DSCTool: a web-service-based framework for statistical comparison of stochastic optimization algorithms

version 1.5

Tome Eftimov, Gašper Petelin, Urban Škvorc, Gorjan Popovski, Peter Korošec

September 28, 2021

The DSCTool was developed and implemented to make all the required knowledge for making different performance evaluations accessible from one place by guiding the user from providing input data (optimisation algorithm results) and selection of desired comparison to the final result of comparison. Further the implementation provides natural progress for all steps of performance evaluation, so no extra knowledge from the user is required.

1 REST web services

A web service is software that supports one or more open protocols and standards defined for exchanging data between applications or systems and makes itself available over the internet. Due to usage of open protocols and standards, software can be written in arbitrary programming language and run on various platforms. This makes web service widely interoperable and decoupled from implementations that uses their functionalities. The protocols most commonly used are the Simple Object Access Protocol (SOAP) and REpresentational State Transfer (REST). Due to its simplicity, flexibility, and lightweightedness, we have decided to use REST. Since REST is based on HTTP, we are able use basic HTTP methods, such as GET, PUT, POST, and DELETE. A REST web service access point is defined by Uniform Resource Identifier (URI), which is a string of characters that unambiguously identifies an individual resource. Data that is being transmitted can use different standard representations, such as XML or JSON.

The web services are accessible from following HTTPS URL `https://ws.ijs.si:8443/dsc-1.5/service/` and are using JSON for input/output data representation.

1.1 Registration service

To prevent uncontrolled abuse of web server, we have incorporated a mandatory registration. A user must register using the following service.

URI manage/user

HTTP method POST

Input JSON

```
{
  "name": "Name Surname",
  "affiliation": "affiliation details",
  "email": "name.surname@email.provider",
  "username": "your_username",
  "password": "your_password"
}
```

All passwords are encrypted before being stored into database. For accessing any other service a username and password using HTTP Basic Authentication is required.

1.2 Ranking service

This service returns ranking provided by the DSC ranking scheme.

URI rank

HTTP method POST

Input JSON

```
{
  "epsilon": double_value,
  "monte_carlo_iterations": integer_value,
  "method": {
    "name": "KS"|"AD",
    "alpha": double_value
  },
  "data":[ {
    "algorithm":"algorithm name",
    "problems":[ {
      "name":"problem name",
      "data":[
        double_value,
        ...
        ]
    }, {
      ...
    }, ... ]
  }, ... ]
}
```

The *epsilon* key values greater than zero initiate pDSC, with number defining practical level. The *monte_carlo_iterations* key values greater than zero initiate Monte Carlo variant of DSC, with number defining the number of permutations. The *method* key defines two-sample test (Kolmogorov-Smirnov test (KS) or Anderson-Darling test (AD)), which determines if two samples come from the same distribution, and its corresponding significance level (*alpha* key). Samples are described with the *data* key where each algorithm's sample set is described by algorithm's name and list of problems containing a problem name and obtained solution values for it by the algorithm.

Output JSON

```
{
  "result": {
    "valid_methods": [
      "omnibus_test_name",
      ...
    ],
    "ranked_matrix": [ {
      "problem": "problem name",
      "result": [ {
        "algorithm": "algorithm name",
        "rank": double_value
```

```
      }, ... ]
    }, ... ],
    "number_algorithms": integer_value,
    "parametric_tests": boolean_value
  },
  "success": boolean_value
}
```

In case of successful ranking (indicated by the *success* key), we receive a list of valid supported omnibus tests (*valid_methods* key), number of algorithms that were ranked (*number_algorithms* key), indication if parametric test is eligible (*parametric_tests* key), and list of ranks (*ranked_matrix* key). Each item in ranked_matrix is defined by the name of the problem (*problem* key) and rankings for each algorithm (*result* key). In case there is not sufficient amount of problems to make omnibus test, the *valid_methods* key will contain a message "Not enough problems for multiple-problem analysis!".

1.3 Multivariate service

To include into analysis also the location of the solutions (not only its values) multivariate service is used.

URI multivariate

HTTP method POST

Input JSON
```
{
  "method": {
    "desired_distribution": integer_value,
    "alpha": double_value
  },
  "data":[ {
    "algorithm":"algorithm name",
    "problems":[ {
      "name":"problem name",
      "data":[
        [
          double_value,
          ...
        ], ... ]
    }, {
      ...
    } ]
  }, ... ]
}
```

The *desired_distribution* key defines the preferred distribution of solutions, with 0 preferring clustered distribution and with 1 preferring sparse distributions. The *alpha* key defines significance level used by statistical test for comparing distributions. The *data* key structure is similar to the one

found in rank service with the difference that the *data* key consists of array of array of double values, describing the solution locations and not its values.

Output JSON Output of this service is exactly the same as rank service.

1.4 Multiobjective service

To limit the influence of selection of quality indicator on statistical analysis a multiobjective service is used, so small (statistically insignificant) differences between the distribution of approximation sets do not influence the final conclusion. **Note: This is really time consuming service and can easily take more than 10 minutes to process so please set timeout appropriately.**

URI multiobjective

HTTP method POST

Input JSON
```
{
  "method": {
    "alpha": double_value
  },
  "data":[ {
    "algorithm":"algorithm name",
    "problems":[ {
      "name":"problem name",
      "approximationSets": [ {
        "preferenceData": double_value,
        "data":[
        [
          double_value,
          ...
        ], ... ]
      }, ... ]

    }, {
      ...
    } ]
  }, ... ]
}
```

The *alpha* key defines significance level used by statistical test for comparing distributions. The *data* key structure, inside the added *approximationSets* structure, is the same as the one found in multivariate service with the difference that array of array of double values describe one approximation set (objectives for each solution in a approximation set) and the *preferenceData* key is added to the structure and consists of calculated quality indicator value from approximation set.

Output JSON Output of this service is exactly the same as rank service.

1.5 Ensemble service

In case rankings of multi-objective algorithms performances according to set of quality indicators are desired, the ensemble service is used. Here, ensemble of different quality indicators is provided according to average, hierarchical majority vote, or data-driven.

URI ensemble

HTTP method POST

Input JSON
```
{
  "method": "average"|"hierarchical"|"data-driven",
  "problems": [
    "problem_name",
    ...
  ],
  "indicators": [ {
    "indicator": "indicator name",
    "data": [ {
      "algorithm": "algorithm name",
      "problems": [
        double_value,
        ...
      ]
    }, ... ]
  }, ... ]
}
```

The *method* key defines the ensemble method (average, hierarchical, or data-driven). The *problems* key represent a list of problems on which algorithms were run. The *ensembles* key represents a list of ensemble items, where each item is represented by indicator name, and list of rank values for each of algorithms achieved for each problem.

Output JSON Output of this service is exactly the same as rank service.

1.6 Omnibus service

To acquire statistical significance according to obtained rankings (either from rank, multivariate or ensemble service) the omnibus service needs to be used.

URI omnibus

HTTP method POST

Input JSON Input JSON takes majority of information from the *result* key of (rank, multivariate, or ensemble services), with the only difference being selection of the actual omnibus statistical test and its significance level through the *method* key.

```
{
  "method": {
    "name": "omnibus test name",
    "alpha": double_value
  },
  "ranked_matrix": [ {
    "problem": "problem name",
    "result": [ {
      "algorithm": "algorithm name",
      "rank": double_value
    }, ... ]
  }, ... ],
  "number_algorithms": integer_value,
  "parametric_tests": boolean_value
}
```

Output JSON

```
{
  "result": {
    "message": "information if hypothesis was rejected or not
        ↪ ",
    "p_value": double_value,
    "t": double_value,
    "method": {
      "name": "omnibus test used",
      "alpha": double_value
    },
    "algorithm_means": [ {
      "algorithm": "algorithm name",
      "mean": double_value
    }, ... ]
  },
  "success": boolean_value
}
```

The result of omnibus service consist of the *message* key, which describes if null hypothesis was rejected or not, the *p_value* key indicating p-value, the *t* key representing test statistic value (if appropriate, otherwise is 0), statistic method being applied, and means of rankings for each algorithm.

We would like to mention that if "wilcoxon-signed-rank-less" method is selected the p-value represents the result of comparison of algorithm with lower mean value vs. algorithm with higher value (i.e., if p-value is lower than alpha value then algorithm with lower mean significantly outperforms the other algorithm, otherwise there is no significance between them).

1.7 Post-hoc service

In case statistical significance is observed in the samples using omnibus test, we need to identify which algorithm perform better than the others using post-hoc service.

URI posthoc

HTTP method POST

Input JSON Input JSON contains *algorithm_means* key returned by omnibus service, with addition of keys related to applied non-parametric Friedman or Friedman-aligned-rank tests.

```
{
  "algorithm_means": [ {
    "algorithm": "algorithm name",
    "mean": double_value
  }, ... ],
  "k": integer_value,
  "n": integer_value,
  "base_algorithm": "control algorithm name",
  "method": {
    "name": "friedman" | "friedman-aligned-rank",
    "alpha": double_value
  }
}
```

The k key represent the number of compared algorithms, the n key represent the number of problems, the *base_algorithm* key defines the control algorithm (one of the algorithms defined inside the *algorithm_means* key), and the *method* key defines corresponding statistics test with respect to selected omnibus test (friedman or friedman-aligned-rank) with desired significance level (*alpha* key). We need to mention here, that in the case when Iman-Davenport statistical test is used, the post-hoc test must be the same as in the case of the Friedman test.

Output JSON

```
{
  "result": {
    "adjusted_p_values": [ {
      "name": "ZValue",
      "algorithms": [ {
        "algorithm": "algorithm name",
        "value": double_value
      }, ... ]
    },
    {
      "name": "UnadjustedPValue",
      "algorithms": [ {
        "algorithm": "algorithm name",
        "value": double_value
      }, ... ]
    },
    {
      "name": "Holm",
      "algorithms": [ {
        "algorithm": "algorithm name",
        "value": double_value
      }, ... ]
    },
    {
      "name": "Hochberg",
```

```
      "algorithms": [ {
        "algorithm": "algorithm name",
        "value": double_value
      }, ... ]
    } ]
  },
  "success": boolean_value
}
```

The result of post-hoc service consists of a list of four results for each of the algorithms with respect to selected control algorithm. Z-value (ZValue) which is the result of selected statistic, unadjusted p-value (UnadjustedP-Value) that is calculated from Z-value, and two adjusted p-values according to Holm and Hochberg procedures.

1.8 Visualize service

To acquire some visualization and understanding from obtained rankings (either from rank, multivariate or ensemble service) the visualize service needs to be used.

URI visualize

HTTP method POST

Input JSON Input JSON takes the *ranked_matrix* of (rank, multivariate, or ensemble services) with the *method* key defining the desired visualization. Currently only performViz visualization is provided.

```
{
  "method":  "performViz",
  "ranked_matrix": [ {
    "problem": "problem name",
    "result": [ {
      "algorithm": "algorithm name",
      "rank": double_value
    }, ... ]
  }, ... ]
}
```

Output PNG Image The result of visualize service consist of the *PNG* image, which shows performViz [] type of visualization. Later on different visualizations will be be added.

2 Supporting web services

2.1 Create/ensemble service

Since creation of input JSON for ensemble web service requires a lot of work to create a valid input JSON from outputs of rank-based web services, a supporting web service was created that simplifies the job.

URI support/create/ensemble

HTTP method POST

Input JSON Input JSON consists of a *method* key defining ensemble method and *ensembles* key defining a list of key/value pairs, where the key defines the name of the indicator, while the value consists of the *ranked_matrix* obtained from the associated output of rank service call.

```
{
  "method": "average|hierarchical|data-driven",
  "ensembles": {
    "indicator name 1":  [
      {
        "problem": "problem name",
        "result": [ {
          "algorithm": "algorithm name",
          "rank": double_value
        }, ... ]
      }, ... ],
    "indicator name 2": ...
  }
}
```

Output JSON

```
{
  "method": "average|hierarchical|data-driven",
  "problems": [
    "problem name",
    ...
  ],
  "indicators": [ {
    "indicator": "indicator name",
    "data": [ {
      "algorithm": "algorithm name",
      "problems": [
        double_value,
        ...
      ]
    }, ... ]
  }, ... ]
}
```

The result of create/ensemble service is an ensemble JSON input, where the *method* consists of all possible values (divided by the | sign) and the user must select only one of them (by deleting the others) before using it as an input JSON for ensemble service.

3 Examples of DSCTool usage

Next, we are going to provide details about how different scenarios, taken from our previous studies, can be done using the proposed web services. The examples were taken from the previous studies because i) we would like to show the

general applicability of the framework in every possible scenario independent of the data ii) the results from using it can be completely different when different data is being analyzed (e.g., statistical significance can be found or not, same is true for the practical, etc.), and iii) to test the validity of the web services implementations, meaning we can check the results with the results reported in the previous studies that were obtained using R scripts. After reproducing the same results reported in the previous studies, to make users familiar with how it can be used, we provide one example for every web service. First, we are going to present an example from [1], where three single-objective optimization algorithms are compared with regard to the obtained solutions' values (i.e. DSC ranking scheme). Second, we will provide an example from [2], where three single-objective optimization algorithms are compared with regard to the distribution of the solutions in the search space (i.e. eDSC ranking scheme). Finally, we are going to present an example that involves multi-objective optimization. Since individual performance indicator ranking is applied in the same manner as single-objective optimization (only solution values are replaced with performance indicator values), we are going to show data-driven ensemble techniques using a set of performance indicators taken from [3, 4]. With these three samples we should cover the basics of how DSCTool should be used in any scenario dealing with single- or multi-objective optimization.

Since the input and output JSONs are too big to be presented in the paper, the reader can find all used example files in the following subsections at `http://cs.ijs.si/dl/dsctool/JSON_FILE`, where JSON_FILE needs to be replaced with appropriate name as stated in the examples below. Additionally, in the following examples one also needs to replace "username" with an actual username and JSON_FILE with the JSON found in it.

3.1 Rank web service

Let us take results of three algorithms, BSif [5], BSqi [5], and CMA-MSR [6] and their results on 22 benchmark functions at dimension 10 from BBOB 2015 competitions [7], as presented in [1]. First we need to transform results into the appropriate JSON structure as can be seen at `http://cs.ijs.si/dl/dsctool/dsc.json`. Here we have inserted appropriate information into data key. In addition, we needed to define if we are interested in practical or statistical significance. Since we are interested in statistical significance, the epsilon key is set to 0, otherwise this would need to be set according to the desired practical level. We are also not interested in Monte Carlo simulations of obtained results, so we set the Monte_carlo_iterations key also to 0, otherwise we would need to set this integer according to the desired number of simulations. Lastly, we needed to define the statistical test which will be applied to determine if there is any significance in the data. In our case, we used Kolmogorov-Smirnov test (KS) with significance level 0.05 (alpha). The actual call to the *rank service* would be the following.

```
curl --user username
```

```
  -H "Accept: application/json"
  -X POST https://ws.ijs.si:8443/dsc-1.5/service/rank
  -H "Content-Type: application/json"
  -d @dsc.json
```

After executing the above call, we receive a result seen at `http://cs.ijs.si/dl/dsctool/dsc.result`. The result consists of the rankings for all algorithms and all benchamrk problems together with two important pieces of information. The first one gives us an information whether a parametric test is applicable as an omnibus statistical test and the second one gives us an information about which statistical tests are appropriate. Using the results from the rank web service, we can use the omnibus statistical web service to see if there is any statistical significance in the data. For this reason, we need to construct an input JSON for the *omnibus web service* as can be seen at `http://cs.ijs.si/dl/dsctool/omnibus.json`. Since we are replicating the tests found in [1], we selected the Friedman test with a significance level of 0.05.

```
curl --user username
  -H "Accept: application/json"
  -X POST https://ws.ijs.si:8443/dsc-1.5/service/omnibus
  -H "Content-Type: application/json"
  -d @omnibus.json
```

The results of the above call to omnibus web service, can be seen at `http://cs.ijs.si/dl/dsctool/omnibus.result`. From it, we can observe that the null hypothesis is rejected, with acquired p-value of 0.007 (as was the case in [1]). Since omnibus test showed that there is a statistical significance between algorithms, we need to make a post-hoc test, to see which algorithms made the difference. In our case we selected the algorithm with lowest mean value (CMA-MSR) as our control algorithm and compare it to the other two algorithms. So, we took algorithm_means from omnibus result, set k key (number of algorithms) to 3, n key (number of benchmark functions) to 22, base_algorithm as CMA-MSR, and as statistical method we selected Friedman test (since omnibus test was also performed using Friedman test) with significance level 0.05.

```
curl --user username
  -H "Accept: application/json"
  -X POST https://ws.ijs.si:8443/dsc-1.5/service/posthoc
  -H "Content-Type: application/json"
  -d @posthoc.json
```

Looking at the results of the above call to the *posthoc web service*, found at `http://cs.ijs.si/dl/dsctool/posthoc.result`, we can see that CMA-MSR significantly outperforms the other algorithms. Since omnibus test and post-hoc procedures are the same for all rankings, we will not show them for the examples in following subsections.

3.2 Multivariate web service

For the *multivariate web service* we are going to look at the example from [2], where we compare three algorithms: the Cauchy-EDA, MCS, and iAMALGAM on results from 22 benchmark functions on dimension 2 taken from the BBOB 2009 competition [8]. To make this comparison, we should create an input JSON as seen at `http://cs.ijs.si/dl/dsctool/multivariate.json`. First we need to insert algorithm results in the data key and determine the method to be applied on the data. To replicate situation from the article, we set desired_distribution key to 0 (to prefer coarser distributions) and significance level to 0.05. After that, we can call the *multivariate service* as:

```
curl --user username
  -H "Accept: application/json"
  -X POST https://ws.ijs.si:8443/dsc-1.5/service/multivariate
  -H "Content-Type: application/json"
  -d @multivariate.json
```

From the result of the above call found at `http://cs.ijs.si/dl/dsctool/multivariate.result`, we acquire the same rankings for the algorithms as found in [2], which can be then used for further omnibus testing. Since the explanation of the results and further steps are the same as for the *rank web service*, we will not discuss them here.

3.3 Ensemble web service

For the *ensemble web service* we are going to take example from [4], where we compare three multi-objective algorithms: the DEMO_SP2, DEMO_NS-II, and NSGA-II on results from 16 test problems with 4 different performance indicators (hypervolume, epsilon, r2, gen-dist) [9]. We will skip the first step which is acquirement of rankings for compared algorithms for all 4 performance indicators by using the rank web service, since this was already shown in Section 3.1. Taking this into account, we can create an input JSON as seen at `http://cs.ijs.si/dl/dsctool/ensemble.json` by setting all the relevant data in problems and indicators keys, and select the desired ensemble method. In our case we decided to select a data-driven one.

```
curl --user username
  -H "Accept: application/json"
  -X POST https://ws.ijs.si:8443/dsc-1.5/service/ensemble
  -H "Content-Type: application/json"
  -d @ensemble.json
```

From the result of the above call found at `http://cs.ijs.si/dl/dsctool/ensemble.result`, we acquire the same rankings for the algorithms as found in [4], which can be then used for further omnibus testing.

4 DSCTool clients

For programming language R there is already a client implemented. The documentation for it can be found at `https://ws.ijs.si:8443/dsc-1.5/R-client.pdf`.

References

[1] T. Eftimov, P. Korošec, B. K. Seljak, A novel approach to statistical comparison of meta-heuristic stochastic optimization algorithms using deep statistics, Information Sciences 417 (2017) 186–215.

[2] T. Eftimov, P. Korošec, A novel statistical approach for comparing meta-heuristic stochastic optimization algorithms according to the distribution of solutions in the search space, Information Sciences 489 (2019) 255–273.

[3] T. Eftimov, P. Korošec, B. K. Seljak, Comparing multi-objective optimization algorithms using an ensemble of quality indicators with deep statistical comparison approach, in: 2017 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2017, pp. 1–8.

[4] T. Eftimov, P. Korošec, B. K. Seljak, Data-driven preference-based deep statistical ranking for comparing multi-objective optimization algorithms, in: International Conference on Bioinspired Methods and Their Applications, Springer, 2018, pp. 138–150.

[5] P. Pošík, P. Baudiš, Dimension selection in axis-parallel brent-step method for black-box optimization of separable continuous functions, in: Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference, ACM, 2015, pp. 1151–1158.

[6] A. Atamna, Benchmarking ipop-cma-es-tpa and ipop-cma-es-msr on the bbob noiseless testbed, in: Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference, ACM, 2015, pp. 1135–1142.

[7] Black-box optimization benchmarking at CEC'2015 (CEC-BBOB), `http://coco.gforge.inria.fr/doku.php?id=cec-bbob-2015`, [Online; accessed 1-May-2019].

[8] N. Hansen, S. Finck, R. Ros, A. Auger, Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions, RR-6829, INRIA, 2009.

[9] T. Tušar, B. Filipič, Differential evolution versus genetic algorithms in multiobjective optimization, Evolutionary Multi-Criterion Optimization. EMO 2007. Lecture Notes in Computer Science 4403 (2007) 257–271.