

SOFTWARE METAPAPER

Rhodium: Python Library for Many-Objective Robust Decision Making and Exploratory Modeling

Antonia Hadjimichael¹, David Gold¹, David Hadka² and Patrick Reed¹

¹ School of Civil and Environmental Engineering, Cornell University, Ithaca, NY, US

² Microsoft, US

Corresponding author: Antonia Hadjimichael (ah986@cornell.edu)

Rhodium is an open source Python library for robust decision making (RDM), many-objective robust decision making (MORDM), and exploratory modeling. These decision-support frameworks enable the identification of robust strategies for the management of complex environmental systems, by evaluating the tradeoffs among candidate strategies, and characterizing their vulnerabilities. Robust strategies refer to management options that perform sufficiently well or acceptably under a range of potential system conditions, rather than optimally in a single, nominal state of the world. Exploratory modeling allows for the simulation of the system under an ensemble of states of the world, so as to discover the ones with consequential effects on the system [1]. Rhodium facilitates rapid application of the RDM and MORDM frameworks by providing a suite of optimization, visualization, scenario discovery, and sensitivity analysis functions. Rhodium is written in Python and can interface with models written in Python, C and C++, Fortran, R, and Excel. The source code is freely available at <https://github.com/Project-Platypus/Rhodium>.

Keywords: many-objective optimization; exploratory modelling; robust decision making; deep uncertainty; Python; visual analytics; sensitivity analysis; SALib; environmental systems; Project Platypus

Funding statement: This work was partially supported by the National Science Foundation under Grant No. (1639268). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

(1) Overview

Introduction

Managing complex environmental systems necessitates identifying management strategies that are robust across many potential future states of the world (SOWs). This task is further complicated by the presence of deep uncertainty, the case where planners or experts cannot agree on prior probability density functions for the parameters of the system model, or even on the model representation itself [2, 3]. In such cases, it is often desirable to search for management strategies that perform sufficiently well under a range of possible system representations and conditions (or SOWs) [4]. This is in contrast to canonical decision making approaches that seek to find the 'optimal' solution for a nominal SOW or a set of SOWs with well-described probabilities of occurrence [5, 6]. These issues have sparked the development of several "bottom-up" decision support frameworks (for recent reviews see [4, 7]). Robust Decision Making (RDM) [8, 9] is one of the seminal bottom-up approaches, which seeks to support in the identification of robust solutions (out of a set of pre-specified candidates) that result in satisfactory performance across a broad ensemble of plausible future

SOWs [10, 3]. RDM uses several "satisficing" or "regret" criteria to rank alternative strategies on their performance across candidate SOWs, rather than identifying those with optimal performance in a single system instantiation [3]. As a result, robust solutions often have to trade optimal performance for reduced sensitivity to incorrect assumptions [3]. RDM also seeks to determine which uncertain parameters are most likely to result in consequential failures for the candidate management strategies in a process termed "scenario discovery" [11].

Several authors have argued that analyzing a set of pre-specified alternatives may potentially overlook innovative sets of candidate actions as well as their inherent performance trade-offs across conflicting stakeholder objectives [12, 13, 14]. The many-objective robust decision making (MORDM) framework overcomes this challenge by combining many-objective evolutionary optimization, RDM, and interactive visual analytics to discover performance tradeoffs, vulnerabilities, and dependencies [12]. The main steps of the MORDM framework are: (1) problem formulation, including the identification of uncertainties, possible actions to be taken by the decision maker, relationships mapping actions to outcomes, and

performance measures to gauge success; (2) generating alternative management actions using multi-objective evolutionary algorithms (MOEAs); (3) using exploratory modeling to broadly sample possible SOWs, perform uncertainty analysis and identify robust solutions; and (4) performing sensitivity analysis and/or scenario discovery to find the key factors that most strongly affect the robustness of candidate strategies. MORDM emphasizes the importance of treating the problem formulation as a learning process, with multiple feedbacks across the four stages of the framework application. This constructive decision aiding approach, allows decision makers to explore what is attainable with regards to performance, as well as how this is shaped by the choice of objectives, constraints and decision variables used in the optimization, and the choice of uncertainty representation [15].

As the aim of MORDM and other decision making under deep uncertainty frameworks is to provide decision support for complex problems, there has been a need for software tools guiding and facilitating the application of these frameworks. The most prominent examples of such tools already available are the EMA Workbench [16] (an open source Python library) and OpenMORDM [14] (an open source R library). The two toolkits facilitate the generation of alternative strategies, exploratory modeling, and the application of sensitivity analysis and scenario discovery methods for complex environmental systems. Both also provide interfaces to existing simulation models, with OpenMORDM able to evaluate R-based models, and EMA Workbench able to analyze simulation models in Python, Vensim, NetLogo, and Excel. This paper introduces Rhodium, an open source Python library, complementing

these two software tools in supporting rapid and iterative application of MORDM for models written in Python, C and C++, Fortran, R, and Excel. **Figure 1** formally illustrates the four main steps of the MORDM framework and their application through the Rhodium library. Furthermore, Rhodium introduces a unified and extensible way to describe models across all the languages (C, C++, Fortran, R and Excel) and handles the invocation of the underlying model, with the use of several wrappers available in the library. All three toolkits allow for the exploration of alternatives by loading external datasets generated as multi-objective optimization outputs. The toolkits also support multi-objective optimization; as with the EMA Workbench, Rhodium employs the Platypus library (<https://github.com/Project-Platypus/Platypus>), a framework for evolutionary computing in Python supporting multiple multi-objective evolutionary algorithms (MOEAs). OpenMORDM only supports multi-objective optimization using the Borg MOEA [17]. As with the other two toolkits, Rhodium also allows for the application of scenario discovery methods, specifically the Patient Rule Induction Method (PRIM) [18] and Classification and Regression Trees (CART) [19]. Multiple sensitivity analysis techniques can also be applied on a Rhodium model through SALib [20]. Finally, Rhodium simplifies MORDM analysis by allowing the user to use expressive language. For example, when “brushing” (i.e., limiting the values of a certain parameter to a specific range to indicate potential preference), one can provide simply input an expression (e.g., “reliability > 0.95”). Rhodium interprets that expression and applies the brushing as well as shows the relevant expression in the

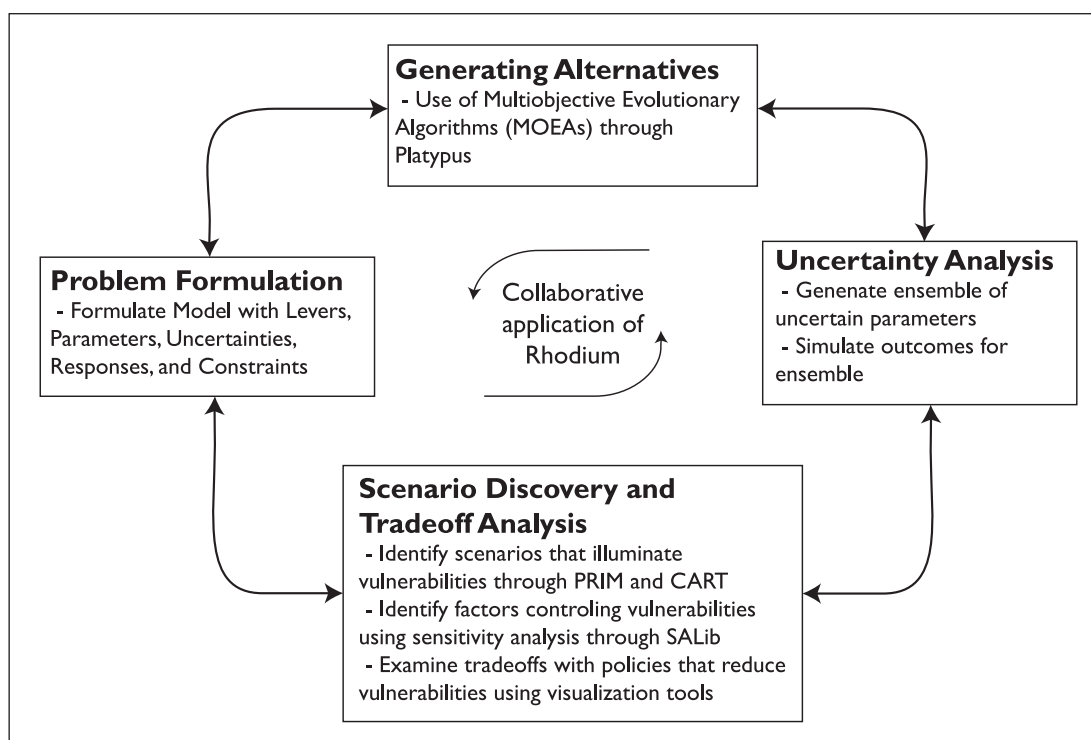


Figure 1: The four steps of the many objective robust decision making (MORDM) framework, as applied using the Rhodium library. The process typically begins with problem formulation. Each step facilitates stakeholder collaboration using the generated visual analytics. Figure adapted from [12].

legend. The parallel development and cross-fertilization between Rhodium and EMA Workbench enable seamless functionality between the two libraries for users that would like to use a mixture of the tools available in each.

Implementation and architecture

Rhodium has been developed as a Python library containing data structures and classes necessary to perform MORDM analysis. It is part of Project Platypus (<https://github.com/Project-Platypus/>), which is a collection of libraries for optimization, data analysis, and decision making, to be used with Python 3.5. The Platypus library (within Project Platypus) supports multi-objective optimization using a variety of MOEAs: NSGA-II, NSGA-III, MOEA/D, IBEA, Epsilon-MOEA, SPEA2, GDE3, OMOPSO, SMPSO, and Epsilon-NSGA-II. J3 is a desktop application written in JavaScript, for producing and sharing high-dimensional, interactive scientific visualizations. It allows the user to visualize and explore thousands of data points by leveraging hardware accelerated graphics, while simultaneously supporting animations and interactivity. J3 is paired with J3Py, a Python module that allows the user to launch J3 from within the Python environment and use it to analyze data output from the other Project Platypus libraries. Finally, the PRIM library can be used to apply the Patient Rule Induction Method [18] for the purposes of scenario discovery, explained in more detail in the following sections.

Rhodium employs these and other scientific computing libraries to support a variety of MORDM tasks, as described in more detail below, and presented in **Figure 2**. The core classes used by Rhodium are:

- Model and filemodel: these two classes enable the definition of the simulation model to be analyzed using Rhodium. The former represents models writ-

ten as Python functions and can be used directly; the latter enables the import of an externally produced file containing the model description in various file formats.

- Parameter: This class can be used to define model parameters that can either be constant, controlled by a lever (see below), or subject to uncertainty. Uncertain model parameters can be used to perform exploratory modeling and scenario discovery, during which the parameters are sampled to be then used for strategy reevaluation.
- Response: Responses represent model outputs. They can be of type MINIMIZE, MAXIMIZE, or INFO. If the type is set to MINIMIZE or MAXIMIZE, then the response may be used during optimization. If the model response is of type INFO, the default, then the response is purely for informative purposes (e.g., exploring how alternative strategies affect the system beyond its objectives) and does not participate in optimization.
- Constraint: This class is used to set “hard constraints” that must be satisfied in order for a candidate solution to be considered feasible. Constraints can be set on any parameter or response. To define a constraint, one can use a valid Python expression that references said parameter or response, or a function using a dictionary of parameters and responses.
- Lever: Defines an adjustable lever that controls a model parameter, to be used during optimization. Five different types of lever are distinguished within the library: RealLever, IntegerLever, CategoricalLever, PermutationLever, SubsetLever. When defining a model lever, the number of decision variables required to represent this lever need to be set by the user to be passed by Rhodium to Platypus for optimization.

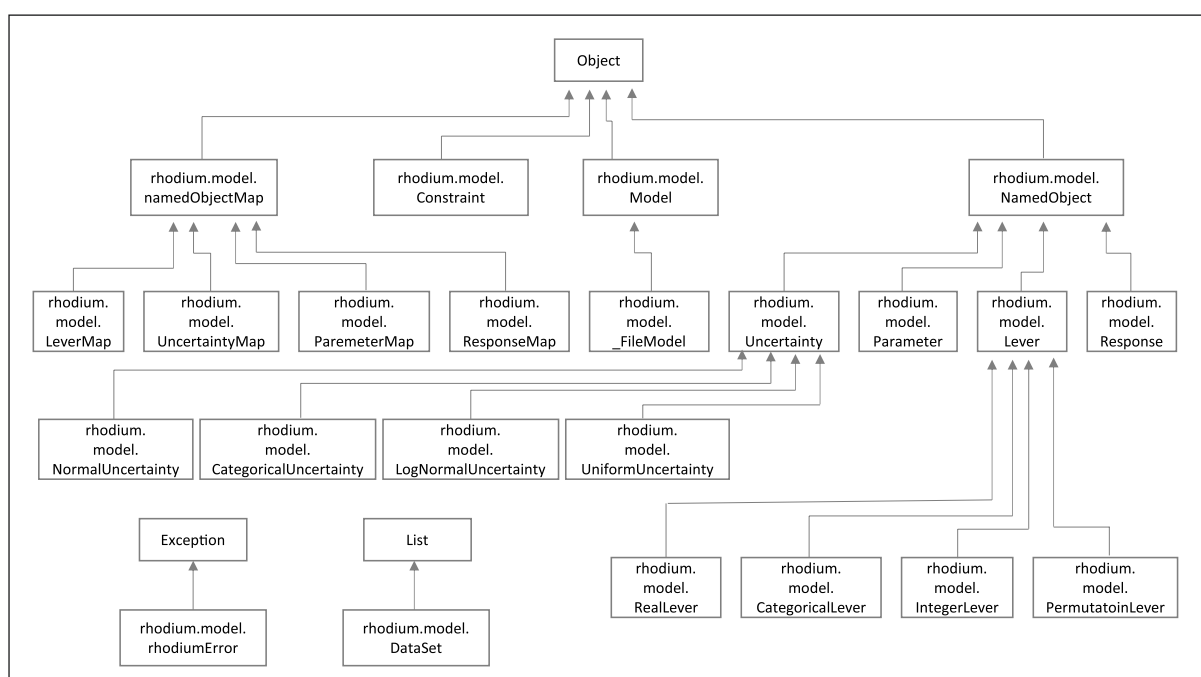


Figure 2: Core classes of the Rhodium library.

- **Uncertainty:** This class is used to set the uncertainty of a model parameter, for the purposes of exploratory modeling and scenario discovery. There are five different types of uncertainty defined in Rhodium: `UniformUncertainty`, `NormalUncertainty`, `LogNormalUncertainty`, `IntegerUncertainty`, and `CategoricalUncertainty`. Depending on the type, minimum and maximum parameter values, distribution parameters, and other arguments might need to be provided.

By using these classes the user can formulate and define a Rhodium model for optimization, scenario discovery, and sensitivity analysis. The library is written in a declarative manner, so all these processes are hidden from the user and handled internally by the library itself. The user only needs to describe the operation to be performed, without needing to specify all the details of how that should be done. For externally built models, several helper classes (wrappers) have been defined, that allow Rhodium to connect to models written in other languages. To perform optimization and generate alternatives, the built-in optimizer employs the Platypus library within Project Platypus. To return the optimization results (i.e., the Pareto-approximate set) use is made of the `Dataset` class.

To explore candidate solutions and tradeoffs across objectives, several tools are available for visualization of performance in two- and three- dimensional space, pairwise plots, kernel density plots, and in parallel coordinates. The library uses model input to make educated guesses on how to generate figures and legends. The plotting functions also allow for user interaction as well as brushing.

The next stage of the MORDM framework is Uncertainty Analysis, where the performance of candidate solutions is explored under more relaxed best-estimate assumptions for the parameter values. To do so, each solution in the Pareto-approximate set is reevaluated in a large number of alternative SOWs. This step can guide the identification of solutions that have acceptable performance across a wide range of plausible future scenarios and alternative system states. There are two sampling functions available in Rhodium, one for Uniform Random Sampling (`sample_uniform`) and one for Latin Hypercube Sampling (`sample_lhs`) [21]. Several other sampling methods are available through SALib, which can be used to generate other kinds of samples to analyze using Rhodium, should the user be interested. The built-in `evaluate` function then allows the analyst to re-simulate any and all candidate solutions in all sampled SOWs, and output a `Dataset` containing the performance of each solution in each SOW.

Scenario discovery typically employs statistical cluster analysis on databases of simulation model outputs to identify simple descriptions of parameter combinations that best predict the SOWs where robust strategies perform poorly [22]. Rhodium provides two methods by which the user can perform scenario discovery: PRIM [18] and Cart [19]. For either method, the user first needs to set thresholds for each of the performance

measures, defined so as to reflect stakeholder preferences. Sampled SOWs that violate these defined thresholds are considered “vulnerable” or “unreliable” [12]. Cart is defined in Rhodium as a class of functions that takes a set of independent variables and a dependent variable in a binary classification (e.g., “reliable” and “unreliable”), and produces a decision tree for classification. To apply PRIM, Rhodium employs the PRIM module included in Project Platypus, which is a standalone version of the PRIM algorithm as implemented in the EMA Workbench, and needs to be installed separately. This version of PRIM allows for user interaction within Matplotlib’s native viewer and can handle a variety of inputs, such as Pandas dataframes, Numpy matrices, or other list-like objects. The algorithm takes a set of independent variables and a dependent variable and produces a set of “PRIM boxes” in the parametric space. Each PRIM box represents different combinations of “coverage” and “density” of reliable SOWs (i.e., boxes containing varying numbers of false-positive and false-negative classifications according to stakeholder preference). Both classification methods (Cart and PRIM) can be used by the analyst to identify easy-to-interpret orthogonal parameter ranges that produce undesirable performance for the candidate policies. Compared to the PRIM functionality within the EMA Workbench, the EMA Workbench implementation also allows for the discovery of multiple “PRIM boxes” instead of a single orthogonal region.

An alternative analysis method available to the Rhodium user is global sensitivity analysis, which can be performed to prioritize the factors (parameters) most significantly affecting the output, and, if desired, fix the factors that appear not to affect the output. The SALib Python Library [20] contains implementations of some of the most commonly used sensitivity analysis methods, including Sobol, Method of Morris, Fourier Amplitude Sensitivity Test (FAST), and Delta Moment-Independent Measure, among others. Rhodium makes use of SALib to allow its user to apply SALib tools to a Rhodium model, by defining the specific output of interest, a candidate policy to study, and the sensitivity analysis method to apply. Internal Rhodium functions then handle the sampling necessary for each sensitivity analysis method and the model evaluations necessary to return sensitivity indices for the Rhodium model. SALib is installed automatically during the Rhodium installation. Rhodium also contains several plotting options for sensitivity analysis results.

Lastly, Rhodium supports parallelization which reduces runtime during analysis. Similar to other components within Rhodium, this is designed to be extensible. Rhodium itself provides support for single-threaded evaluation and multi-process evaluation on a single computer using the `ProcessPoolEvaluator`. This is powered by Python’s built-in multiprocessing module. It also integrates with other Python libraries, including `MPIPool` and `JobLib`, for scaling out to cloud or high-performance computing architectures. Since the optimization and re-evaluation tasks are typically embarrassingly parallel, substantial speedup can be achieved, significantly reducing the time to run experiments.

Quality control

Rhodium has been successfully tested on Linux Ubuntu and CentOS, MacOS Mavericks and El Capitan, and Windows 7 and 10, with Python versions 3.5 and higher. Before using Rhodium to one's unique case study, users are advised to first execute the example Lake Problem application provided in the examples directory (under master/examples/Basic/example.py). The Lake Problem is a model of lake nutrient dynamics developed by [23], representing a pollution control problem in which a theoretical town needs to develop an emissions policy that balances its economic benefits and the quality of the lake. The system used in this example is presented and analyzed extensively by [24], and is included in the Rhodium repository for the purposes of framework illustration and verification. We present the dynamics of the system and how it can be analyzed using Rhodium below.

The lake water quality in this system can transition between an oligotrophic (healthy) equilibrium and a eutrophic (unhealthy) equilibrium, as determined by the following dimensionless equation:

$$X_{t+1} = X_t + a_t + Y_t + \frac{X_t^q}{1 + X_t^q} - bX_t \quad (1)$$

where X is the concentration of phosphorus in the lake, a are anthropogenic phosphorus inputs (controlled by emissions policy), $Y \sim LN(\mu, \sigma^2)$ are natural phosphorus inputs (uncontrolled), q is the phosphorus recycling rate in the lake, and b is the rate of phosphorus loss in the lake. Finally, this discrete-time model is using $t \in \{0, 1, 2, \dots\}$

as the time index. As detailed by [24], the term $\frac{X_t^q}{1 + X_t^q}$ represents phosphorus recycling by the lake sediment, as a function of the current phosphorus level and the term bX represents the losses of phosphorus due to sediment adsorption. As q increases, the change in recycling rate as a function of the phosphorus concentration is more precipitous, whereas as b increases, phosphorus losses become larger. The system dynamics without natural and anthropogenic inputs are presented in **Figure 3**, where the rates of phosphorus recycling (in orange) and leaving the lake (in black) are plotted as a function of the phosphorus concentration. Including the natural and anthropogenic inputs would move the curved line upwards. The points where the recycling rate equals the sinking rate (the intersections of the two lines) are considered equilibria (denoted in black and white points), two of which are stable and one is unstable. The stable equilibrium with low phosphorus concentration (bottom left) is an oligotrophic equilibrium, considered attractive. Therefore, even with higher phosphorus concentrations (moving rightward), as long as the sink flux is higher than the recycling flux, the lake concentration will return to the equilibrium. If anthropogenic emissions increase the phosphorus concentration enough to cross the unstable equilibrium (white point), then the lake concentration is irreversibly attracted to the eutrophic stable equilibrium (upper right). Management needs to therefore identify a policy that maximizes profits derived from emissions, while keeping phosphorus concentration low enough so as to not cross this tipping point to a permanently polluted lake.

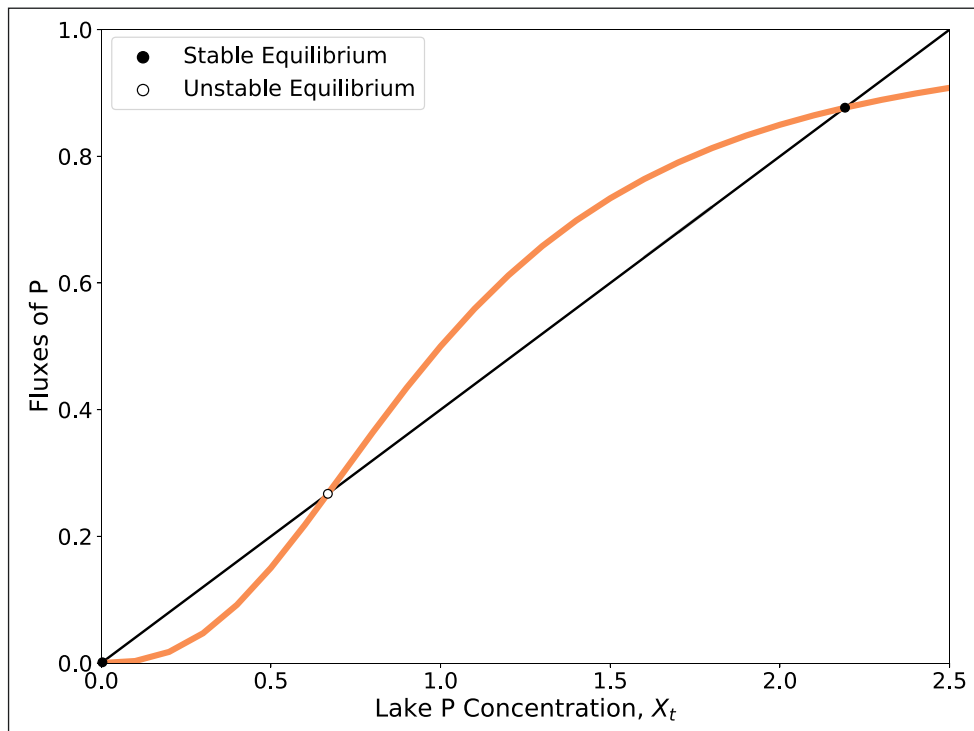


Figure 3: Non-linear dynamics of the irreversible lake model, with the phosphorus recycling in orange and phosphorus sink in black. The equilibria of this system are presented as points, with black denoting the stable equilibria and white denoting the unstable equilibrium, i.e., the tipping point of this system. Increasing the phosphorus concentration and crossing this tipping point puts the system at an irreversible eutrophic state.

To analyze this system using Rhodium, the user needs to execute the file in a Python environment. The script performs multi-objective optimization on the simulation model, trying to identify emissions policies that maximize utility (profits), minimize the maximum phosphorus concentration over the time horizon, maximize policy inertia, and maximize reliability (time below tipping point). The formulation of this problem follows that of [24] (referred to therein as “Intertemporal”), which also elaborates on the objective equations chosen. The optimization is performed in the example using the NSGAI MOEA for 10,000 NFE (number of function evaluations). The generated alternatives (management policies) are stored and can be visualized using a suite of visualization options presented in **Figure 4**.

Figure 4 (a) and **(b)** present the generated alternatives in two-dimensional scatter plots; **Figure 4 (a)** shows each point (candidate strategy) colored by its performance on a third objective, whereas **Figure 4 (b)** displays the “brushing” functionality allowing users to indicate preference on one or more model responses. In this particular example, **Figure 4 (b)** presents solutions meeting preference criteria of reliability ≥ 0.5 and utility > 0.5 , with solutions not meeting either criterion set to grey color. **Figure 4 (a)** and **(b)** show that with increasing profits (utility), the phosphorus concentration in the lake must also increase, indicating a strong tradeoff between the two. When the phosphorus concentration increases

so as to cross the tipping point (left side of the figures), higher utility values are achieved by those solutions, albeit in a eutrophic lake. In **Figure 4 (b)**, one can see that those solutions actually achieve very low values in the reliability objective (shown in grey on the right). The solutions in grey on the left side of the figure are those failing to meet the brushing criterion of utility > 0.5 . These results are in general agreement with those reported by [24].

Figure 4 (c) displays the performance of each alternative in a three-dimensional scatter plot and **Figure 4 (d)** shows the performance of the solutions in a parallel axis plot. This style of plot represents the performance on each objective by a vertical axis. The points where each line (candidate solution) crosses a vertical axis indicate the performance value for that objective. The figure is oriented so an upward shift in one of the vertical axes indicates increased preference in the equivalent objective performance. Brushing can also be applied here, indicating, in this case a stakeholder preference for the reliability objective to only have values above 0.2. All solutions not meeting that criterion are then set to grey. Even though there is a stochastic component to both the system equations and the optimization, if the installation was performed successfully, analysts testing the library using this example should be able to generate figures that look very similar without errors.

To test the library’s scenario discovery functionality, the example guides the user on how to perform a

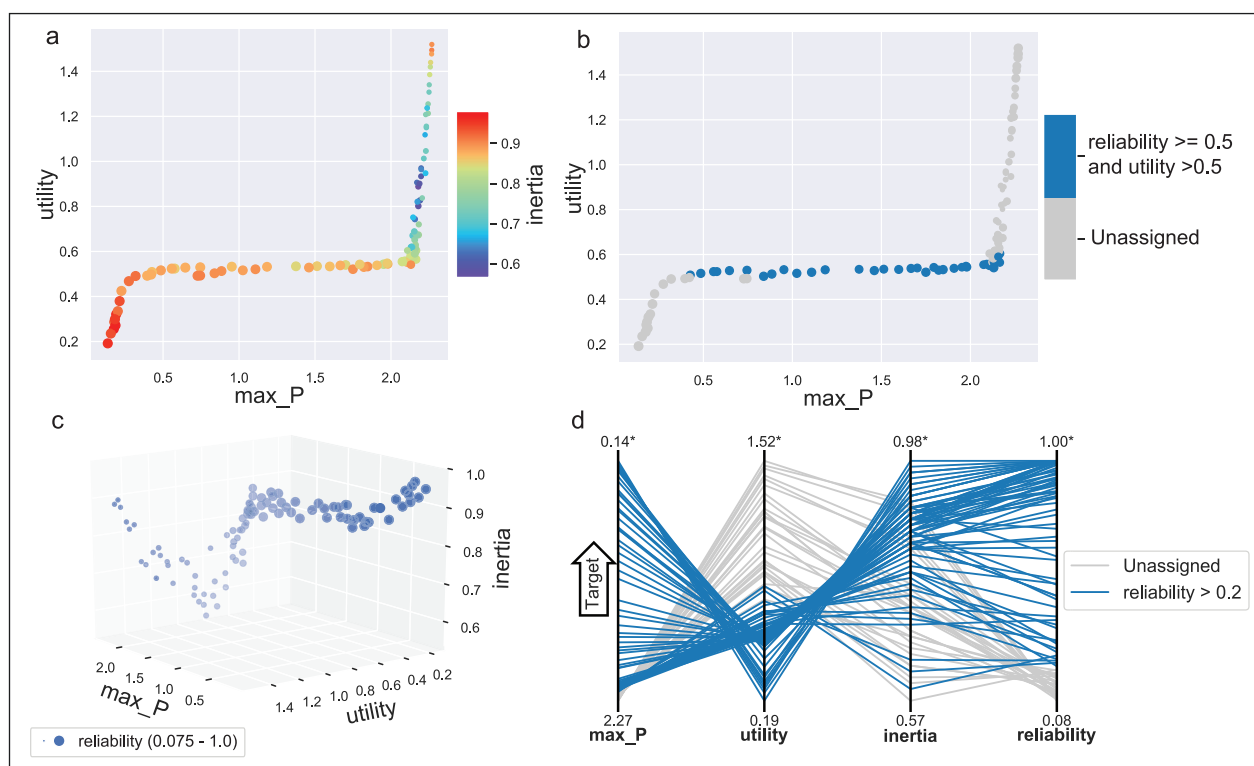


Figure 4: Four alternative visualization options available in the Rhodium library. Each panel presents the performance of each candidate solution in: **(a)** a two-dimensional scatter plot with the color of each point set by the performance of each solution on one of the objectives; **(b)** a two-dimensional scatter plot with the color of each point set by whether it meets user-set preference criteria; **(c)** a three-dimensional scatter plot with the size of each point set by the performance of each solution on one of the objectives; **(d)** a parallel axis plot, where each objective is represented by a vertical axis and the performance of each solution (each line) is indicated by the point where it crosses each axis.

Latin Hypercube Sample on the uncertain parameters, generating 100 alternative SOWs. For the illustrative purposes of this example, one of the candidate policies is selected and re-evaluated in all sampled SOWs. The criterion used to evaluate success and failure in the sampled SOWs is whether the reliability of the policy (time below tipping point) is ≥ 0.9 . Users can then perform scenario discovery through PRIM and Cart by executing the respective commands. Output figures from executing these two scenario discovery methods are presented in **Figure 5**. **Figure 5 (a)** presents one of the PRIM boxes identified, representing parameter ranges where the policy chosen is meets the stakeholder-set criteria (the SOWs indicated in red). The user can interact with this figure and navigate to different PRIM boxes representing parameter ranges that capture more of the reliable parameter combinations

but also introduce some less reliable combinations. As explained above, this functionality allows stakeholders to explore how many false-positives and false-negatives they are willing to accept in their classification of reliable SOWs. **Figure 5 (b)** shows the classification tree produced by the application of Cart. This method produces similar orthogonal divisions of the parametric space, with easy-to-interpret “larger than” and “smaller than” conditions on the parameters. Stakeholders can navigate the tree by moving downward, which represents additional splits on the parameter values. Users replicating this example should be able to easily generate these figures following the script provided. The results in **Figure 5** show that the most important parameters controlling the success or failure of the selected policy are b and q , also in agreement with the findings by [24]. This is explained by the fact that

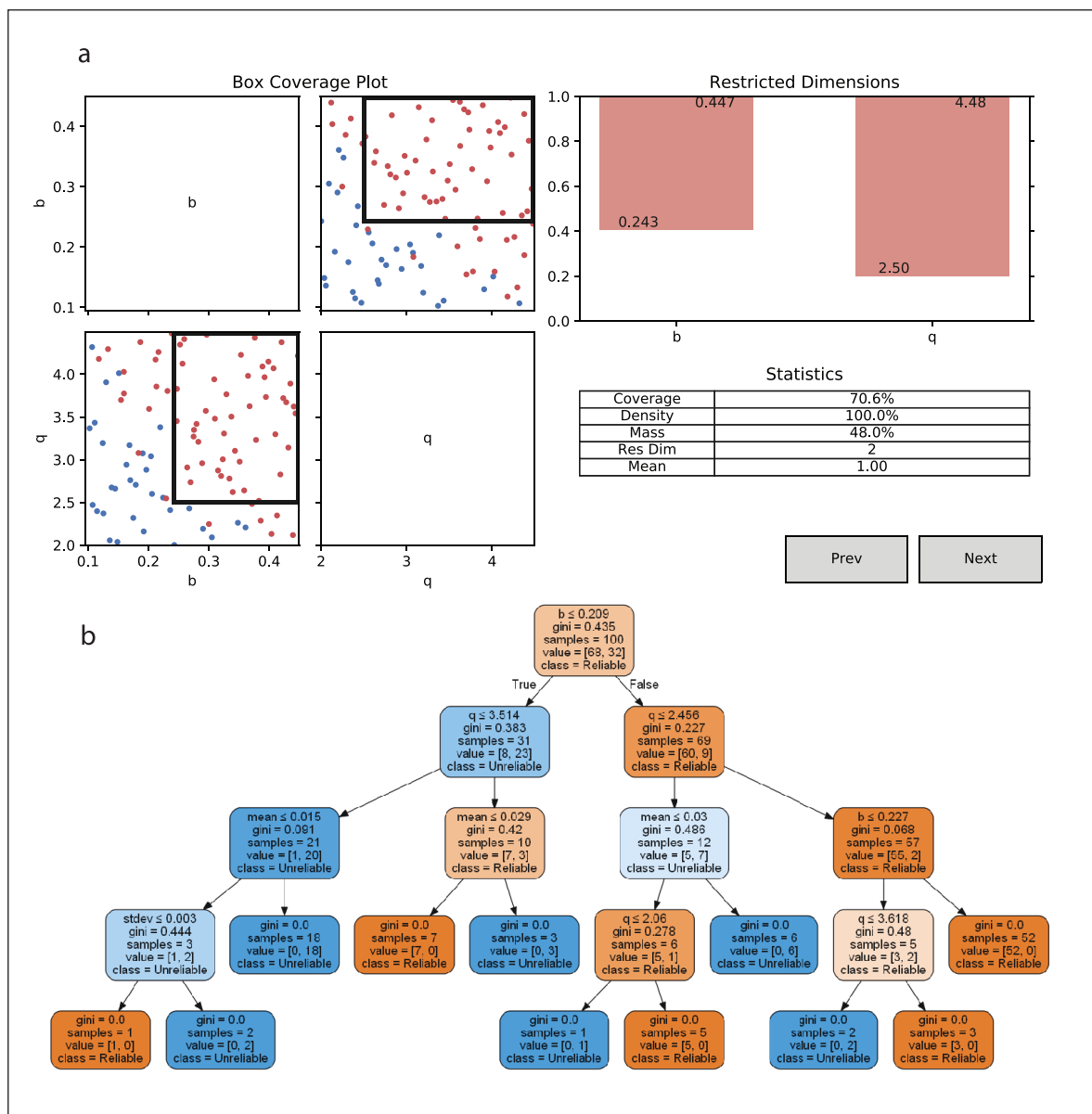


Figure 5: Scenario discovery results as produced by PRIM and Cart, using the Rhodium library. **(a)** One of the identified PRIM boxes, representing parameter ranges where the policy chosen is always reliable. The “Prev” and “Next” buttons allow the user to navigate to other PRIM boxes with different coverage and density of reliable SOWs. **(b)** The classification tree produced by Cart. Each node in the tree represents an orthogonal split on the range of one of the parameters. Moving downward on the tree indicates additional divisions of the parametric space.

the values of b and q determine the tipping point of this system, which is intrinsically related to the criterion set (time below tipping point).

To test the sensitivity analysis functionality, new users are guided to the `sensitivity_analysis.py` example script. This script uses the same model to demonstrate four sensitivity analysis methods available in the SALib library [20]. SALib is a Python library containing implementations of multiple global sensitivity analysis methods. This library can be used in simulation, optimization, and modeling applications to calculate the influence of model inputs or parameters on outputs of interest. This example applies the Fourier Amplitude Sensitivity Test (FAST) [25], the Delta Moment-Independent Method [26], Sobol Sensitivity Analysis [27], and the Method of Morris [28]. Executing the script correctly should produce sensitivity indices resulting from each method as well as their confidence intervals.

(2) Availability

Operating system

Rhodium can run on Linux Ubuntu and CentOS, MacOS Mavericks and El Capitan, and Windows 7 and 10, with Python versions 3.5 and higher installed.

Programming language

Python 3.5+

Additional system requirements

None.

Dependencies

Platypus

PRIM

The following dependencies are handled during installation:

matplotlib

mpldatacursor

numpy

pandas

pydot

SALib

scipy

seaborn

six

sklearn

List of contributors

David Hadka

Software location

Archive

Name: GitHub

Persistent identifier: <https://github.com/Project-Platypus/Rhodium/>

Licence: GNU General Public License v3.0

Publisher: David Hadka

Version published: 1.0

Date published: 25/10/2015

Code repository

Name: GitHub

Persistent identifier: <https://github.com/Project-Platypus/Rhodium/>

Licence: GNU General Public License v3.0

Date published: 25/10/2015

Language

English.

(3) Reuse potential

Rhodium is a flexible library of functions that can be applied in any RDM, MORDM, and exploratory modeling application, where robust management strategies need to be identified. The library can be used as an interface with existing simulation models, define computational experiments to be performed with such models through optimization, and visualize and analyze the optimization outputs. The functionality also supports exploratory modeling, as well as scenario discovery and sensitivity analysis methods. Users interested only in the investigation of externally obtained multi-objective optimization results can simply import the data and make use of functionality provided for analysis. Even though its presentation in this paper and the illustrative example were focused on complex environmental systems, Rhodium (as with the MORDM framework) is applicable to any decision-making problem with many objectives in the presence of deep uncertainties. The library has been applied and tested on multiple examples that can be found in the examples directory of the repository, including the lake problem example, as elaborated in the Quality control section.

Acknowledgements

We would like to thank Jan Kwakkel (TU Delft) for his valuable feedback during the development of the Rhodium framework.

Competing Interests

The authors have no competing interests to declare.

References

1. **Bankes S.** Exploratory Modeling for Policy Analysis. *Operations Research*. 1993; 41(3): 435–449. DOI: <https://doi.org/10.1287/opre.41.3.435>
2. **Knight FH.** Risk, Uncertainty and Profit. Courier Dover Publications; 1921.
3. **Lempert RJ, Collins MT.** Managing the Risk of Uncertain Threshold Responses: Comparison of Robust, Optimum, and Precautionary Approaches. *Risk Analysis*. 2007; 27(4): 1009–1026. DOI: <https://doi.org/10.1111/j.1539-6924.2007.00940.x>
4. **Herman JD, Reed PM, Zeff HB, Characklis GW.** How Should Robustness Be Defined for Water Systems Planning under Change? *Journal of Water Resources Planning and Management*. 2015; 141(10): 04015012. DOI: [https://doi.org/10.1061/\(ASCE\)WR.1943-5452.0000509](https://doi.org/10.1061/(ASCE)WR.1943-5452.0000509)

5. **Borgomeo E, Mortazavi-Naeini M, Hall JW, Guillod BP.** Risk, Robustness and Water Resources Planning Under Uncertainty. *Earth's Future*. 2018; 6(3): 468–487. DOI: <https://doi.org/10.1002/2017EF000730>
6. **Herman JD, Quinn JD, Steinschneider S, Giuliani M, Fletcher S.** Climate Adaptation as a Control Problem: Review and Perspectives on Dynamic Water Resources Planning Under Uncertainty. *Water Resources Research*. 2020; 56(2): e24389. DOI: <https://doi.org/10.1029/2019WR025502>
7. **Dittrich R, Wreford A, Moran D.** A survey of decision-making approaches for climate change adaptation: Are robust methods the way forward? *Ecological Economics*. 2016; 122: 79–89. DOI: <https://doi.org/10.1016/j.ecolecon.2015.12.006>
8. **Lempert RJ, Groves DG, Popper SW, Bankes SC.** A general, analytic method for generating robust strategies and narrative scenarios. *Management science*. 2006; 52(4): 514–528. DOI: <https://doi.org/10.1287/mnsc.1050.0472>
9. **Lempert RJ, Popper SW, Bankes SC.** Robust decision making: coping with uncertainty. *The Futurist*. 2010; 44(1): 47.
10. **Lempert RJ.** A new decision sciences for complex systems. *Proceedings of the National Academy of Sciences*. 2002; 99(suppl 3): 7309–7313. DOI: <https://doi.org/10.1073/pnas.082081699>
11. **Bryant BP, Lempert RJ.** Thinking inside the box: A participatory, computer-assisted approach to scenario discovery. *Technological Forecasting and Social Change*. 2010; 77(1): 34–49. DOI: <https://doi.org/10.1016/j.techfore.2009.08.002>
12. **Kasprzyk JR, Nataraj S, Reed PM, Lempert RJ.** Many objective robust decision making for complex environmental systems under-going change. *Environmental Modelling & Software*. 2013; 42: 55–71. DOI: <https://doi.org/10.1016/j.envsoft.2012.12.007>
13. **Herman JD, Zeff HB, Reed PM, Characklis GW.** Beyond optimality: Multistakeholder robustness tradeoffs for regional water portfolio planning under deep uncertainty. *Water Resources Research*. 2014; 50(10): 7692–7713. DOI: <https://doi.org/10.1002/2014WR015338>
14. **Hadka D, Herman J, Reed P, Keller K.** An open source framework for many-objective robust decision making. *Environmental Modelling & Software*. 2015; 74: 114–129. DOI: <https://doi.org/10.1016/j.envsoft.2015.07.014>
15. **Kasprzyk J, Reed PM, Kirsch BR, Characklis GW.** Managing population and drought risks using many-objective water portfolio planning under uncertainty. *Water Resources Research*. 2009; 45(12). DOI: <https://doi.org/10.1029/2009WR008121>
16. **Kwakkel JH.** The Exploratory Modeling Workbench: An open source toolkit for exploratory modeling, scenario discovery, and (multi-objective) robust decision making. *Environmental Modelling & Software*. 2017; 96: 239–250. DOI: <https://doi.org/10.1016/j.envsoft.2017.06.054>
17. **Hadka D, Reed P.** Borg: An Auto-Adaptive Many-Objective Evolutionary Computing Framework. *Evolutionary Computation*. 2013; 21(2): 231–259. DOI: https://doi.org/10.1162/EVCO_a_00075
18. **Friedman JH, Fisher NI.** Bump hunting in high-dimensional data. *Statistics and Computing*. 1999; 9(2): 123–143. DOI: <https://doi.org/10.1023/A:1008894516817>
19. **Breiman L.** Classification and Regression Trees. New York: Routledge; 1984. DOI: <https://doi.org/10.1201/9781315139470>
20. **Herman JD, Usher W.** SALib: An open-source Python library for Sensitivity Analysis. *J Open Source Software*. 2017; 2(9): 97. DOI: <https://doi.org/10.21105/joss.00097>
21. **McKay MD, Beckman RJ, Conover WJ.** A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*. 1979; 21(2): 239–245. DOI: <https://doi.org/10.2307/1268522>
22. **Lempert R.** Scenarios that illuminate vulnerabilities and robust responses. *Climatic Change*. 2013; 117(4): 627–646. DOI: <https://doi.org/10.1007/s10584-012-0574-6>
23. **Carpenter SR, Ludwig D, Brock WA.** Management of Eutrophication for Lakes Subject to Potentially Irreversible Change. *Ecological Applications*. 1999; 9(3): 751–771. DOI: [https://doi.org/10.1890/1051-0761\(1999\)009\[0751:MOEFLS\]2.0.CO;2](https://doi.org/10.1890/1051-0761(1999)009[0751:MOEFLS]2.0.CO;2)
24. **Quinn JD, Reed PM, Keller K.** Direct policy search for robust multi-objective management of deeply uncertain socio-ecological tipping points. *Environmental Modelling & Software*. 2017; 92: 125–141. DOI: <https://doi.org/10.1016/j.envsoft.2017.02.017>
25. **Cukier RI, Fortuin CM, Shuler KE, Petschek AG, Schaibly JH.** Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. I Theory. *The Journal of chemical physics*. 1973; 59(8): 3873–3878. DOI: <https://doi.org/10.1063/1.1680571>
26. **Borgonovo E.** A new uncertainty importance measure. *Reliability Engineering & System Safety*. 2007; 92(6): 771–784. DOI: <https://doi.org/10.1016/j.res.2006.04.015>
27. **Sobol IM.** Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. *Mathematics and computers in simulation*. 2001; 55(1–3): 271–280. DOI: [https://doi.org/10.1016/S0378-4754\(00\)00270-6](https://doi.org/10.1016/S0378-4754(00)00270-6)
28. **Morris MD.** Factorial sampling plans for preliminary computational experiments. *Technometrics*. 1991; 33(2): 161–174. DOI: <https://doi.org/10.1080/00401706.1991.10484804>

How to cite this article: Hadjimichael A, Gold D, Hadka D, Reed P 2020 Rhodium: Python Library for Many-Objective Robust Decision Making and Exploratory Modeling. *Journal of Open Research Software*, 8: 12. DOI: <https://doi.org/10.5334/jors.293>

Submitted: 23 August 2019

Accepted: 30 April 2020

Published: 09 June 2020

Copyright: © 2020 The Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC-BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. See <http://creativecommons.org/licenses/by/4.0/>.

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press.

OPEN ACCESS 