From Standardized Data Formats to Standardized Tools for Optimization Algorithm Benchmarking

Thomas Weise
Institute of Applied Optimization, Faculty of Computer Science and Technology,
Hefei University, Hefei 230601, Anhui, China
tweise@hfuu.edu.cn

***Abstract*—Benchmarking, the empirical algorithm performance comparison, is usually the only feasible way to find which algorithm is good for a given problem. Benchmarking consists of two steps: First, the algorithms are applied to the benchmarking problems and data is collected. Second, the collected data is evaluated. There is little guidance for the first and a lack of tool support for the second step. Researchers investigating new problems need to implement both data collection and evaluation. We make the case for defining standard directory structures and file formats for the performance data and metadata of experiments with optimization algorithms. Such formats must be easy to read, write, and to incorporate into existing setups. They would allow more general tools to emerge. Researchers then would no longer need to implement their own evaluation programs. We derive suitable formats by analyzing what existing tools do and what information they need. We present a general tool, the *optimizationBenchmarking.org* framework, including an open source library for reading and writing data in our format. Since our framework obtains its data from a general file format, it can assess the performance of arbitrary algorithms implemented in arbitrary programming languages on arbitrary single-objective optimization problems.**

I. Introduction

Optimization is a field of incredible versatility with many application areas, including logistics, supply chain management, production scheduling, finances, engineering design, data mining, medicine, and algorithm synthesis – to just name a few. For almost every real-world application of optimization, a new approach needs to be designed as any scenario has specific requirements [2].

This naturally leads to the question which approach is best for a certain application. Answering such a question based on mathematics and theoretical computer science is rarely possible [3], leaving experimentation and benchmarking as the technology of choice.

Due to the wide variety of problem types, a wide variety of benchmark problems has evolved. During the last decade, a set of tools for assessing algorithm performance has emerged as well. These tools are usually bound to a certain problem or algorithm type. This means that the vast majority of statistics over experimental results reported in papers has been obtained manually by researchers, either by writing their own set of problem-specific tools or by only relying on simple (and often brittle) sample statistics and using software such as Excel.

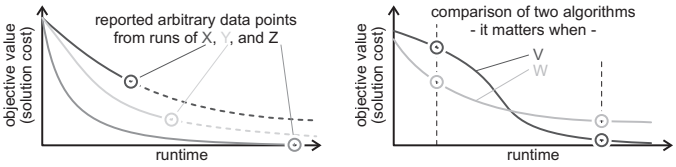
We believe that the key to developing tools which are applicable to many different problem and algorithm types is to define simple standard file formats. We propose the following requirements:

1) Data in such formats must be easy to read or write, both programmatically as well as manually.
2) It must be very simple to modify existing experimental setups to produce data in these formats.
3) It must be very simple to read and write data in such formats using existing tools such as Excel, R, and Matlab.
4) It must be possible to represent both raw data gathered from experiments and metadata such as algorithm parameters and instance features. While the former may be sufficient for descriptive statistics, inferential statistics require the latter as well.

Such file formats can improve the rigor in experimentation, as they indirectly guide researchers to collect data sufficient for a sound analysis. They furthermore make it much easier to exchange data. We analyze the requirements for such formats for both raw measurements and experiment metadata in-depth. We provide example solutions for these requirements based on our own *optimizationBenchmarking.org* framework, showing that a tool working with such formats by default can assess the performance of almost arbitrary algorithms implemented in arbitrary programming languages applied to arbitrary single-objective problems.

We first discuss *a)* the need to measure more than just "final" results and *b)* the advantages of using multiple time measures during experimentation in Section II. From these requirements, we derive a simple text-based format (space-separated values) to represent the measurements from single runs of optimization algorithms. We complement this format with a specification of what time measures were applied in XML [4]. Since both formats are text-based and to be evaluated by tools after the experiments, the programming languages used to implement algorithms becomes irrelevant. Producing data in these formats is also very easy.

In Section III, we list several existing tools for visualizing performance data and for providing descriptive statistics. Given the above information, these or similar tools could be



(a) Based on the reported results, methods X, Y, Z may be misinterpreted as alternatives for different computational budgets in the Pareto sense, but Z is always better.

(b) Comparing two anytime algorithms at only one point in time might yield entirely different results, depending on how that point is chosen.

Fig. 1: Example for problems with algorithm comparisons based on single (runtime, objective value) tuples.

applied to more general datasets if the location of the data files was clear. Hence, we derive a simple folder structure so that a tool can find the data gathered in an experiment regardless of the type of algorithm applied and regardless of the problem under investigation.

The main goal of research is to find the reasons that cause the phenomena observed by the scientist. In the context of research on optimization, this could mean, for instance, to suggest which feature renders a problem hard or why a particular algorithm setup performs well. Besides our *optimizationBenchmarking.org* framework, there are few tools in this line of research and most of them consider only final results from experiments or can only deal with different setups of a single algorithm, as outlined Section IV. Nevertheless, such tools require meta-information about the experiments, namely the problem instance features and algorithm parameters. We discuss how these information can be represented in an XML format.

In Section V, we give a more formal definition of the above derived data structures. We then describe how our *optimizationBenchmarking.org* framework works with such data in Section VI. This framework is open source and provides a Java library for reading and writing all involved formats. We summarize our discussions and give pointers to future work in Section VII.

II. Performance Data of Single-Objective Algorithms

A. Experimentation with Anytime Algorithms

A very large fraction of the available optimization algorithms are *anytime algorithms* [5]: Evolutionary Algorithms (EAs) [6, 7], Local Search (LS) [7, 8], Ant Colony Optimization (ACO) [7, 9], and even some exact methods like Branch and Bound (B&B) [10, 11] belong to this category. Anytime algorithms can provide an approximate solution for a problem at any point during their execution. The quality of this approximation may improve over time. Many machine learning approaches proceed in such a manner as well. This includes algorithms that involve iterative learning, such as backpropagation [12, 13], or algorithms which iteratively refine their data structures as done in, e.g., k-means clustering [14]. Due to this anytime character, it is possible that an algorithm V initially produces worse solutions than an algorithm W, but eventually reaches better approximations. Yet, many if not most publications report only statistics based on results measured at a *selected* single point in the algorithm execution, which cannot represent such situations. This may lead to problems such as those shown in Figure 1 and pointed out in [15–17] for optimization and in [18] for machine learning.

Besides these problems, it is also methodically wrong: Metaheuristics are usually applied to problems where either the objective value of the global optimum is not known and often even a lower bound cannot be provided and/or in cases where solving the problem exactly takes too long. This means we can often not expect to find the global optimum and in many problems (in particular those proven to be $\mathcal{NP}$-hard), the runtime needed actually find the global optimum would be way too long. In other words, for any finite computational budget, we would hope that optimization process would keep improving. While the improvements may become exponentially smaller (similar to the law of *diminishing returns* [19]), we would hope that they do exist. In such a scenario, there is no natural "end" and the decision when to actually stop the optimization process becomes arbitrary, often based on the computational resources available to the particular experimenter. Listing "results" would imply some non-existing form of finality and unintentionally obscure this reality.

In order to make robust decisions which algorithms to use for varying computational budgets, sufficient data to reasonably-well approximate the "runtime-approximation quality" relationships of optimization processes should be collected. Based on such data, it can also become visible when further improvements become negligible. Having only singular results forces us to trust that the experimenter hopefully has made the right decisions (and did not intentionally or unintentionally create situations as depicted in Figure 1).

B. Measuring Runtime

This leads to the question of how runtime should be measured. We can distinguish machine-dependent and machine-independent ways for doing so. Traditionally, runtime is either measured in CPU seconds or the number of generated candidate solutions (i.e., objective "function evaluations", or FEs in short). The problem with using CPU time is that it is strongly machine-dependent, being influenced by the hardware and software setup, and that results obtained on different machines are thus inherently incomparable. FEs, on the other hand, are a machine-independent time measure and results provided in terms of FEs will remain valid. However, unlike the computational complexity of an algorithm, which they are intended to somewhat approximate empirically, FEs cannot give information about the actual overall runtime of an algorithm, since they do not include any setup and overhead time and because 1 FE may have different computational complexities in different algorithms.

Let us discuss the latter issue on the example of the Traveling Salesman Problem (TSP), where the goal is to find the shortest round-trip visiting sequence of n cities [17]. Here,

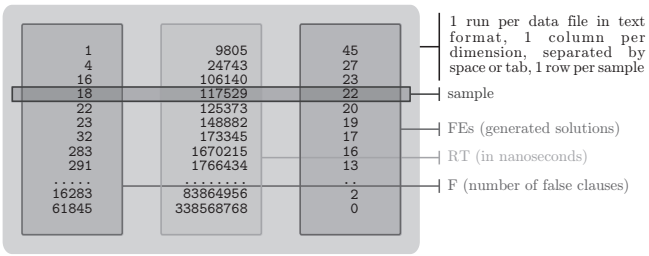


Fig. 2: One example of a text file with data points describing the progress of one run of a hill climbing algorithm on a maximum satisfiability problem instance.

```
<?xml version="1.0" encoding="UTF-8"?>
<dimensions xmlns="http://www.optimizationBenchmarking.org/formats/...">

  <dimension name="FEs"
    description="The number of function evaluations, i.e., the amount of
        generated candidate solutions."
    dimensionType="iterationFE" direction="increasingStrictly" dataType="long"
    iLowerBound="1" />

  <dimension name="RT" description="The elapsed runtime in nanoseconds."
    dimensionType="runtimeCPU" direction="increasing" dataType="long"
    iLowerBound="0" />

  <dimension name="F" description="The number of unsatisfied clauses."
    dimensionType="qualityProblemDependent" direction="decreasing"
    dataType="int" iLowerBound="0" iUpperBound="2000" />

</dimensions>
```

Fig. 3: An example of the dimension annotation with the *optimizationBenchmarking.org* XML format EDI.

an LS algorithm may obtain a new solution from an existing tour of known length by swapping two cities, which has the complexity of $\mathcal{O}(1)$. A crossover operator in an EA would usually create a new tour in a more complicated way and thus need to calculate the length of that tour, i.e., would usually need at least $\mathcal{O}(n)$ steps [20]. In ACO for the TSP, the creation of one new solution has time complexity $\mathcal{O}(n^2)$. In other words, if the ACO and LS would reach the same tour length in the same number of FEs, saying that they are as same as good would be grossly unfair towards the LS. This issue is surely not limited to the TSP, but we have never seen it addressed in any publication basing its results on FEs.

One way to make runtime measurements "more" comparable over different machines, at least to a certain degree, would be to normalize them by machine- and problem-specific performance indicators [17]. One way to obtain better machine-independent measures would be to develop finer-grained, problem-specific time measures, such as the number of distance matrix accesses for the TSP [17] or the number of bit flips for Boolean satisfiability problems [21]. Using both machine-dependent and independent time measures at once might be the best solution.

C. Data Format

We find that it is necessary to gather data describing the progress of an optimization method over time. We also find that measuring time with only time measure will potentially not tell us the whole story [17, 22].

1) Raw Experimental Data: A data format for describing one run of one optimization algorithm on one problem instance should allow us to collect multiple data points. Each data point should represent the solution quality reached (obviously monotonously improving)[1] and the time consumed (monotonously increasing). While a single-objective problem demands for a single quality value per data point, there might be multiple time values, namely one per time measure (e.g., consumed runtime in ms, consumed runtime in FEs, ...). Since this is a very simple structure, using simple text files to store such data, one file per algorithm run per problem instance, as sketched in Figure 2, seems to be a reasonable choice. The columns inside the text files are separated by spaces or tabs. The last column should be the column with the objective values. Such files are easy to read and to write and can be parsed by common tools such as R, Matlab, and Excel.

2) Meta-Data about Measurement Dimensions: While this format itself is sufficiently general, it may also make sense to provide additional metadata. Since there might be arbitrarily many columns, as we can measure time in several different ways, the metadata should specify which column stands for which dimension and maybe assign a name to a column. Our *optimizationBenchmarking.org* framework therefore uses an XML dialect called *Experiment Data Interchange* (EDI) [23] which can provide these information, but also allows us to specify the monotonous "direction" of the dimension, its basic data type, and value range limits. Figure 3 provides the metadata file `dimensions.xml` accompanying the experiment from which also Figure 2 stems.

In the *optimizationBenchmarking.org* framework, dimension names can be used in mathematical expressions, allowing arbitrary transformations of axes and specifying which data columns to use in certain diagrams. The metadata is also used to create compact data structures with very fast lookup and search functionality on the fly via in-memory code generation and for strict sanity checks. A researcher may choose to write such a `dimensions.xml` file manually or by using the graphical editor of the *optimizationBenchmarking.org* GUI requiring no XML knowledge.

Still, providing the dimension information is optional: It is not required for simple evaluations, especially in the case of a single time dimension. This means we propose a set of data formats which can complement each other, but a researcher may also choose to stick to a very simple baseline.

III. Mining Descriptive Information

A. Existing Software Tools

As we have discussed in the previous section, if the whole runtime behavior of algorithms should be compared, quite a

[1]While it is true that several optimization methods such as Simulated Annealing or generational EAs with, e.g., (μ, λ)-strategies, may "lose" the best solution found so far *from their population*, this holds only from the perspective of the optimization algorithm. Any sane implementation would, in such cases, remember the best solution in an additional variable not used inside the algorithm, in order to eventually return it to the user. This variable would just not influence the algorithm in any way, i.e., should not be mistaken for *elitism*.

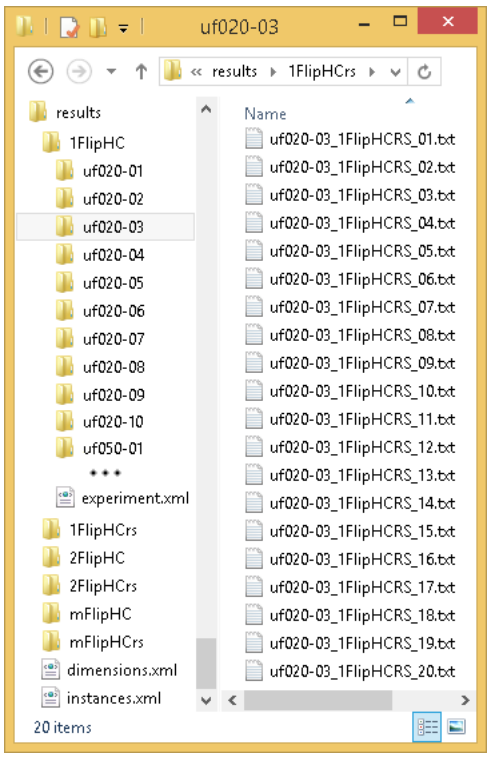


Fig. 4: An example of a general folder structure for experiments.

lot of data needs to be collected in the experiments. Extracting all necessary information from this data by hand will no longer be feasible [17]. Several automated tools have been developed to compute statistics and draw diagrams based on such comprehensive datasets.

The COCO system [24] for numerical optimization, used in the Black-Box Optimization Benchmarking (BBOB) workshops, is one of the first such approaches. Its evaluation procedure generates statically structured papers that contain diagrams such as Empirical Cumulative Distribution Functions (ECDFs) [24, 25] and Estimated Running Time (ERT) [24] charts. The necessary data is automatically collected from automatically executed experiments. The no longer supported AOAB [26] system provided similar functionality for algorithms solving large-scale numerical problems.

The TSP Suite [17] is a holistic experimentation framework for the TSP. Its evaluation process makes use of diagrams similar to those in COCO. All selected evaluation criteria are combined into algorithm rankings and text-based discussions and conclusions are generated, resulting in comprehensive reports instead of rigidly structured papers. Another feature of the TSP Suite is that it measures runtime in several different ways at once, in order to analyze both the machine-independent and machine-dependent algorithm runtime behavior, as proposed in Section II-B.

B. Data Format

The existing tools listed in the selection above all are applicable only to specific problems. However, these tools could in principle work with data provided as specified in Section II-C. One obstacle is that they require data to be stored in a specific structure of file names and folders. This structure would be different when conducting experiments with arbitrary algorithms on arbitrary benchmark instances.

This problem can easily be solved by prescribing a folder structure as follows: In the root folder of the experiment data, there be the file `dimensions.xml` (see Figure 3). Furthermore, for each algorithm setup, one folder with a corresponding name be created. Inside this folder, there be one sub-folder for each benchmark instance. In such a sub-folder, there be one text file of arbitrary name with one column per measurement dimension and one data sample per row, as sketched in Figure 2. Figure 4 illustrates such a structure for an experiment where six algorithms (`1FlipHC` to `mFlipHCrs`) were compared on nine benchmark instances (`uf020-01` to `uf020-09`) and twenty runs were conducted per algorithm/instance combination. This structure is again easy to implement, intuitive to understand, and easy to process by tools.

IV. Mining Inferential Information

A. Existing Tool Support

The work of a researcher includes inference from and generalization of the phenomena observed in experiments. This involves finding the reasons for these phenomena as well as predicting possible outcomes in yet untested scenarios.

Predicting algorithm performance [27, 28] is a common task in algorithm selection [29, 30] and algorithm portfolios [31, 32]. Here, the goal often is to choose the algorithm most suitable to solve a given problem in terms of a single performance metric. Usually, it is attempted to either

1) predict this metric for each algorithm in the portfolio based on the features of the problem instance to be solved or to
2) classify problem instances based on their features, while knowing from previous experiments which algorithm is likely to be most suitable for a given class.

The hardness of a problem instance can be assessed based on the performance of an algorithm [33] on it. A logical continuation and combination of algorithm performance and instance hardness prediction is provided by Hutter *et al.* [28]. They propose an automated way to identify the key parameters of algorithms, the key features of problem instances and their interactions, and to predict parameter values which are likely to yield good performance.

Fawcett and Hoos [34] developed an automated method to explain which parameters contribute the most to the difference between two existing setups in terms of a single performance metric. This is particularly useful if one of the setup is the result of automated parameter tuning (see, e.g., [35]), but requires that the two setups stem from the same algorithm.

Smith-Miles *et al.* [16, 36, 37] generate diverse benchmark instances of a given problem type with an EA. They can then cluster the instances by the search effort required by the investigated solvers [37] and predict areas (called the footprint) in the instance feature space where an algorithm performs well [16]. The power of the solvers can be measured by comparing the overlap and relative size of their footprints. The impact on benchmark features on the performance labels gets analyzed by using machine learning.

Qi *et al.* [38] first model the performance of an optimization algorithm and then mine the relationship of the model parameters to both, algorithm behaviors and instance hardness. They can predict the behavior of algorithms on unseen problem instances based on the modeled behavior on already analyzed on instances by learning the mapping of instance features to model parameters.

B. Data Format

The data collection scheme we prescribe so far would be feasible for these tools as well. However, in order to discover the impact of an algorithm setup parameter or of a benchmark instance feature, a system for inferential analysis must additionally know about them. We assume that there may be arbitrarily many problem instance features and algorithm parameters, each being numerical or nominal.

If a problem instance feature is defined, it is clear that a value for this feature must exist for each instance of the problem. In the TSP, the number n of cities could be a feature and clearly this value is known for each TSP instance. Features could also be descriptive statistics on the instance structure: Another feature of TSPs could be the coefficient of variation of the distances between any two cities, which clearly is defined for any TSP instance with $n > 1$ cities.

For algorithm parameters, the situation is slightly different: An EA can have a parameter "crossover operator" which would be undefined in a LS. Therefore, we propose to introduce the special parameter value $\emptyset$ to be used in such situations (many machine learning algorithms can handle unspecified values).

This also raises the question how to deal with optimization processes measured from different algorithms. We propose to solve this issue by introducing the nominal parameter "algorithm" identifying the applied algorithm. Considering the algorithm itself as a parameter of an experiment has the effect that it can become subject to inferential analysis just like any other algorithm parameter, i.e., requires no special treatment.

Assume that we conduct experiments on a satisfiability problem with a LS and an EA, both using one of the two different search operators, "one-flip" and "two-flip". The inferential analysis might then find, for instance, that the choice of search operator is more important than the choice of algorithm, vice versa, or the one operator works well with one algorithm while other setups do not perform well.

Based on these considerations, we notice an interesting duality: Algorithms have parameters, instances have features. If a software tool can discover the feature setting that makes benchmark problems hard (easy) for the investigated algorithms, the very same tool could be used to discover which algorithm parameter settings lead to bad (good) algorithm performance. Both cases are two sides of the same medal, as both influence the runtime-quality relationships of optimization processes.

```
<?xml version="1.0" encoding="UTF-8"?>
<instances xmlns="http://www.optimizationBenchmarking.org/formats/...">
  <instance name="uf020-01"
    description="A␣uniformly␣randomly␣generated␣satisfiable␣3-SAT␣instance␣
        with␣20␣variables␣and␣91␣clauses.">
    <feature name="n" value="20" />
    <feature name="k" value="91" />
  </instance>
...
  <instance name="uf050-01"
    description="A␣uniformly␣randomly␣generated␣satisfiable␣3-SAT␣instance␣
        with␣50␣variables␣and␣218␣clauses.">
    <feature name="n" value="50" />
    <feature name="k" value="218" />
  </instance>
....
  <instance name="uf075-01"
    description="A␣uniformly␣randomly␣generated␣satisfiable␣3-SAT␣instance␣
        with␣75␣variables␣and␣325␣clauses.">
    <feature name="n" value="75" />
    <feature name="k" value="325" />
  </instance>
...
</instances>
```

Fig. 5: An example of the benchmark instance feature specification with the *optimizationBenchmarking.org* XML format EDI [23], as to be placed into the root data folder as sketched in Figure 4.

```
<?xml version="1.0" encoding="UTF-8"?>
<experiment
  xmlns="http://www.optimizationBenchmarking.org/formats/..."
  name="1FlipHC" description="An␣experiment␣with␣a␣1-flip␣Hill␣Climber␣
      without␣restarts.">
  <parameter name="algorithm" value="HC" />
  <parameter name="operator" value="1-flip" />
  <parameter name="restart" value="false" />
</experiment>
```

Fig. 6: An example of the algorithm setup parameter specification with the *optimizationBenchmarking.org* XML format EDI [23], one of which would be to be placed into each algorithm setup folder, as sketched in Figure 4.

The EDI format of our *optimizationBenchmarking.org* framework allows specifying both algorithm parameters and instance features as discussed above. A file named `instances.xml` should be placed in the root folder (see Section III-B) and it defines the feature values for each benchmark problem instance. The names of the benchmark instances used in this file must be exactly (case-sensitive) the same as the names of the instance folders in the folder structure. An example is given in Figure 5. Besides specifying instance features, this format also allows for specifying lower and upper bounds for each measurement dimension for an instance, which, e.g., can be used to specify the objective values of global optima, if any.

Similarly, each folder for an algorithm setup can contain one file `experiment.xml` describing the parameters of the algorithm applied (and the algorithm itself as parameter). This structure allows for re-using benchmark instance descriptions and for easily adding new algorithm setups. Using it is entirely optional, i.e., allows the researcher to choose whether she wants to provide the metadata to use more sophisticated tools

or not. An example is given in Figure 6

V. Formal Description

Let us now describe the anecdotally derived data format from the previous three sections in a more formal way. Experiments are performed with a set A or setups of algorithms $a \in A$. An algorithm setup a is characterized by a vector $\vec{p}(a) = (p_1(a), p_2(a), \ldots, p_{n_P}(a))$ of values $p(a)$ of parameters p. A parameter may be nominal or numerical. The space of all parameter vectors from a set of experiments be $\mathbb{P}$.

If multiple different algorithms are investigated, the algorithm applied in an experiment can be identified by a nominal parameter. This way, the same analysis methods used to assess the impact of the algorithm parameters can be applied to assess the impact of the choice of algorithm itself. Since different algorithms may have different parameters, a parameter can also take on the unspecified value $\emptyset$. We refer to "algorithm setup" as *algorithm* synonymously.

Each algorithm is applied to a set I of problem instances $i \in I$. A problem instance is characterized by a vector $\vec{f}(i) = (f_1(i), f_2(i), \ldots, f_{n_F}(i))$ of values $f(i)$ of features f. A feature may be nominal or numerical. The space of all feature vectors be $\mathbb{F}$. Different from parameters, all features must be specified for all instances.

Instance features and algorithm parameters are similar from the perspective of analysis. If we can analyze the impact of an algorithm parameter p on the performance based on the observed behavior on all problem instances $i \in I$, we can apply the same method to analyze the impact of a problem feature f based on the behavior of all algorithms $a \in A$.

In the experiments, data is collected during the execution of several independent runs $r_{a,i}$ for each algorithm $a \in A$ on each problem instance $i \in I$. The number of collected samples ρ may differ from run to run. Samples may be collected at specific points in time, e.g., after certain amounts of FEs [39, 40] or milliseconds have passed, when certain goal solution qualities have been reached [24], or all of the above [17], which we consider the best idea.

We limit our considerations to two types of measurement dimensions, namely *time* and *quality* measures. T be the set of time dimensions $t \in T$. At least one time dimension must be measured in the experiments. Q be the set of solution quality dimensions measured in the experiments. A quality dimension $q \in Q$ represents the values of an objective function. We assume that an optimization algorithm never "loses" its best-so-far solution. (A generational EA could store its best solution in an additional variable even if it drops out of the population.) We limit our considerations to single-objective problems with $|Q| = 1$. Other types of dimensions such as the memory consumed by the algorithm are not considered at the moment.

At first glance, this way of representing algorithm runtime behavior may seems to be similar to time series [41, 42], but there are significant differences: There can be multiple time dimensions, multiple runs form one semantic unit, and the measured samples may not be equidistant in any dimension. More similarities are shared with the functional data [43, 44] concept, although the overall structure of runs belonging to algorithm setups applied to problem instances and the need to also consider the metadata makes the inferential analysis of algorithm behaviors significantly more complex and challenging. Evaluating optimization algorithms in a robust, rigorous, methodologically clean, and statistically sound way is indeed an endeavor bringing us right to the frontier of statistics and machine learning!

VI. Implementation: optimizationBenchmarking.org

In 2014, we devised an open source benchmarking system for the TSP called the TSP Suite [3, 11, 17]. This framework enabled the benchmarking and sound comparison of different TSP solvers. We now have developed a generalized software, the *optimizationBenchmarking.org* framework, which can be applied to *arbitrary* optimization problems to compare *arbitrary* algorithms. We found that the enabler of such a general approach is a clear and simple and versatile file format as introduced in this paper.

An overview of the data evaluation procedure of our framework is given in Figure 7. The system operates on the data formats introduced here and the directory structure given in Figure 4. All XML/EDI files needed can be generated using a comfortable editor form provided by the user interface (Figure 8) of our system, which does not require any previous knowledge of or direct contact with XML.

Given all these data, the user can now choose what information she would like to obtain by selecting and configuring any number of the available evaluation modules (`evaluation.xml`). After selecting suitable output (LaTeX, XHTML, text export) and graphics (pdf, eps, svg, png, ...) format and data locations, she can start the evaluation procedure. As a result, she will obtain a report, not unlike a small article, containing textual descriptions of the extracted results and automated findings.

The *optimizationBenchmarking.org* framework is an open source software provided at `http://optimizationBenchmarking.github.io`. It comes either as Java executable, which can automatically detect and use LaTeX and `R` installations under both Windows and Linux, or as Docker [45] image, which does not require any additional installations and can be used under Linux, Windows, and Mac OS. A web-based user interface which can be accessed via any common browser is provided.

The framework is designed in a modular way and divided into several libraries which are managed at https://www.github.com/optimizationBenchmarking. The library `evaluator-base` provides I/O modules for reading, manipulating, and writing EDI data.

VII. Conclusions

Benchmarking of algorithms is important. What makes benchmark results *good*? From the perspective of research,

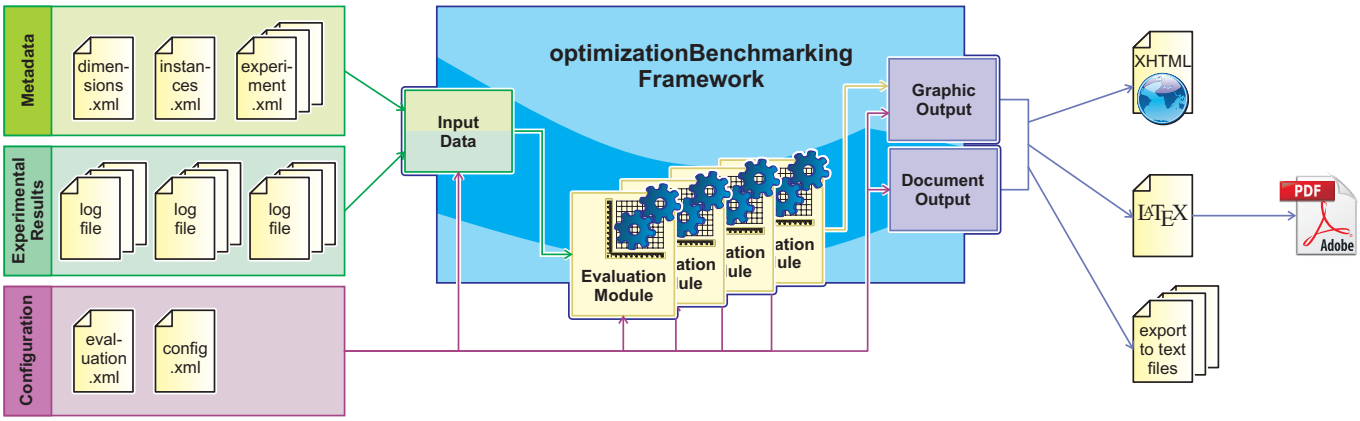


Fig. 7: The flow of using the *optimizationBenchmarking.org* framework: The data measured from runs R for each algorithm $a \in A$ applied to each problem instance $i \in I$ are provided as text files. The metadata, i.e., the parameters, features, and the properties of the time dimensions $t \in T$ and quality dimension q can be specified in the XML format EDI [23]. Via additional XML files, the user can choose the evaluation modules to apply as well as the graphics and output formats. For the latter, LaTeX, XHTML, and a text format for data export to other application is supported.

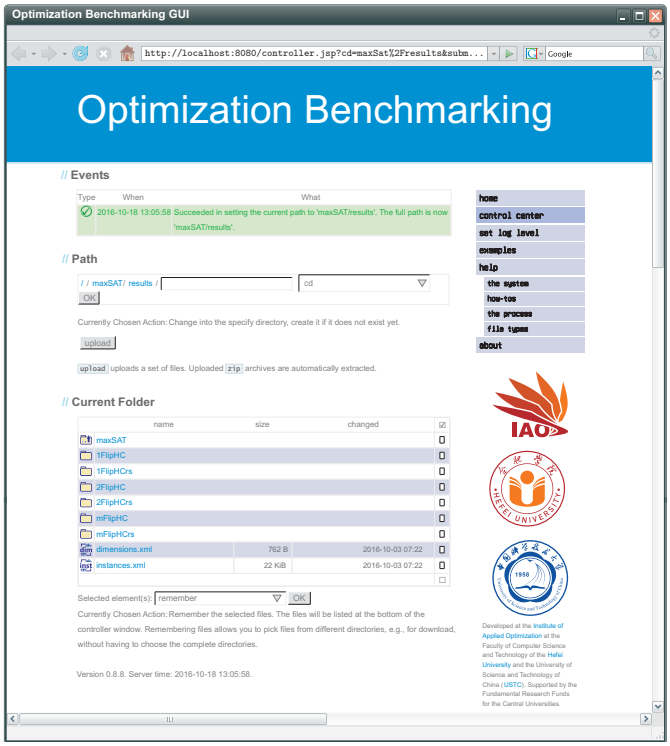


Fig. 8: A sketch of the web-based user interface of our *optimizationBenchmarking.org* framework.

good results are those which are rigorously analyzed, statistically sound, and stem from methodologically clean experiments and evaluation procedures. We brought forward several issues that arise when collecting data in experiments with anytime algorithms in Section II. Proper experimentation can lead to lots of data, which requires tool support. Experimental results, being summaries of the data, e.g., diagrams, descriptive, or inferential statistics, can thus only be as *good* as the tools used to extract them. Unfortunately, the field of optimization is extremely wide with a vast amount of different problems and existing tool support is often focused on narrow problem domains or specific algorithms.

More general tools are needed, because tools which can be used by more researchers in more scenarios would be tremendously helpful. Also, such tools would be assessed by more researchers so that methodological errors are more likely to be discovered and the tools get better over time. We make the case for trying to reach a consensus regarding file formats for representing data from a wide range of experiments, namely all kinds of single-objective problems. If we can agree on such data formats and both researchers on optimization as well as researchers on evaluation tools use them, this would bring us automatically much closer to this goal.

In Sections III and IV we discuss the requirements for such a data format and propose an example solution as used in our *optimizationBenchmarking.org* framework, with which we are able to analyze a very wide range of algorithms (LS, EAs, hybrid algorithms, ACO, B&B...) on a wide range of problems (TSP, satisfiability, numerical optimization...). We hope that this can be a good starting point for further discussions.

Acknowledgments We acknowledge support from the National Natural Science Foundation of China under Grants 61673359, 61150110488, and 71520107002 and the Fundamental Research Funds for the Central Universities.

References

[1] T. Weise, "From standardized data formats to standardized tools for optimization algorithm benchmarking," in *Proceedings of the 16th IEEE Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC'17), Jul. 26–28, 2017, University of Oxford, Oxford, UK*. Los Alamitos, CA, USA: IEEE Computer Society Press.

[2] *Variants of Evolutionary Algorithms for Real-World Applications*. Berlin Heidelberg: Springer, 2011.

[3] T. Weise, Y. Wu, R. Chiong, K. Tang, and J. Lässig, "Global versus local search: The impact of population sizes on evolutionary algorithm performance," *Journal of Global Optimization*, vol. 66, pp. 511–534, 2016.

[4] B. DuCharme, *XML: The Annotated Specification*. Upper Saddle River, USA: Prentice Hall, 1999.

[5] M. S. Boddy and T. L. Dean, "Solving time-dependent planning problems," Brown University, Department of Computer Science, Providence, USA, Tech. Rep. CS-89-03, 1989. [Online]. Available: ftp://ftp.cs.brown.edu/pub/techreports/89/cs89-03.pdf

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

[7] T. Weise, *Global Optimization Algorithms – Theory and Application*. it-weise.de (self-published), 2009. [Online]. Available: http://www.it-weise.de/projects/book.pdf

[8] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications*. San Francisco, USA: Morgan Kaufmann, 2005.

[9] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, USA: MIT Press, 2004.

[10] W. Zhang, "Truncated and anytime depth-first branch and bound: A case study on the asymmetric traveling salesman problem," in *AAAI Spring Symp. Series: Search Techniques for Problem Solving Under Uncertainty and Incomplete Information*, ser. AAAI Technical Report, vol. SS-99-07. AAAI, Menlo Park, USA, 1999, pp. 148–155.

[11] Y. Jiang, T. Weise, J. Lässig, R. Chiong, and R. Athauda, "Comparing a hybrid branch and bound algorithm with evolutionary computation methods, local search and their hybrids on the TSP," in *Proc. of the IEEE Symp. on Computational Intelligence in Production and Logistics Systems (CIPLS'14), Dec. 9–12, 2014, Orlando, USA*. IEEE, Los Alamitos, USA, pp. 148–155.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[13] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.

[14] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[15] B. A. Huberman, R. M. Lukose, and T. Hogg, "An economics approach to hard computational problems," *Science*, vol. 275, no. 5296, pp. 51–54, 1997.

[16] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis, "Towards objective measures of algorithm performance across instance space," *Computers & Operations Research*, vol. 45, pp. 12–24, 2014.

[17] T. Weise, R. Chiong, K. Tang, J. Lässig, S. Tsutsui, W. Chen, Z. Michalewicz, and X. Yao, "Benchmarking optimization algorithms: An open source framework for the traveling salesman problem," *IEEE Computational Intelligence Magazine (CIM)*, vol. 9, no. 3, pp. 40–52, Aug. 2014.

[18] G.-C. Christophe, "Beyond predictive accuracy: What?" in *Workshop on Upgrading Learning to Meta-Level: Model Selection and Data Transformation at the 10th European Conf. on Machine Learning (ECML-98), Apr. 21–23, 1998, Chemnitz, Germany*, 1998, pp. 78–85.

[19] P. A. Samuelson and W. D. Nordhaus, *Microeconomics*. McGraw-Hill, Boston, USA, 2001.

[20] D. Whitley, D. Hains, and A. E. Howe, "Tunneling between optima: Partition crossover for the traveling salesman problem," in *Proc. of the Genetic and Evolutionary Computation Conference, GECCO '09, Jul. 8-12, 2009, Montreal, Canada*. ACM, New York, USA, pp. 915–922.

[21] D. A. D. Tompkins and H. H. Hoos, "UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT," in *Revised Selected Papers from the 7th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'04), May 10–13, 2004, Vancouver, Canada*. Springer, Berlin, Germany, pp. 306–320.

[22] Y. Wu, T. Weise, and R. Chiong, "Local search for the traveling salesman problem: A comparative study," in *Proceedings of the 14th IEEE Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC'15), Jul. 6-8, 2015, Beijing, China*. IEEE, pp. 213–220.

[23] T. Weise, "The XML schema for the Experiment Data Interchange data format, version 1.0," 2016, xML Schema provided as XSD file `experimentDataInterchange.1.0.xsd` as part of the *optimizationBenchmarking.org* framework [46]. [Online]. Available: https://github.com/optimizationBenchmarking/evaluator-base

[24] N. Hansen, A. Auger, S. Finck, and R. Ros, "Real-parameter black-box optimization benchmarking: Experimental setup," Université Paris Sud, INRIA Futurs, Orsay, France, Tech. Rep., Mar. 24, 2012. [Online]. Available: http://coco.lri.fr/BBOB-downloads/download11.05/bbobdocexperiment.pdf

[25] H. H. Hoos and T. Stützle, "Evaluating las vegas algorithms – pitfalls and remedies," in *Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence (UAI'98), Jul. 24–26, 1998, Madison, WI, USA*. Morgan Kaufmann, San Francisco, USA, pp. 238–245.

[26] T. Weise, L. Niu, and K. Tang, "AOAB – automated optimization algorithm benchmarking," in *Companion Publication of the Genetic and Evolutionary Computation Conf. (GECCO'10), Jul. 7–11, 2010, Portland, USA*. ACM, New York, USA, pp. 1479–1486.

[27] F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown, "Performance prediction and automated tuning of randomized and parametric algorithms," in *Proc. of the 12th Intl. Conf. on Principles and Practice of Constraint Programming (CP 2006), Sep. 25-29, 2006, Nantes, France*. Springer, Berlin, Germany, pp. 213–228.

[28] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Identifying key algorithm parameters and instance features using forward selection," in *Revised Selected Papers of the 7th Intl. Conf. on Learning and Intelligent Optimization (LION 7), Jan. 7-11, 2013, Catania, Italy*. Springer, Berlin, Germany, pp. 364–381.

[29] J. R. Rice, "The algorithm selection problem," in *Advances in Computers*. New York, USA: Academic Press, 1976, vol. 15, pp. 65–118.

[30] J. Kanda, A. Carvalho, E. Hruschka, and C. Soares, "Selection of algorithms to solve traveling salesman problems using meta-learning," *Intl. Journal of Hybrid Intelligent Systems (IJHIS)*, vol. 8, no. 3, pp. 117–128, 2011.

[31] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "SATzilla: Portfolio-based algorithm selection for SAT," *Journal of Artificial Intelligence Research (JAIR)*, vol. 32, pp. 565–606, 2008.

[32] F. Peng, K. Tang, G. Chen, and X. Yao, "Population-based algorithm portfolios for numerical optimization," *IEEE Transactions on Evolutionary Computation*, vol. 14, pp. 782–800, 2010.

[33] K. Leyton-Brown, E. Nudelman, and Y. Shoham, "Learning the empirical hardness of optimization problems," in *Proc. of the 8th Intl. Conf. on Principles and Practice of Constraint Programming (CP 2002), Sep. 9-13, 2002, Ithaca, USA*. Springer, Berlin, Germany, pp. 556–572.

[34] C. Fawcett and H. H. Hoos, "Analysing differences between algorithm configurations through ablation," *Journal of Heuristics*, vol. 22, no. 4, pp. 431–458, 2016.

[35] T. Bartz-Beielstein, O. Flasch, P. Koch, and W. Konen, "SPOT: A toolbox for interactive and automatic tuning in the `R` environment," in *Proc. of the 20th Workshop on Computational Intelligence, Dec. 1-3, 2010, Dortmund, Germany*. KIT Scientific Publishing, Karlsruhe, Germany, pp. 264–273.

[36] K. Smith-Miles and J. I. van Hemert, "Discovering the suitability of optimisation algorithms by learning from evolved instances," *Annals of Mathematics and Artificial Intelligence*, vol. 61, no. 2, pp. 87–104, 2011.

[37] K. Smith-Miles, J. I. van Hemert, and X. Y. Lim, "Understanding TSP difficulty by learning from evolved instances," in *Selected Papers from the 4th Intl. Conf. on Learning and Intelligent Optimization (LION 4), Jan. 18-22, 2010, Venice, Italy*. Springer, Berlin, Germany, pp. 266–280.

[38] Q. Qi, T. Weise, and B. Li, "Modeling optimization algorithm runtime behavior and its applications," in *Proc. of the Genetic and Evolutionary Computation Conf. (GECCO'17), Jul. 15–19, 2017, Berlin, Germany*. ACM, New York, USA, accepted for publication.

[39] K. Tang, Z. Yang, and T. Weise, "Special session on evolutionary computation for large scale global optimization at 2012 IEEE world congress on computational intelligence," University of Science and Technology of China (USTC), School of Computer Science and Technology, Hefei, Anhui, China, Tech. Rep., Jun. 14, 2012.

[40] 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," University of Science and Technology of China (USTC), School of Computer Science and Technology, Hefei, Anhui, China, Tech. Rep., Jan. 8, 2010.

[41] P. Esling and C. Agon, "Time-series data mining," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, 2012.

[42] T. W. Liao, "Clustering of time series data—a survey," *Pattern Recognition*, vol. 38, pp. 1857–1874, 2005.

[43] J. Jacques and C. Preda, "Functional data clustering: A survey," *Advances in Data Analysis and Classification*, vol. 8, pp. 231–255, 2014.

[44] J. Ramsay and B. W. Silverman, *Functional Data Analysis*, 2nd ed. New York, USA: Springer, 2005.
[45] A. Mouat, *Using Docker: Developing and Deploying Software with Containers*. Sebastopol, USA: O'Reilly, 2016.
[46] T. Weise, X. Wang, Q. Qi, B. Li, and K. Tang, "Automating scientific work in optimization," 2017, under review.


```
@inproceedings{W2017FSDFTSDFOAB,
  author    = {Thomas Weise},
  title     = {From Standardized Data Formats to
               Standardized Tools for Optimization
               Algorithm Benchmarking},
  booktitle = {Proceedings of the 16th IEEE
               Conference on Cognitive Informatics \&
               Cognitive Computing (ICCI*CC'17), } #
               jul # {~26--28, 2017, University of
               Oxford, Oxford, UK},
  address   = {Los Alamitos, CA, USA},
  publisher = {IEEE Computer Society Press},
}
```