flaccogui: Exploratory Landscape Analysis for Everyone

Christian Hanster
University of Münster
Leonardo-Campus 3
48149 Münster, Germany
c_hans08@uni-muenster.de

Pascal Kerschke
University of Münster
Leonardo-Campus 3
48149 Münster, Germany
kerschke@uni-muenster.de

ABSTRACT

Finding the optimal solution for a given problem has always been an intriguing goal and a key for reaching this goal is sound knowledge of the problem at hand. In case of single-objective, continuous, global optimization problems, such knowledge can be gained by *Exploratory Landscape Analysis* (ELA), which computes features that quantify the problem's landscape prior to optimization. Due to the various backgrounds of researches that developed such features, there nowadays exist numerous implementations of feature sets across multiple programming languages, which is a blessing and burden at the same time.

The recently developed R-package `flacco` takes multiple of these feature sets (from the different packages and languages) and combines them within a single R-package. While this is very beneficial for R-users, users of other programming languages are left out. Within this paper, we introduce `flaccogui`, a graphical user interface that does not only make `flacco` more user-friendly, but due to a platform-independent web-application also allows researchers that are not familiar with R to perform ELA and benefit of the advantages of `flacco`.

CCS CONCEPTS

•Mathematics of computing → Statistical software; Continuous optimization; •Computing methodologies → *Supervised learning;*

KEYWORDS

Exploratory Landscape Analysis, Graphical User Interface, R-Package, Continuous Optimization, Automated Algorithm Selection

ACM Reference format:
Christian Hanster and Pascal Kerschke. 2017. `flaccogui`: Exploratory Landscape Analysis for Everyone. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3067695.3082477

1 INTRODUCTION

In recent years, the interest in research fields related to the (statistical) analysis of data – such as machine learning, optimization or algorithm selection – has been growing steadily and usually, their results can be improved by further knowledge of the underlying problems. For instance, when trying to solve the *Algorithm Selection Problem* [3, 32] – whose goal is the prediction of the best algorithm $A \in \mathcal{A}$ out of a (pre-defined) portfolio or set of optimization algorithms $\mathcal{A}$ for a given instance $I \in \mathcal{I}$ – the knowledge of problem-specific properties or characteristics is useful to train an enhanced algorithm selection model [4, 17].

The choice of characteristics is obviously domain-dependent, in a lot of cases even problem-dependent, but luckily there often exist several groups of characteristics which can be used. For instance, there exist features for single-objective combinatorial [7, 29], continuous [22, 27] or multi-objective optimization problems [8, 16, 18], but also for other domains such as the Traveling Salesperson Problem [10, 23, 30].

While it is definitely profitable that researchers from all over the world contribute to their specific domain of interest, it also comes with a major drawback when looking at it from a practical point of view: each group's features are – if at all – only provided within a single package of the respective group's favorite programming language. Therefore, if one wants to use features from different research groups, one has to deal with multiple interfaces. As the latter is not very user-friendly, people tend to avoid features from different sources and rather opt to restrict their experiments to features from a single source. Consequently, the conducted experiments are less powerful as their results are not comparable to experiments that were based on a different set of features. Also, all the information, which might be hidden within any of the neglected feature sets, is discarded; the same holds for possible interactions among features from different feature sets.

At least in the context of single-objective continuous optimization – in which the respective feature-generating process is often called *Exploratory Landscape Analysis* (ELA, [22]) – this obstacle has been mainly resolved by the development of `flacco` [12], which combines the respective features from multiple research groups within a single `R`-package [31]. Unfortunately, `R` is not very user-friendly (given its command-based interface) and thus, only people who are familiar with this programming language are currently able to profit from the package. In order to tackle this obstacle, we created a graphical user interface (GUI), which lets one profit from all the functionalities of `flacco` without any knowledge of `R`.

Following a brief introduction into Exploratory Landscape Analysis and its integration into `flacco` within Section 2, we provide a description of the GUI in Section 3 and conclude our work in Section 4.

GECCO '17 Companion, Berlin, Germany

978-1-4503-4939-0/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3067695.3082477

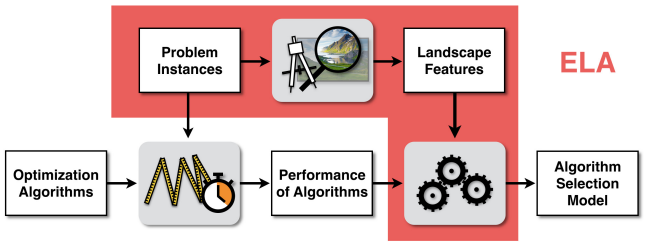


Figure 1: Schematic representation of the usage of ELA for training algorithm selection models.

2 EXPLORATORY LANDSCAPE ANALYSIS

As mentioned in the previous section, *Exploratory Landscape Analysis* (ELA, [22]) is a sophisticated approach, which aims at characterizing the landscapes of single-objective, continuous, global optimization problems by means of – not necessarily intuitively understandable – numerical features. These numbers could then for instance be used to guide optimization algorithms towards a landscape's optimum or even automatize the algorithm selection process for a given set of problem instances as depicted in Figure 1.

The importance of ELA becomes more obvious when taking into consideration that it is usually used in the context of black-box optimization, i.e., the underlying problems are unknown and function evaluations are (often) considered to be highly expensive. Therefore, features which solely rely on an initial sample can be very beneficial as long as the size of the initial sample is low. In an ideal scenario, the initial sample could be identical to a well-distributed initial population of an evolutionary algorithm and thus, its information could be used prior to even running the algorithm itself.

Although the term "Exploratory Landscape Analysis" is still rather new, contributions to this field, e.g., [11, 19, 20], have been made well before its first public mentioning in 2010 [24]. However, as there exist no obvious traits that clearly characterize a landscape, researchers from all over the world invent new sets of landscape features on a regular base, such as [1, 13, 25–28].

In addition, there have also been successful attempts of employing landscape features within algorithm selection [4] or for detecting whether a problem possesses an underlying funnel structure [14], i.e., a landscape whose local optima pile up to an upside-down version of a mountain.

As the previous examples show, there exist numerous sets of landscape features and also different fields of application. However, they are usually programmed in different languages – mainly in R [31], python [33] or Matlab [21] – and thereby exclude people who are not familiar with the respective programming language.

2.1 Integration of ELA into Flacco

In order to overcome the obstacle of the feature sets being distributed across different programming languages, we collected several of them and combined them (publicly accessible) within a single R-package for **f**eature-based **l**andscape **a**nalysis of **c**ontinuous and **c**onstraint **o**ptimization problems: `flacco` [12]. Currently, the package holds a total of 17 feature sets or equivalently more than 300 landscape features and can either be used via its stable release

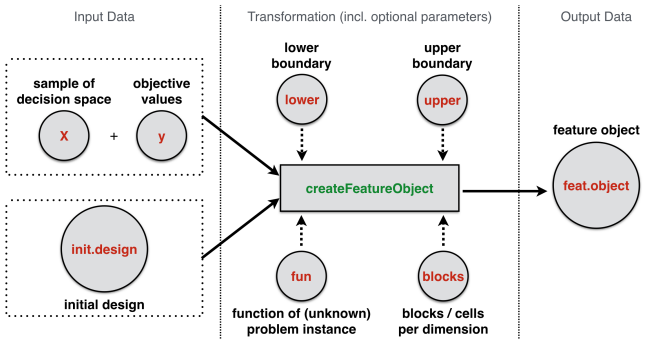


Figure 2: Creating the so-called "feature object", which is the basis for all further computations within `flacco`, cf. [15].

from CRAN[1] or as a development version which can be found on GitHub[2]. Note that the latter also provides a link to a tutorial.

Aside from the feature values itself, the package also automatically tracks the costs (in function evaluations *and* runtime) per feature set. Also, `flacco` provides functions for visualizing certain feature sets and hopefully, these plots improve the understanding of the often not intuitively comprehensible numerical values.

The essential ingredient of all computations and visualizations within `flacco` is the so-called "feature object". As shown in Figure 2 this object basically stores the information of the initial design, either provided by (a) a matrix or data frame with the input variables and a vector with the corresponding objective values, or (b) a data frame, which merged the two parts of (a) within a single object. This input data – eventually enhanced by some additional information such as the exact lower and upper bounds of the input variables or the exact definition of the (usually unknown) problem – is then transformed into the "feature object" by using the function `createFeatureObject`. Note that the exact function definition is only relevant for feature sets, which can not be computed without further additional function evaluations, such as the local search search features of [22].

For better usability, the package also ships with a function called `createInitialSample` which creates an initial sample (of points within the decision space). By defining the function's arguments, the user can define the number of observations (= points) as well as the number of dimensions (= input variables) d. Note that per default, the points are sampled random uniformly within the d-dimensional hypercube $[0, 1]^d$, but these default settings can easily be changed by the user. That is, the user can use a latin hypercube sample [2] rather than sampling the points random uniformly and the values of the boundaries can be set to any other value.

The code snippet below shows how one could create the feature object that is based on an initial design consisting of 100 two-dimensional points and whose respective objective values are exemplarily defined as $y_n = \sum_{i=1}^{2} x_{n,i} \cdot (x_{n,i} + 1)$:

```
> # compute the initial sample
> X = createInitialSample(n.obs = 100, dim = 2)
> # define the function
```

[1] Link to stable release: https://cran.r-project.org/package=flacco
[2] Link to development version: https://github.com/kerschke/flacco

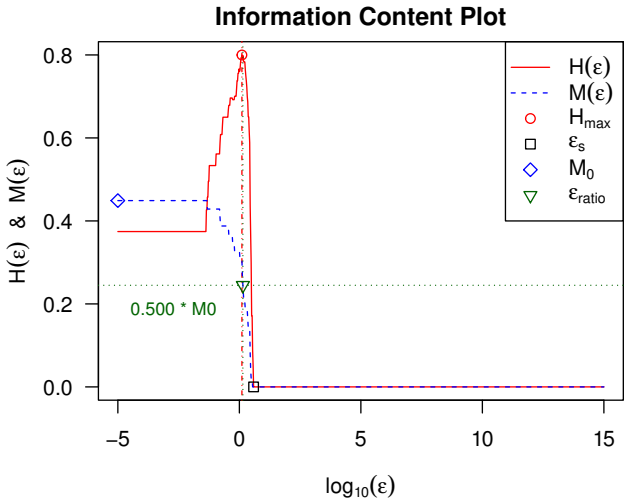


Figure 3: Example of an information content plot, cf. [27].

```
> fun = function(x) sum(x * (x + 1))
> # compute the objective values
> y = apply(X, 1, fun)
>
> # compute the feature object
> feat.obj = createFeatureObject(X = X, y = y,
+   fun = fun)
```

Given this feature object, one now can compute specific feature sets, such as the local search features (`ela_local`) from [22] or the dispersion features (`disp`) from [19]:

```
> feats.local = calculateFeatureSet(
+   feat.object = feat.obj, set = "ela_local")
> feats.disp = calculateFeatureSet(
+   feat.object = feat.obj, set = "disp")
```

Similarly to the feature computation, one can also create related plots, such as the information content plot by [27]:

```
> plotInformationContent(feat.object = feat.obj)
```

The corresponding plot for the example above is shown in Figure 3. The idea of the information content approach is to perform a random walk across the problem's fitness landscape and measure specific values based on the changes occurring in the objective space. Within the plot, some of these measures, such as the *information content* $H(\varepsilon)$ and *partial information content* $M(\varepsilon)$, are shown in dependency of the parameter for the *information sensitivity* ε. For further details, the interested reader is referred to [27]. Note that again, all computations completely rely on the feature object.

3 A GUI FOR FLACCO

As shown in the previous chapter, the R-package `flacco` can be seen as a powerful library to carry out landscape analysis. However, for users who do not have experiences in using R it is quite difficult

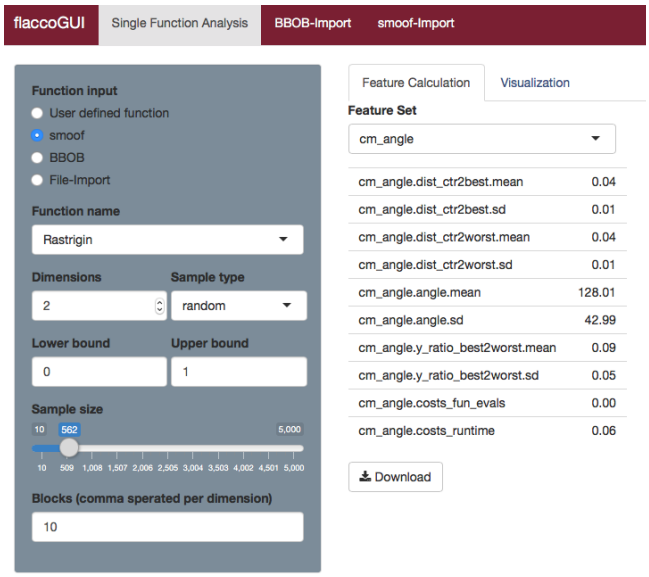


Figure 4: Exemplary screenshot of the `flaccogui` web-application.

to use the package. With `flaccogui` we therefore designed an application, which provides a graphical user interface for `flacco`. The idea of this GUI is that also people, who do not have experiences in using R, can profit of the functionalities of `flacco`. And although `flaccogui` is also implemented in R it can be run with just a few clicks. Moreover, there exists a hosted version of the `flaccogui` application online (https://flaccogui.shinyapps.io/flaccogui/), which allows the usage of the GUI without any installation (or even knowledge) of R.

The implementation of `flaccogui` makes use of the R-package `shiny` [6], which was designed to create web applications in R so that the user can develop web applications using R-functions without any knowledge of web-technologies. Note that `shiny` apps are written completely in R. For a better code structure shiny-applications are divided into two parts: a `ui`-function for the description of the user interface and a `server`-function for the application logic. When the application is started, `shiny` starts a web-server and transforms the components described in the `ui`-function into a web-application (consisting of HTML, CSS and JavaScript) which can be used in every browser.

For the application logic the shiny-web-server runs the `server`-function of the app. The `server`-function is automatically connected (through ajax) with the browser session so it can calculate results in R and display them in the application based on the user input. The `shiny`-package also offers the possibility to cluster application parts into *modules*, so that they can be reused in other (`shiny`-) apps. In order to integrate a module into a `shiny` app, the main application only has to use two functions provided by the module. One will be used for describing the UI, the other will control the logic.

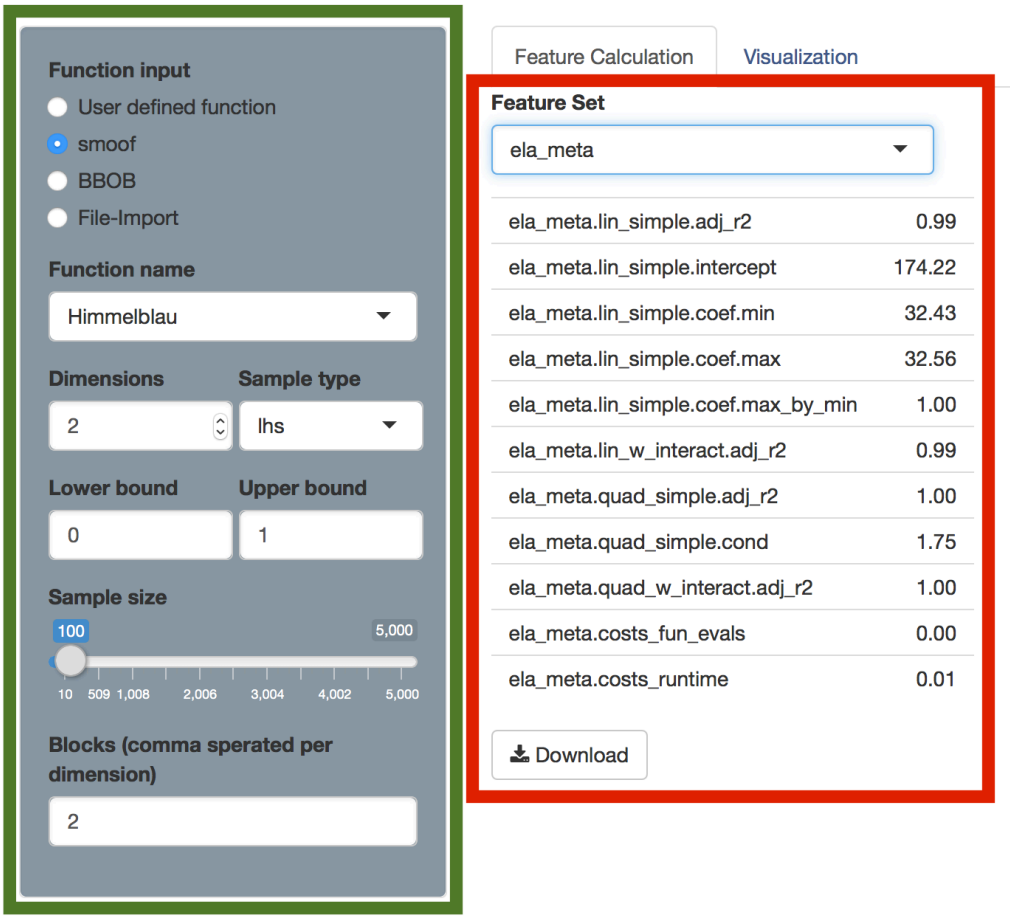


Figure 5: Within the `functionInput`-module on the left side (green box), the user can select (or configure) the input data and within the `featureCalculation`-module on the right side (red), the user can compute the desired feature sets.

3.1 Using `flaccogui`

To use the `flaccogui` interface the user has different options. The easiest option is to use the hosted version of `flaccogui` at shinyapps.io[3]. In that case, the user does not need to install any other software but just needs to open the web page and perform the landscape analysis online.

If one prefers to run the application locally on its own machine, one can do so by executing the following R-code snippet:

```
> # first, install the "flacco"-package and its depen-
> # dencies from CRAN
> install.packages("flacco", dependencies = TRUE)
>
> # then, load the package and start the app
> library(flacco)
> runFlaccoGUI()
```

The `library`-command loads the `flacco`-package in R and the following command will start the web-application right out of R (as shown in Figure 4). The application includes all functionalities (i.e., modules) of `flacco`, i.e., feature calculation and visualization, but also extends them by allowing the user to choose among various test problems such as the ones from the *Black-Box Optimization Benchmark* (BBOB, [9]) or any other test problems provided by the R-package `smoof`[4] [5].

Note that the modular build of `flaccogui` comes with an additional advantage: it facilitates the integration of parts of the package into other `shiny` applications.

3.2 Creating the Feature Object

As explained before, the first step to perform landscape analysis with `flacco` is to create an initial sample and the corresponding *feature object*. Following this workflow, `flaccogui` provides a module that creates a *featureObject* as shown in the left part of Figure 5 (highlighted by a green box).

As there are different application contexts to perform landscape analysis, our GUI offers different options to create the *featureObject*:

[3] Link to the hosted version of `flaccogui`: https://flaccogui.shinyapps.io/flaccogui/

[4] `smoof` integrates well-known optimization problems from the literature so that the exact mathematical function definition does not need to be entered by the user.

(1) Describe the optimization problem by a mathematical term within a plain text field.
(2) Select an optimization problem that is already defined within the R-package `smoof` [5].
(3) Configure a function from the *Black-Box Optimization Benchmark* (BBOB, [9]). Although the BBOB functions are also part of the `smoof`-package, we separate them from the others as they are frequently used when performing ELA.
(4) CSV-import of the initial design. Whenever the exact functions are unknown – which is the case for most real-world scenarios – and also in the cases, in which the calculation of the function values is very cost intensive, it makes sense to just import the sample (which has been generated before).

Based on the selection how to create the feature object, the user needs to put in different configuration parameters (e.g., lower and upper bound, dimensions, number of blocks for cell mapping features, etc.). Our GUI will use this input to create a *feature object*, which can be used for the calculation of any of the feature sets from `flacco`.

The integration of the module is very easy. The following code snippet shows a basic `shiny` app, which can create a *feature object*:

```
> ui <- sidebarLayout(
+   featureObject_sidebar("feature_function")
+ )
>
> server <- function(input, output) {
+   featureObject <- callModule(
+     functionInput,
+     "feature_function",
+     stringsAsFactors = FALSE)
+ }
```

3.3 Calculation of a Feature Set

Beside the creation of the *feature object* our GUI provides modules for the calculation and visualization of feature sets. The red box in Figure 5 shows the `FeatureSetCalculationComponent`. Based on a *feature object*, which has to be handed over to the module, the module can calculate feature sets. As shown in Section 3.2, this *feature object* can be created by the `functionInput`-module of our GUI, but – given the modularized structure of the app – it can also be created through another way.

When using the GUI, the user can select a specific feature set from a drop down menu and the respective feature values will be calculated directly after the user made a choice. There is also an option for the calculation of all feature sets (at once), which are currently available in `flacco`. For a better integration in the development workflow, the calculated feature sets can also be downloaded (in CSV-format). So, if a user wants to use the ELA results in another application or programming environment he/she just needs to import the CSV-file generated by `flaccogui`.

3.4 Landscape analysis visualizations

As mentioned in Section 2.1, `flacco` also provides options for the visualization of specific feature sets, such as plots for general cell mapping, barrier trees (in 2D and 3D), as well as an information content graph.

Similar to the `FeatureSetCalculation`-component there is a `FeatureSetVisualization`-component, which generates the different plots. Based on the *feature object*, which the module uses as input, the module shows the different graph options that are available for the current *feature object* – for instance, the cell-mapping plots and the barrier-trees are only available for two-dimensional problems. Depending on the user's selection `flaccogui` will generate the corresponding plot, such as the cell mapping plot shown in Figure 6.

For creating an application as the cell mapping plot that is shown in Figure 6, one only needs a few lines of code, as shown below:

```
> ui <- sidebarLayout(
+   featureObject_sidebar("feature_function"),
+   mainPanel(
+     # Tabset-panel for the different possibilities
+     # providing certain input in the app
+     tabsetPanel(
+       tabPanel("Feature Calculation",
+         FeatureSetCalculationComponent(
+           "featureSet_Calculation")),
+       tabPanel("Visualization",
+         FeatureSetVisualizationComponent(
+           "featureSet_Visualization"))
+ )))
>
> server <- function(input, output) {
+   featureObject <- callModule(
+     functionInput,
+     "feature_function",
+     stringsAsFactors = FALSE)
+   callModule(
+     FeatureSetCalculation,
+     "featureSet_Calculation",
+     stringsAsFactors = FALSE,
+     reactive(featureObject()))
+   callModule(
+     FeatureSetVisualization,
+     "featureSet_Visualization",
+     stringsAsFactors = FALSE,
+     reactive(featureObject()))
+ }
```

3.5 BBOB CSV-import

For advanced users, the `flaccogui`-package also includes modules for batch-import of functions and their corresponding parameters. Especially when analyzing multiple BBOB-functions, the analysis should be carried out for different functions and instances.

Therefore `flaccogui` incorporates a module (shown in Figure 7), which imports a CSV-file whose rows represent the input parameters of a BBOB function, namely the function ID (FID), instance ID (IID) and dimension (dim). Along with the CSV-file, the user can set some additional parameters, such as the number of replicates (how often should the features be calculated per function), which

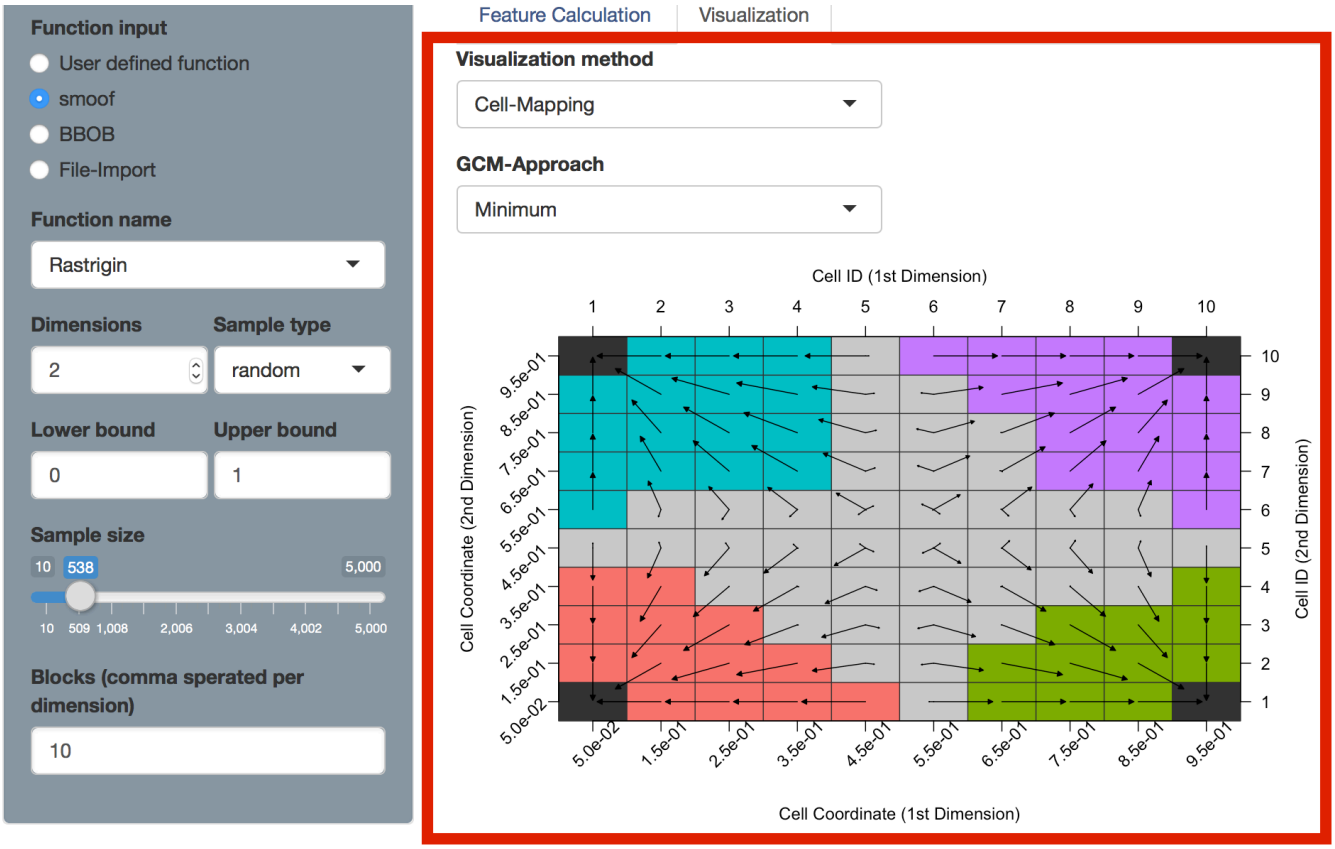
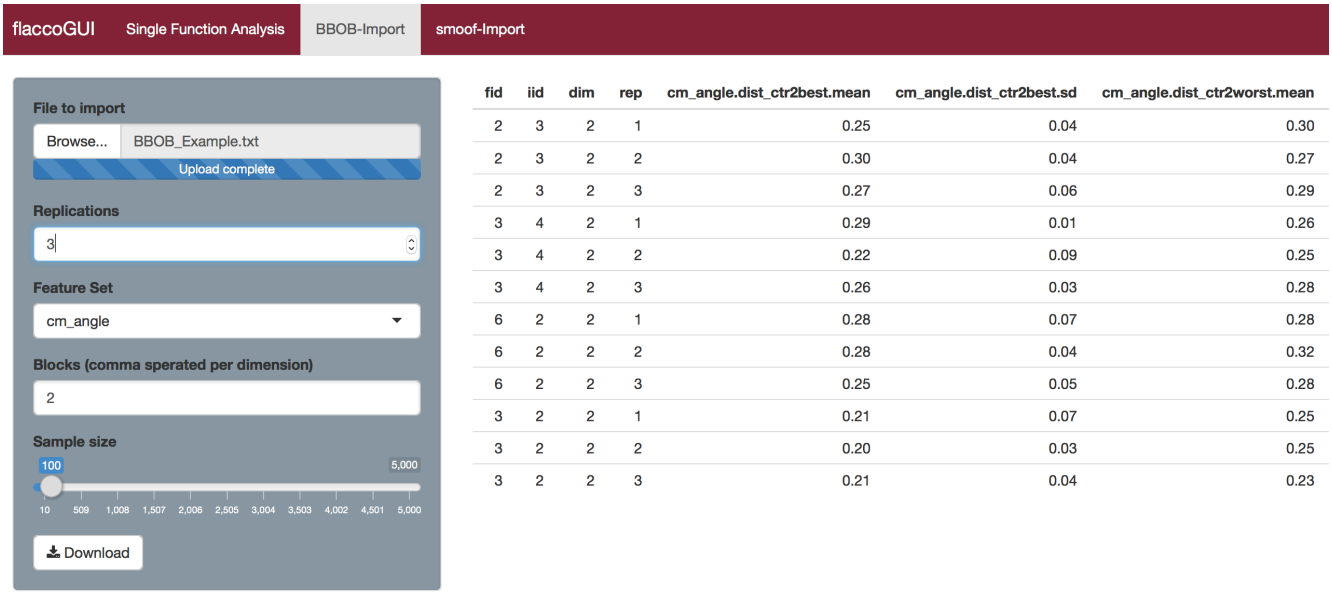


Figure 6: `featureVisualization`-module (red box) displaying a two-dimensional cell-mapping plot. The plot is exemplarily shown for an instance of the two-dimensional Rastrigin function within the box constraints $[0,\ 1] \times [0,\ 1]$.



fid	iid	dim	rep	cm_angle.dist_ctr2best.mean	cm_angle.dist_ctr2best.sd	cm_angle.dist_ctr2worst.mean
2	3	2	1	0.25	0.04	0.30
2	3	2	2	0.30	0.04	0.27
2	3	2	3	0.27	0.06	0.29
3	4	2	1	0.29	0.01	0.26
3	4	2	2	0.22	0.09	0.25
3	4	2	3	0.26	0.03	0.28
6	2	2	1	0.28	0.07	0.28
6	2	2	2	0.28	0.04	0.32
6	2	2	3	0.25	0.05	0.28
3	2	2	1	0.21	0.07	0.25
3	2	2	2	0.20	0.03	0.25
3	2	2	3	0.21	0.04	0.23

Figure 7: `flaccogui` allows to import the parameters of several BBOB functions via a CSV-file and thereby enables the calculation of multiple feature sets across several optimization problems at once.

are relevant when computing stochastic features (especially when the sample size is quite small). Our application will then calculate the selected feature sets for each of the replicates and also provides a possibility to download the results.

4 CONCLUSION

This paper introduces `flaccogui`, a graphical user interface to `flacco`, which is an R-package for *Exploratory Landscape Analysis*. Our GUI has recently been integrated in the R-package `flacco`, but is also available as a platform-independent web-application. The latter allows to enjoy the advantages of `flacco` – most importantly calculating different landscape features – without needing any knowledge of R. As a result, researchers can reuse already developed source code and tools for feature calculation without the burden of interfacing or re-implementing the same features in another language.

In the future, `flaccogui` will be developed hand-in-hand with `flacco`, which allows to keep its content up-to-date with the most recent developments in ELA, while sustaining the benefits of a platform-independent graphical user interface.

ACKNOWLEDGMENTS

The authors acknowledge support from the European Research Center for Information Systems (ERCIS).[5]

REFERENCES

[1] Tinus Abell, Yuri Malitsky, and Kevin Tierney. 2013. Features for Exploiting Black-Box Optimization Problem Structure. In *Proceedings of 7th International Conference on Learning and Intelligent Optimization (LION)*, Giuseppe Nicosia and Panos Pardalos (Eds.). Springer, 30 – 36. DOI: http://dx.doi.org/doi.org/10.1007/978-3-642-44973-4_4

[2] Brian K Beachkofski and Ramana V Grandhi. 2002. Improved Distributed Hypercube Sampling. In *43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*.

[3] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Thomas Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hendrik Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. 2016. ASlib: A Benchmark Library for Algorithm Selection. *Artificial Intelligence* 237 (2016), 41 – 58. DOI: http://dx.doi.org/doi.org/10.1016/j.artint.2016.04.003

[4] Bernd Bischl, Olaf Mersmann, Heike Trautmann, and Mike Preuss. 2012. Algorithm Selection Based on Exploratory Landscape Analysis and Cost-Sensitive Learning. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO '12)*. ACM, New York, NY, USA, 313–320. DOI: http://dx.doi.org/doi.org/10.1145/2330163.2330209

[5] Jakob Bossek. 2016. *smoof: Single and Multi-Objective Optimization Test Functions*. https://github.com/jakobbossek/smoof R-package version 1.4.

[6] Winston Chang, Joe Cheng, JJ Allaire, Yihui Xie, and Jonathan McPherson. 2016. *shiny: Web Application Framework for R*. https://CRAN.R-project.org/package=shiny R package version 0.14.1.

[7] Fabio Daolio, Sébastien Verel, Gabriela Ochoa, and Marco Tomassini. 2012. Local optima networks and the performance of iterated local search. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. ACM, 369–376.

[8] Deon Garrett and Dipankar Dasgupta. 2007. Multiobjective Landscape Analysis and the Generalized Assignment Problem. In *Proceedings of 2nd International Conference on Learning and Intelligent Optimization (LION)*, Vittorio Maniezzo, Roberto Battiti, and Jean-Paul Watson (Eds.). Lecture Notes in Computer Science, Vol. 5313. Springer, 110 – 124. DOI: http://dx.doi.org/doi.org/10.1007/978-3-540-92695-5_9

[9] Nikolaus Hansen, Anne Auger, Steffen Finck, and Raymond Ros. 2010. *Real-Parameter Black-Box Optimization Benchmarking 2010: Experimental Setup*. Technical Report RR-7215. INRIA. http://hal.inria.fr/docs/00/46/24/81/PDF/RR-7215.pdf

[10] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. 2014. Algorithm runtime prediction: Methods & evaluation. *Journal of Artificial Intelligence* 206, 0 (2014), 79–111.

[11] Terry Jones. 1995. *Evolutionary algorithms, fitness landscapes and search*. Ph.D. Dissertation. Citeseer.

[12] Pascal Kerschke. 2017. *flacco: Feature-Based Landscape Analysis of Continuous and Constraint Optimization Problems*. https://github.com/kerschke/flacco R-package version 1.5.

[13] Pascal Kerschke, Mike Preuss, Carlos Hernández, Oliver Schütze, Jian-Qiao Sun, Christian Grimme, Günter Rudolph, Bernd Bischl, and Heike Trautmann. 2014. Cell Mapping Techniques for Exploratory Landscape Analysis. In *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*. Springer, 115–131. DOI: http://dx.doi.org/doi.org/10.1007/978-3-319-07494-8_9

[14] Pascal Kerschke, Mike Preuss, Simon Wessing, and Heike Trautmann. 2016. Low-Budget Exploratory Landscape Analysis on Multiple Peaks Models. In *Proceedings of the 18th Annual Conference on Genetic and Evolutionary Computation*. ACM. DOI: http://dx.doi.org/10.1145/2908812.2908845

[15] Pascal Kerschke and Heike Trautmann. 2016. The R-Package FLACCO for Exploratory Landscape Analysis with Applications to Multi-Objective Optimization Problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*. IEEE.

[16] Pascal Kerschke, Hao Wang, Mike Preuss, Christian Grimme, André Deutz, Heike Trautmann, and Michael T. M. Emmerich. 2016. Towards Analyzing Multimodality of Multiobjective Landscapes. In *Proceedings of the 14th International Conference on Parallel Problem Solving from Nature (PPSN XIV) (Lecture Notes in Computer Science)*, Julia Handl, Emma Hart, Peter R. Lewis, Manuel López-Ibáñez, Gabriela Ochoa, and Ben Paechter (Eds.). Springer, 962–972. DOI: http://dx.doi.org/10.1007/978-3-319-45823-6_90

[17] Lars Kotthoff, Pascal Kerschke, Holger Hendrik Hoos, and Heike Trautmann. 2015. Improving the State of the Art in Inexact TSP Solving Using Per-Instance Algorithm Selection. In *Proceedings of 9th International Conference on Learning and Intelligent Optimization (LION) (Lecture Notes in Computer Science)*, Clarisse Dhaenens, Laetitia Jourdan, and Marie-Eléonore Marmion (Eds.), Vol. 8994. Springer, 202 – 217. DOI: http://dx.doi.org/doi.org/10.1007/978-3-319-19084-6_18

[18] Arnaud Liefooghe, Sébastien Verel, Fabio Daolio, Hernán Aguirre, and Kiyoshi Tanaka. 2015. A Feature-Based Performance Analysis in Evolutionary Multiobjective Optimization. In *Proceedings of the 8th International Conference on Evolutionary Multi-Criterion Optimization (EMO)*, António Gaspar-Cunha, Carlos Henggeler Antunes, and Carlos A. Coello Coello (Eds.). Lecture Notes in Computer Science, Vol. 9019. Springer, 95 – 109. DOI: http://dx.doi.org/doi.org/10.1007/978-3-319-15892-1_7

[19] Monte Lunacek and Darrell Whitley. 2006. The Dispersion Metric and the CMA Evolution Strategy. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. ACM, 477–484. DOI: http://dx.doi.org/doi.org/10.1145/1143997.1144085

[20] Katherine M Malan and Andries P Engelbrecht. 2009. Quantifying ruggedness of continuous landscapes using entropy. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE, 1440 – 1447. DOI: http://dx.doi.org/doi.org/10.1109/CEC.2009.4983112

[21] MATLAB. 2013. *Version 8.2.0 (R2013b)*. The MathWorks Inc., Natick, MA, USA.

[22] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. 2011. Exploratory Landscape Analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11)*. ACM, New York, NY, USA, 829–836. DOI: http://dx.doi.org/doi.org/10.1145/2001576.2001690

[23] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Markus Wagner, Jakob Bossek, and Frank Neumann. 2013. A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. *Annals of Mathematics and Artificial Intelligence* 69, 2 (2013), 151–182. DOI: http://dx.doi.org/10.1007/s10472-013-9341-2

[24] Olaf Mersmann, Mike Preuss, and Heike Trautmann. 2010. Benchmarking Evolutionary Algorithms: Towards Exploratory Landscape Analysis. In *PPSN XI: Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (Lecture Notes in Computer Science 6238)*, R. Schaefer and others (Eds.). Springer, 71–80. DOI: http://dx.doi.org/doi.org/10.1007/978-3-642-15844-5_8

[25] Rachael Morgan and Marcus Gallagher. 2015. Analysing and Characterising Optimization Problems Using Length Scale. *Soft Computing* (2015), 1 – 18. DOI: http://dx.doi.org/doi.org/10.1007/s00500-015-1878-z

[26] Christian L. Müller and Ivo F. Sbalzarini. 2011. Global Characterization of the CEC 2005 Fitness Landscapes Using Fitness-Distance Analysis. In *Applications of Evolutionary Computation*. Springer, 294 – 303. DOI: http://dx.doi.org/doi.org/10.1007/978-3-642-20525-5_30

[27] Mario Andrés Muñoz, Michael Kirley, and Saman K Halgamuge. 2015. Exploratory Landscape Analysis of Continuous Space Optimization Problems using Information Content. *Evolutionary Computation, IEEE Transactions on* 19, 1 (2015), 74–87. DOI: http://dx.doi.org/doi.org/10.1109/TEVC.2014.2302006

[28] Mario Andrés Muñoz, Yuan Sun, Michael Kirley, and Saman K. Halgamuge. 2015. Algorithm Selection for Black-Box Continuous Optimization Problems: A Survey on Methods and Challenges. *Information Sciences* 317 (2015), 224 – 245. DOI:

[5] Link to ERCIS: https://www.ercis.org/

http://dx.doi.org/doi.org/10.1016/j.ins.2015.05.010

[29] Eugene Nudelman, Kevin Leyton-Brown, Holger H. Hoos, Alex Devkar, and Yoav Shoham. 2004. *Understanding Random SAT: Beyond the Clauses-to-Variables Ratio.* Springer, 438–452. DOI: http://dx.doi.org/10.1007/978-3-540-30201-8_33

[30] Josef Pihera and Nysret Musliu. 2014. Application of Machine Learning to Algorithm Selection for TSP. In *Proceedings of the IEEE 26th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE Computer Society, 47–54. DOI: http://dx.doi.org/10.1109/ICTAI.2014.18

[31] R Core Team. 2016. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org/

[32] John R Rice. 1976. The Algorithm Selection Problem. *Advances in Computers* 15 (1976), 65–118.

[33] Guido VanRossum and The Python Development Team. 2015. *The Python Language Reference – Release 3.5.0.* Python Software Foundation.