storn and K. V. Price. Differential Evolution – A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Technical Report TR-95-012, International Computer Science Institute, Berkely, CA, USA, 1995.

[34] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Technical Report May-30-05, Nanyang Technological University (NTU), Singapore, 2005.

[35] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y.-P. Chen, C.-M. Chen, and Z. Yang. Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization. Technical report, University of Science and Technology of China, Nature Inspired Computation and Applications Laboratory (NICAL), 2007.

[36] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise. Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization. Technical report, University of Science and Technology of China, Nature Inspired Computation and Applications Laboratory (NICAL), Héféi, Ānhuī, China, 2010.

[37] T. Weise. *Global Optimization Algorithms – Theory and Application*. self-published, 2009. URL `http://www.it-weise.de/`.

[38] D. H. Wolpert and W. G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 1(1):67–82, 1997.

[39] X. Yao, Y. Liu, and G. Lin. Evolutionary Programming Made Faster. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 3(2):82–102, 1999.

[40] F. Yates. *The Design and Analysis of Factorial Experiments*. Imperial Bureau of Soil Science, Commonwealth Agricultural Bureaux, Harpenden, England, UK, 1937.