Optimization Problem Inspector: A Tool for Analysis of Industrial Optimization Problems and Their Solutions

Tea Tušar
Jordan N. Cork
Andrejaana Andova
Bogdan Filipič
Jožef Stefan Institute and Jožef Stefan International Postgraduate School
Ljubljana, Slovenia
{tea.tusar,jordan.cork,andrejaana.andova,bogdan.filipic}@ijs.si

Abstract

This paper presents the Optimization Problem Inspector (OPI) tool for assisting researchers and practitioners in analyzing industrial optimization problems and their solutions. OPI is a highly interactive web application requiring no programming knowledge to be used. It helps the users to better understand their problem by: 1) comparing the landscape features of the analyzed problem with those of some well-understood reference problems, and 2) visualizing the values of solution variables, objectives, constraints and any other user-specified solution parameters. The features of OPI are presented using a bi-objective pressure vessel design problem as an example.

Keywords

optimization, black-box problems, sampling, problem characterization, visualization

1 Introduction

Industrial optimization problems often require simulations to evaluate solutions. For example, in electrical motor design [18, 19], assessing the efficiency and electromagnetic performance of a proposed design is done by running a simulator that analyzes the motor magnetic field and flux distribution. Such evaluations are black boxes to the user and the optimization algorithm alike, i.e., the underlying functions cannot be explicitly expressed, which makes the problem hard to understand and solve.

The established way to gain a better understanding of industrial problems is through the analysis of their solutions. Depending on the problem at hand, this can be a challenging task, as industrial problems often have a large number of variables, multiple objectives and constraints [20].

The Optimization Problem Inspector (OPI) presented in this paper is a tool conceived to ease this task for both problem experts and optimization algorithm developers. OPI provides two ways to further the understanding of an optimization problem:

(1) It computes a set of landscape features of the analyzed problem and compares them to those of well-understood reference problems.
(2) It provides visualizations of solutions through the values of their variables, objectives, constraints and any other user-specified solution parameters.

OPI is a web application, implemented by a Python library called `optimization-problem-inspector` included in the PyPi Python package index[1]. It is highly interactive and requires no programming knowledge to be used.

Freely available contemporary software tools for multiobjective optimization, such as DESDEO [12], jMetal [7] (and jMetalPy [2]), the MOEA Framework [8], ParadisEO-MOEO [10], platEMO [17], pygmo [3], pymoo [4], and Scilab [15], provide the implementation of various optimization algorithms and test problems. While the majority of them include some visualization of solutions, the plots are mostly focused on showing algorithm results for the purpose of comparing algorithm performance and not to increase problem understanding. In addition, none of these tools compute additional problem features as OPI does. Therefore, OPI brings a unique perspective to optimization problem analysis and understanding.

Next, Section 2 presents the real-world problem that will be used to showcase the features of OPI in Section 3. The paper concludes with some remarks in Section 4.

2 Real-World Use Case

Our chosen real-world problem is a version of the well-known pressure vessel design problem, first proposed more than 30 years ago [16]. In this work, we adapt the formulation from [5] to handle the pressure vessel volume as a constraint, as well as an objective. We also remove one unnecessary constraint and use the original boundary constraints for the first two variables.

A pressure vessel is a tank, designed to store compressed gasses or liquids. It consists of a cylindrical middle part capped at both ends by hemispherical heads. The pressure vessel has four design variables (see Figure 1): the shell thickness, $x_1 = T_s$, the head thickness, $x_2 = T_h$, the inner radius, $x_3 = R$, and the length of the cylindrical section of the vessel, $x_4 = L$. The two thickness variables are integer multiples of 0.0625 inches, which correspond to the available thicknesses of rolled steel plates, while the length and the radius are continuous. The problem has three constraints, two on the search variables and one on

[1] https://pypi.org/project/optimization-problem-inspector/

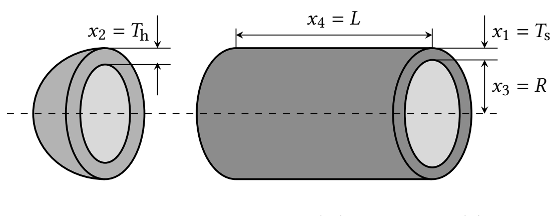

Figure 1: Pressure vessel design variables.

Information Society 2024, 7–11 October 2024, Ljubljana, Slovenia
© 2024 Copyright held by the owner/author(s).
https://doi.org/10.70314/is.2024.scai.8265

the volume. Its two objectives are to minimize the total costs, including the costs of the material, forming and welding, and to maximize the volume. The problem is formally defined as follows:

$$\begin{aligned}
\min \quad & f_1(\mathbf{x}) = 0.6224 z_1 x_3 x_4 + 1.7781 z_2 x_3^2 + 3.1661 z_1^2 x_4 \\
& \qquad + 19.84 z_1^2 x_3 \\
\max \quad & f_2(\mathbf{x}) = \pi x_3^2 x_4 + \frac{4}{3}\pi x_3^3 \\
\text{subject to} \quad & g_1(\mathbf{x}) = 0.0193 x_3 - z_1 \le 0 \\
& g_2(\mathbf{x}) = 0.00954 x_3 - z_2 \le 0 \\
& g_3(\mathbf{x}) = f_2(\mathbf{x}) \ge 1\,296\,000 \\
& x_1 \in \{18, \dots, 32\} \\
& x_2 \in \{10, \dots, 32\} \\
& x_3, x_4 \in [10, 200] \\
\text{where} \quad & z_1 = 0.0625 x_1 \\
& z_2 = 0.0625 x_2
\end{aligned}$$

3 Optimization Problem Inspector Features

OPI is a web application, organized into five functional sections and a help section, providing guidance to the user. OPI expects the user to provide the problem specification and its data—evaluated problem solutions. Then, it generates and visualizes comparisons to artificial reference problems and visualizes the provided data.

Next, we will describe the main features of OPI through its five content sections: problem specification, sample generation, data, comparison to reference problems, and data visualization.

3.1 Problem Specification

In the first OPI section, the user can provide the specification of the industrial problem to be studied. The tool needs this information to properly generate the samples, described in the Section 3.2, and setup the visualisations.

The problem specification must be given in the `yaml` file format and needs to contain some basic information about problem parameters (variables, objectives, constraints) to be included in the analysis. OPI can handle one or more objectives and zero or more constraints. In addition to variables, objectives and constraints, the user can specify any number of other parameters that they want analyzed and visualized, for example, the name of the algorithm that found a solution or the time required to evaluate a solution.

For each of the parameters, the user needs to specify its name and its grouping (whether it is a variable, objective, constraint or something else). For variables, their type (continuous, integer or categorical) and the upper and lower bounds (for non-categorical types) are also required. An example `yaml` file, specifying a constrained multiobjective problem with several variables, is already provided within the tool to guide the user.

For the pressure vessel design problem, we can input four variables (first two are integer and last two are continuous), two objectives and three constraints. Alternatively, we can decide to skip the individual constraints and only use the total constraint violation instead.

3.2 Sample Generation

In OPI, a sample is a set of x-values, corresponding to the variables set in the problem specification section. In other words, a sample is a set of non-evaluated solutions.

If needed, the sample can be generated by the tool itself, based on the variable information provided in the problem specification step. However, this is not a required step in using OPI. A user that already has a set of (evaluated) solutions to work with can skip it and input the data directly (see Section 3.3).

Sample generation requires one to choose the number of desired samples, set to a default of 100, and the sample generation method. Three sample generation methods are supported: random, Sobol and Latin Hypercube, with random sampling being the default. The user may alter the settings of these sampling methods, such as the random generator seed. Selecting the button to generate and download the sample will download it in a `csv`-formatted file.

In the pressure vessel use case, OPI warns the user that not all sample generation methods are appropriate. In fact, the Sobol sampler and the Latin Hypercube Sampler are not compatible with non-continuous parameters. If used nevertheless, they may produce unexpected results.

3.3 Data

In OPI, the data is essentially a set of evaluated solutions, where each solution must contain a value for all objectives, constraints and other parameters included in the problem specification. The evaluation is conducted externally to the tool.

The data needs to be uploaded in a file in `csv` format. If any parameters from the problem specification are missing from the data, the tool will display a warning message. Any data parameters that are not included in the problem specification, are ignored without raising any warnings. When correctly input, the user will be able to view the data they have input, inspecting it in tabular format.

Inputting the data completes the setup stage of the process. The user may then begin generating visualisations to assist them in understanding their problem.

3.4 Comparison to Reference Problems

The first visualization mechanism provided by OPI visually compares the problem to a set of artificial reference problems with known properties. This is conducted by displaying the landscape features of the user-defined problem alongside the same features of each of the reference problems in a parallel coordinates plot. The plot is interactive—the user can highlight some of the problems by brushing along one of the parallel axes. In addition, the feature values can be viewed in a table and downloaded to a file in `csv` format.

The reference problems can be set by the user, however, confined within the collection labelled here as GBBOB, i.e., generalised BBOB, where BBOB stands for the well-known suite of 24 Black-Box Optimization Benchmarking problems with diverse properties [9]. OPI provides a generator of GBBOB problems that match the analyzed problem in terms of the number of variables and objectives and the presence or absence of constraints. For objectives and (optionally) the constraint, any single-objective BBOB problem instance can be used. The user can specify the desired GBBOB problems in the `yaml` format. OPI already contains five GBBOB problems to start.

A problem can be characterized by a large number of features, most hard to interpret by a human. In OPI, we included the following problem landscape features that are understandable to an expert user [1, 11, 13, 14]: `CorrObj`, `MinCV`, `FR`, `constr_obj_corr`, `H_MAX`, `UPO_N`, `PO_N` and a set of neighborhood features. `CorrObj`

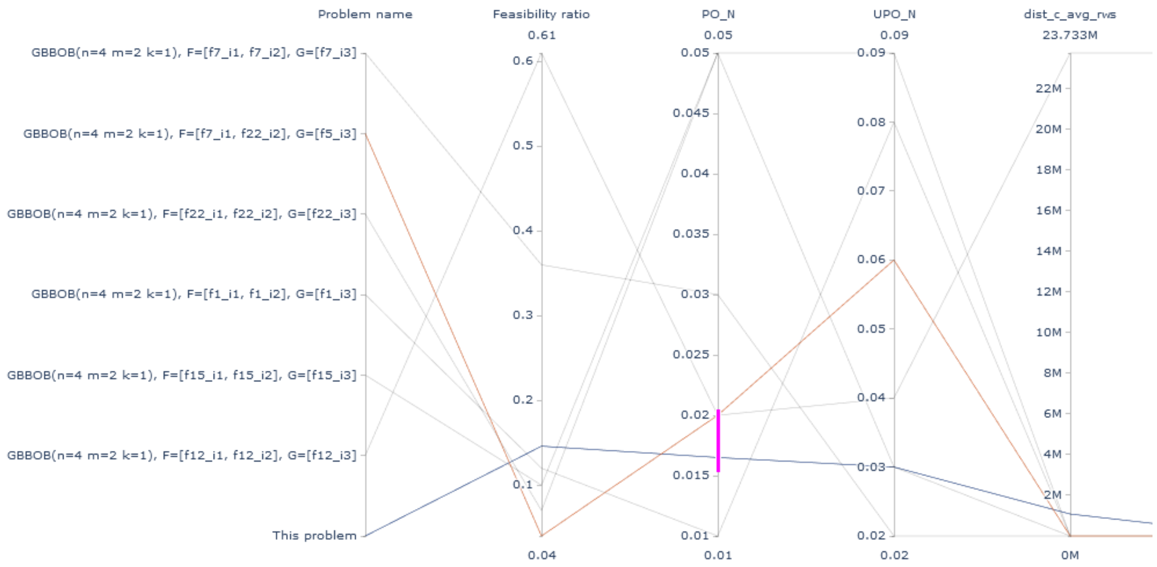

Figure 2: The initial part of the parallel coordinate plot visualizing feature values for the analyzed problem and the chosen set of artificial test problems.

is a feature that shows the correlation between the objectives. `MinCV` represents the minimum constraint violation among all solutions in the population. `FR` represents the proportion of feasible solutions in the population. `constr_obj_corr` presents the maximum correlation between the constraints and all the problem objectives. `H_MAX` is the maximum information content among all objectives. `UPO_N` is the proportion of unconstrained non-dominated solutions, while `PO_N` is the proportion of the constrained non-dominated solutions. The neighbourhood features denoted by `neighbourhood_feats` are a collection of features explaining the neighborhood of solutions, e.g., how many neighbors of a solution dominate the solution, how many neighbors are dominated by the solution, how many are incomparable to the solution, how close the neighboring solutions are, etc. OPI offers a total of 16 features, but the user can choose which to compute and visualize.

Figure 2 shows the initial part of the parallel coordinates plot (as the entire plot would not fit the paper) for the pressure vessel problem. In the comparison, we use the default five GBBOB reference problems as well as a custom created one. We notice that the pressure vessel problem is most similar to the custom GBBOB problem with the first objective equal to the step ellipsoid function f_7, the second to the multimodal peaks function f_{22}, and the linear constraint f_5. This similarity might be due to our mixed-integer problem containing plateaus in the continuous landscape space in which the features are computed, which is similar to the step ellipsoid function, and having linear constraints.

3.5 Data Visualization

In the data visualization section of the web application, the supplied data can be visualized using either a scatter plot matrix or a parallel plot. In both cases, the user can choose which problem parameters to visualize among all those listed in problem specification. Additionally, a simple data filtering that limits any variable between the desired minimum and maximum values is also supported and can be manipulated via the OPI interface in `yaml` format. The parameter used for coloring the solutions, as well as the color map, can also be specified by the user. Both visualizations support interaction and can be downloaded in `html` or `png` format.

3.5.1 Scatter Plot Matrix. The scatter plot matrix consists of n^2 plots for n chosen problem parameters as it contains 2-D scatter plots for all possible parameter pairs. In OPI, the user can apply brushing and linking to select the desired solutions in one or more of the scatter plots. These are then highlighted in all scatter plots in the matrix.

Figure 3 shows such a scatter plot for our pressure vessel problem. This visualization includes data from two sources. The first comes from a random sampling of the search space (shown in light blue) and the second from running the NSGA-II algorithm [6] on this problem for $2 \cdot 10^6$ function evaluations to achieve a good approximation of the Pareto front (shown in black). The two sources are set apart by a custom parameter that is then used for coloring the solutions. Some solutions from Figure 3 are highlighted – see the rectangle in the (x_3, x_1) scatter plot (third from the left in the top row).

These plots clearly show the linear relationship of the near-optimal solutions between x_1 and x_2 as well as x_1 and x_3. When only f_1 and f_2 are chosen, it is distinctively visible that the Pareto set approximation is piece-wise linear and disconnected.

3.5.2 Parallel Coordinates Plot. The parallel coordinates plot shows all chosen parameters as parallel coordinates and solutions as lines in the plot. Similarly as with the scatter plot matrix, interaction via brushing and linking is supported to select solutions that fit the desired values.

4 Conclusions

This work presented the features of Optimization Problem Inspector – a web application to support problem experts and algorithm designers in gaining a better understanding of industrial optimization problems. The tool provides comparisons to well-understood reference problems and interactive and highly-customizable visualizations, which can be exported in `html` and `png` formats.

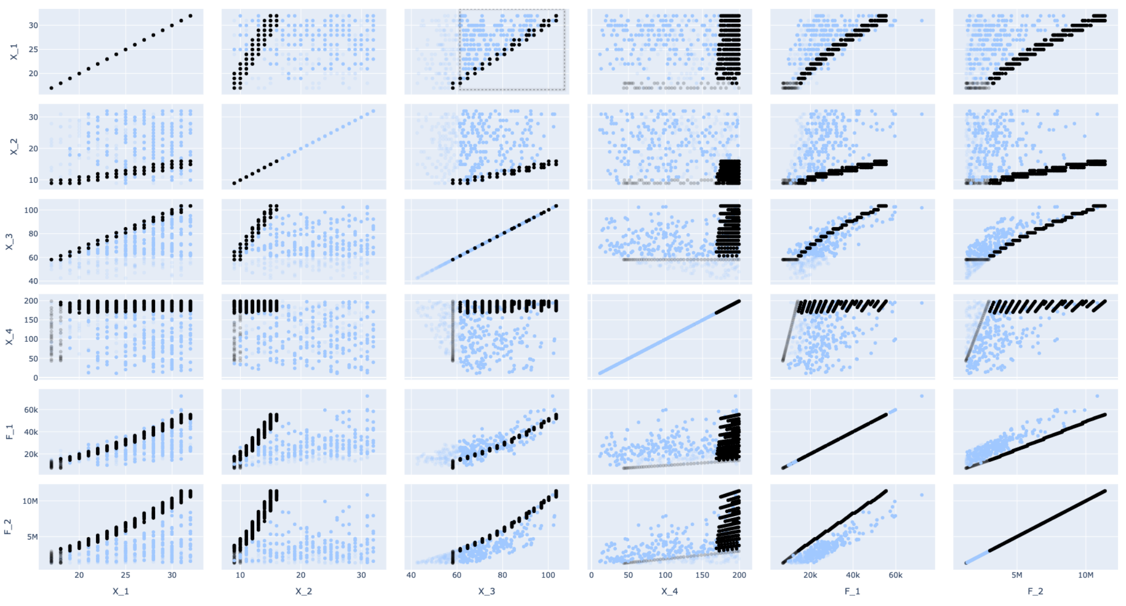

Figure 3: Random (light blue) and near-optimal (black) solutions of the pressure vessel design problem visualized in OPI with a scatter plot matrix containing variables x_1 to x_4 and objectives f_1 and f_2.

Samples can be exported and solutions imported using the standard csv format, which makes the data exchange between OPI and various optimization software easy to do. OPI functionality is made to be simple and at the same time flexible. Therefore, it is utilisable by non-experts and experts, alike, providing a wide range of angles from which to view the problems.

Acknowledgements

The authors acknowledge the financial support from the Slovenian Research and Innovation Agency (research core funding No. P2-0209 "Artificial Intelligence and Intelligent Systems", and project No. N2-0254 "Constrained Multiobjective Optimization Based on Problem Landscape Analysis") and from the Jožef Stefan Innovation Fund (project "A Tool for Analysis of Industrial Optimization Problems and Their Solutions"). This publication is also based upon work from COST Action "Randomised Optimisation Algorithms Research Network" (ROAR-NET), CA22137, supported by COST (European Cooperation in Science and Technology). We are grateful to Jernej Zupančič for implementing the core functionalities of the Optimization Problem Inspector.

References

[1] Hanan Alsouly, Michael Kirley, and Mario Andrés Muñoz. 2023. An instance space analysis of constrained multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 27, 5, 1427–1439. DOI: 10.1109/TEVC.2022.3208595.

[2] Antonio Benítez-Hidalgo, Antonio J. Nebro, José García-Nieto, Izaskun Oregi, and Javier Del Ser. 2019. jMetalPy: A Python framework for multi-objective optimization with metaheuristics. *Swarm and Evolutionary Computation*, 51, 100598. DOI: 10.1016/J.SWEVO.2019.100598.

[3] Francesco Biscani and Dario Izzo. 2020. A parallel global multiobjective framework for optimization: pagmo. *Journal of Open Source Software*, 5, 53, 2338. DOI: 10.21105/joss.02338.

[4] Julian Blank and Kalyanmoy Deb. 2020. Pymoo: Multi-objective optimization in Python. *IEEE Access*, 8, 89497–89509. DOI: 10.1109/ACCESS.2020.2990567.

[5] Carlos A. Coello Coello. 2002. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191, 11, 1245–1287. DOI: https://doi.org/10.1016/S0045-7825(01)00323-1.

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6, 2, 182–197. DOI: 10.1109/4235.996017.

[7] Juan José Durillo and Antonio J. Nebro. 2011. jMetal: A Java framework for multi-objective optimization. *Advanced Engineering Software*, 42, 10, 760–771. DOI: 10.1016/J.ADVENGSOFT.2011.05.014.

[8] David Hadka. 2024. MOEA Framework: A free and open source Java framework for multiobjective optimization. https://github.com/MOEAFramework/MOEAFramework. Computer software, version 4.4. (2024).

[9] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. 2009. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829. INRIA. https://hal.inria.fr/inria-00362633v2.

[10] Arnaud Liefooghe, Laetitia Jourdan, and El-Ghazali Talbi. 2011. A software framework based on a conceptual unified model for evolutionary multiobjective optimization: ParadisEO-MOEO. *European Journal of Operational Research*, 209, 2, 104–112. DOI: 10.1016/J.EJOR.2010.07.023.

[11] K. M. Malan, J. F. Oberholzer, and A. P. Engelbrecht. 2015. Characterising constrained continuous optimisation problems. In *Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC 2015)*, 1351–1358. DOI: 10.1109/CEC.2015.7257045.

[12] Giovanni Misitano, Bhupinder Singh Saini, Bekir Afsar, Babooshka Shavazipour, and Kaisa Miettinen. 2021. DESDEO: The modular and open source framework for interactive multiobjective optimization. *IEEE Access*, 9, 148277–148295. DOI: 10.1109/ACCESS.2021.3123825.

[13] Mario A. Muñoz, Michael Kirley, and Saman K. Halgamuge. 2015. Exploratory landscape analysis of continuous space optimization problems using information content. *IEEE Transactions on Evolutionary Computation*, 19, 1, 74–87. DOI: 10.1109/TEVC.2014.2302006.

[14] Cyril Picard and Jürg Schiffmann. 2021. Realistic constrained multiobjective optimization benchmark problems from design. *IEEE Transactions on Evolutionary Computation*, 25, 2, 234–246. DOI: 10.1109/TEVC.2020.3020046.

[15] Philippe Roux and Perrine Mathieu. 2016. Scilab: I. Fundamentals. In *Scilab, from theory to practice*. D-Booker Editions.

[16] E. Sandgren. 1990. Nonlinear integer and discrete programming in mechanical design optimization. *Journal of Mechanical Design*, 112, 2, 223–229.

[17] Ye Tian, Ran Cheng, Xingyi Zhang, and Yaochu Jin. 2017. PlatEMO: A MATLAB platform for evolutionary multi-objective optimization. *IEEE Computational Intelligence Magazine*, 12, 4, 73–87. DOI: 10.1109/MCI.2017.2742868.

[18] Tea Tušar, Peter Korošec, and Bogdan Filipič. 2023. A multi-step evaluation process in electric motor design. In *Slovenian Conference on Artificial Intelligence, Proceedings of the 26th International Multiconference Information Society (IS 2023)*. Vol. A. Jožef Stefan Institute, Ljubljana, Slovenia, 48–51.

[19] Tea Tušar, Peter Korošec, Gregor Papa, Bogdan Filipič, and Jurij Šilc. 2007. A comparative study of stochastic optimization methods in electric motor design. *Applied Intelligence*, 27, 2, 101–111. DOI: 10.1007/S10489-006-0022-2.

[20] Koen van der Blom, Timo M. Deist, Vanessa Volz, Mariapia Marchi, Yusuke Nojima, Boris Naujoks, Akira Oyama, and Tea Tušar. 2023. Identifying properties of real-world optimisation problems through a questionnaire. In *Many-Criteria Optimization and Decision Analysis: State-of-the-Art, Present Challenges, and Future Perspectives*. Natural Computing Series. Dimo Brockhoff, Michael Emmerich, Boris Naujoks, and Robin C. Purshouse, editors. Springer, 59–80. DOI: 10.1007/978-3-031-25263-1_3.