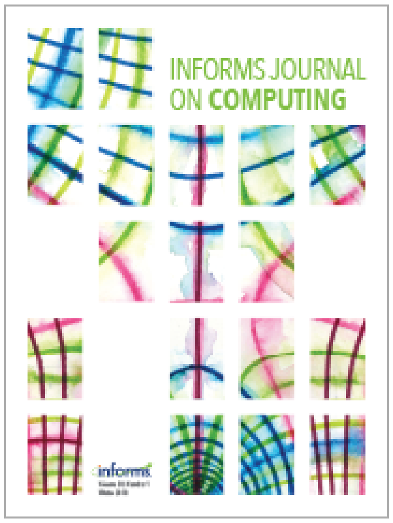


INFORMS Journal on Computing

ROC++: Robust Optimization in C++

Phebe Vayanos, Qing Jin, George Elissaios

To cite this article:
Phebe Vayanos, Qing Jin, George Elissaios (2022) ROC++: Robust Optimization in C++. INFORMS Journal on Computing 34(6):2873-2888. https://doi.org/10.1287/ijoc.2022.1209

https://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Vol. 34, No. 6, November–December 2022, pp. 2873–2888
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

ROC++: Robust Optimization in C++

Phebe Vayanos,[a,*] **Qing Jin,**[a] **George Elissaios**[a]

[a] CAIS Center for Artificial Intelligence in Society, University of Southern California, Los Angeles, California 90089
*Corresponding author

Contact: phebe.vayanos@usc.edu, https://orcid.org/0000-0001-7800-7235 (PV); qingjin@usc.edu, https://orcid.org/0000-0003-3013-4205 (QJ); elissaios@gmail.com (GE)

Received: June 10, 2020
Revised: May 7, 2021; December 19, 2021; March 17, 2022; May 3, 2022
Accepted: May 13, 2022
Published Online in Articles in Advance: September 6, 2022

https://doi.org/10.1287/ijoc.2022.1209

Copyright: © 2022 INFORMS

Abstract. Over the last two decades, robust optimization has emerged as a popular means to address decision-making problems affected by uncertainty. This includes single-stage and multi-stage problems involving real-valued and/or binary decisions and affected by exogenous (decision-independent) and/or endogenous (decision-dependent) uncertain parameters. Robust optimization techniques rely on duality theory potentially augmented with approximations to transform a (semi-)infinite optimization problem to a finite program, the *robust counterpart*. Whereas writing down the model for a robust optimization problem is usually a simple task, obtaining the robust counterpart requires expertise. To date, very few solutions are available that can facilitate the modeling and solution of such problems. This has been a major impediment to their being put to practical use. In this paper, we propose ROC++, an open-source C++ based platform for automatic robust optimization, applicable to a wide array of single-stage and multi-stage robust problems with both exogenous and endogenous uncertain parameters, that is easy to both use and extend. It also applies to certain classes of stochastic programs involving continuously distributed uncertain parameters and endogenous uncertainty. Our platform naturally extends existing off-the-shelf deterministic optimization platforms and offers ROPy, a Python interface in the form of a callable library, and the ROB file format for storing and sharing robust problems. We showcase the modeling power of ROC++ on several decision-making problems of practical interest. Our platform can help streamline the modeling and solution of stochastic and robust optimization problems for both researchers and practitioners. It comes with detailed documentation to facilitate its use and expansion. The latest version of ROC++ can be downloaded from https://sites.google.com/usc.edu/robust-opt-cpp/.

Summary of Contribution: The paper "ROC++: Robust Optimization in C++" proposes a new open-source C++ based platform for modeling, automatically reformulating, and solving robust optimization problems. ROC++ can address both single-stage and multi-stage problems involving exogenous and/or endogenous uncertain parameters and real- and/or binary-valued adaptive variables. The ROC++ modeling language is similar to the one provided for the deterministic case by state-of-the-art deterministic optimization solvers. ROC++ comes with detailed documentation to facilitate its use and expansion. It also offers ROPy, a Python interface in the form of a callable library. The latest version of ROC++ can be downloaded from https://sites.google.com/usc.edu/robust-opt-cpp/.

History: Accepted by Ted Ralphs, Area Editor for Software Tools.
Funding: This material is based upon work supported by the National Science Foundation under Grant No. 1763108. This support is gratefully acknowledged.
Supplemental Material: The software that supports the findings of this study is available within the paper and its Supplementary Information (https://pubsonline.informs.org/doi/suppl/10.1287/ijoc.2022.1209) or is available from the IJOC GitHub software repository (https://github.com/INFORMSJoC) at (https://dx.doi.org/10.5281/zenodo.6360996).

Keywords: robust optimization • sequential decision-making • exogenous uncertainty • endogenous uncertainty • decision-dependent uncertainty • decision-dependent information discovery • C++ • Python

Notation

We denote vectors (matrices) by boldface lowercase (uppercase) letters. The kth element of a vector $\boldsymbol{x} \in \mathbb{R}^n$ ($k \le n$) is denoted by x_k. Scalars are denoted by letters—for example, α or N. We let $\mathcal{L}_k^n$ ($\mathcal{B}_k^n$) represent the space of all functions from $\mathbb{R}^k$ to $\mathbb{R}^n$ ($\{0,1\}^n$). Given two vectors of equal length, $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, we let $\boldsymbol{x} \circ \boldsymbol{y}$ denote their Hadamard product.

1. Introduction

1.1. Motivation

Robust optimization (RO) is a discipline that develops models and algorithms for solving decision problems affected by uncertainty; see, for example, Ben-Tal et al. (2009) and Bertsimas et al. (2010). It studies problems with worst-case objective and robust constraints that must hold for all possible realizations of the uncertain problem parameters.

The simplest problems studied by RO are *single-stage* problems, where all decisions are made *before* the uncertain parameters are revealed, and involve only *exogenous* uncertainty. These arise, for example, in inventory management (Ardestani-Jaafari and Delage 2016), healthcare (Bandi et al. 2018), and biodiversity conservation (Haider et al. 2018).

Single-stage robust problems with exogenous uncertainty are representable as

$$\text{minimize}\left\{\max_{\boldsymbol{\xi}\in\Xi}\ \boldsymbol{c}(\boldsymbol{\xi})^\top\boldsymbol{y}+\boldsymbol{d}(\boldsymbol{\xi})^\top\boldsymbol{z}\ :\ \boldsymbol{y}\in\mathcal{Y},\ \boldsymbol{z}\in\mathcal{Z},\right.$$
$$\left.\boldsymbol{A}(\boldsymbol{\xi})\boldsymbol{y}+\boldsymbol{B}(\boldsymbol{\xi})\boldsymbol{z}\ \le\ \boldsymbol{h}(\boldsymbol{\xi})\quad\forall\boldsymbol{\xi}\in\Xi\right\},\qquad(1)$$

where $\boldsymbol{y}\in\mathcal{Y}\subseteq\mathbb{R}^n$ and $\boldsymbol{z}\in\mathcal{Z}\subseteq\{0,1\}^\ell$ stand for the vectors of real- and binary-valued (static) decisions, respectively, that must be made before the uncertain parameters $\boldsymbol{\xi}\in\mathbb{R}^k$ are observed. The set $\Xi\subseteq\mathbb{R}^k$ denotes the *uncertainty set*, which represents the set of all realizations of $\boldsymbol{\xi}$ against which the decision-maker wishes to be immunized. Here, $\boldsymbol{c}(\boldsymbol{\xi})\in\mathbb{R}^n$ and $\boldsymbol{d}(\boldsymbol{\xi})\in\mathbb{R}^\ell$ can be interpreted as cost vectors, whereas $\boldsymbol{h}(\boldsymbol{\xi})\in\mathbb{R}^m$, and $\boldsymbol{A}(\boldsymbol{\xi})\in\mathbb{R}^{m\times n}$ and $\boldsymbol{B}(\boldsymbol{\xi})\in\mathbb{R}^{m\times\ell}$ represent the right-hand-side vector and constraint coefficient matrices, respectively. It is usually assumed, without much loss of generality, that $\boldsymbol{c}(\boldsymbol{\xi})$, $\boldsymbol{d}(\boldsymbol{\xi})$, $\boldsymbol{A}(\boldsymbol{\xi})$, $\boldsymbol{B}(\boldsymbol{\xi})$, and $\boldsymbol{h}(\boldsymbol{\xi})$ are all linear in $\boldsymbol{\xi}$. The goal of the decision-maker is to select, among all decisions that are robustly feasible, one that achieves the smallest value of the cost in the worst-case. The uncertainty set usually admits a conic representation, being expressible as

$$\Xi := \{\boldsymbol{\xi}\in\mathbb{R}^k : \exists\boldsymbol{\zeta}^s\in\mathbb{R}^{k_s}, s=1,\dots,S :$$
$$\boldsymbol{P}^s\boldsymbol{\xi}+\boldsymbol{Q}^s\boldsymbol{\zeta}^s+\boldsymbol{q}^s\in\mathcal{K}^s, s=1,\dots,S\},\qquad(2)$$

for some matrices $\boldsymbol{P}^s\in\mathbb{R}^{r_s\times k}$ and $\boldsymbol{Q}^s\in\mathbb{R}^{r_s\times k_s}$, and vector $\boldsymbol{q}^s\in\mathbb{R}^{r_s}$, where $\mathcal{K}^s$ are closed convex pointed cones in $\mathbb{R}^{r_s}$, $s=1\dots,S$. This model includes as special cases budget uncertainty sets (Ben-Tal et al. 2009), uncertainty sets based on the central limit theorem (Bandi and Bertsimas 2012), and ellipsoidal uncertainty sets (Ben-Tal et al. 2009).

Under some mild assumptions, the semi-infinite problem (1) is equivalent to a finite program, the *robust counterpart* (RC), that can be solved with off-the-shelf solvers. This robust counterpart can be obtained by reformulating each semi-infinite constraint in (1) equivalently as a finite set of constraints using duality theory. Although the data in the RC are the same as those in Problem (1), the RC will typically not resemble at all the original problem, and converting one to the other is a tedious task.

A slight generalization to Problem (1) where Ξ is allowed to depend on decision variables can model single-stage problems with *endogenous* uncertainty; see Nohadani and Sharma (2018) and Lappas and Gounaris (2018). These arise for example, in radiation therapy (Nohadani and Roy 2017) and in certain classes of clinical trial planning problems (Lappas and Gounaris 2018). Their RC is a finite optimization problem involving products of binary and real-valued variables. If Ξ depends only on binary variables, these products can be linearized to yield a mixed-binary conic program that can be solved with off-the-shelf solvers. In that sense, obtaining the RC of such problems is more involved than obtaining the RC of (1).

The RO community has also extensively studied *multi-stage* problems with *exogenous* uncertainty, where uncertain parameters are revealed sequentially over time and decisions are allowed to *adapt* to the history of observations (Ben-Tal et al. 2004). These arise for example, in vehicle routing (Gounaris et al. 2013), energy (Rocha and Kuhn 2012, Jiang et al. 2014), and inventory management (Ben-Tal et al. 2005, Mamani et al. 2017).

A multi-stage robust optimization problem with exogenous uncertainty over the finite planning horizon $t\in\mathcal{T}:=\{1,\dots,T\}$ is representable as

$$\begin{array}{ll}
\min & \displaystyle\max_{\boldsymbol{\xi}\in\Xi}\left[\sum_{t\in\mathcal{T}}\boldsymbol{c}_t^\top\boldsymbol{y}_t(\boldsymbol{\xi})+\boldsymbol{d}_t(\boldsymbol{\xi})^\top\boldsymbol{z}_t(\boldsymbol{\xi})\right]\\
\text{s.t.} & \boldsymbol{y}_t\in\mathcal{L}_k^{n_t},\ \boldsymbol{z}_t\in\mathcal{B}_k^{\ell_t}\quad\forall t\in\mathcal{T}\\
& \displaystyle\sum_{\tau=1}^{t}\boldsymbol{A}_{t\tau}\boldsymbol{y}_\tau(\boldsymbol{\xi})+\boldsymbol{B}_{t\tau}(\boldsymbol{\xi})\boldsymbol{z}_\tau(\boldsymbol{\xi})\ \le\ \boldsymbol{h}_t(\boldsymbol{\xi})\\
& \qquad\qquad\qquad\qquad\forall\boldsymbol{\xi}\in\Xi, t\in\mathcal{T}\\
& \boldsymbol{y}_t(\boldsymbol{\xi})=\boldsymbol{y}_t(\boldsymbol{\xi}'),\ \boldsymbol{z}_t(\boldsymbol{\xi})=\boldsymbol{z}_t(\boldsymbol{\xi}')\\
& \quad\forall t\in\mathcal{T},\ \forall\boldsymbol{\xi},\boldsymbol{\xi}'\in\Xi:\boldsymbol{w}_{t-1}\circ\boldsymbol{\xi}=\boldsymbol{w}_{t-1}\circ\boldsymbol{\xi}',
\end{array}\qquad(3)$$

where $\boldsymbol{y}_t(\boldsymbol{\xi})\in\mathbb{R}^{n_t}$ and $\boldsymbol{z}_t(\boldsymbol{\xi})\in\{0,1\}^{\ell_t}$ represent the vectors of real- and binary-valued decisions for time t, respectively. The adaptive nature of the decisions is modelled mathematically by allowing them to depend on the observed realization of $\boldsymbol{\xi}\in\mathbb{R}^k$. The vectors $\boldsymbol{c}_t\in\mathbb{R}^{n_t}$ and $\boldsymbol{d}_t(\boldsymbol{\xi})\in\mathbb{R}^{\ell_t}$ can be interpreted as cost vectors, $\boldsymbol{h}_t(\boldsymbol{\xi})\in\mathbb{R}^{m_t}$ are the right-hand-side vectors, and $\boldsymbol{A}_{t\tau}\in\mathbb{R}^{m_t\times n_t}$ and $\boldsymbol{B}_{t\tau}(\boldsymbol{\xi})\in\mathbb{R}^{m_t\times\ell_t}$ are the constraint coefficient matrices. Without much loss, we assume that $\boldsymbol{d}_t(\boldsymbol{\xi})$, $\boldsymbol{h}_t(\boldsymbol{\xi})$, and $\boldsymbol{B}_{t\tau}(\boldsymbol{\xi})$ are all linear in $\boldsymbol{\xi}$. The binary vector $\boldsymbol{w}_t\in\{0,1\}^k$ represents the *information base* for time $t+1$—that is, it encodes the information revealed up to time t. Specifically, $w_{t,i}=1$ if and only if $\boldsymbol{\xi}_i$ is observed at some time $\tau\in\{0,\dots,t\}$, and $\boldsymbol{w}_0=\boldsymbol{0}$. As information is never forgotten, $\boldsymbol{w}_t\ge\boldsymbol{w}_{t-1}$ for all $t\in\mathcal{T}$. The last set of constraints in (3) enforces nonanticipativity, stipulating that $\boldsymbol{y}_t$ and $\boldsymbol{z}_t$ must be constant in parameters that have not been observed by time t.

Problems of the form (3) are generally intractable, and much of the research in the RO community has focused on devising conservative approximations. Most authors have studied problems involving only real-valued adaptive decisions and devised *decision rule approximations* that restrict the adjustable decisions to those presenting, for example, constant, linear (Ben-Tal et al. 2004), piecewise linear (Vayanos et al. 2011, Georghiou et al. 2015), or polynomial (Bampou and Kuhn 2011, Bertsimas et al. 2011, Vayanos et al. 2012) dependence on $\boldsymbol{\xi}$. We use the shorthands CDR and LDR for constant and linear decision rules, respectively. Under such approximations, Problem (3) reduces to a single-stage problem, and approaches from single-stage RO can be used to solve it. More recently, several authors have investigated problems involving binary adaptive decision variables. Some papers have proposed piecewise constant decision rule approximations over either a static (Vayanos et al. 2011) or adaptive (Bertsimas and Georghiou 2015, Bertsimas and Dunning 2016, Bertsimas and Georghiou 2018) partitions of the uncertainty set. Others have studied the more flexible *finite adaptability* approximation that consists of selecting a moderate number of candidate strategies today and implementing the best of those strategies in an adaptive fashion once the uncertain parameters are revealed (Bertsimas and Caramanis 2010, Hanasusanto et al. 2015, Vayanos et al. 2020). Although writing down the model for a multi-stage problem is usually a simple task (akin to formulating a deterministic problem), obtaining the RC of a conservative approximation to (3) is typically tedious and requires expertise in robust optimization.

Recently, there has been increased interest in *multi-stage* robust optimization problems involving *endogenous* uncertainty (Jonsbråten 1998). This includes problems with decision-dependent uncertainty sets, where the decision-maker can control the set of possible realizations of $\boldsymbol{\xi}$, and problems involving decision-dependent non-anticipativity constraints, where the decision-maker can control the time of information discovery. The latter are particularly relevant in practice, where oftentimes uncertain parameters only become observable after a costly investment. It has applications in R&D project selection (Solak et al. 2010), clinical trial planning (Colvin and Maravelias 2008), offshore oilfield exploration (Goel and Grossman 2004), and preference elicitation (Vayanos et al. 2020, 2021), among others.

Multi-stage problems with endogenous uncertainty set are simple variants of (3), where Ξ is allowed to depend on *adaptive* decision variables. These have been studied by Bertsimas and Vayanos (2014), who proposed piecewise constant and piecewise linear decision rule approximations over both preselected and adaptive partitions of the uncertainty set and showed that the resulting problem can be reformulated as a mixed-integer conic program.

Multi-stage robust optimization problems with endogenous information discovery constitute a variant of Problem (3), where the information base for each time $t \in \mathcal{T}$ is kept flexible and under the control of the decision-maker. Thus, the information base is modeled as an adaptive decision variable that is itself allowed to depend on $\boldsymbol{\xi}$, and we denote it by $\boldsymbol{w}_t(\boldsymbol{\xi}) \in \mathcal{W}_t \subseteq \{0,1\}^k$. Multi-stage robust optimization problems with endogenous information discovery (ID) are expressible as

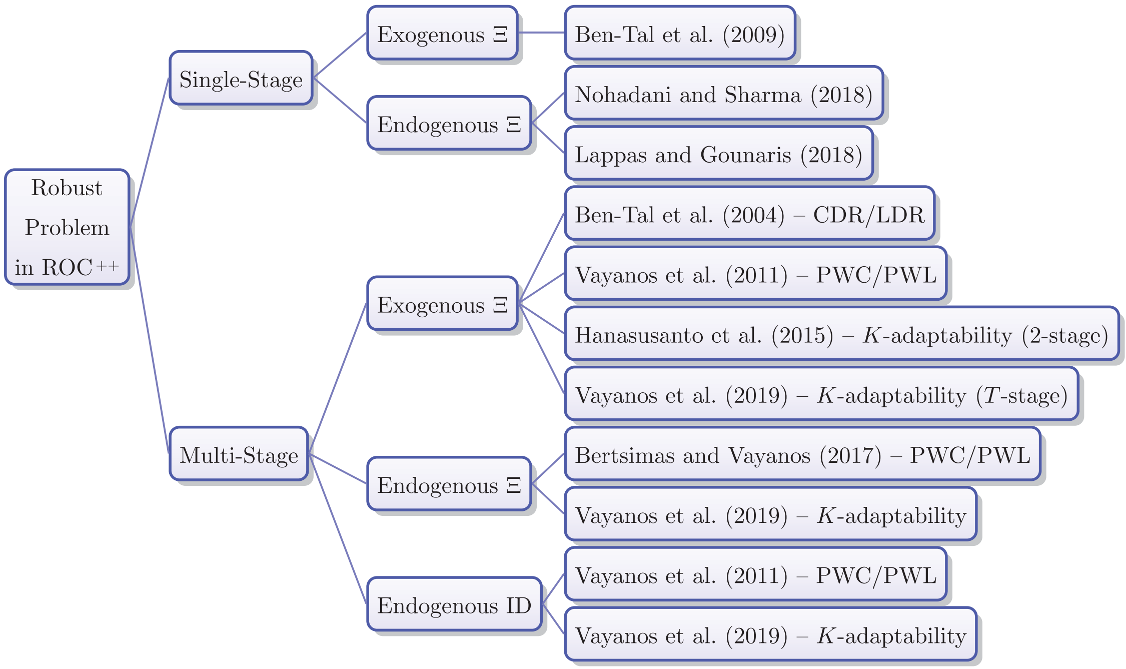
$$\begin{aligned}
\min \quad & \max_{\boldsymbol{\xi}\in\Xi} \left[\sum_{t\in\mathcal{T}} \boldsymbol{c}_t^\top \boldsymbol{y}_t(\boldsymbol{\xi}) + \boldsymbol{d}_t(\boldsymbol{\xi})^\top \boldsymbol{z}_t(\boldsymbol{\xi}) + \boldsymbol{f}_t(\boldsymbol{\xi})^\top \boldsymbol{w}_t(\boldsymbol{\xi}) \right] \\
\text{s.t.} \quad & \boldsymbol{y}_t \in \mathcal{L}_k^{n_t},\ \boldsymbol{z}_t \in \mathcal{B}_k^{\ell_t},\ \boldsymbol{w}_t \in \mathcal{B}_k^{k} \quad \forall t \in \mathcal{T} \\
& \left. \begin{array}{l} \displaystyle\sum_{\tau=1}^{t} \boldsymbol{A}_{t\tau}\boldsymbol{y}_\tau(\boldsymbol{\xi}) + \boldsymbol{B}_{t\tau}(\boldsymbol{\xi})\boldsymbol{z}_\tau(\boldsymbol{\xi}) \\ \quad + \boldsymbol{C}_{t\tau}(\boldsymbol{\xi})\boldsymbol{w}_\tau(\boldsymbol{\xi}) \le \boldsymbol{h}_t(\boldsymbol{\xi}) \\ \boldsymbol{w}_t(\boldsymbol{\xi}) \in \mathcal{W}_t \\ \boldsymbol{w}_t(\boldsymbol{\xi}) \ge \boldsymbol{w}_{t-1}(\boldsymbol{\xi}) \end{array} \right\} \quad \forall \boldsymbol{\xi} \in \Xi, t \in \mathcal{T} \\
& \left. \begin{array}{l} \boldsymbol{y}_t(\boldsymbol{\xi}) = \boldsymbol{y}_t(\boldsymbol{\xi}') \\ \boldsymbol{z}_t(\boldsymbol{\xi}) = \boldsymbol{z}_t(\boldsymbol{\xi}') \\ \boldsymbol{w}_t(\boldsymbol{\xi}) = \boldsymbol{w}_t(\boldsymbol{\xi}') \end{array} \right\} \begin{array}{l} \forall t \in \mathcal{T},\ \forall \boldsymbol{\xi}, \boldsymbol{\xi}' \in \Xi : \\ \quad \boldsymbol{w}_{t-1}(\boldsymbol{\xi}) \circ \boldsymbol{\xi} = \boldsymbol{w}_{t-1}(\boldsymbol{\xi}') \circ \boldsymbol{\xi}', \end{array}
\end{aligned} \tag{4}$$

where $f_{t,i}(\boldsymbol{\xi}) \in \mathbb{R}$ can be interpreted as the cost of including the uncertain parameter $\boldsymbol{\xi}_i$ in the information base at time t and $\boldsymbol{C}_{t\tau}(\boldsymbol{\xi})$ collects the coefficients of $\boldsymbol{w}_\tau$ in the time t constraint. The third constraint ensures that information observed in the past cannot be forgotten, whereas the last set of constraints are *decision-dependent non-anticipativity constraints* that model the requirement that decisions can only depend on information that the decision-maker chose to observe in the past. Without much loss, we assume that $\boldsymbol{d}_t(\boldsymbol{\xi}), \boldsymbol{f}_t(\boldsymbol{\xi}), \boldsymbol{B}_{t\tau}(\boldsymbol{\xi}) \in \mathbb{R}^{m_t \times \ell_\tau}$, and $\boldsymbol{C}_{t\tau}(\boldsymbol{\xi}) \in \mathbb{R}^{m_t \times k}$ are all linear in $\boldsymbol{\xi}$.

To tackle problems of the form (4), decision rule and finite adaptability approximations have been proposed. Vayanos et al. (2011) studied piecewise constant (PWC) and piecewise linear (PWL) decision rule approximations to the real- and binary-valued decisions, respectively. Accordingly, Vayanos et al. (2020) generalized the finite adaptability approximation to this setting. In both cases, the authors reformulated the problem as a mixed-integer linear problem that can be solved in practical times with off-the-shelf solvers. Although effective, these approaches are quite difficult to implement, as they rely on approximations, on the introduction of new decision variables, and on duality theory, making them inaccessible to practitioners and difficult for researchers to implement.

Robust optimization techniques have been extended to address certain multi-stage stochastic programs

Figure 1. (Color online) Classes of Robust Problems and Methods that Can Be Handled by ROC++


Robust Problem in ROC++
Single-Stage
Multi-Stage
Exogenous Ξ
Endogenous Ξ
Exogenous Ξ
Endogenous Ξ
Endogenous ID
Ben-Tal et al. (2009)
Nohadani and Sharma (2018)
Lappas and Gounaris (2018)
Ben-Tal et al. (2004) – CDR/LDR
Vayanos et al. (2011) – PWC/PWL
Hanasusanto et al. (2015) – K-adaptability (2-stage)
Vayanos et al. (2019) – K-adaptability (T-stage)
Bertsimas and Vayanos (2017) – PWC/PWL
Vayanos et al. (2019) – K-adaptability
Vayanos et al. (2011) – PWC/PWL
Vayanos et al. (2019) – K-adaptability

involving *continuously distributed* uncertain parameters and affected by exogenous (Kuhn et al. 2009, Bodur and Luedtke 2022) and even endogenous (Vayanos et al. 2011) uncertainty. In recent years, the field of *distributionally robust optimization* (DRO) has burgeoned, which immunizes decision-makers against ambiguity in the distribution of $\boldsymbol{\xi}$ (Wiesemann et al. 2014, Rahimian and Mehrotra 2019). Similarly to RO, deterministic reformulations of DRO problems can be obtained based on duality theory.

In spite of RO's success at addressing diverse problems and the difficulty of implementing these solutions, few platforms are available, and they provide only limited functionality.

1.2. Contributions

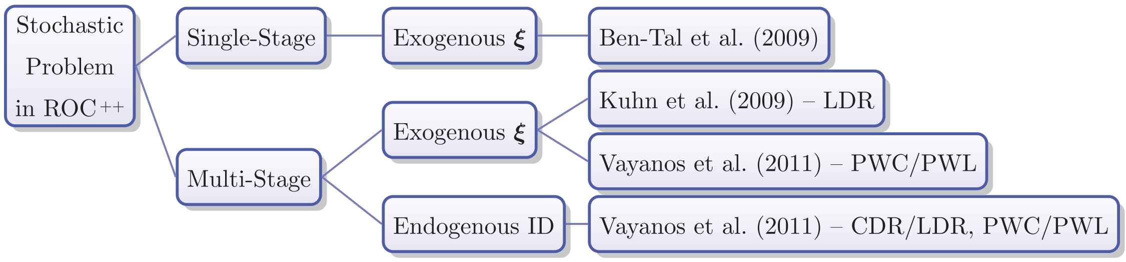
We now summarize our main contributions and the key advantages of our platform:

a. We propose ROC++, a C++ based platform for modeling, automatically reformulating, and solving robust optimization problems; see Vayanos et al. (2022). Our platform is the first capable of addressing both single-stage and multi-stage problems involving exogenous and/or endogenous uncertain parameters and real- and/or binary-valued adaptive variables. It can also be used to address certain classes of stochastic programs involving continuously distributed uncertainties. The suite of models and methods currently available in ROC++ is summarized in Figures 1 and 2 for the robust and stochastic settings, respectively. ROC++ can also interface with a variety of solvers. Presently, ROC++ offers an interface to Gurobi[1] and SCIP.[2]

b. ROC++ provides a Python library, ROPy, that features all the main functionality.

c. Thanks to operator overloading, ROC++ and ROPy are both very easy to use. We illustrate the

Figure 2. (Color online) Classes of Stochastic Problems and Methods that Can Be Handled by ROC++



Note. Support is currently limited to uniformly distributed uncertain parameters.

flexibility and ease of use of our platform on several stylized problems.

d. Through our design choices in ROC++, we aim to align with the SOLID principles of object-oriented programming to facilitate maintainability and extendability (Martin 2003). ROC++ leverages the power of C++ and in particular of polymorphism to code dynamic behavior, allowing the user to select their reformulation strategy at runtime and making it easy to extend the code with additional methods.

e. We propose the ROB file format, the first file format for storing and sharing general robust optimization problems, that is also interpretable and easy to use.

f. Our platform comes with detailed documentation (created with Doxygen[3]) to facilitate its use and expansion. Our framework is open-source. The latest version of the code, installation instructions, and dependencies of ROC++ are available at https://sites.google.com/usc.edu/robust-opt-cpp/. A snapshot of the software and data that were used in the research reported in this paper can be found at the software's DOI (Vayanos et al. 2022).

Through these capabilities, our platform lays the foundation to help facilitate research in, and real-life applications of, robust optimization.

1.3. Related Literature

Our robust optimization platform ROC++ most closely relates to several tools released in recent years for modeling and solving robust optimization problems. All of these tools present a similar structure: They provide a modeling platform combined with an approximation/reformulation toolkit that can automatically obtain the robust counterpart, which is then solved by using existing open-source and/or commercial solvers. The platform that most closely relates to ROC++ is called ROC[4] and is based on the paper of Bertsimas et al. (2019). It can be used to solve single-stage and multi-stage (distributionally) robust optimization problems with real-valued adaptive variables. It tackles this class of problems by approximating the adaptive decisions by linear decision rules or enhanced linear decision rules and solves the resulting problem using CPLEX.[5] Contrary to our platform, it cannot solve problems with endogenous uncertainty nor with binary adaptive variables. It appears to be harder to extend because the problems that it can model are a lot more limited (e.g., no decisions in the uncertainty set, no decision-dependent information discovery, and no binary adaptive variables) and because it does not provide a general framework for building new approximations/reformulations. Moreover, it does not provide a Python interface. The majority of the remaining platforms are based on the MATLAB modeling language. One tool is the robust optimization module of YALMIP (Löfberg 2012), which provides support for single-stage problems with exogenous uncertainty. A notable advantage of YALMIP is that the robust counterpart output by the platform can be solved by using any one of a variety of open-source or commercial solvers. Other platforms, like ROME[6] and RSOME[7], are entirely motivated by the (stochastic) robust optimization modeling paradigm (see Goh and Sim 2011 and Chen et al. 2020) and provide support for both single-stage and multi-stage (distributionally) robust optimization problems affected by exogenous uncertain parameters and involving only real-valued adaptive variables. The robust counterparts output by ROME can be solved with CPLEX, Mosek,[8] and SDPT3;[9] those output by RSOME, with CPLEX, Gurobi, and Mosek. We note that RSOME is not open-source. Recently, JuMPeR[10] has been proposed as an add-on to JuMP; see Dunning et al. (2017). It can be used to model and solve single-stage problems with exogenous uncertain parameters. JuMPeR can be connected to a large variety of open-source and commercial solvers. On the commercial front, AIMMS[11] is currently equipped with an add-on that can be used to model and automatically reformulate robust optimization problems. It can tackle both single-stage and multi-stage problems with exogenous uncertainty. To the best of our knowledge, none of the available platforms can address (neither model nor solve) problems involving endogenous uncertain parameters (decision-dependent uncertainty sets nor decision-dependent information discovery). None of them can tackle (neither model nor solve) problems presenting binary adaptive variables. A summary of the functionality offered by these tools is provided in Table 1.

ROC++ also relates, albeit more loosely, to platforms for modeling and solving stochastic programming problems (Birge and Louveaux 2000), such as PySP,[15] MSPP (Ding et al. 2019), SDDP[16] (Dowson and Kapelevich 2021), and SMI.[17] To the best of our knowledge, all such platforms assume that the uncertain parameters are discretely distributed or provide mechanisms for building a scenario tree approximation to the problem. This differs from our approach, as we work with the true distribution, but approximate the adaptive decisions.

In our work, we propose an entirely new platform for modeling and solving a wide array of RO problems that is also easy to extend with new models and new approximation and reformulation techniques as they are proposed in the literature. The main motivation for doing so is that new models and solution techniques are constantly being devised, whereas existing tools offer limited capability and were not built with extensibility in mind. Moreover, as there is a tight coupling between problem class and solution method in robust optimization (see Section 1.1), a strong abstraction is needed over the different problem types and reformulations/approximations to enable extensibility. ROC++ provides a framework built with the

Table 1. Summary of Tools for Modeling, Reformulating, and Solving Robust Optimization Problems

Software	Multi-stage support	Approximation schemes	Endogenous uncertainty	Binary adaptive variables	DRO	Language	Solvers
JuMPeR	No	—	No	No	No	Julia	Clp,[12] Cbc,[13] GLPK,[14] Gurobi, Mosek, CPLEX
YALMIP add-on	No	—	No	No	No	MATLAB	Almost any open-source or commercial solver
AIMMS add-on (commercial)	Yes	LDR	No	No	No	AIMMS	CPLEX recommended
ROME	Yes	LDR, bideflected LDR	No	No	Yes	MATLAB	CPLEX, Mosek, SDPT3
RSOME (not open-source)	Yes	LDR, event-wise static event-wise affine	No	No	Yes	MATLAB	CPLEX, Gurobi, Mosek
ROC	Yes	LDR, enhanced LDR	No	No	Yes	C++	CPLEX
ROC++	Yes	CDR/LDR, PWC/PWL, K-adaptability	Yes	Yes	No	C++, Python	Gurobi, SCIP

needs of extensibility and usability in mind from the onset and accounting for the coupling between problem class and solution method that is characteristic of RO. We chose to build the platform in C++ to: (a) leverage the benefits of object-oriented programming, including inheritance and polymorphism, to build strong class abstractions; (b) take advantage of its competitive speed at runtime to be able to more effectively tackle large scale problems; and, (c) to have the flexibility to compile libraries for other languages, as showcased by our ROPy interface. These factors combined were the main drivers behind this decision.

1.4. Organization of the Paper

Section 2 discusses the software design and the design rationale of ROC++. A sample model created and solved using ROC++ is provided in Section 3. Section 4 presents extensions to the core model that can also be tackled by ROC++, introduces the ROB file format, and briefly highlights the ROPy interface. Finally, Section 5 concludes. Additional sample models handled by ROC++ are provided in the online appendix.

2. ROC++ Software Design and Design Rationale

The software design and design rationale of ROC++ are motivated by the literature, which has shown that new problem classes, reformulation strategies, and approximation schemes are constantly being developed. This implies that our code should be easy to maintain and extend and that optimization models and approximation/reformulation schemes should not be restricted to a tight standard form. For these reasons, ROC++ aims to align with the SOLID principles of object-oriented programming, which establish practices for developing software with considerations for maintainability and extendability (see Martin 2003). Moreover, because of the tight coupling between problem class and solution method (see Section 1), the software needs a strong abstraction over the different problem types and reformulations/approximations. These considerations are the main motivation for the design choices behind ROC++. We now describe the software design while highlighting how some of our choices help serve these considerations.

2.1. Classes Involved in the Modeling of Optimization Problems

The main building blocks to model optimization problems in the ROC++ platform are the optimization model interface class, `ROCPPOptModelIF`; the constraint interface class, `ROCPPConstraintIF`; the decision variable interface class, `ROCPPVarIF`; the objective function interface class, `ROCPPObjectiveIF`; their derived classes; and the uncertain parameter class, `ROCPPUnc`. These classes mainly act as containers to which several reformulations, approximations, and solvers can be applied as appropriate; see Sections 2.2–2.4. Inheritance in the aforementioned classes implements the "is a" relationship. In particular, in our design choices for these classes, we subscribe to the Open-Close Principle (OCP) of SOLID, as we encapsulate abstract concepts in base classes so that additional functionality can be added by subclassing without changing the previously written code. We now give a more detailed description of some of these classes and how they relate to one another.

The `ROCPPVarIF` class is an abstract base class that provides a common interface to all decision variable types. Its class diagram is provided in Figure 3. Its children are the abstract classes, `ROCPPStaticVarIF` and `ROCPPAdaptVarIF`, that model static and adaptive variables, respectively. Each of these present three children, each of which model static (respectively (resp.), adaptive) real-valued, binary, or integer variables (see Figure 3). This structure of the `ROCPPVarIF` class ensures that we can pass objects of a subtype

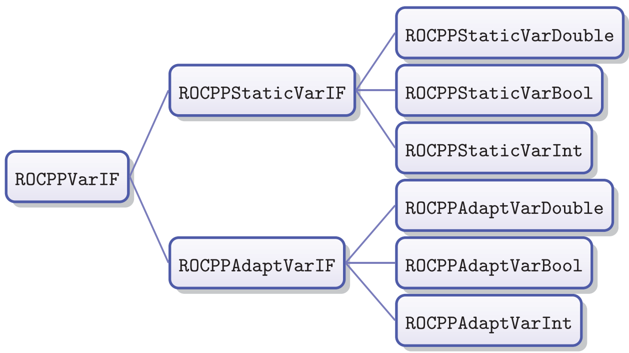


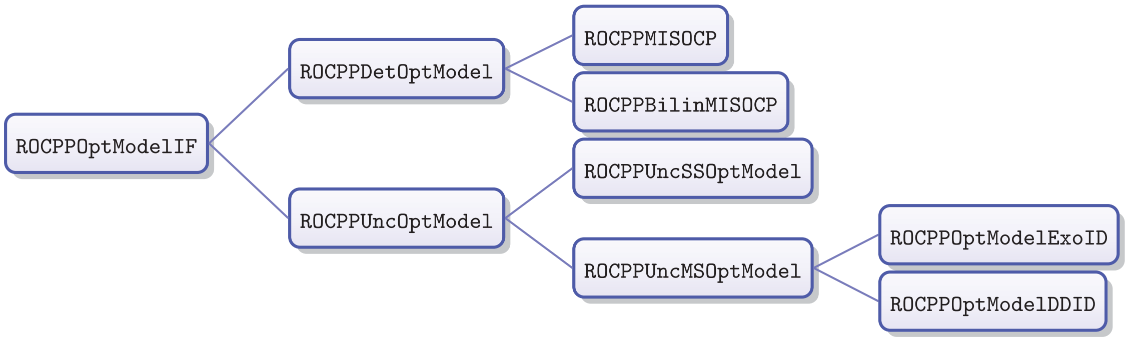
Figure 3. (Color online) Inheritance Diagram for the `ROCPPVarIF` Class

(e.g., `ROCPPStaticVarReal`) whenever an object of a supertype (e.g., `ROCPPVarIF`) is specified. This implies that a decision variable declaration will always have the type of the interface, `ROCPPVarIF`, and that the class realizing the details will be referenced only once, when it is instanciated. In particular, our modules will only depend on the abstraction. Thus, optimization problems, for example, store `ROCPPVarIF` types and are indifferent as to the precise type of the decision variable. This idea aligns with the Liskov Substitution Principle (LSP) and with the Dependency Inversion Principle (DIP) of SOLID.

The `ROCPPConstraintIF` class is an abstract base class with a single child: the interface class `ROCPPClassicConstraint`, whose two children, `ROCPPEqConstraint` and `ROCPPIneqConstraint`, model equality and inequality constraints, respectively. We are currently working to add `ROCPPSOSConstraint` and `ROCPPIfThenConstraint` as derived classes to `ROCPPConstraintIF`, to model Special Ordered Set and logical forcing constraints, respectively. Through the interface definition, all constraint types are forced to provide certain functionality. For example, they must implement the `mapVars` function, which maps the decision variables in the constraint to an expression; these are used in the reformulations/approximations. Constraints can either be main problem constraints or define the uncertainty set and may involve decision variables and/or uncertain parameters. The `ROCPPObjectiveFunctionIF` abstract base class presents two children, `ROCPPSimpleObjective` and `ROCPPMaxObjective`, that model linear and piecewise linear convex objective functions, respectively. The key building block for the `ROCPPConstraintIF` and `ROCPPObjectiveFunctionIF` classes is the `ROCPPExpr` class that models an expression, which is a sum of terms of abstract base type `ROCPPCstrTermIF`. The `ROCPPCstrTermIF` class has two children: `ROCPPProdTerm`, which is used to model monomials, and `ROCPPNorm`, which is used to model the two-norm of an expression. These class structures align with LSP and DIP. In particular, a constraint declaration will always have the type of the interface, `ROCPPConstraintIF`, and optimization problems will store `ROCPPConstraintIF` types while allowing any of the subtypes (e.g., `ROCPPIneqConstraint`), being indifferent as to the type of the constraint. Similarly, expressions will store arbitrary constraint terms without concern for their specific types.

The `ROCPPOptModelIF` is an abstract base class that provides a common and standardized interface to all optimization problem types. It consists of decision variables, constraints, an objective function, and potentially uncertain parameters. Its class diagram is shown in Figure 4. `ROCPPOptModelIF` presents two derived classes, `ROCPPDetOptModel` and `ROCPPUncOptModel`, which are used to model deterministic optimization models and optimization models involving uncertain parameters, respectively. Although `ROCPPDetOptModel` can involve arbitrary deterministic constraints, its derived classes, `ROCPPMISOCP` and `ROCPPBilinMISOCP`, can only model mixed-integer second-order cone problems (MISOCPs) and MISOCPs that also involve bilinear terms. The `ROCPPUncOptModel` class presents two derived classes, `ROCPPUncSSOptModel` and `ROCPPUncMSOptModel`, that are used to model single-stage and multi-stage problems respectively. Finally, `ROCPPUncMSOptModel` has two derived classes, `ROCPPOptModelExoID` and `ROCPPOptModelDDID`, that can model multi-stage optimization problems where the time

Figure 4. (Color online) Inheritance Diagram for the `ROCPPOptModelIF` Class



of information discovery is exogenous and endogenous, respectively. Thus, in accordance with OCP, we encapsulate abstract concepts, such as constraint containers, in base classes and add more functionality on the subclasses. As usual, the interface classes list a set of tools that all derived classes must provide. For example, all optimization problem types are forced to implement the `checkCompatibility` function that checks whether the constraint being added is compatible with this problem type. The main role of inheritance here is to ensure that the problems constructed are of types to which the available tools (reformulators, approximators, or solvers) can apply. Naturally, if the platform is augmented with more such tools that enable the solution of different/more general optimization problems, the existing inheritance structure can be leveraged to easily extend the code.

2.2. Dynamic Behavior via Strategy Pattern

2.2.1. Reformulation Strategies. The central objective of our platform is to convert (potentially through approximations) the original uncertain problem input by the user to a form that it can be fed into and solved by an off-the-shelf solver. This is achieved in our code through the use of reformulation strategies applied sequentially to the input problem. Currently, our platform provides a suite of such strategies, all of which are derived from the abstract base class `ReformulationStrategyIF`. The main approximation strategies are: the linear and constant decision rule approximations, provided by the classes `ROCPPLinearDR` and `ROCPPConstantDR`, respectively; the piecewise decision rule approximation, provided by the `ROCPPPWDR` class; and the *K*-adaptability approximation, provided by the `ROCPPKAdapt` class. The main equivalent reformulation strategies are: the `ROCPPRobustifyEngine`, which can convert a single-stage robust problem to its deterministic counterpart; and the `ROCPP-MItoMB` class, which can linearize bilinear terms involving products of binary and real-valued (or other) decisions. In accordance with LSP and DIP, the `ReformulationStrategyIF` module accepts optimization problems of any type derived from `ROCP-POptModelIF`.

2.2.2. Reformulation Orchestrator. In ROC++, the user can select at runtime which strategies to apply to their input problem and the sequence in which these strategies should be used. This is achieved by using the idea of a *strategy pattern*, which allows an object to change its behavior based on some unpredictable factor; see, for example, Perez (2018). To implement the strategy pattern, we provide, in addition to the reformulation strategies discussed above, the class `ROCPPOrchestrator` that will act as the *client*, being aware of the existence of strategies, but not needing to know what each strategy does. At runtime, an optimization problem, the *context*, is provided to the `ROCPPOrchestrator` together with a strategy or set of strategies to apply to the context, and the orchestrator applies the strategies in sequence, after checking that they can apply to the input problem.

2.2.3. Using and Extending the Code. Thanks to the idea of the strategy pattern, the code is very easy to use (the user simply needs to provide the input problem and the sequence of reformulation strategies). It is also very easy to extend; a researcher can create more reformulation strategies and leverage the existing client code to apply these strategies at runtime to the input problem. All that needs to be provided by the new reformulation strategy are implementations of the `Reformulate`, `isApplicable`, and `getName` functions, which do the reformulation, check that the reformulation can be applied to the problem input, and return the name of the approximation, respectively. Usability and extendability were the key factors that influenced this design choice.

2.3. Solver Interface

The ROC++ platform provides an abstract base class, `ROCPPSolverInterface`, which is used to convert deterministic MISOCPs in ROC++ format to a format that is recognized and solved by a commercial or open-source solver. Currently, there is support for two solvers: Gurobi, through the `ROCPPGurobi` class, and SCIP, through the `ROCPPSCIP` class. Gurobi and SCIP (with COIN-OR's Ipopt[18] solver) can solve all problem types output by our platform. Both of these classes are children of `ROCPPSolverInterface` and allow for changing the solver parameters, solving the problem, retrieving an optimal solution, etc. New solvers can conveniently be added by creating children classes to `ROCPPSolverInterface` and implementing its pure virtual member functions. This aligns with the OCP, LSP, and DIP principles, facilitating both use and expansion. Note that we also considered using interfaces such as the Osi Open Solver Interface[19] directly, but did not include it, as it does not currently provide support for conic optimization problems.

2.4. Tools to Facilitate Extension

The ROC++ platform comes with several classes that can be leveraged to construct new reformulation strategies, such as polynomial decision rules—see, for example, Bampou and Kuhn (2011) and Vayanos et al. (2012)—or constraint sampling approximations—see Campi and Garatti (2008). The key classes that can help construct new approximators and reformulators are the abstract base class `ROCPPVariableConverterIF` and its abstract derived classes `ROCPPOneToOneVarConverterIF` and `ROCPPOneToExprVarConverterIF`, which can map variables in the problem to other variables and variables to expressions, respectively. For example, one of the derived classes of `ROCPPOneToExprVarConverterIF` is `ROCPPPredefO2EVarConverter`, which takes a map from variable to expression as input and maps all variables in the problem to their corresponding expressions in the map. We have used it to implement the linear decision rule by passing a map from adaptive variables to affine functions of uncertain parameters. New decision rule approximations, such as polynomial decision rules, can be added in a similar way. This framework aligns with the OCP principle.

2.5. Interpretable Problem Input Through Operator Overload

ROC++ leverages operator overloading in C++ to enable the creation of problem expressions and constraints in a highly interpretable, human-readable format. `ROCPPExpr`, `ROCPPCstrTermIF`, `ROCPPVarIF`, and `ROCPPUnc` objects can be added or multiplied together to form new `ROCPPExpr` objects that can be used as left-hand sides of constraints. The double equality ("==") or inequality ("<=") signs can be used to create constraints. This framework effectively generalizes the modeling setup of modern solvers like Gurobi or CPLEX to the uncertain setting; see Section 3 for examples.

3. Modeling and Solving Decision-Making Problems in ROC++

In this section, we showcase the ease of use of our platform through a concrete example. Additional examples are provided in Online Appendix B. A snapshot of the software and data that were used to generate these results can be found at the software's DOI (Vayanos et al. 2022).

3.1. Robust Pandora's Box: Problem Description

We consider a robust variant of the celebrated stochastic Pandora's Box (PB) problem due to Weitzman (1979). This problem models selection from a set of unknown, alternative options, when evaluation is costly. There are I boxes indexed in $\mathcal{I} := \{1,\dots,I\}$ that we can choose or not to open over the planning horizon $\mathcal{T} := \{1,\dots,T\}$. Opening box $i \in \mathcal{I}$ incurs a cost $\boldsymbol{c}_i \in \mathbb{R}_+$. Each box has an unknown value $\boldsymbol{\xi}_i \in \mathbb{R}$, $i \in \mathcal{I}$, which will only be revealed if the box is opened. At the beginning of each time $t \in \mathcal{T}$, we can either select a box to open or keep one of the opened boxes, earn its value (discounted by θ^{t-1}), and stop the search.

We assume that the box values are restricted to lie in the set

$$\Xi := \{\boldsymbol{\xi} \in \mathbb{R}^I : \exists \boldsymbol{\zeta} \in [-1,1]^M,\ \boldsymbol{\xi}_i = (1+\Phi_i^\top \boldsymbol{\zeta}/2)\overline{\boldsymbol{\xi}}_i \quad \forall i \in \mathcal{I}\},$$

where $\boldsymbol{\zeta} \in \mathbb{R}^M$ represent M risk factors, $\Phi_i \in \mathbb{R}^M$ represent the factor loadings, and $\overline{\boldsymbol{\xi}} \in \mathbb{R}^I$ collects the nominal box values.

In this problem, the box values are endogenous uncertain parameters, whose time of revelation can be controlled by the box open decisions. Thus, the information base, encoded by the vector $\boldsymbol{w}_t(\boldsymbol{\xi}) \in \{0,1\}^I$, $t \in \mathcal{T}$, is a decision variable. In particular, $\boldsymbol{w}_{t,i}(\boldsymbol{\xi}) = 1$ if and only if box $i \in \mathcal{I}$ has been opened on or before time $t \in \mathcal{T}$ in scenario $\boldsymbol{\xi}$. We assume that $\boldsymbol{w}_0(\boldsymbol{\xi}) = \mathbf{0}$, so that no box is opened before the beginning of the planning horizon. We denote by $\boldsymbol{z}_{t,i}(\boldsymbol{\xi}) \in \{0,1\}$ the decision to keep box $i \in \mathcal{I}$ and stop the search at time $t \in \mathcal{T}$.

The requirement that, at most, one box be opened at each time $t \in \mathcal{T}$ and that no box be opened if we have stopped the search can be captured by the constraint

$$\sum_{i\in\mathcal{I}} (\boldsymbol{w}_{t,i}(\boldsymbol{\xi}) - \boldsymbol{w}_{t-1,i}(\boldsymbol{\xi})) \leq 1 - \sum_{\tau=1}^{t}\sum_{i\in\mathcal{I}} \boldsymbol{z}_{\tau,i}(\boldsymbol{\xi}) \quad \forall t \in \mathcal{T}. \tag{5}$$

The requirement that only one of the opened boxes can be kept is expressible as

$$\boldsymbol{z}_{t,i}(\boldsymbol{\xi}) \leq \boldsymbol{w}_{t-1,i}(\boldsymbol{\xi}) \quad \forall t \in \mathcal{T},\ \forall i \in \mathcal{I}. \tag{6}$$

The objective of the PB problem is to select the sequence of boxes to open and the box to keep so as to maximize worst-case net profit. Because the decision to open box i at time t can be expressed as the difference $(\boldsymbol{w}_{t,i} - \boldsymbol{w}_{t-1,i})$, the objective of the PB problem is

$$\max\ \min_{\boldsymbol{\xi}\in\Xi} \sum_{t\in\mathcal{T}}\sum_{i\in\mathcal{I}} \theta^{t-1}\boldsymbol{\xi}_i \boldsymbol{z}_{t,i}(\boldsymbol{\xi}) - \boldsymbol{c}_i(\boldsymbol{w}_{t,i}(\boldsymbol{\xi}) - \boldsymbol{w}_{t-1,i}(\boldsymbol{\xi})).$$

The mathematical model for this problem can be found in Online Appendix D.1.

3.2. Robust Pandora's Box: Model in ROC++

We present the ROC++ model for the PB problem. We assume that the data of the problem have been defined in C++, as summarized in Table 2. We discuss how to construct the key elements of the problem here. The full code can be found in Online Appendix D.2.

The PB problem is a multi-stage robust optimization problem involving uncertain parameters whose time of revelation is decision-dependent. Such models can be stored in the `ROCPPOptModelDDID` class, which is derived from `ROCPPOptModelIF`. We note that in ROC++, all optimization problems are minimization problems. All models are pointers to the interface class `ROCPPOptModelIF`. Thus, the robust PB problem can be initialized as:

(Color online)

```
// Create an empty robust model with T periods for the PB problem
ROCPPOptModelIF_Ptr PBModel(new ROCPPOptModelDDID(T, robust));
```

Table 2. List of Model Parameters and Their Associated C++ Variables for the PB Problem

Model parameter	C++ name	C++ variable type	C++ map keys
θ	`Theta`	`double`	NA
$T\,(t)$	`T(t)`	`uint`	NA
$I\,(i)$	`I(i)`	`uint`	NA
$M\,(m)$	`M(m)`	`uint`	NA
$\boldsymbol{c}_i,\ i \in \mathcal{I}$	`CostOpen`	`map<uint,double>`	`i=1…I`
$\overline{\boldsymbol{\xi}}_i,\ i \in \mathcal{I}$	`NomVal`	`map<uint,double>`	`i=1…I`
$\boldsymbol{\Phi}_{im},\ i \in \mathcal{I},\ m \in \mathcal{M}$	`FactorCoeff`	`map<uint,map<uint,double> >`	`i=1…I, m=1…M`

Note. NA, not applicable.

Next, we create the ROC++ variables associated with uncertain parameters and decision variables in the problem. The correspondence between variables is summarized in Table 3.

The uncertain parameters of the PB problem are $\boldsymbol{\xi} \in \mathbb{R}^I$ and $\boldsymbol{\zeta} \in \mathbb{R}^M$. We store the ROC++ variables associated with these in the `Value` and `Factor` maps, respectively. Each uncertain parameter is a pointer to an object of type `ROCPPUnc`. The constructor of the `ROCPPUnc` class takes two input parameters: the name of the uncertain parameter and the period when that parameter is revealed (first time stage when it is observable). As $\boldsymbol{\xi}$ has a time of revelation that is decision-dependent, we can omit the second parameter when we construct the associated ROC++ variables. The `ROCPPUnc` constructor also admits a third (optional) parameter with default value true that indicates if the uncertain parameter is observable. As $\boldsymbol{\zeta}$ is an auxiliary uncertain parameter, we set its time period as being, for example, one and indicate through the third parameter in the constructor of `ROCPPUnc` that this parameter is not observable.

(Color online)

```
// Create empty maps to store the uncertain parameters
map<uint, ROCPPUnc_Ptr> Value, Factor;
for (uint i = 1; i <= I; i++)
    // Create the uncertainty associated with box i and add it to Value
    Value[i] = ROCPPUnc_Ptr(new ROCPPUnc("Value_"+to_string(i)));
for (uint m = 1; m <= M; m++)
    // The risk factors are not observable
    Factor[m]= ROCPPUnc_Ptr(new ROCPPUnc("Factor_"+to_string(m),1,false));
```

The decision variables of the problem are the measurement variables $\boldsymbol{w}$ and the variables $\boldsymbol{z}$, which decide on the box to keep. We store these in the maps `MeasVar` and `Keep`, respectively. In ROC++, the measurement variables are created automatically for all time periods in the problem by calling the `add_ddu()` function, which is a public member of `ROCPPOptModelIF`. This function admits four input parameters: an uncertain parameter, the first and last time period when the decision-maker can choose to observe that parameter, and the cost for observing the parameter. In this problem, the cost for observing $\boldsymbol{\xi}_i$ is equal to $\boldsymbol{c}_i$. The measurement variables constructed in this way can be recovered by using the `getMeasVar()` function, which admits as inputs the name of an uncertain parameter and the time period for which we want to recover the measurement variable associated with that uncertain parameter.

(Color online)

```
map<uint, map<uint, ROCPPVarIF_Ptr> > MeasVar;
for (uint i = 1; i <= I; i++) {
    // Create the measurement variables associated with the value of box i
    PBModel->add_ddu(Value[i], 1, T, obsCost[i]);
    // Get the measurement variables and store them in MeasVar
    for (uint t = 1; t <= T; t++)
        MeasVar[t][i] = PBModel->getMeasVar(Value[i]->getName(), t);
}
```

Table 3. List of Model Variables and Uncertainties and Their Associated C++ Variables for the PB Problem

Variable	C++ name	C++ type	C++ map keys
$z_{t,i}$, $i \in \mathcal{I}$, $t \in \mathcal{T}$	`Keep`	`map<uint,map<uint,ROCPPVarIF_Ptr> >`	`t=1…T, i=1…I`
$w_{t,i}$, $i \in \mathcal{I}$, $t \in \mathcal{T}$	`MeasVar`	`map<uint,map<uint,ROCPPVarIF_Ptr> >`	`t=1…T, i=1…I`
ζ_m, $m \in \mathcal{M}$	`Factor`	`map<uint,ROCPPUnc_Ptr>`	`m=1…M`
ξ_i, $i \in \mathcal{I}$	`Value`	`map<uint,ROCPPUnc_Ptr>`	`i=1…I`

The boolean Keep variables can be built in ROC++ by using the constructors of the `ROCPPStaticVarBool` and `ROCPPAdaptVarBool` classes for the static and adaptive variables, respectively. The constructor of `ROCPPStaticVarBool` admits one input parameter: the name of the variable. The constructor of `ROCPPAdaptVarBool` admits two input parameters: the name of the variable and the time period when the decision is made.

(Color online)

```
map<uint, map<uint, ROCPPVarIF_Ptr> > Keep;
for (uint t = 1; t <= T; t++) {
    for (uint i = 1; i <= I; i++) {
        if (t == 1)  // In the first period, the Keep variables are static
            Keep[t][i] = ROCPPVarIF_Ptr(new ROCPPStaticVarBool("Keep_"+to_string(t
                )+"_"+to_string(i)));
        else   // In the other periods, the Keep variables are adaptive
            Keep[t][i] = ROCPPVarIF_Ptr(new ROCPPAdaptVarBool("Keep_"+to_string(t)
                +"_"+to_string(i), t));
    }
}
```

Having created the decision variables and uncertain parameters, we turn to adding the constraints to the model. To this end, we use the `StoppedSearch` expression, which tracks the running sum of the `Keep` variables, to indicate whether at any given point in time, we have already decided to keep one box and stop the search. We also use the `NumOpened` expression that, at each period, stores the expression for the total number of boxes that we choose to open in that period. Using these expressions, the constraints can be added to the problem using the following code.

(Color online)

```
 // Create the constraints and add them to the problem
ROCPPExpr_Ptr StoppedSearch(new ROCPPExpr());
for (uint t = 1; t <= T; t++) {
    // Create the constraint that at most one box be opened at t (none if the
        search has stopped)
    ROCPPExpr_Ptr NumOpened(new ROCPPExpr());
    // Update the expressions and and the constraint to the problem
    for (uint i = 1; i <= I; i++) {
        StoppedSearch = StoppedSearch + Keep[t][i];
        if (t>1)
            NumOpened = NumOpened + MeasVar[t][i] - MeasVar[t-1][i];
        else
            NumOpened = NumOpened + MeasVar[t][i];
    }
    PBModel->add_constraint( NumOpened <= 1. - StoppedSearch );
    // Constraint that only one of the open boxes can be kept
    for (uint i = 1; i <= I; i++)
        PBModel->add_constraint( (t>1) ? (Keep[t][i] <= MeasVar[t-1][i]) : (Keep[t
            ][i] <= 0.));
}
```

Next, we create the uncertainty set and the objective function.

(Color online)

```
// Create the uncertainty set constraints and add them to the problem
// Add the upper and lower bounds on the risk factors
for (uint m = 1; m <= M; m++) {
    PBModel->add_constraint_uncset(Factor[m] >= -1.0);
    PBModel->add_constraint_uncset(Factor[m] <= 1.0);
}
// Add the expressions for the box values in terms of the risk factors
for (uint i = 1; i <= I; i++) {
    ROCPPExpr_Ptr ValueExpr(new ROCPPExpr());
    for (uint m = 1; m <= M; m++)
        ValueExpr = ValueExpr + RiskCoeff[i][m]*Factor[m];
    PBModel->add_constraint_uncset(  Value[i] == (1.+0.5*ValueExpr) * NomVal[i] );
}
// Create the objective function expression
ROCPPExpr_Ptr PBObj(new ROCPPExpr());
for (uint t = 1; t <= T; t++)
    for (uint i = 1; i <= I; i++)
        PBObj = PBObj + pow(theta,t-1)*Value[i]*Keep[t][i];
// Set objective (multiply by -1 for maximization)
PBModel->set_objective(-1.0*PBObj);
```

We emphasize that the observation costs were automatically added to the objective function when we called the add_ddu() function.

3.3. Robust Pandora's Box: Solution in ROC++

The PB problem is a multi-stage robust problem with decision-dependent information discovery; see Vayanos et al. (2011, 2019). ROC++ offers two options for solving this class of problems: finite adaptability and piecewise constant decision rules; see Online Appendix Section A. Here, we illustrate how to solve PB using the finite adaptability approach; see Online Appendix Section A.2. Mathematically, the finite adaptability approximation of a problem is a multi-stage robust optimization problem, wherein in the first period, a collection of contingency plans $z_t^{k_1,\dots,k_t} \in \{0,1\}^{\ell_t}$ and $w_t^{k_1,\dots,k_t} \in \{0,1\}^k$, $k_t \in \{1,\dots, K_t\}$, $t \in \mathcal{T}$ for the variables $z_t(\boldsymbol{\xi})$ and $w_t(\boldsymbol{\xi})$ is chosen. Then, at the beginning of each period $t \in \mathcal{T}$, one of the contingency plans for that period is selected to be implemented, in an adaptive fashion; see Online Appendix A for more details. We let Kmap store the number of contingency plans K_t per period—the index in the map indicates the time period t, and the value it maps to corresponds to the choice of K_t. The process of computing the optimal contingency plans is streamlined in ROC++.

(Color online)

```
// Construct the reformulation orchestrator
ROCPPOrchestrator_Ptr pOrch(new ROCPPOrchestrator());
// Construct the finite adaptability reformulation strategy with 2 candidate
    policies in the each time stage
ROCPPStrategy_Ptr pKadaptStrategy(new ROCPPKAdapt(Kmap));
// Construct the robustify engine reformulation strategy
ROCPPStrategy_Ptr pRE (new ROCPPRobustifyEngine());
//Construct the linearization strategy based on big M constraints
ROCPPStrategy_Ptr pBTR (new ROCPPBTR_bigM());
// Approximate the adaptive decisions using the linear/constant decision rule
    approximator and robustify
vector<ROCPPStrategy_Ptr> strategyVec {pKadaptStrategy, pRE, pBTR};
ROCPPOptModelIF_Ptr PBModelKAadapt = pOrch->Reformulate(PBModel, strategyVec);
// Construct the solver and solve the problem
ROCPPSolver_Ptr pSolver(new ROCPPGurobi(SolverParams()));
pSolver->solve(PBModelKAdapt);
```

We consider the instance of PB detailed in Online Appendix D.3, for which $T = 4$, $M = 4$, and $I = 5$. For $K_t = 1$ (resp., $K_t = 2$ and $K_t = 3$) for all $t \in \mathcal{T}$, the problem takes under half a second (resp., under half a second and six seconds) to approximate and robustify. Its objective value is 2.12 (resp., 9.67 and 9.67). Note that with $T = 4$ and $K_t = 2$ (resp., $K_t = 3$), the total number of contingency plans is eight (resp., 27).

Next, we showcase how optimal contingency plans can be retrieved in ROC++.

(Color online)

```
// Retrieve the optimal solution from the solver
map<string,double> optimalSln(pSolver->getSolution());
// Print the optimal decision (from the original model)
// Print decision rules for variable Keep_4_2 from the original problem
    automatically
ROCPPKAdapt_Ptr pKadapt = static_pointer_cast<ROCPPKAdapt>(pKadaptStrategy);
pKadapt->printOut(PBModel, optimalSln, Keep[4][2]);
```

When executing this code, the values of all variables $z_{2,4}^{k_1\ldots k_t}$ used to approximate $z_{2,4}$ under all contingency plans $(k_1,\ldots,k_t) \in \times_{\tau=1}^{t} \{1,\ldots,K^\tau\}$ are printed. We show here the subset of the output associated with contingency plans where $z_{2,4}(\boldsymbol{\xi})$ equals one (for the case $K = 2$).

```
Value of variable Keep_4_2 under contingency plan (1-2-2-1) is: 1
```

Thus, at time 4, we will keep the second box if and only if the contingency plan we choose is $(k_1,k_2,k_3,k_4) = (1,2,2,1)$. We can display the first time that an uncertain parameter is observed using the following ROC++ code.

(Color online)

```
// Prints the observation decision for uncertain parameter Value_2
pKadapt->printOut(PBModel, optimalSln, Value[2]);
```

When executing this code, the time when $\boldsymbol{\xi}_2$ is observed under each contingency plan $(k_1,\ldots,k_T) \in \times_{\tau\in\mathcal{T}} \mathcal{K}^t$ is printed. In this case, part of the output we get is as follows.

```
Parameter Value_2 under contingency plan (1-1-1-1) is never observed
Parameter Value_2 under contingency plan (1-2-2-1) is observed at time 2
```

Thus, in an optimal solution, $\boldsymbol{\xi}_2$ is opened at time 2 under contingency plan $(k_1,k_2,k_3,k_4) = (1,2,1,1)$. On the other hand, it is never opened under contingency plan $(1,1,1,1)$.

4. Extensions

4.1. ROB File Format

Given a robust/stochastic optimization problem expressed in ROC++, our platform can generate a file displaying the problem in human readable format. We show the first few lines from the file obtained by printing the Pandora's Box problem of Section 3 to illustrate this format, with extension ".rob".

(Color online)

```
# minimize either expected or worst-case costs, as indicated by E or max
Objective:
min max -1 Keep_1_1 Value_1 -1 Keep_1_2 Value_2 -1 Keep_1_3 Value_3  ...
Constraints:
c0: -1 mValue_2_1 +1 mValue_1_1 <= +0 ...
Uncertainty Set:
c0: -1 Factor_1 <= +1 ...
# For decision variable, list its name, type, adaptability, time stage,
# and associated uncertain parameter if it's a measurement variable
Decision Variables:
Keep_1_1: Boolean, Static, 1, Non-Measurement
mValue_2_2: Boolean, Adaptive, 2, Measurement, Value_2
Bounds:
0 <= Keep_1_1 <= 1 ...
# For uncertain parameter, list its name, observability, time stage,
# the first and last observable stages if it's decision dependent
Uncertainties:
Factor_4: Not Observable, 1, Non-DDU
Value_1: Observable, 1, DDU, 1, 4 ...
```

4.2. Integer Decision Variables

ROC++ can solve problems involving integer decision variables. In the case of the CDR/PWC approximations, integer adaptive variables are directly approximated by constant/piecewise constant decisions that are integer on each subset of the partition. In the case of the finite adaptability approximation, bounded integer variables must first be expressed as finite sums of binary variables before the approximation is applied. This can be achieved through the reformulation strategy `ROCPPBinaryMItoMB`.

4.3. Stochastic Programming Capability

ROC++ currently provides limited support for solving stochastic programs with exogenous and/or decision-dependent information discovery based on the paper Vayanos et al. (2011). In particular, the approach from Vayanos et al. (2011) is available for the case where the uncertain parameters are uniformly distributed in a box. We showcase this functionality via an example on a stochastic best box problem in Online Appendix Section B.2.

4.4. Limited Memory Decision Rules

For problems involving long time-horizons (>100), the LDR/CDR and PWL/PWC decision rules can become computationally expensive. Limited memory decision rules approximate adaptive decisions by linear functions of the *recent* history of observations. The memory parameter of the `ROCPPConstantDR`, `ROCPPLinearDR`, and `ROCPPPWDR` can be used in ROC++ to trade off optimality with computational complexity.

4.5. The ROPy Python Interface

We use pybind11,[20] a lightweight header-only library, to create Python bindings of the C++ code. With the Python interface we provide, users can generate a Python library called ROPy, which contains all the functions needed for creating decision variables, constraints, and models supported by ROC++. ROPy also implements the dynamic behavior via strategy pattern. It includes all reformulation strategies of ROC++ and uses the reformulation orchestrator to apply the strategy sequentially. The concise grammar of Python makes ROPy easy to use. Code extendability is guaranteed by pybind11. Developers may directly extend the library ROPy (by, e.g., deriving new classes) in Python without looking into the C++ code or by rebuilding the library after making changes in C++. ROPy code to all the examples in our paper can be found in our GitHub repository.

5. Conclusion

We proposed ROC++, an open-source platform for automatic robust optimization in C++ that can be used to solve single-stage and multi-stage robust optimization problems with binary and/or real-valued variables, with exogenous and/or endogenous uncertainty set and with exogenous and/or endogenous information discovery. ROC++ is very easy to use, thanks to operator overloading in C++ that allows users to enter constraints to a ROC++ model in the same way that they look on paper and thanks to the strategy pattern that allows users to select the reformulation strategy to employ at runtime. ROC++ is also very easy to extend, thanks to extensive use of inheritance throughout and thanks to the numerous hooks that are available (e.g., new reformulation strategies and new solvers). We also provide a Python library to ROC++, named ROPy. ROPy is easy to extend either directly in Python or in C++. We believe that ROC++ can facilitate the use of robust optimization among both researchers and practitioners.

Some desirable extensions to ROC++ that we plan to include in future releases are unit test capability, support for distributionally robust optimization, polynomial decision rules, and constraint sampling. We also hope to generalize the classes of stochastic programming problems that can be addressed by ROC++ by adding support for problems where the mean and covariance of the uncertain parameters are known. Finally, we are constantly working to improve usability and extendability, following the SOLID principles of object oriented programming.

Acknowledgments

The authors are grateful to Dimitrios Elissaios, Daniel Kuhn, Berç Rustem, and Wolfram Wiesemann for valuable discussions that helped shape this work. The authors also thank the entire review team, including the area editor, the associate editor, and four anonymous reviewers, for their constructive comments that helped improve the quality of both our software and paper.

Endnotes

[1] See https://www.gurobi.com.

[2] See https://www.scipopt.org.

[3] See https://www.doxygen.nl/index.html.

[4] See https://github.com/g0900971/RobustOptimization.

[5] See https://www.ibm.com/analytics/cplex-optimizer.

[6] See https://robustopt.com.

[7] See https://www.rsomerso.com.

[8] See https://www.mosek.com.

[9] See https://www.math.cmu.edu/~reha/sdpt3.html.

[10] See https://jumper.readthedocs.io/en/latest/jumper.html.

[11] See https://www.aimms.com.

[12] See https://www.coin-or.org/Clp/.

[13] See https://www.coin-or.org/Cbc/.

[14] See https://www.gnu.org/software/glpk/.

[15] See https://pyomo.readthedocs.io/en/stable/modeling_extensions/stochastic_programming.html.

[16] See https://odow.github.io/SDDP.jl/stable/.

[17] See https://github.com/coin-or/Smi.

[18] See https://coin-or.github.io/Ipopt/.

[19] See https://github.com/coin-or/Osi.

[20] See https://pybind11.readthedocs.io/en/stable/.

References

Ardestani-Jaafari A, Delage E (2016) Robust optimization of sums of piecewise linear functions with application to inventory problems. *Oper. Res.* 64(2):474–494.

Bampou D, Kuhn D (2011) Scenario-free stochastic programming with polynomial decision rules. *Proc. 50th IEEE Conf. Decision Control* (IEEE, Piscataway, NJ), 7806–7812.

Bandi C, Bertsimas D (2012) Tractable stochastic analysis in high dimensions via robust optimization. *Math. Program.* 134:23–70.

Bandi C, Trichakis N, Vayanos P (2018) Robust multiclass queuing theory for wait time estimation in resource allocation systems. *Management Sci.* 65(1):152–187.

Ben-Tal A, El Ghaoui L, Nemirovski A (2009) *Robust Optimization, Princeton Series in Applied Mathematics* (Princeton University Press, Princeton, NJ).

Ben-Tal A, Golany B, Nemirovski A, Vial JP (2005) Retailer-supplier flexible commitments contracts: A robust optimization approach. *Manufacturing Service Oper. Management* 7(3):248–271.

Ben-Tal A, Goryashko A, Guslitzer E, Nemirovski A (2004) Adjustable robust solutions of uncertain linear programs. *Math. Program.* 99(2):351–376.

Bertsimas D, Caramanis C (2010) Finite adaptability for linear optimization. *IEEE Trans. Automatic Control* 55(12):2751–2766.

Bertsimas D, Dunning I (2016) Multi-stage robust mixed-integer optimization with adaptive partitions. *Oper. Res.* 64(4):980–998.

Bertsimas D, Georghiou A (2015) Design of near optimal decision rules in multi-stage adaptive mixed-integer optimization. *Oper. Res.* 63(3):610–627.

Bertsimas D, Georghiou A (2018) Binary decision rules for multi-stage adaptive mixed-integer optimization. *Math. Program.* 167(2):395–433.

Bertsimas D, Vayanos P (2014) Data-driven learning in dynamic pricing using adaptive robust optimization. Preprint, submitted October 11, http://www.optimization-online.org/DB_HTML/2014/10/4595.html.

Bertsimas D, Brown D, Caramanis C (2010) Theory and applications of robust optimization. *SIAM Rev.* 53(3):464–501.

Bertsimas D, Iancu D, Parrilo P (2011) A hierarchy of near-optimal policies for multi-stage adaptive optimization. *IEEE Trans. Automatic Control* 56(12):2809–2824.

Bertsimas D, Sim M, Zhang M (2019) Adaptive distributionally robust optimization. *Management Sci.* 65(2):604–618.

Birge JR, Louveaux F (2000) *Introduction to Stochastic Programming, Springer Series in Operational Research and Financial Engineering* (Springer, New York).

Bodur M, Luedtke JR (2022) Two-stage linear decision rules for multi-stage stochastic programming. *Math. Program.* 191:347–380.

Campi MC, Garatti S (2008) The exact feasibility of randomized solutions of robust convex programs. *SIAM J. Optim.* 19(3):1211–1230.

Chen Z, Sim M, Xiong P (2020) Robust stochastic optimization made easy with RSOME. *Management Sci.* 66(8):3329–3339.

Colvin M, Maravelias CT (2008) A stochastic programming approach for clinical trial planning in new drug development. *Comput. Chem. Engrg.* 32(11):2626–2642.

Ding L, Ahmed S, Shapiro A (2019) A Python package for multi-stage stochastic programming. Preprint, submitted May 7, http://www.optimization-online.org/DB_HTML/2019/05/7199.html.

Dowson O, Kapelevich L (2021) SDDP.jl: A Julia package for stochastic dual dynamic programming. *INFORMS J. Comput.* 33(1):27–33.

Dunning I, Huchette J, Lubin M (2017) JuMP: A modeling language for mathematical optimization. *SIAM Rev.* 59(2):295–320.

Georghiou A, Wiesemann W, Kuhn D (2015) Generalized decision rule approximations for stochastic programming via liftings. *Math. Program.* 152(1):301–338.

Goel V, Grossman IE (2004) A stochastic programming approach to planning of offshore gas field developments under uncertainty in reserves. *Comput. Chem. Engrg.* 28(8):1409–1429.

Goh J, Sim M (2011) Robust optimization made easy with ROME. *Oper. Res.* 59(4):973–985.

Gounaris CE, Wiesemann W, Floudas CA (2013) The robust capacitated vehicle routing problem under demand uncertainty. *Oper. Res.* 61(3):677–693.

Haider Z, Charkhgard H, Kwon C (2018) A robust optimization approach for solving problems in conservation planning. *Ecol. Model.* 368:288–297.

Hanasusanto GA, Kuhn D, Wiesemann W (2015) K-adaptability in two-stage robust binary programming. *Oper. Res.* 63(4):877–891.

Jiang R, Zhang M, Li G, Guan Y (2014) Two-stage network constrained robust unit commitment problem. *Eur. J. Oper. Res.* 234(3):751–762.

Jonsbråten TW (1998) Optimization models for petroleum field exploitation. Unpublished PhD thesis, Norwegian School of Economics and Business Administration, Bergen, Norway.

Kuhn D, Wiesemann W, Georghiou A (2009) Primal and dual linear decision rules in stochastic and robust optimization. *Math. Program.* 130(1):177–209.

Lappas NH, Gounaris CE (2018) Robust optimization for decision-making under endogenous uncertainty. *Comput. Chem. Engrg.* 111:252–266.

Löfberg J (2012) Automatic robust convex programming. *Optim. Methods Software* 27(1):115–129.

Mamani H, Nassiri S, Wagner MR (2017) Closed-form solutions for robust inventory management. *Management Sci.* 63(5):1625–1643.

Martin RC (2003) *Agile Software Development: Principles, Patterns, and Practices* (Prentice Hall PTR, Upper Saddle River, NJ).

Nohadani O, Roy A (2017) Robust optimization with time-dependent uncertainty in radiation therapy. *IISE Trans. Healthcare Systems Engrg.* 7(2):81–92.

Nohadani O, Sharma K (2018) Optimization under decision-dependent uncertainty. *SIAM J. Optim.* 28(2):1773–1795.

Perez S (2018) Coding dynamic behavior with the strategy pattern. Accessed May 1, 2021, https://severinperez.medium.com/coding-dynamic-behavior-with-the-strategy-pattern-c0bebaee6671#.

Rahimian H, Mehrotra S (2019) Distributionally robust optimization: A review. Preprint, submitted August 13, https://arxiv.org/pdf/1908.05659.pdf.

Rocha P, Kuhn D (2012) Multi-stage stochastic portfolio optimization in deregulated electricity markets using linear decision rules. *Eur. J. Oper. Res.* 216(2):397–408.

Solak S, Clarke JP, Johnson EL, Barnes ER (2010) Optimization of R&D project portfolios under endogenous uncertainty. *Eur. J. Oper. Res.* 207(1):420–433.

Vayanos P, Georghiou A, Yu H (2020) Robust optimization with decision-dependent information discovery. Preprint, submitted April 18, https://arxiv.org/pdf/2004.08490.pdf.

Vayanos P, Jin Q, Elissaios G (2022) ROCPP version v2020.0140. Accessed March 17, 2022, http://dx.doi.org/10.5281/zenodo.6360996.

Vayanos P, Kuhn D, Rustem B (2011) Decision rules for information discovery in multi-stage stochastic programming. *Proc. 50th IEEE Conf. Decision Control* (IEEE, Piscataway, NJ), 7368–7373.

Vayanos P, Kuhn D, Rustem B (2012) A constraint sampling approach for multi-stage robust optimization. *Automatica J. IFAC* 48(3):459–471.

Vayanos P, Ye Y, McElfresh D, Dickerson J, Rice E (2021) Robust active preference elicitation. Preprint, submitted March 4, https://arxiv.org/abs/2003.01899.

Weitzman ML (1979) Optimal search for the best alternative. *Econometrica* 47(3):641–654.

Wiesemann W, Kuhn D, Sim M (2014) Distributionally robust convex optimization. *Oper. Res.* 62(6):1358–1376.