Environmental Modelling & Software 96 (2017) 239–250

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

The Exploratory Modeling Workbench: An open source toolkit for exploratory modeling, scenario discovery, and (multi-objective) robust decision making

Jan H. Kwakkel

Faculty of Technology, Policy and Management, Jaffalaan 5, PO Box 5015, 2600 GA, Delft, Netherlands

ARTICLE INFO

Article history:
Received 4 February 2017
Received in revised form
30 June 2017
Accepted 30 June 2017

Keywords:
Deep uncertainty
Exploratory modeling
Scenario discovery
Many-objective robust decision making

ABSTRACT

There is a growing interest in model-based decision support under deep uncertainty, reflected in a variety of approaches being put forward in the literature. A key idea shared among these is the use of models for exploratory rather than predictive purposes. Exploratory modeling aims at exploring the implications for decision making of the various presently irresolvable uncertainties. This is achieved by conducting series of computational experiments that cover how the various uncertainties might resolve. This paper presents an open source library supporting this. The Exploratory Modeling Workbench is implemented in Python. It is designed to (i) support the generation and execution of series of computational experiments; and (ii) support the visualization and analysis of the results from the computational experiments. The Exploratory Modeling Workbench enables users to easily perform exploratory modeling with existing models, identify the policy-relevant uncertainties, assess the efficacy of policy options, and iteratively improve candidate strategies.

Software availability

Name of Software: Exploratory Modeling Workbench
Description: The Exploratory Modeling Workbench is an open source Python library for exploratory modeling. Exploratory modeling underpins the various model-based approaches for decision making under deep uncertainty. The library can be used to develop interfaces to existing simulation models, define computational experiments to conduct with those models, analyze the results of these experiments, and store the results. The software is available through pip, a Python package manager: pip install ema_workbench. It depends on the Python scientific computing stack (numpy, scipy, matplotlib) as well as seaborn and ipyparallel (both are available through the conda package manager). Optional dependencies are platypus (available through github) for many-objective optimization, SALib (available through the pip package manager) for global sensitivity analysis, and mpld3 (available through conda) for interactive visualizations of PRIM
Developer: Jan H. Kwakkel (j.h.kwakkel@tudelft.nl) with contributions from M. Jaxa-Rozen, S. Eker, W. Auping, E. Pruyt, and C. Hamarat
Source language: Python
Supported systems: unix, linux, windows, Mac
License: BSD 3 clause

1. Introduction

In many planning problems, planners face major challenges in coping with uncertain and changing physical conditions, and rapid unpredictable socio-economic developments. How should planners and decision makers address this confluence of uncertainty? Given the presence of irreducible uncertainties, there is no straightforward answer to this question. Effective decisions must be made under unavoidable uncertainty (Dessai et al., 2009; Lempert et al., 2003). In recent years, this has been labeled as decision making under deep uncertainty: the various parties to a decision do not know or cannot agree on the system and its boundaries; the outcomes of interest and their relative importance; the prior probability distribution for uncertain inputs to the system (Lempert et al., 2003; Walker et al., 2013); or decisions are made over time in dynamic interaction with the system and cannot be considered

E-mail address: j.h.kwakkel@tudelft.nl.

http://dx.doi.org/10.1016/j.envsoft.2017.06.054

independently (Haasnoot et al., 2013; Hallegatte et al., 2012).

There is a rapidly growing interest in the challenge of offering decision support under deep uncertainty (Maier et al., 2016). A variety of approaches have been put forward including Robust Decision Making (Groves and Lempert, 2007; Lempert et al., 2006), Many Objective Robust Decision Making (Kasprzyk et al., 2013), Adaptive Policy Making (Hamarat et al., 2013; Kwakkel et al., 2010; Walker et al., 2001), Dynamic Adaptive Policy Pathways (Haasnoot et al., 2013), Info-Gap decision analysis (Ben Haim, 2001), Real Options (de Neufville and Scholtes, 2011), and Decision Scaling (Brown et al., 2012; LeRoy Poff et al., 2015). There are three ideas that underpin this literature:

1. *Exploratory modeling*: in the face of deep uncertainty, one should explore the consequences of the various presently irreducible uncertainties for decision making (Lempert et al., 2006; Weaver et al., 2013). This exploration uses model-based scenario techniques for the systematic exploration of a very large ensemble of plausible futures (Bankes, 1993, 2002; Bankes et al., 2013; Van Asselt and Rotmans, 2002).
2. *Adaptive planning*: decision robustness is to be achieved through plans that can be adapted over time in response to how the future actually unfolds (Haasnoot et al., 2013; Kwakkel et al., 2010; Wilby and Dessai, 2010)
3. *Decision Support*: the aim of the various deep uncertainty approaches is to enable joint sense-making amongst the various parties to decide on the basis of thousands to millions of simulation model results, covering impacts on a wide variety of outcomes of interest, regarding the concerns of the various actors within the decision problem and the consequences of various means of resolving these concerns (Herman et al., 2015; Tsoukiàs, 2008).

In parallel to the development of various approaches for supporting decision making under deep uncertainty, software tools are being developed to support the application of these approaches. Examples include the closed-source Computer Assisted Reasoning software used by the RAND Corporation, the open source Scenario Discovery Toolkit (Bryant, 2014), and the openMORDM library (Hadka et al., 2015).

From an analytical perspective, all model-based approaches for supporting decision making under deep uncertainty are rooted in the idea of exploratory modeling (Bankes, 1993; Bankes et al., 2013). Traditionally, model-based decision support is based on a predictive use of models. Simulation models are used to predict future consequences, and decisions are optimized in light of this. Under deep uncertainty, this predictive use of models is highly misleading. Instead, models should be used in an exploratory fashion, for what-if scenario generation, for learning about system behavior, and for the identification of critical combinations of assumptions that make a difference for policy (Weaver et al., 2013).

In this paper, we introduce the Exploratory Modeling Workbench. The Exploratory Modeling Workbench is an open source library for performing exploratory modeling. By extension, the workbench can be used for the various model-based approaches for decision making under deep uncertainty. In scope, it is fairly similar to the openMORDM toolkit (Hadka et al., 2015), although there are some interesting differences in the approach taken to supporting exploratory modeling. The workbench is implemented in Python. It is compatible with both Python 2.7, and Python 3.5 and higher. The latest version is available through GitHub, and a stable version can be installed using pip, one of the standard package managers for Python.

The remainder of this paper is structured as follows. Section 2 introduces a theoretical framework that underpins the design of the Exploratory Modeling Workbench. Section 3 discusses the design and key implementation details of the workbench; these are then compared and contrasted to some of the other available open source tools for mode-based decision support under deep uncertainty. Section 4 demonstrates the use of the workbench for the Lake Problem (Hadka et al., 2015; Lempert and Collins, 2007; Quinn et al., 2017; Singh et al., 2015; Ward et al., 2015). Section 5 contains some concluding remarks and a discussion of future extensions.

2. Framework

There are three key ideas that jointly underpin the design of the Exploratory Modeling Workbench. These are the XLRM framework, the use of simulation models as if they were a function, and a taxonomy of robustness frameworks. We now elaborate these three ideas and how they influence the design of the workbench.

The first idea which underpins the workbench is the system diagram (Walker, 2000), or XLRM framework (Lempert et al., 2003). This diagram is shown in Fig. 1, where X stands for the exogenous or external factors. These are factors that are outside the control of the decision-makers. L stands for policy levers. R stands for relationships inside the system, and M stands for performance metrics or outcomes of interest. In the Exploratory Modeling Workbench, this framework is used for structuring the relevant information. In exploratory modeling, the system is always conceptualized such that all uncertainties can be handled as external factors. For example, if there is structural uncertainty about specific relations inside the system, this is transformed into an exogenous parameter where each value stands for one particular representation of this relation. Exogenous factors are called uncertainties, policy levers are called levers, and performance metrics are called outcomes. These three are attributes of a model, which contains the relationships in the system.

The second idea behind the design of the Exploratory Modeling Workbench is the idea of running a simulation model as if it were a function. Adopting the XLRM notation, a simulation model is simply a function called with a set of parameters $\boldsymbol{X}$ and $\boldsymbol{L}$. The return of the function is a set of outcomes of interest $\boldsymbol{M}$. So

$$\boldsymbol{M} = \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{L}) \tag{1}$$

In the workbench, there is a wrapper around the actual model. This wrapper presents a consistent interface to the underlying model that the other parts of the Exploratory Modeling Workbench can use. This interface enables running the simulation model as if it were a function. This enables the workbench to interface with any modeling or simulation package that exposes some kind of API. For example, Vensim, a package for System Dynamics modeling exposes a DLL. The workbench can use this DLL to perform exploratory modeling with simulation models implemented in Vensim. Presently, the workbench comes with off the shelf interfaces to Vensim (system dynamics), NetLogo (agent based modeling), and Excel. Proof of concept wrappers are available for Simio and Arena (discrete event simulation), PCRaster (spatio-temporal

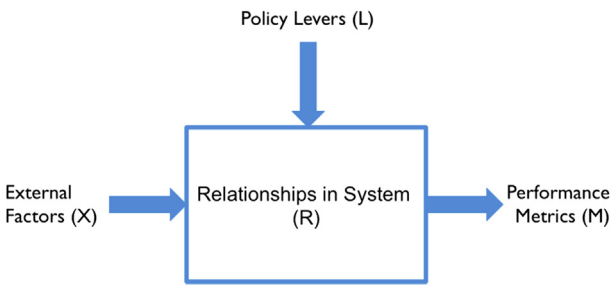

Fig. 1. The XLRM framework.

environmental models), Modflow (groundwater modeling), and GAMS (mathematical optimization).

By combining the XLRM framework and the idea of running a model as a simple function, it is possible to use the workbench to explore uncertainty pertaining to relationships within the model. This can be done either by parameterizing the uncertain relationship through some categorical variable where each category represents one possible realization of the uncertain relationship, or by working with multiple distinct simulation models. The workbench can be used for both, and implementation concerns dictate the most efficient way of exploring model uncertainty.

The third foundational idea for the workbench is a taxonomy of robustness frameworks. Most recently, Herman et al. (2015) presented such a taxonomy. The workbench evolved separately from this specific taxonomy, but is coherent with it. In the taxonomy of Herman et al. (2015), there are four components:

3. *Generation of policy options*; policy options can be pre-specified, they can be identified through design of experiments, through (many-objective) search, or they can be developed using iterative vulnerability analysis. The Exploratory Modeling Workbench can be used for all four.
4. *Generation of states of the world*; States of the world are the scenarios against which candidate policy options are evaluated. These can be pre-specified, or generated using methods for design of experiments. The workbench supports the use of user-specified states of the world, as well as Monte Carlo sampling, Latin Hypercube sampling, and Full Factorial sampling. More specialized samplers for the Sobol, Morris, and FAST sensitivity analysis techniques can also be applied using SALib (Herman and Usher, 2017).
5. *Vulnerability analysis*; Vulnerability analysis aims at identifying the relative influence of the various uncertain factors on policy robustness. This can be done through factor prioritization based approaches as found in the sensitivity analysis literature, or through scenario discovery. The workbench offers support for Scenario Discovery using either Classification and Regression Trees (Breiman et al., 1984) or the Patient Rule Induction Method (Friedman and Fisher, 1999). In addition, it implements a random forest based feature scoring approach (Breiman, 2001; Geurts et al., 2006), which can be used as an alternative to traditional global sensitivity analysis techniques, as well as a regional sensitivity analysis technique (Spear and Hornberger, 1980; Young et al., 1978).
6. *Robustness evaluation*; There is a wide literature on robustness metrics. Broadly speaking, three families of metrics can be identified: (i) satisficing metrics; (ii) regret metrics; and (iii) descriptive statistics of the distribution of outcomes over the states of the world (Kwakkel et al., 2016a). The workbench does not include pre-defined robustness metrics, but it is straightforward to implement any robustness metric in Python as part of the analysis of the results.

3. The Exploratory Modeling Workbench

The Exploratory Modeling Workbench is implemented in Python. Python is a widely used high-level open source programming language that supports various programming paradigms. Python places a strong emphasis on code readability and code expressiveness. It is increasingly popular for scientific computing purposes due to the rapidly expanding scientific computing ecosystem available for Python. The workbench extensively uses libraries from this scientific computing ecosystem and can be used with both Python 2.7, and Python 3.5 and higher.

The workbench is composed of four packages: em_framework, analysis, connectors, and util. In addition, there is a folder with a wide variety of examples. The em_framework package contains the core classes and functions of the workbench. These key components are

- *Model and FileModel*; two classes that can be used directly or as a starting point for a more detailed wrapper class of an underlying simulation model. The difference is that FileModel has a working directory where one or more files that are used for the simulation model reside. This can be the model itself, if the model is implemented in another programming language or in a simulation package (e.g. Vensim, NetLogo), or data files used by the model.
- *Evaluators*; A singleton class that is responsible for performing the computational experiments. The workbench comes with three types of evaluators: SequentialEvaluator, MultiprocessingEvaluator (for simple parallelization on a single machine using the multiprocessing library), and IpyparallelEvaluator (for distributed parallelization using the ipyparallel library).
- *Parameters*; In order to support both sampling and search over either the levers or the uncertainties, the workbench has generic Parameter classes that can be used for both. The workbench distinguishes three types of parameters: RealParameter, IntegerParameter, and CategoricalParameter.
- *Outcomes*; the workbench distinguishes between ScalarOutcomes and TimeSeriesOutcomes. In case one is using search, only ScalarOutcomes are considered as part of the optimization.
- *Data storage*; a basic easily extendable mechanism for storing the results of the computational experiments.
- *Samplers*; the workbench offers support for Monte Carlo, Latin Hypercube, and Full Factorial designs, as well as Sobol, Morris and FAST using SALib.

With these components, it is possible to realize the idea of running a simulation model as if it was a function. The parallelization support reduces runtime. The data storage is straightforward, simple, and human readable, and has proven to be sufficient in virtually all use cases.

The analysis package contains a variety of functions and classes for visualization of results, and more in-depth analysis. Visualization focuses on displaying time series data, and primarily supports the rapid exploration of the results. The analysis functionality can be decomposed into feature scoring metrics, regional sensitivity analysis, and scenario discovery. Feature scoring is a machine-learning alternative to global sensitivity analysis and can be used to get insight into the relative influence of the various uncertain factors on model outcomes. For scenario discovery, both the Patient Rule Induction Method (Friedman and Fisher, 1999) and Classification and Regression Trees (Breiman et al., 1984) are supported, and employ as much as possible an identical interface.

The connector package contains off the shelf connectors for Excel, Vensim, and NetLogo. With these off the shelf connectors, it becomes very easy to perform exploratory modeling on simulation models implemented in these simulation packages. In the trivial case, it is sufficient to only specify the uncertain factors, their ranges or sets of values, and the outcomes. Other simulation packages can easily be added if desired by extending either the Model or FileModel class.

The util package contains various utility functions. Most importantly it offers support for logging which functions even

when running in parallel on either a single machine or a high performance cluster. It also offers functionality for serializing results and storing them to disk, and loading the results back into memory. The storage format currently is a simple tarbal of csv files, with separate metadata text files. This is a format that is human readable, cross platform, and simple.

There exist various other tools for supporting exploratory modeling and scenario discovery. The closest in spirit is the openMORDM toolkit. This toolkit is implemented in R, has strong support for interactive visualization, and good support for working with BORG (Hadka and Reed, 2013) and other state of the art multi-objective optimization algorithms. The Exploratory Modeling Workbench supports both robust optimization and multi-objective optimization of outcomes for policy search, using the platypus library (https://github.com/Project-Platypus/Platypus not yet available on pypi). The interactive web-based visualization of openMORDM is better developed, in particular with respect to the visualization of multi-dimensional data. OpenMORDM comes with interactive parallel coordinates plotting functionality, something that the Exploratory Modeling Workbench lacks. In contrast, the workbench is much better developed when it comes to supporting the design and execution of experiments, including the off the shelf support for parallelization on single machines and high performance clusters. The Scenario Discovery support in the workbench is also more elaborate.

Regarding scenario discovery, the standard alternative to the workbench is the scenario discovery toolkit of Ben Bryant. This toolkit is implemented in R. The main difference presently is that the workbench does not implement the resampling statistic (Bryant and Lempert, 2010). In contrast, the workbench implementation is much easier to extend, as evidenced by its use for f-PRIM (Hallegatte et al., 2015), the ease with which it was possible to implement the multi-boxes modification suggested by Guivarch et al. (2016), and its use in implementing a random forest inspired extension (Kwakkel and Cunningham, 2016). The implementation of PRIM offered by the Exploratory Modeling Workbench also support PCA-PRIM (Dalal et al., 2013).

A new and upcoming project is Platypus. This is project on GitHub (https://github.com/Project-Platypus) containing a collection of libraries for many objective optimization (somewhat confusingly this library is also named platypus), sampling (rhodium), and scenario discovery (prim, a stand-alone port of the prim code from the Exploratory Modeling Workbench). Project Platypus is essentially a Python implementation of the MORDM toolkit and has the same developers. There is ongoing cross-fertilization between project Platypus and the Exploratory Modeling Workbench, as evidenced by the fact that the workbench uses the many-objective optimization library from Project Platypus while project Platypus has adapted the prim code from the workbench.

4. The lake problem

The Exploratory Modeling Workbench includes an example folder. This folder contains a variety of examples that demonstrate the functionality of the workbench. Many of these examples have been drawn from published cases. Here, we use the Lake Problem as an example for demonstrating some of the key functionality of the workbench. A repository with the source code and data can be found on GitHub (https://github.com/quaquel/lake_problem).

The lake problem is a stylized and hypothetical decision problem where the population of a city has to decide on the amount of annual pollution it will put into a lake. If the pollution in the lake passes a threshold, it will suffer irreversible eutrophication. In recent years, the Lake problem, in various forms, has been used to test and demonstrate several approaches for supporting decision making under deep uncertainty (Hadka et al., 2015; Lempert and Collins, 2007; Singh et al., 2015; Ward et al., 2015). The model's basic behavior is given by:

$$X_{t+1} = X_t + a_t + \frac{X_t^q}{1 + X_t^q} - bX_t + \varepsilon_t \tag{2}$$

where X_t is the pollution at time t, a_t is the rate of anthropogenic pollution at time t, b is the lake's natural recycling rate, q controls the rate of recycling of phosphor from the sediment, ε_t is the rate of natural pollution at time t and this is modeled following Singh et al. (2015), as a log normal distribution with mean μ and standard deviation σ. The rate of anthropogenic pollution a_t is the decision variable and is somewhere between 0, and 0.1. So $a_t \varepsilon [0,\ 0.1]$.

There are four outcomes of interest. The first is the average concentration of phosphor in the lake.

$$f_{phosphor} = \frac{1}{|T|} \sum_{t \in T} X_t \tag{3}$$

where $|T|$ is the cardinality of the set of points in time.

The second objective is the economic benefit derived from polluting the lake. Following Singh et al. (2015), this is defined as the discounted benefit of pollution minus the costs of having a polluted lake

$$f_{economic} = \sum_{t \in T} \left(\alpha a_t - \beta X_t^2 \right) \delta^t \tag{4}$$

where α is the utility derived from polluting, β is the cost of having polluted lake, and δ is the discount rate. By default, α is 0.04, and β is 0.08.

The third objective is related to the year over year change in the anthropogenic pollution rate.

$$f_{inertia} = \frac{1}{|T| - 1} \sum_{t=1}^{|T|} I(|a_t - a_{t-1}| < \tau) \tag{5}$$

where I is an indicator function that is zero if the statement is false, and 1 if the statement is true; τ is the threshold that is deemed undesirable, and is for illustrative purposes set to 0.02. Effectively, $f_{inertia}$ is the fraction of years where the absolute value of the change in anthropogenic pollution is smaller then. τ.

The fourth objective is the fraction of years where the pollution in the lake is below the critical threshold.

$$f_{reliability} = \frac{1}{|T|} \sum_{t \in T} I(X_t < X_{crit}) \tag{6}$$

where I is an indicator function that is zero if the statement is false, and 1 if the statement is true; X_{crit} is the critical threshold of pollution and is a function of b and q.

The lake problem is characterized by both stochastic uncertainty

Table 1
Deeply uncertain parameters.

Parameter	Range	Default value
μ	0.01–0.05	0.02
σ	0.001–0.005	0.0017
β	0.1–0.45	0.42
q	2–4.5	2
δ	0.93–0.99	0.98

and deep uncertainty. The stochastic uncertainty arises from the natural inflow. To reduce this stochastic uncertainty, multiple replications are performed and the average over the replication is taken. That is, running many replications and taking summary statics over these replications handles the stochastic uncertainty. Deep uncertainty is presented by uncertainty about the mean μ and standard deviation σ of the lognormal distribution characterizing the natural inflow, the natural removal rate of the lake β, the natural recycling rate of the lake q, and the discount rate δ. Table 1 specifies the ranges for the deeply uncertain factors, as well as their best estimate or default values.

5. Analyzing the lake problem using the Exploratory Modeling Workbench

In this section, the Exploratory Modeling Workbench is demonstrated using a stylized case. Throughout this example, it is assumed that various standard Python scientific computing packages are imported accordingly:

```
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

5.1. Specifying the lake model

The first step in performing exploratory modeling with the workbench is to implement the lake model. This implementation can be split into two separate parts. The first part is the lake model itself, as described above. We have chosen to implement this as a separate function that takes various parameters as keyword arguments with default values, and returns the four specified outcomes of interest. We also include in the function signature the possibility of passing policy lever values. For this we wrap any additional keyword arguments into a dict, and next process this dict assuming it contains the policy lever values.

```
from scipy.optimize import brentq
import math

def lake_problem(
         b = 0.42,          # decay rate for P in lake (0.42 = irreversible)
         q = 2.0,           # recycling exponent
         mean = 0.02,       # mean of natural inflows
         stdev = 0.001,     # future utility discount rate
         delta = 0.98,      # standard deviation of natural inflows
         alpha = 0.4,       # utility from pollution
         nsamples = 100,    # Monte Carlo sampling of natural inflows
         **kwargs):
    '''
    this code has been adapted from a gist provided by jdherman on
    GitHub: https://gist.github.com/jdherman/d519b3ef81840fc27d96
    '''
    decisions = [kwargs[str(i)] for i in range(100)]

    Pcrit = brentq(lambda x: x**q/(1+x**q) - b*x, 0.01, 1.5)
    nvars = len(decisions)
    X = np.zeros((nvars,))
    average_daily_P = np.zeros((nvars,))
    decisions = np.array(decisions)
    reliability = 0.0

    for _ in range(nsamples):
        X[0] = 0.0

        natural_inflows = np.random.lognormal(
                math.log(mean**2 / math.sqrt(stdev**2 + mean**2)),
                math.sqrt(math.log(1.0 + stdev**2 / mean**2)),
                size = nvars)

        for t in range(1,nvars):
            X[t] = (1-b)*X[t-1] + X[t-1]**q/(1+X[t-1]**q) + decisions[t-1] +\
                    natural_inflows[t-1]

                average_daily_P[t] += X[t]/float(nsamples)

            reliability += np.sum(X < Pcrit)/float(nsamples*nvars)

        max_P = np.max(average_daily_P)
        utility = np.sum(alpha*decisions*np.power(delta,np.arange(nvars)))
        inertia = np.sum(np.diff(decisions) > -0.02)/float(nvars-1)

        return max_P, utility, inertia, reliability
```

The second part is the specification of the Model object, which makes it possible to perform exploratory modeling on the lake model using the workbench. This involves instantiating a Model object, with a name and the lake_problem function we just defined. Next, we specify the time horizon over which we run the model. Given the model, we can now specify the uncertainties, the levers, the outcomes, and constants. Of these, only uncertainties and outcomes are always required. Constants give us the possibility to overwrite default values of parameters that are not uncertain. Here, for example, we set nsamples to 150, meaning that we will run 150 replications of the lake model for each computational experiment.

```
from ema_workbench import (Model, RealParameter, ScalarOutcome, Constant)

#instantiate the model
lake_model = Model('lakeproblem', function=lake_problem)
lake_model.time_horizon = 100

#specify uncertainties
lake_model.uncertainties = [RealParameter('b', 0.1, 0.45),
                            RealParameter('q', 2.0, 4.5),
                            RealParameter('mean', 0.01, 0.05),
                            RealParameter('stdev', 0.001, 0.005),
                            RealParameter('delta', 0.93, 0.99)]

# set levers, one for each time step
lake_model.levers = [RealParameter(str(i), 0, 0.1) for i in
                     range(lake_model.time_horizon)]

#specify outcomes
lake_model.outcomes = [ScalarOutcome('max_P', kind=ScalarOutcome.MINIMIZE),
                       ScalarOutcome('utility', kind=ScalarOutcome.MAXIMIZE),
                       ScalarOutcome('inertia', kind=ScalarOutcome.MAXIMIZE),
                       ScalarOutcome('reliability', kind=ScalarOutcome.MAXIMIZE)]

# override some of the defaults of the model
# nsamples controls the number of replications for the stochastic uncertainty
lake_model.constants = [Constant('alpha', 0.41),
                        Constant('nsamples', 150)]
```

This completes the specification of the model, which we can now use for performing exploratory modeling.

If we want to run in parallel on a single machine, we can use the MultiprocessingEvaluator.

```
from ema_workbench import (perform_experiments, ema_logging, save_results,
                           load_results)
from ema_workbench.em_framework import samplers

# turn on logging
ema_logging.log_to_stderr(ema_logging.INFO)

# perform experiments
n_scenarios = 1000
n_policies = 4
fn = './data/{} scenarios {} policies.tar.gz'.format(n_scenarios, n_policies)

counter = util.Counter()
policies = samplers.sample_levers(lake_model, n_policies,
                                  sampler=samplers.MonteCarloSampler(),
                                  name=counter)
with MultiprocessingEvaluator(lake_model) as evaluator:
    results = evaluator.perform_experiments(n_scenarios, policies)
save_results(results, fn)
```

5.2. Generation of states of the world and vulnerability analysis

The next step is to use the model for performing a first series of experiments. To do this, we first turn on the default logging functionality that comes with the workbench. Next we specify the number of scenarios and create a file name using this information. For our first set of experiments, we are going to use four randomly generated release policies. We do this by sampling over the policy levers using a Monte Carlo sampler. Given these four random policies and the number of scenarios, we can now perform the experiments. In case only a number is used to indicate the number of scenarios or the number of policies, the workbench defaults to using Latin Hypercube sampling. The workbench comes with various evaluators, which handle the execution of the experiments.

5.2.1. Visual analysis of results

The foregoing exploration results in a data set of 4000 computational experiments. We can now analyze this data set. A first

Table 2
Feature scoring using extra trees feature scoring.

	Uncertain factor	Feature score
0	b	0.889158
1	q	0.067018
2	mean	0.022405
3	policy	0.014576
4	delta	0.003477
5	stdev	0.003366

simple visualization in this case is to look at the pairwise scatter plot for each of the four objectives. This can easily be achieved with the pairs_scatter function, which is built on top of the matplotlib plotting library (Hunter, 2007). The workbench contains a variety of other convenient visualization functions for rapid visual exploration of the results, many of which are specifically useful for time series outcomes. All plotting functions return the figure instance, as well as a dictionary with all the axes, giving the user full control over the further layout etc. of the figures.

```
import ema_workbench.analysis.pairs_plotting as pairs

fig, axes = pairs.pairs_scatter(results, group_by='policy', legend=False)
plt.show()
```

5.2.2. Feature scoring

More elaborate analyses involve feature scoring and scenario discovery. Feature scoring is a family of machine learning techniques for identifying the relative importance of various features for a certain outcome or class of outcomes. Various feature scoring techniques are included in the scikit–learn library (Pedregosa et al., 2011). The workbench provides a convenient function that wraps some of the scikit-learn feature scoring techniques. As an example, we perform feature scoring on the concentration of phosphor in the lake at the end of the simulation. To achieve this, we first need to preprocess the experiment array. Since we sampled over the policy levers, this array contains the sampled values for each lever, as well as a shorthand name for each policy. In this particular case, the levers are the values for the anthropogenic release for each year. Because we have generated random values for these, we first remove them from the experiment array. We still keep the name of the policy, so if the results are particularly sensitivity to a policy, we can still detect this.

```
y = outcomes
fs_all = feature_scoring.get_feature_scores_all(x, y)
sns.heatmap(fs_all, annot=True, cmap='viridis')
plt.show()
```

```
from ema_workbench.analysis import feature_scoring

# filter out the levers from the experiment array
experiments, outcomes = results
experiments = rf.drop_fields(experiments, [str(i) for i in range(100)],
                             asrecarray=True)

x = experiments
y = outcomes['max_P']
fs, _ = feature_scoring.get_rf_feature_scores(x,y,
                                              mode=feature_scoring.REGRESSION)
```

The results of this analysis are shown in Table 2. As can be seen, the parameter b is by far the most important factor affecting the terminal value of the concentration of phosphor in the lake. The second and third parameter are the policy and the parameter q. Interestingly, the mu and sigma, which together specify the lognormal distribution of the natural inflow, have only a limited impact on the final concentration of phosphor.

This result is only applicable for a single outcome of interest. It might be convenient to have some insight into the sensitivity of each of the four outcomes of interest to the various uncertainties as well as the policy. Again, this is easily achieved with the workbench, using the get_feature_scores_all function. By default, this function uses the Extra Trees feature scoring algorithm, although other options are provided as well. The result of this code is shown in Fig. 3. Note here that the table includes a row titled model. The Exploratory Modeling workbench offers support for designing and executing experiments over more than one model, considering either the intersection or the union of the uncertainties associated with the various models. Demonstrating this functionality is beyond the scope of the present paper, but examples of this can be found in Kwakkel et al. (2013), Pruyt and Kwakkel (2014), and Auping et al. (2015).

5.2.3. Scenario discovery

A second analysis technique is to perform scenario discovery (Bryant and Lempert, 2010) using either the Patient Rule Induction Method (PRIM) (Friedman and Fisher, 1999) or Classification and Regression Trees (CART) (Breiman et al., 1984). Here we show the application of PRIM. The implementation of PRIM, which comes with the workbench, is designed for interactive use through jupyter notebook (Pérez and Granger, 2007). In practice, the code would thus be executed in several steps and the user would make choices based on the results shown. The first step in applying scenario discovery is to specify which experiments are of interest. In a typical application, one would focus on the subset of experiments where one or more objectives have not been achieved. For the lake problem, this would mean the cases where the anthropogenic pollution is high. However, as can be seen in Fig. 2 there are many experiments where this is the case. PRIM is not particularly useful if there are many experiments of interest. Therefore instead we have chosen to focus on the

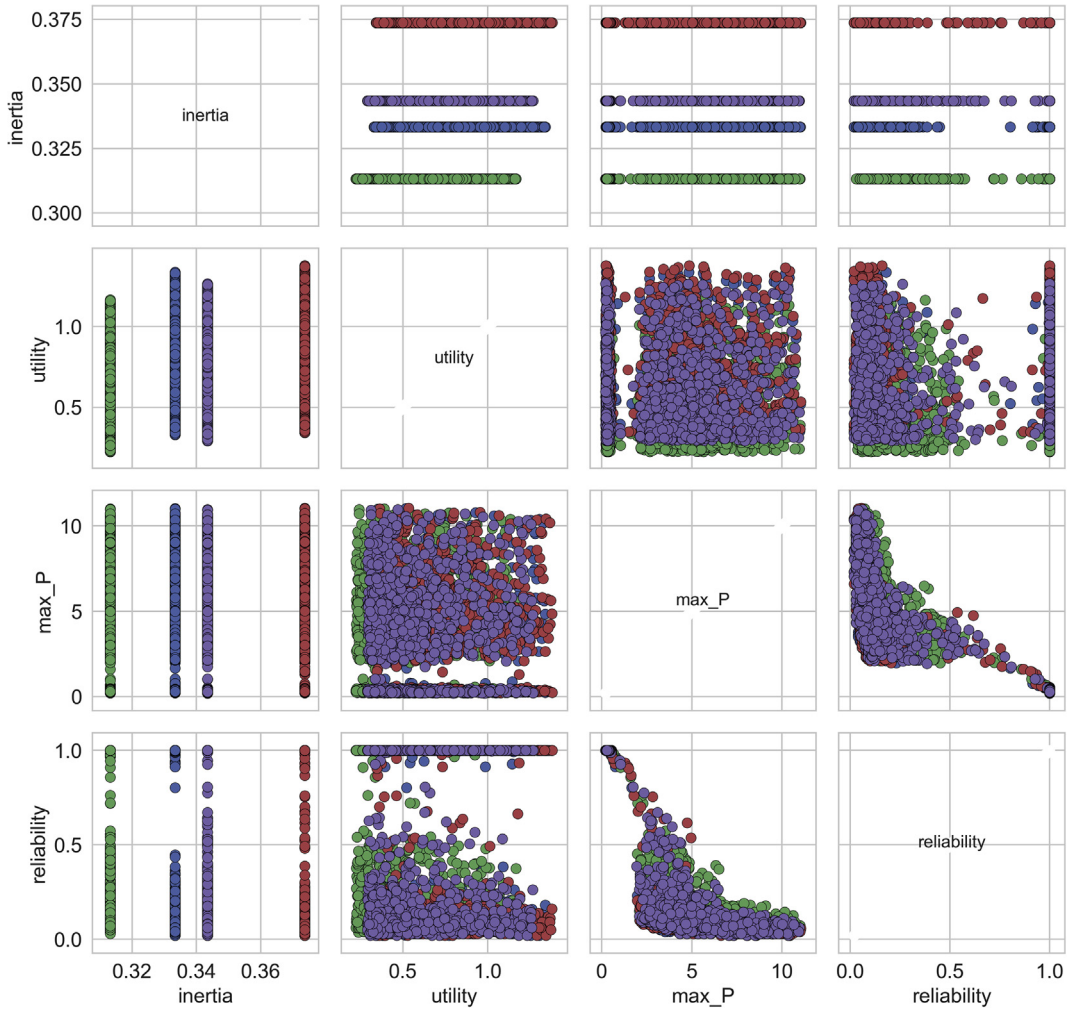

Fig. 2. A pairwise scatter plot of the performance of the four policies for each experiment.

experiments where the pollution remains below 1. Any experiment where the concentration in the lake is below 1 is of interest.

```
from ema_workbench.analysis import prim
import mpld3 # enables interaction with trade off curve

# a simple classification of the outcomes
y = outcomes['max_P'] < 1

prim_alg = prim.Prim(x,y, threshold=0.8)

box1 = prim_alg.find_box()

box1.show_tradeoff() # can be displayed with mpld3 for interactivity
box1.inspect(style='graph')
plt.show()
```

Fig. 4 shows the results for PRIM. On the left, we see the trade-off between coverage and density, and color coded information on the number of restricted dimensions. On the right, we see the identified box, including information on the quasi p-values of each of the limits, and the coverage and density of the box. For more details on the meaning of these numbers and how they can be used when performing Scenario Discovery, see Bryant and Lempert (2010) and Kwakkel and Jaxa-Rozen (2016). This particular box is the last one (top left) on the peeling trajectory shown on the left.

5.2.4. Global sensitivity analysis

A third analysis technique is to use global sensitivity analysis. The SALib library (Herman and Usher, 2017) in Python makes this

particularly easy. All we need to do is generate some reference policy, in this case without any release, and pass the sampling technique we want to use to perform experiments. This will generate a Sobol sample using SALib, and execute these.

```
from ema_workbench.em_framework import Policy
from ema_workbench.em_framework import SOBOL

policy = Policy('no release', **{str(i):0.1 for i in range(100)})

# perform experiments
n_scenarios = 1000

with MultiprocessingEvaluator(lake_model) as evaluator:
    results = evaluator.perform_experiments(n_scenarios, policy,
                                            sampling=SOBOL)
```

Any subsequent analysis requires using SALib directly. To make this easier, the workbench comes with a convenience function that will generate the SALib problem object given the uncertainties in the model. Here we calculate the Sobol indices, and extract the total order and first order indices including confidence intervals. The results are shown in Table 3.

```
from ema_workbench.em_framework import get_SaLib_problem
from SALib.analyze import sobol

experiments, outcomes = results

problem = get_SALib_problem(lake_model.uncertainties)
y = outcomes['max_P']
sobol_indices = sobol.analyze(problem, y)
sobol_stats = {key:sobol_indices[key] for key in ['ST', 'ST_conf', 'S1', 'S1_conf']}
sobol_stats = pd.DataFrame(sobol_stats, index=problem['names'])
sobol_stats.sort_values(by='ST', ascending=False)
```

5.3. Generating policy options using search

The workbench is designed to enable sampling and search over both the uncertainties and the levers. In contrast to sampling, where the workbench comes with a variety of standard sampling techniques, the workbench does not provide any (many-objective) optimization algorithm itself. The Exploratory Modeling Workbench has the library platypus as an optional dependency. If platypus is installed, the workbench can use it for many-objective (robust) optimization. Many-objective optimization can be used to search over either the levers or the uncertainties. The former is used in many-objective robust decision making (Kasprzyk et al., 2013; Watson and Kasprzyk, 2017), while the later enables worst-case discovery (Halim et al., 2016). Many-objective robust optimization searches over the levers, while the performance of a candidate strategy is a function of a set of scenarios (see e.g. Kwakkel et al., 2015, 2016b).

If we would like to find a set of candidate release strategies for a reference scenario as is done in many-objective robust decision making, we can use the following.

```
from ema_workbench import MultiprocessingEvaluator
from pandas.plotting import parallel_coordinates

with MultiprocessingEvaluator(lake_model) as evaluator:
    optimization_results = evaluator.optimize(searchover='levers',
                      nfe=100000, epsilons=[0.1,]*len(lake_model.outcomes))

parallel_coordinates(optimization_results, '0', cols=['max_P', 'utility',
                     'inertia','reliability'])
plt.show()
```

By default ε-NSGAII (Kollat and Reed, 2006) is used, but any of the other algorithms that are available in platypus can be used as well. ε-NSGAII requires the specification of epsilons. For a more in-depth discussion on the meaning of this and how to choose these see e.g. Kollat and Reed (2006). The optimize method takes

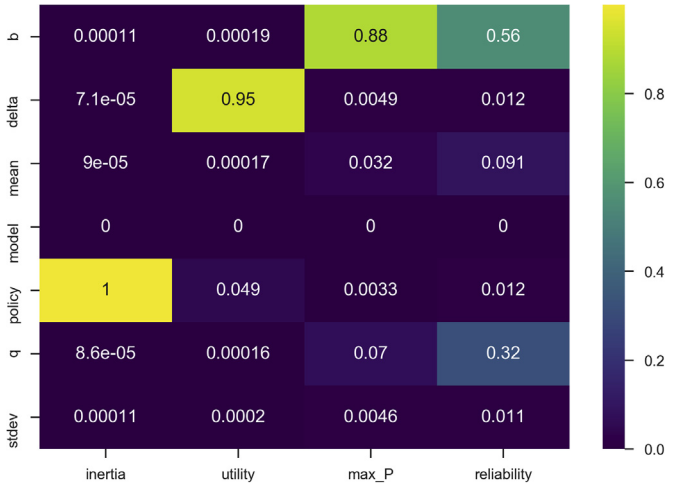

Fig. 3. Extra trees feature scoring for all outcomes of interest.

want to use for calculating our robustness. We also need to specify the scenarios or number of scenarios over which we want to define our robustness function(s). Below is a proof of principle. Using the lake model, we define four robustness metrics: the mean of the maximum pollution, the standard deviation of the maximum pollution, the signal to noise ratio of reliability, and the 10th percentile for utility. A detailed discussion of these ways of operationalizing robustness is beyond the scope of this paper. For a broad introduction of a wide variety of metrics see Giuliani and Castelletti (2016); Herman et al. (2015); Kwakkel et al. (2016a). Next, we generate a number of scenarios, in this case 10. In real world applications this is something to be established through trial and error, or by assessing how the robustness value changes as a function of the number of random scenarios (Hamarat et al., 2014). Instead of specifying the scenarios up front, we could also pass the number of scenarios to the robust optimization method, in which case new random scenarios are generated for each iteration of the optimization algorithm. With this setup we can now call the robust optimization method.

```
import functools
from ema_workbench.em_framework.samplers import sample_uncertainties

def signal_to_noise(data):
    mean = np.mean(data)
    std = np.std(data)
    sn = mean/std
    return sn

# a percentile based minimax robustness function
percentile10 = functools.partial(np.percentile, q=10)

MAXIMIZE = ScalarOutcome.MAXIMIZE
MINIMIZE = ScalarOutcome.MINIMIZE
robustnes_functions = [ScalarOutcome('mean p', kind=MINIMIZE,
                                     variable_name='max_P', function=np.mean),
                       ScalarOutcome('std p', kind=MINIMIZE,
                                     variable_name='max_P', function=np.std),
                       ScalarOutcome('sn reliability', kind=MAXIMIZE,
                                     variable_name='reliability',
                                     function=signal_to_noise),
                       ScalarOutcome('10th percentile utility', kind=MAXIMIZE,
                                     variable_name='reliability',
                                     function=percentile10),
                      ]
n_scenarios = 10 # for demo purposes only, should in practice be higher
scenarios = sample_uncertainties(lake_model, n_scenarios)
nfe = 1000

with MultiprocessingEvaluator(lake_model) as evaluator:
    robust_results = evaluator.robust_optimize(robustnes_functions,
                               scenarios, nfe=nfe, population_size=25,
                               epsilons=[0.1,]*len(robustnes_functions))
```

an options reference scenario argument, which can be used to specify explicitly the scenario under which one wants to optimize. This is useful for example in case of performing search for multiple different scenarios in MORDM, as suggested by Watson and Kasprzyk (2017), The evaluator returns a pandas dataframe with the levers and outcomes as columns. We can easily visualize this using the parallel coordinates plot functionality that comes with pandas, the resulting figure is shown in Fig. 5.

Performing many-objective robust optimization is a bit more involved. Most importantly, we need to specify the function(s) we

6. Concluding remarks

This paper has introduced the Exploratory Modeling Workbench. We explained the key ideas that underpin the design of the workbench, and demonstrated the key functionality of the workbench using the Lake Problem. This demonstration only shows the essential functionality using a relatively simple case. A reader interested in learning more about the workbench and what it can do is kindly referred to the examples that come with the

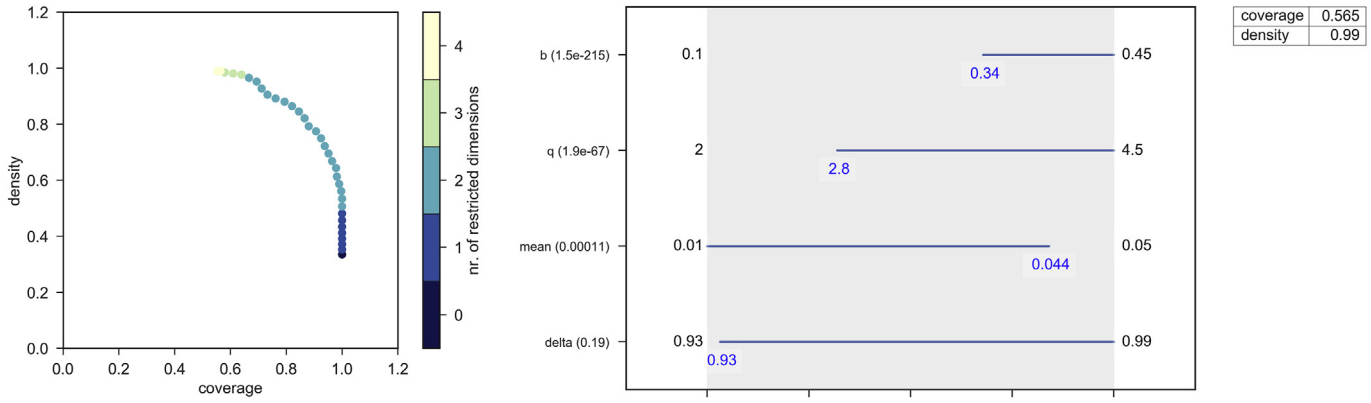

Fig. 4. Results of PRIM analysis, with the tradeoff between coverage and density (left), and the box limits including quasi-p values for each restriction and coverage and density information (right).

Table 3
Sobol indices for the lake problem.

	S1	S1_conf	ST	ST_conf
b	0.963029423	0.199196639	1.004861419	0.071067392
q	−0.000116163	0.000151987	1.65E-06	1.89E-06
mean	0.020551529	0.024681916	0.019212303	0.005819627
stdev	0.021225864	0.040653511	0.046739934	0.007108621
delta	0.000428036	0.000526972	2.12E-05	3.15E-05

workbench.

The Exploratory Modeling Workbench is one amongst several tools that can be used to support decision making under deep uncertainty. If we compare the Exploratory Modeling Workbench with the openMORDM toolkit, several differences stand out. Obviously, openMORDM uses R while the Exploratory Modeling Workbench uses Python. Both are fine programming languages for scientific computing and their respective strengths are quite complementary. OpenMORDM includes links to a variety of web based tools for visual analytics, something that is less developed in the Exploratory Modeling Workbench. The openMORDM toolkit was designed to support the many-objective robust decision making process. The design of the workbench, in contrast, is more generic since it starts from a taxonomy of model-based approaches for supporting decision making under deep uncertainty.

Compared to project Platypus, the main difference is that the workbench offers support for working with simulation models that need to access the file system. We have found this to be quite convenient, for it enables analysts to support decision making under deep uncertainty with existing simulation models rather than having to first recode the entire model. As we have shown, one can easily combine the workbench with the optimization part of Platypus. Presently, there is a fertile exchange of ideas between platypus and the workbench, with platypus building on ideas from the workbench and extending them further, and vice versa.

Going forward, there is an ongoing discussion of working towards an integration of platypus and the workbench. The extent to which this will be realized is dependent on time, as well as considerations regarding the exact scope of both projects. In particular, the explicit inclusion of support for models that require interaction with the file system creates various technical challenges that might be outside the scope of project platypus. Similarly, the workbench considers the implementation of many objective optimization algorithms to be outside its scope. A possible direction to reconcile these differences is to develop the workbench into a collection of related libraries rather than a single integrated library.

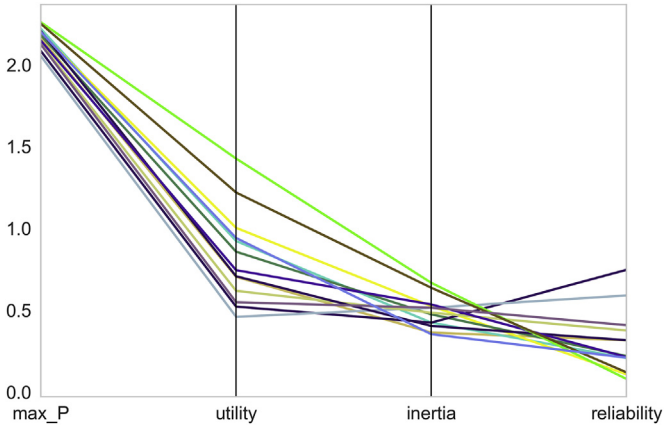

Fig. 5. Parallel coordinate plot of optimization results for a reference scenario.

Acknowledgements

The research reported in this paper has been funded by the Dutch National Science Foundation under grant number 451-13-018. The workbench has been the result of ongoing research over several years. MSc. Students, PhD students, and colleagues at the faculty of Technology, Policy and Management of Delft University of Technology have been invaluable in critically discussing the code, and testing it in various projects. The workbench has also benefited from fruitful discussions with Dave Hadka, who is the lead developer of project Platypus. An earlier version of this paper has been presented at iEMSs 2016.

References

Auping, W., Pruyt, E., Kwakkel, J.H., 2015. Societal ageing in The Netherlands: a robust system dynamics approach. Syst. Res. Behav. Sci. 32 (4), 485–501.
Bankes, S.C., 1993. Exploratory modeling for policy analysis. Operations Res. 4 (3), 435–449.
Bankes, S.C., 2002. Tools and techniques for developing policies for complex and uncertain systems. Proc. Natl. Acad. Sci. U. S. A. 99 (3), 7263–7266.
Bankes, S.C., Walker, W.E., Kwakkel, J.H., 2013. Exploratory modeling and analysis. In: Gass, S., Fu, M.C. (Eds.), Encyclopedia of Operations Research and Management Science, third ed. Springer, Berlin, Germany.
Ben Haim, Y., 2001. Information-Gap Decision Theory: Decision under Severe Uncertainty. Academic Press, London, UK.
Breiman, L., 2001. Random forests. Mach. Learn. 45, 5–23.
Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J., 1984. Classification and Regression Trees. Wadsworth, Monterey, CA.
Brown, C., Ghile, Y., Laverty, M., Li, K., 2012. Decision scaling: linking bottom-up

vulnerability analysis with climate projections in the water sector. Water Resour. Res. 48 (9), 1–12.
Bryant, B.P., 2014. Sdtoolkit: Scenario Discovery Tools to Support Robust Decision Making (v2.33-1).
Bryant, B.P., Lempert, R.J., 2010. Thinking inside the Box: a participatory computer-assisted approach to scenario discovery. Technol. Forecast. Soc. Change 77 (1), 34–49.
Dalal, S., Han, B., Lempert, R., Jaycocks, A., Hackbarth, A., 2013. Improving scenario discovery using orthogonal rotations. Environ. Model. Softw. 48, 49–64.
de Neufville, R., Scholtes, S., 2011. Flexibility in Engineering Design. The MIT Press, Cambridge, Massachusetts.
Dessai, S., Hulme, M., Lempert, R.J., Pielke jr, R., 2009. Do we need better predictions to adapt to a changing climate? EOS 90 (13), 111–112.
Friedman, J.H., Fisher, N.I., 1999. Bump hunting in high-dimensional data. Statistics Comput. 9 (2), 123–143.
Geurts, P., Ernst, D., Wehenkel, L., 2006. Extremely randomized trees. Mach. Learn. 63 (1), 3–42.
Giuliani, M., Castelletti, A., 2016. Is robustness really robust? How different definitions of robustness impact decision-making under climate change. Clim. Change 135 (3–4), 409–424.
Groves, D.G., Lempert, R.J., 2007. A new analytic method for finding policy-relevant scenarios. Glob. Environ. Change 17 (1), 73–85.
Guivarch, C., Rozenberg, J., Schweizer, V., 2016. The diversity of socio-economic pathways and CO2 emissions scenarios: insights from the investigation of a scenarios database. Environ. Model. Softw. 80, 336–353.
Haasnoot, M., Kwakkel, J.H., Walker, W.E., ter Maat, J., 2013. Dynamic adaptive policy pathways: a method for crafting robust decisions for a deeply uncertain world. Glob. Environ. Change 23 (2), 485–498.
Hadka, D., Herman, J.D., Reed, P.M., Keller, K., 2015. OpenMORDM: an open source framework for many objective robust decision making. Environ. Model. Softw. 74, 114–129.
Hadka, D., Reed, P.M., 2013. Borg: an auto-adaptive many-objective evolutionary computing framework. Evol. Comput. 21 (2), 231–259.
Halim, R.A., Kwakkel, J.H., Tavasszy, L.A., 2016. A scenario discovery study of the impact of uncertainties in the global container transport system on European ports. Futures 81, 148–160.
Hallegatte, S., Bangalore, M., Bonzanigo, L., Fay, M., Tamaro, K., Narloch, U., Rozenberg, J., Treguer, D., Vogt-Schilb, A., 2015. Shock Waves - Managing the Impact of Climate Change on Povert. The World Bank, Washington, DC.
Hallegatte, S., Shah, A., Lempert, R., Brown, C., Gill, S., 2012. Investment decision making under deep uncertainty: application to climate change. The World Bank.
Hamarat, C., Kwakkel, J.H., Pruyt, E., 2013. Adaptive robust design under deep uncertainty. Technol. Forecast. Soc. Change 80 (3), 408–418.
Hamarat, C., Kwakkel, J.H., Pruyt, E., Loonen, E., 2014. An exploratory approach for adaptive policymaking by using multi-objective robust optimization. Simul. Model. Pract. Theory 46, 25–39.
Herman, J.D., Reed, P.M., Zeff, H.B., Characklis, G.W., 2015. How should robustness be defined for water systems planning under change. J. Water Resour. Plan. Manag. 141 (10).
Herman, J.D., Usher, W., 2017. SALib: an open-source Python library for sensitivity analysis. J. Open Source Softw. 2 (9).
Hunter, J.D., 2007. Matplotlib: a 2D graphics environment. Comput. Sci. Eng. 9 (3), 90–95.
Kasprzyk, J.R., Nataraj, S., Reed, P.M., Lempert, R.J., 2013. Many objective robust decision making for complex environmental systems undergoing change. Environ. Model. Softw. 42, 55–71.
Kollat, J.B., Reed, P.M., 2006. Comparing state-of-the-art evolutionary multi-objective algorithms for long-term groundwater monitoring design. Adv. Water Resour. 29 (6), 792–807.
Kwakkel, J.H., Auping, W.L., Pruyt, E., 2013. Dynamic scenario discovery under deep uncertainty: the future of copper. Technol. Forecast. Soc. Change 80 (4), 789–800.
Kwakkel, J.H., Cunningham, S.C., 2016. Improving scenario discovery by bagging random boxes. Technol. Forecast. Soc. Change 111, 124–134.
Kwakkel, J.H., Eker, S., Pruyt, E., 2016a. How robust is a robust Policy? Comparing alternative robustness metrics for robust decision-making. In: Doumpos, M., Zopounidis, C., Grigoroudis, E. (Eds.), Robustness Analysis in Decision Aiding, Optimization, and Analytics. Springer.
Kwakkel, J.H., Haasnoot, M., Walker, W.E., 2015. Developing Dynamic Adaptive Policy Pathways: a computer-assisted approach for developing adaptive strategies for a deeply uncertain world. Clim. Change 132 (3), 373–386.
Kwakkel, J.H., Haasnoot, M., Walker, W.E., 2016b. Comparing robust decision-making and dynamic adaptive policy pathways for model-based decision support under deep uncertainty. Environ. Model. Softw. 86, 168–183.
Kwakkel, J.H., Jaxa-Rozen, M., 2016. Improving scenario discovery for handling heterogeneous uncertainties and multinomial classified outcomes. Environ. Model. Softw. 79, 311–321.
Kwakkel, J.H., Walker, W.E., Marchau, V.A.W.J., 2010. Adaptive airport strategic planning. Eur. J. Transp. Infrastructure Res. 10 (3), 227–250.
Lempert, R.J., Collins, M., 2007. Managing the risk of uncertain threshold response: comparison of robust, optimum, and precautionary approaches. Risk Anal. 24 (4), 1009–1026.
Lempert, R.J., Groves, D.G., Popper, S.W., Bankes, S.C., 2006. A general, analytic method for generating robust strategies and narrative scenarios. Manag. Sci. 52 (4), 514–528.
Lempert, R.J., Popper, S., Bankes, S., 2003. Shaping the Next One Hundred Years: New Methods for Quantitative, Long Term Policy Analysis. RAND, Santa Monica, CA.
LeRoy Poff, N., Brown, C., Grantham, T.E., Matthews, J.H., Palmer, M.A., Spence, C.M., Wilby, R.L., Haasnoot, M., Mendoza, G.F., Dominique, K.C., Baeza, A., 2015. Sustainable water management under future uncertainty with eco-engineering decision scaling. Nat. Clim. Change 6 (1), 25–34.
Maier, H.R., Guillaume, J.H.A., van Delden, H., Riddell, G.A., Haasnoot, M., Kwakkel, J.H., 2016. An uncertain future, deep uncertainty, scenarios, robustness and adaptation: how do they fit together? Environ. Model. Softw. 81, 154–164.
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830.
Pérez, F., Granger, B.E., 2007. IPython: a system for interactive scientific computing. Comput. Sci. Eng. 9 (3), 21–29.
Pruyt, E., Kwakkel, J.H., 2014. Radicalization under deep uncertainty: a multi-model exploration of activism, extremism and terrorism. Syst. Dyn. Rev. 30 (1–2), 1–28.
Quinn, J.D., Reed, P.M., Keller, K., 2017. Direct policy search for robust multi-objective management of deeply uncertain socio-ecological tipping points. Environ. Model. Softw. 92, 125–141.
Singh, R., Reed, P.M., Keller, K., 2015. Many-objective robust decision making for managing an ecosystem with a deeply uncertain threshold response. Ecol. Soc. 20 (3).
Spear, R.C., Hornberger, G.M., 1980. Eutrophication in peel inlet. II. Identification of critical uncertainties via generalized sensitivity analysis. Water Res. 14 (1), 43–49.
Tsoukiàs, A., 2008. From decision theory to decision aiding methodology. Eur. J. Operational Res. 187, 138–161.
Van Asselt, M.B.A., Rotmans, J., 2002. Uncertainty in integrated assessment modelling. From positivism to pluralism. Clim. Change 54 (1–2), 75–105.
Walker, W.E., 2000. Policy analysis: a systematic approach to supporting policy-making in the public sector. J. Multicriteria Decis. Analysis 9 (1–3), 11–27.
Walker, W.E., Lempert, R., Kwakkel, J.H., 2013. Deep uncertainty. In: Gass, S., Fu, M. (Eds.), Encyclopedia of Operations Research and Management Science, third ed. Springer, Berlin, Germany, pp. 395–402.
Walker, W.E., Rahman, S.A., Cave, J., 2001. Adaptive policies, policy analysis, and policymaking. Eur. J. Operational Res. 128 (2), 282–289.
Ward, V.L., Singh, R., Reed, P.M., Keller, K., 2015. Confronting tipping points: can multi-objective evolutionary algorithms discover pollution control tradeoffs given environmental thresholds? Environ. Model. Softw. 73 (1), 27–43.
Watson, A.A., Kasprzyk, J.R., 2017. Incorporating deeply uncertain factors into the many objective search process. Environ. Model. Softw. 89, 159–171.
Weaver, C.P., Lempert, R.J., Brown, C., Hall, J.W., Revell, D., Sarewitz, D., 2013. Improving the contribution of climate model information to decision making: the value and demands of robust decision making frameworks. WIREs Clim. Change 4, 39–60.
Wilby, R.L., Dessai, S., 2010. Robust adaptation to climate change. Weather 65 (7), 180–185.
Young, P.C., Spear, R.C., Hornberger, G.M., 1978. Modeling Badly Defined Systems: Some Further Thoughts (Proceedings SIMSIG Conference: Canberra).