RETROSPECTIVE APPROXIMATION ALGORITHMS FOR MULTI-OBJECTIVE SIMULATION OPTIMIZATION ON INTEGER LATTICES

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Kyle Cooper

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Susan R. Hunter, Chair

School of Industrial Engineering, Purdue University

Dr. Andrew Liu

School of Industrial Engineering, Purdue University

Dr. Hong Wan

School of Industrial Engineering, Purdue University

Dr. Jeffrey Tew

Tata Consultancy Services

Dr. Raghu Pasupathy

Department of Statistics, Purdue University

Approved by:

Dr. Steven J. Landry

Head of the School Graduate Program

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Cooper, Kyle PhD, Purdue University, May 2019. Retrospective Approximation Algorithms for Multi-Objective Simulation Optimization on Integer Lattices. Major Professor: Susan R. Hunter.

We consider multi-objective simulation optimization (MOSO) problems, that is, nonlinear optimization problems in which multiple simultaneous objective functions can only be observed with stochastic error, e.g., as output from a Monte Carlo simulation model. In this context, the solution to a MOSO problem is the efficient set, which is the set of all feasible decision points for which no other feasible decision point is at least as good on all objectives and strictly better on at least one objective. We are concerned primarily with MOSO problems on integer lattices, that is, MOSO problems where the feasible set is a subset of an integer lattice.

In the first study, we propose the Retrospective Partitioned Epsilon-constraint with Relaxed Local Enumeration (R-PεRLE) algorithm to solve the bi-objective simulation optimization problem on integer lattices. R-PεRLE is designed for sampling efficiency. It uses a retrospective approximation (RA) framework to repeatedly call the PεRLE sample-path solver at a sequence of increasing sample sizes, using the solution from the previous RA iteration as a warm start for the current RA iteration. The PεRLE sample-path solver is designed to solve the sample-path problem only to within a tolerance commensurate with the sampling error. It comprises a call to each of the Pε and RLE algorithms, in sequence. First, Pε searches for new points to add to the sample-path local efficient set by solving multiple constrained single-objective optimization problems. Pε places constraints to locate new sample-path local efficient points that are a function of the standard error away, in the objective space, from those already obtained. Then, the set of sample-path local efficient

points found by Pε is sent to RLE, which is a local crawling algorithm that ensures the set is a sample-path approximate local efficient set. As the number of RA iterations increases, R-PεRLE provably converges to a local efficient set with probability one under appropriate regularity conditions. We also propose a naive, provably-convergent benchmark algorithm for problems with two or more objectives, called R-MinRLE. R-MinRLE is identical to R-PεRLE except that it replaces the Pε algorithm with an algorithm that updates one local minimum on each objective before invoking RLE. R-PεRLE performs favorably relative to R-MinRLE and the current state of the art, MO-COMPASS, in our numerical experiments. Our work points to a family of RA algorithms for MOSO on integer lattices that employ RLE for certification of a sample-path approximate local efficient set, and for which the convergence guarantees are provided in this study.

In the second study, we present the PyMOSO software package for solving multi-objective simulation optimization problems on integer lattices, and for implementing and testing new simulation optimization (SO) algorithms. First, for solving MOSO problems on integer lattices, PyMOSO implements R-PεRLE and R-MinRLE, which are developed in the first study. Both algorithms employ pseudo-gradients, are designed for sampling efficiency, and return solutions that, under appropriate regularity conditions, provably converge to a local efficient set with probability one as the simulation budget increases. PyMOSO can interface with existing simulation software and can obtain simulation replications in parallel. Second, for implementing and testing new SO algorithms, PyMOSO includes pseudo-random number stream management, implements algorithm testing with independent pseudo-random number streams run in parallel, and computes the performance of algorithms with user-defined metrics. For convenience, we also include an implementation of R-SPLINE for problems with one objective. The PyMOSO source code is available under a permissive open source license.

1. INTRODUCTION

We consider of multi-objective simulation optimization (MOSO) problems, which are nonlinear optimization problems where each objective function can only be observed with stochastic error. MOSO problems emerge, for example, when decision-makers use a Monte Carlo simulation model to design and optimize a complex stochastic system in the presence of multiple simultaneous and conflicting objectives. MOSO problems with one objective function are single-objective simulation optimization (SOSO) problems. The solution is a feasible point which optimizes the expected value of the objective function. Except in the trivial case where there is no conflict between the objectives, no single feasible point can simultaneously optimize every objective function of a MOSO problem. Thus, in the case of conflicting objectives, the solution is the set of optimal feasible points called the *efficient set*, the set of feasible decision points for which no other feasible decision point is at least as good on all objectives and strictly better on at least one objective. Under a neighborhood structure, feasible sets may contain *local efficient sets*. Members of efficient sets and members of local efficient sets are called efficient points and local efficient points, respectively. We formally define efficient sets, local efficient sets, efficient points, and local efficient points in §1.2.2.

Although no feasible decision point will simultaneously optimize every performance measure, a decision-maker must decide which point to implement in practice. The desired solution to a MOSO problem can depend on many factors including the nature of the feasible points, the number of objectives, and the stage in the decision-making process in which a decision-maker expresses preferences. In the context of deterministic multi-objective optimization (MOO), Miettinen (1999) classifies solution methods based on when in the decision-making process the decision-maker's prefer-

ences are revealed within a multi-criteria decision-making (MCDM) framework. No-preference methods generate a single efficient point regardless of the decision-maker's preferences. A-posteriori methods generate the entire efficient set, or a characterization of the efficient set, without preferences and then allows the decision-maker to choose which point to implement as part of the MCDM process. A-priori methods consider the decision-maker's preferences before optimization. Finally, interactive methods query the decision-maker throughout the optimization process. Unless otherwise noted, we henceforth consider MOSO in the a-posteriori context, where the solution to a MOSO problem is the entire efficient set.

Formally, we define the MOSO problem as

$$\begin{aligned} \text{Problem } M\text{: minimize} \quad & \mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_d(\mathbf{x})) = (\mathbb{E}[G_1(\mathbf{x}, \boldsymbol{\xi})], \ldots, \mathbb{E}[G_d(\mathbf{x}, \boldsymbol{\xi})]) \\ \text{s.t.} \quad & \mathbf{x} \in \mathcal{X}, \end{aligned}$$

where $(\mathbb{E}[G_1(\mathbf{x}, \boldsymbol{\xi})], \ldots, \mathbb{E}[G_d(\mathbf{x}, \boldsymbol{\xi})])$ is a vector of $d \geq 2$ objective functions, the feasible set $\mathcal{X}$ is nonempty and known, and $\boldsymbol{\xi}$ is a random vector with support Ξ, such that $\mathbb{P}\{\boldsymbol{\xi} \in \Xi\} = 1$. In the minimization context, efficient points are such that no other feasible point maps to an objective vector which is at least as small on all objectives, and strictly smaller on at least one objective. The solution to Problem M is the efficient set, defined in §1.2.2.

MOSO problems are important because they exist in many application domains, especially in those domains that tend to employ Monte Carlo simulation models. We discuss three examples of MOSO problems.

Example 1 Zhou et al. (2018) solve a bi-objective scheduling problem to reduce congestion in a lighterage terminal. They model the lighterage terminal in a Monte Carlo simulation with objectives to minimize both expected truck time and expected barge time on the port.

Example 2 Seeking both to reduce the manual intervention required in scheduling and to choose an optimal schedule for a camshaft machining line, Andersson et al.

(2007) model two objectives: (1) a penalty function for expected shortage of the safety stock to be minimized and, (2) a representation of the expected process throughput to be maximized.

Example 3 Chen and Wang (2016) consider a medical resource allocation problem for a Taiwan hospital's emergency department. Hospital visitors continue to rise even as hospital resources, including doctors, funds, and administrators, remain constant. Chen and Wang (2016) represent the emergency department in a Monte Carlo simulation model which outputs two performance objectives: a function representing the expected average patient's length of stay to minimize, and and function representing the expected wasted medical resources (e.g. an idle doctor) to minimize.

We adapt Table 1.1 from Hunter et al. (2019), which classifies application papers based on their fit within 2017 Winter Simulation Conference application tracks (Chan et al., 2017). As shown in Table 1.1, efficient MOSO solution methods can substantially improve problem-solving capabilities in many application areas. Although MOSO problems are common, solution methods are relatively sparse when compared to those in MOO or in SOSO (Hunter et al., 2019). We discuss MOSO categories and existing methods in Chapter 2.

Table 1.1.
The table categorizes example MOSO applications by application area (Hunter et al., 2019).

Area	MOSO Application Papers
agriculture	plant breeding (Hunter and McClosky, 2016) irrigation design (Crespo et al., 2010) land management under climate change (Klein et al., 2013)
architecture & construction	earthmoving operations (Zhang, 2008) *review:* building performance analysis, usually $d \in \{2, 3\}$ (Nguyen et al., 2014)
aviation	aircraft spare part management (Li et al., 2015c,a) aircraft flight scheduling (Lee et al., 2007) aircraft spare part management (Lee et al., 2008)
energy	oil drilling (Kim, 2014) design of burners in the combustion chamber of a gas turbine (Büche et al., 2002) power plant design (Subramanyan et al., 2011) energy pricing in an electricity market as a stochastic collaborative game between d players (Fliege and Xu, 2011, p. 158)
environment & sustainability	groundwater remediation design (Singh and Minsker, 2008) dynamic flood control operation in a river-reservoir system with up to $d = 5$ (Prakash et al., 2015)
healthcare	capacity allocation in an emergency department or obstetrics ward (Chen and Wang, 2016; Lucidi et al., 2016) portable ultrasound machine allocation (Huang, 2016, p. 90) patient flow between healthcare centers and hospital with $d = 8$ reduced to $d = 2$ (Song et al., 2016) patient flow and capacity in a cancer treatment center (Baesler and Sepulveda, 2001) hospital inpatient flow process, solved as three sets of paired objectives (Wang et al., 2015) cadaveric liver allocation policies (Feng et al., 2013)
logistics, supply chain & transportation	supply chain management (Ding et al., 2006) reduce congestion in a lighterage terminal (Zhou et al., 2018) differentiated service inventory management (Chew et al., 2009) supply chain management (Joines et al., 2002; Amodeo et al., 2009; Li et al., 2017) train traction system design (Dullinger et al., 2017)
manufacturing	production line scheduling (Andersson et al., 2007) injection molding (Villarreal-Marroquín et al., 2013)
military	fighter aircraft maintenance (Mattila and Virtanen, 2014) military ground vehicle design for safety (Hoffenson et al., 2014)

1.1 Overview and Contributions

In the remainder of Chapter 1, we formally define the solution to a MOSO problem, both in the global and local sense. In Chapter 2, we provide an introduction to retrospective approximation (RA), the framework we use for our algorithms; we briefly introduce MOO and SOSO concepts; and we categorize existing MOSO literature by the nature of the feasible sets for which the methods were designed.

In Chapter 3, we present a family of algorithms which converge almost surely to a local efficient set. In particular, we present the Retrospective Partitioned Epsilon-constraint with Relaxed Local Enumeration (R-PεRLE) algorithm, a new state-of-the-art algorithm for solving a bi-objective MOSO problems on integer lattices. Chapter 3 also includes R-MinRLE, a competitive algorithm for solving MOSO problems on integer lattices. For simulation sampling efficiency, the algorithms employ a RA framework (see §2.1). Within the RA framework, we scalarize the MOO problem to a sequence of single-objective optimization problems using the ε-constraint method. We solve each single-objective optimization problem using the pseudo-gradient-based SPLINE algorithm, introduced in Wang et al. (2013) in the SOSO context.

To aid MOSO practitioners (such as the authors of papers listed in Table 1.1) and MOSO researchers (such as the authors of methods covered in Hunter et al. 2019), we present PyMOSO in Chapter 4, a software framework for solving MOSO problems and creating MOSO algorithms. Primarily, PyMOSO provides an off-the-shelf implementation of R-PεRLE for solving real MOSO problems, and a framework for quickly creating and testing new MOSO algorithms. The source code and user manual are found at `https://github.com/HunterResearch/PyMOSO`.

The work presented in this thesis has resulted in several publications. The work in Chapter 3 has resulted in the papers Cooper et al. (2017), which appeared in the Proceedings of the 2017 Winter Simulation Conference, and the journal article Cooper et al. (2018), which is currently under first revision. The work in Chapter 4 resulted in the journal article Cooper and Hunter (2018), which is currently under

review. Finally, the literature review information related to MOSO in Chapters 1 and 2 was inspired by Hunter et al. (2019), which is an accepted journal article that provides an introduction to MOSO and began as a collaborative course project for IE 690 at Purdue University in Fall 2016.

1.2 Preliminaries

In this section, we discuss the following: terminology and notation, optimality concepts for MOSO, and a formal problem statement.

1.2.1 Terminology and Notation

We adopt the terminology of Hunter et al. (2019) in that *efficient points* are in the decision space and their image, the *Pareto optimal points* or *Pareto points*, are in the objective function space. The set of all d-dimensional integer-valued vectors is $\mathbb{Z}^d \subset \mathbb{R}^d$. Usually, capital letters denote random variables (X), script capital letters denote sets ($\mathcal{A}$), vectors appear in bold ($\mathbf{x}$), and random vectors appear in capital bold ($\boldsymbol{X}$). The d-dimensional vector of zeros is $\mathbf{0}_d$. If $\mathbf{g}\colon \mathcal{X} \subseteq \mathbb{R}^q \to \mathbb{R}^d$ is a vector-valued function, then for some set $\mathcal{S} \subseteq \mathcal{X}$, the set $\mathbf{g}(\mathcal{S})$ is the image of the set $\mathcal{S}$, $\mathbf{g}(\mathcal{S}) := \{\mathbf{g}(\mathbf{x})\colon \mathbf{x} \in \mathcal{S}\}$. The sum of two sets $\mathcal{A}$ and $\mathcal{B}$ is the Minkowski sum, $\mathcal{A} + \mathcal{B} := \{\mathbf{a} + \mathbf{b} : \mathbf{a} \in \mathcal{A},\ \mathbf{b} \in \mathcal{B}\}$. We say that a sequence of random variables $X_1, X_2, \ldots$ converges with probability 1 (w.p.1) to a random variable X if for all $\varepsilon > 0$, $\mathbb{P}\{\lim_{n\to\infty} |X_n - X| < \varepsilon\} = 1$ (Casella and Berger, 2002, p. 234). For a sequence of events $\{E_n\}$ defined in a probability space, we say E_n infinitely often (i.o.) if infinitely many of E_n occur, where E_n i.o. $= \limsup_n E_n = \cap_{n=1}^{\infty} \cup_{j=n}^{\infty} E_j$ (Billingsley, 1995, p. 52–53). We require notions of distance. Let $\mathcal{A} \subset \mathbb{R}^q$ and $\mathcal{B} \subset \mathbb{R}^q$ be two nonempty, bounded sets. Then (a) $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||$ is the Euclidean distance between two points $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^q$; (b) $d(\mathbf{x}, \mathcal{B}) = \inf_{\mathbf{x}' \in \mathcal{B}} ||\mathbf{x} - \mathbf{x}'||$ is the distance from the point $\mathbf{x} \in \mathbb{R}^q$ to the set $\mathcal{B}$; (c) $\mathbb{D}(\mathcal{A}, \mathcal{B}) = \sup_{x \in \mathcal{A}} d(x, \mathcal{B})$ is the distance from set $\mathcal{A}$ to set $\mathcal{B}$; and

(d) $\mathbb{H}(\mathcal{A},\mathcal{B}) := \max\{\mathbb{D}(\mathcal{A},\mathcal{B}),\mathbb{D}(\mathcal{B},\mathcal{A})\}$ is the Hausdorff distance between sets $\mathcal{A}$ and $\mathcal{B}$.

1.2.2 Optimality Concepts

In this section, we define global and local solutions to Problem M, as in Miettinen (1999). First, to define concepts of global optimality in Problem M, we must define dominance.

Definition 1. *Let* $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$ *and* $d \geq 2$. *For vectors* $\mathbf{g}(\mathbf{x}_1)$ *and* $\mathbf{g}(\mathbf{x}_2)$, *we say that*

1. $\mathbf{g}(\mathbf{x}_1)$ weakly dominates $\mathbf{g}(\mathbf{x}_2)$, *written as* $\mathbf{g}(\mathbf{x}_1) \leqq \mathbf{g}(\mathbf{x}_2)$, *if* $g_k(\mathbf{x}_1) \leq g_k(\mathbf{x}_2)$ *for all* $k = 1, \ldots, d$.

2. $\mathbf{g}(\mathbf{x}_1)$ dominates $\mathbf{g}(\mathbf{x}_2)$, *written as* $\mathbf{g}(\mathbf{x}_1) \leq \mathbf{g}(\mathbf{x}_2)$, *if* $\mathbf{g}(\mathbf{x}_1) \leqq \mathbf{g}(\mathbf{x}_2)$ *and* $\mathbf{g}(\mathbf{x}_1) \neq \mathbf{g}(\mathbf{x}_2)$.

3. $\mathbf{g}(\mathbf{x}_1)$ strictly dominates $\mathbf{g}(\mathbf{x}_2)$, *written as* $\mathbf{g}(\mathbf{x}_1) < \mathbf{g}(\mathbf{x}_2)$, *if* $g_k(\mathbf{x}_1) < g_k(\mathbf{x}_2)$ *for all* $k = 1, \ldots, d$.

Using the concept of dominance, we now define efficient and weakly efficient points.

Definition 2. *A decision point* $\mathbf{x}^* \in \mathcal{X}$ *is*

1. *an* efficient point *if there does not exist another point* $\mathbf{x} \in \mathcal{X}$ *such that* $\mathbf{g}(\mathbf{x}) \leq \mathbf{g}(\mathbf{x}^*)$.

2. *a* weakly efficient point *if there does not exist another point* $\mathbf{x} \in \mathcal{X}$ *such that* $\mathbf{g}(\mathbf{x}) < \mathbf{g}(\mathbf{x}^*)$.

Notice that all efficient points are also weakly efficient points. We define a *Pareto point* and a *weakly Pareto point* as the image of an efficient point and a weakly efficient point, respectively.

We collect the non-dominated points into sets as follows.

Definition 3. *We define the following sets:*

1. *The* efficient set, $\mathcal{E} \subseteq \mathcal{X}$, *is the set of all efficient points.*
2. *The* weakly efficient set, $\mathcal{E}^{\mathrm{w}}$, *where* $\mathcal{E} \subseteq \mathcal{E}^{\mathrm{w}} \subseteq \mathcal{X}$, *is the set of all weakly efficient points.*

The *Pareto set* is the image of the efficient set $\mathcal{P} := \mathbf{g}(\mathcal{E})$, and the *weakly Pareto set* is the image of the weakly efficient set $\mathcal{P}^{\mathrm{w}} := \mathbf{g}(\mathcal{E}^{\mathrm{w}})$, where $\mathcal{P} \subseteq \mathcal{P}^{\mathrm{w}}$.

If a neighborhood structure can be defined on the feasible set, then there may exist local solutions to Problem M. First, points may be locally efficient if they are efficient within a neighborhood.

Definition 4. *Let* $\mathbf{x}^* \in \mathcal{X}$. *Given an appropriate neighborhood* $\mathcal{N}(\mathbf{x}^*)$, *we say* $\mathbf{x}^*$ *is*

1. *a* local efficient point *on* $\mathcal{N}$ *if there does not exist* $\mathbf{x} \in \mathcal{N}(\mathbf{x}^*) \cap \mathcal{X}$ *such that* $\mathbf{g}(\mathbf{x}) \leq \mathbf{g}(\mathbf{x}^*)$.
2. *a* local weakly efficient point *on* $\mathcal{N}$ *if there does not exist* $\mathbf{x} \in \mathcal{N}(\mathbf{x}^*) \cap \mathcal{X}$ *such that* $\mathbf{g}(\mathbf{x}) < \mathbf{g}(\mathbf{x}^*)$.

Locally efficient points may form local efficient sets. To define these concepts formally, we require definitions for the neighborhood of a set and the deleted neighborhood of a set. Given the previous notion of an appropriate neighborhood of a point $\mathcal{N}(\mathbf{x})$, we define the neighborhood of a set $\mathcal{S}$ as $\mathcal{N}(\mathcal{S}) := \cup_{\mathbf{x} \in \mathcal{S}} \mathcal{N}(\mathbf{x})$. The *deleted neighborhood* of a set $\mathcal{S}$ is $\mathcal{N}'(\mathcal{S}) := \mathcal{N}(\mathcal{S}) \setminus \mathcal{S}$. Then we have the following definitions (see Deb, 1999; Kim and Ryu, 2011a; Li et al., 2015a; Cooper et al., 2017, 2018).

Definition 5. *Given an appropriate neighborhood* $\mathcal{N}(\mathbf{x})$ *for each* $\mathbf{x} \in \mathcal{X}$,

1. *A set* $\mathcal{L}$ *is a* local efficient set *on* $\mathcal{N}$ *if (a) for all* $\mathbf{x}^* \in \mathcal{L}$, $\mathbf{x}^*$ *is a local efficient point on* $\mathcal{N}(\mathbf{x}^*)$, *(b) no points in* $\mathbf{g}(\mathcal{L})$ *dominate other points in* $\mathbf{g}(\mathcal{L})$, *and (c) for each* $\mathbf{x} \in \mathcal{N}'(\mathcal{L}) \cap \mathcal{X}$, *there exists* $\mathbf{x}^* \in \mathcal{L}$ *such that* $\mathbf{g}(\mathbf{x}^*) \leq \mathbf{g}(\mathbf{x})$.

2. *A set* $\mathcal{L}^{\mathrm{w}}$ *is a* local weakly efficient set *on* $\mathcal{N}$ *if (a) for all* $\mathbf{x}^* \in \mathcal{L}^{\mathrm{w}}$, $\mathbf{x}^*$ *is a local weakly efficient point on* $\mathcal{N}(\mathbf{x}^*)$, *(b) no points in* $\mathbf{g}(\mathcal{L}^{\mathrm{w}})$ *strictly dominate other points in* $\mathbf{g}(\mathcal{L}^{\mathrm{w}})$, *and (c) for each* $\mathbf{x} \in \mathcal{N}'(\mathcal{L}^{\mathrm{w}}) \cap \mathcal{X}$, *there exists* $\mathbf{x}^* \in \mathcal{L}^{\mathrm{w}}$ *such that* $\mathbf{g}(\mathbf{x}^*) \leqq \mathbf{g}(\mathbf{x})$.

Thus each point in a local efficient set must be a local efficient point on $\mathcal{N}$, the image of each point in the set should not be dominated by the images of any other points in the set, and each point in the deleted neighborhood of the local efficient set must be dominated by a point in the local efficient set. Further, the local Pareto set is $\mathcal{L} := \mathbf{g}(\mathcal{E}_{\mathcal{N}})$ and the local weakly Pareto set is $\mathcal{P}^{\mathrm{w}}_{\mathcal{N}} := \mathbf{g}(\mathcal{L}^{\mathrm{w}})$.

1.2.3 Problem Statement

We now introduce the broad problem statement considered in this thesis. For each objective function in Problem M we assume the existence of an oracle capable of producing consistent estimators for each $\mathbf{x} \in \mathcal{X}$. That is, we assume the oracle produces estimators $\hat{G}_k(\mathbf{x}, n)$ where $\hat{G}_k(\mathbf{x}, n) \to g_k(\mathbf{x})$ w.p.1 as the simulation effort $n \to \infty$ for all $k = 1, \ldots, d$ and all $\mathbf{x} \in \mathcal{X}$; let $\hat{\boldsymbol{G}}(\mathbf{x}, n) := (\hat{G}_1(\mathbf{x}, n), \ldots, \hat{G}_d(\mathbf{x}, n))$. Then the MOSO problem statement is, *given* an oracle that produces the consistent estimator $\hat{\boldsymbol{G}}(\mathbf{x}, n)$ of $\mathbf{g}(\mathbf{x})$ for each $\mathbf{x} \in \mathcal{X}$, *find* the solution to Problem M, which is the efficient set $\mathcal{E}$.

In Chapter 3 we propose a similar but narrower problem statement.

2. OVERVIEW OF RELATED LITERATURE AND CONCEPTS

In this chapter, we provide context for the MOSO problems and solution strategies considered in this thesis. First, in §2.1, we introduce the RA framework by formalizing the sample-path MOSO problem and providing an example, generic RA algorithm to solve MOSO problems. Second, as MOSO exists at the intersection of deterministic MOO and SOSO, we briefly discuss both along with their solution strategies in §2.2.1 and §2.2.2. MOSO solution methods tend to target problems based on the nature of the feasible set: finite, countable, or uncountable. Thus, we categorize MOSO problems as in Hunter et al. (2019), where we discuss MOSO on finite sets in §2.2.3; MOSO on countable, integer-ordered sets in §2.2.4; and MOSO on uncountable, continuous sets in §2.2.5.

2.1 Retrospective Approximation

Sample average approximation (SAA) is an algorithm framework in which the sample-path version of a MOSO problem is solved at some sample size. SAA is not an algorithm, since it does not specify a routine to solve the sample-path problem itself. The sample-path MOSO problem for a simulation budget m is

$$\text{Problem } \hat{M}_m\text{:} \qquad \underset{\mathbf{x}\in\mathcal{X}}{\text{minimize}} \;\; \bar{\boldsymbol{G}}_m(\mathbf{x}) := (\bar{G}_{1,m}(\mathbf{x}), ..., \bar{G}_{d,m}(\mathbf{x})),$$

where $\bar{G}_{k,m}(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} G_k(\mathbf{x}, \xi_i)$ for every $k \in \{1, ..., d\}$ and $G_k(\mathbf{x}, \xi_i)$ is an observation retrieved from the simulation oracle, and $\bar{G}_{k,m}(\mathbf{x}) \to g_k(\mathbf{x})$ w.p.1 for all k. The sample-path solution to the Problem $\hat{M}_m$ is the estimated efficient set $\hat{\mathcal{E}}$ defined similarly to the efficient set as $\hat{\mathcal{E}} := \{\mathbf{x} \in \mathcal{X} : \nexists \mathbf{x}' \in \mathcal{X} \ni \bar{\boldsymbol{G}}_m(\mathbf{x}') \leq \bar{\boldsymbol{G}}_m(\mathbf{x})\}$.

The RA algorithm framework is a version of SAA in which an algorithm solves a sequence of independently realized sample-path problems, each with increasing sample sizes. The generic algorithm below illustrates the common structure of RA algorithms in the MOSO context. The algorithm uses the sample-path solver SPMOSOSolver to solve a deterministic, multi-objective optimization in each retrospective iteration ν. The sample-path problem is deterministic since the objective values of every visited feasible point within a RA iteration are estimated with a sample size of m_ν. Note that SPMOSOSolver needs to be specified for the algorithm to work. Such algorithms tend to converge as the simulation budget approaches infinity (Pasupathy and Ghosh, 2013) and the convergence rate can be fine-tuned as in Pasupathy (2010).

Algorithm 1: Generic RA framework to solve the MOSO problem

Require:
initial solutions for each objective $\mathbf{x}_0$; a schedule of sample sizes $\{m_\nu\}$ to expend on each visited solution at iteration ν

Ensure: collection of sample-path efficient points $\hat{\mathcal{E}}_\nu$

1: Initialize $\hat{\mathcal{E}}_0 = \mathbf{x}_0$
2: **for** $\nu = 1, 2, \ldots$ **do**
3: $\quad \hat{\mathcal{E}}_\nu = SPMOSOSolver(\hat{\mathcal{E}}_{\nu-1}, m_\nu)$
4: **end for**

Although the ideas for RA appear much earlier, RA frameworks have appeared in the context of stochastic root-finding (Chen and Schmeiser, 2001; Pasupathy and Schmeiser, 2009; Pasupathy, 2010; Pasupathy and Kim, 2011) and have been used in both continuous (Pasupathy, 2010) and integer-ordered contexts (Wang et al., 2013; Nagaraj and Pasupathy, 2016). See Pasupathy and Ghosh (2013) for a complete retrospective on RA.

RA algorithms are designed to be efficient. They "cheaply" obtain solutions in early iterations due the low sample sizes at each visited point. These cheaply acquired solutions are used as the "warm start" to the subsequent RA iteration, which reduces the required sampling effort to acquire another sample-path solution. Since we use

the same sample-size at every visited point, simulation replications can be taken both in parallel and using common random numbers (Law, 2015).

2.2 MOSO

MOSO problems can be thought of as a generalized version of both MOO problems and SOSO problems. First, in generalizing from MOO, MOSO allows objective functions that can only be observed with stochastic error. To adapt MOO solution methods to operate on MOSO problems, the methods require a sampling strategy to account for the error in the objective functions, which increases the computational complexity of the solution method. Second, in generalizing from SOSO, MOSO allows any number of objective functions. To adapt SOSO solution methods to operate on MOSO problems, the methods require a scalarization strategy to convert the multi-objective problem into a sequence of single-objective problems, which increases the computational complexity of the solution method. Thus, MOSO problems tend to be more difficult than MOO or SOSO problems, due to being more computationaly intensive. In constructing a MOSO solution method, we desire to choose efficient components from both the MOO and SOSO literature that can alleviate this computational burden. In the remainder of this section, we discuss solution strategies for both MOO and SOSO, and then existing strategies for MOSO.

2.2.1 Deterministic MOO

MOO solution methods are either scalarization methods or non-scalarization methods. According to Miettinen (1999), scalarization methods are most common and we primarily focus on scalarization methods since we employ the ε-constraint method in Chapter 3. For a discussion of non-scalarization methods, we refer to Wiecek et al. (2016).

Scalarization methods reduce multi-objective problems to a sequence of single-objective problems. The well-known linear weighted sum method is an example of

a scalarization method. In the linear weighted sum method, each objective function $k \in \{1, ..., d\}$ is assigned a weight $w_k \in \mathbb{R}$. Then, each weighted objective function is added to the others to form a single objective function such that the MOO problem $\text{minimize}_{\mathbf{x}\in\mathcal{X}}(g_1(\mathbf{x}), ..., g_d(\mathbf{x}))$ becomes the single-objective optimization problem $\text{minimize}_{\mathbf{x}\in\mathcal{X}} \sum_{k=1}^{d} w_k g_k(\mathbf{x})$. Solving the latter problem with different weights may yield different efficient points.

The ε-constraint method is another scalarization method that reduces the MOO problem into constrained, single-objective objective problems with constraints $(\varepsilon_1, ..., \varepsilon_d)$. We formulate an individual ε-constraint problem as

$$\begin{aligned} \text{Problem } S(k^*, \varepsilon): \quad & \text{minimize}_{\mathbf{x}\in\mathcal{X}} \ \bar{G}_{k^*,n}(\mathbf{x}) \\ & \text{s.t.} \quad g_{k^{\text{con}}}(\mathbf{x}) \leq \varepsilon \text{ for } k^{\text{con}} \in \{1, ..., d\}, k^{\text{con}} \neq k^*. \end{aligned}$$

When Problem $S(k^*, \varepsilon)$ is feasible, its solution is a weakly efficient point (Miettinen, 1999, p.85, Theorem 3.2.1). If a weakly efficient solution is also a solution to a Problem $S(k^*, \varepsilon)$ for every $k \neq k^*$ then the the point is an efficient point (Miettinen, 1999, p.85 Theorem 3.2.2). We must solve d ε-constraint problems to certify that a solution is efficient, and we can generate every efficient point if ε's are chosen carefully. To aid in choosing ε's, it is useful to know the bounds of the efficient set. The *ideal* vector is constructed by minimizing each objective function. The resulting vector $\mathbf{g}^{\text{ideal}}$ is defined as

$$\mathbf{g}^{\text{ideal}} := (\min_{\mathbf{x}\in\mathcal{X}} g_1(\mathbf{x}), \min_{\mathbf{x}\in\mathcal{X}} g_2(\mathbf{x}), ..., \min_{\mathbf{x}\in\mathcal{X}} g_d(\mathbf{x})).$$

The ideal vector represents a lower bound for each objective function. It is the solution to a MOO problem only if none of the objectives are conflicting. To obtain an upper bound on the efficient set, we can attempt to generate the *nadir* vector, defined as

$$\mathbf{g}^{\text{nadir}} := (\max_{\mathbf{x}\in\mathcal{E}} g_1(\mathbf{x}), \max_{\mathbf{x}\in\mathcal{E}} g_2(\mathbf{x}), ..., \max_{\mathbf{x}\in\mathcal{E}} g_d(\mathbf{x})).$$

In bi-objective problems, the nadir is trivially generated from the ideal (Ehrgott and Tenfelde-Podehl, 2003). However, in the presence of more than two objectives it

is difficult, if not impossible, to generate the nadir exactly. Thus, an ε-constraint method for MOO problems with more than two objective functions must rely on approximations of the nadir when attempting to locate the entire efficient set.

2.2.2 SOSO

Like MOSO methods, SOSO methods can be categorized by their feasible sets as in Hong and Nelson (2009) and in Pasupathy and Ghosh (2013). Solution methods for finite feasible sets are called ranking and selection methods. Ranking and selection methods acquire simulation replications from every feasible point. Goldsman (2015) provide an introduction to SOSO ranking and selection methods. Nelson (2010) introduces methods for SOSO problems with integer-ordered feasible sets. For SOSO problems with continuous decision variables, we refer to §5 of Pasupathy and Ghosh (2013).

2.2.3 MOSO on Finite Sets

As in the SOSO context, MOSO problems with finite sets are ranking and selection problems, called multi-objective ranking and selection (MORS) problems. The decision variables in MORS problems may be categorical, and are usually called systems. Methods typically fall into one of two categories, as in Pasupathy and Ghosh (2013): those that guarantee sampling efficiency given a constraint on the simulation budget, called fixed-budget procedures; and those that provide a probabilistic guarantee on the optimality gap of the solution, called fixed-precision procedures. MORS procedures are the most well-developed MOSO solution methods (Hunter et al., 2019). In what follows, we discuss three recent fixed-budget procedures and one fixed-precision procedure.

Arguably, the most popular MORS method is Multi-objective Optimal Computing Budget Allocation (MOCBA; Lee et al., 2010a). MOCBA is a multi-objective extension of OCBA (Chen et al., 2000) and is among the first MOSO methods to retrieve

the entire efficient set. MOCBA provides a simulation allocation rule that attempts to minimize the probability of misclassification, that is, the probability that 1) a system is classified as Pareto despite being dominated or that 2) a system is classified as dominated despite being non-dominated. Due to the difficulty of simultaneously minimizing both types of misclassification, MOCBA uses a heuristic to minimize the largest estimated bound of the two misclassification types.

Sampling Criteria for Optimization using Rate Estimators (SCORE) exactly formulates the simulation allocation of each system that maximizes the decay rate of the probability of misclassification (Applegate et al., 2018; Feldman and Hunter, 2018). The formulation, which accounts for correlation between objectives, is posed as a concave maximization problem where decision variables are the percent of the total simulation budget allocated to each system. Due to the computational difficulty of solving the problem formulation, Applegate et al. (2018) present two approximately optimal allocation schemes: one for "small" problems, perhaps with few objectives; and one for "large" problems. The scheme for large problems assumes independence between objective functions to reduce the computational burden. Applegate et al. (2018) provide a sequential procedure to compute the allocations.

The Myopic Multi-Objective Budget Allocation (M-MOBA, M-MOBA-HV) algorithms uses a Bayesian procedure to repeatedly choose the "best" system from which to obtain additional samples (Branke and Zhang, 2015; Branke et al., 2016). In M-MOBA, the best system is that with the highest probability of changing the efficient set. In M-MOBA-HV, the best system is that which causes the largest change in the expected hypervolume of the Pareto set.

The Generalized Sequential Probability Ratio Test (GSPRT) framework is preliminary work of a fixed-budget procedure that guarantees a Probability of Correct Selection (PCS) of the efficient set (Wang and Wan, 2017). GSPRT uses the generalized likelihood to compute the probability that a system is misclassified.

2.2.4 MOSO on Integer-Ordered Sets

Few methods exist for MOSO on integer-ordered sets where the feasible set is a subset of the integer lattice, $\mathcal{X} \subseteq \mathbb{Z}^q$. Since methods may seek local optimality, we define an integer-ordered, Euclidean neighborhood structure, as in Wang et al. (2013), on which our local optimality holds (see §1.2.2). For $\mathbf{x} \in \mathcal{D} \subseteq \mathbb{Z}^q$ and neighborhood size parameter $a \in \overline{\mathbb{R}}$, $a \geq 1$, define the $\mathcal{N}_a$-neighborhood of a point $\mathbf{x}$ as $\mathcal{N}_a(\mathbf{x}) := \{\mathbf{x}' \in \mathbb{Z}^q : d(\mathbf{x}, \mathbf{x}') \leq a\}$.

The most notable existing algorithm is the Multi-objective Convergent Optimization via Most Promising Area Stochastic Search (MO-COMPASS) algorithm (Li et al., 2015b), a multi-objective extension of the COMPASS algorithm (Hong and Nelson, 2006). MO-COMPASS converges to a local efficient set given an integer-ordered MOSO problem utilizing the *most promising area* concept of the COMPASS algorithm family. The most promising area is a set containing points closer to a point in the estimated efficient set than to any estimated dominated point. MO-COMPASS requires that no two points take the same value on the same objective and takes as input a simulation allocation rule that samples at least once from every "new" point in an iteration. MO-COMPASS operates on any number of objectives.

Multi-objective Probabilistic Branch and Bound (MOPBnB), an extension of PBnB, operates on bounded and ordered feasible sets which may be integer-ordered, continuous, or mixed (Huang and Zabinsky, 2014). In MOPBnB, the "good", not-discarded, subregions contain points with estimated values that are either in the estimated efficient set or near the estimated efficient set. The discarded subregions contain only points estimated to be dominated. The good subregions converge to the global efficient set.

2.2.5 MOSO on Continuous Sets

MOSO problems on continuous sets are MOSO problems with continuous decision variables. Because the efficient set may be uncountable, the goal of continuous MOSO

methods is to generate a characterization of the efficient set. In this context, the few existing algorithms tend to incorporate sample average approximation (SAA, see §2.1) frameworks. Bonnel and Collonge (2014) provide conditions and convergence results for MOSO algorithms in SAA frameworks.

Kim and Ryu (2011b) provide a trust-region algorithm in a SAA framework for bi-objective MOSO. The algorithm seeks a characterization of a local efficient set by repeatedly identifying the most "isolated" estimated efficient point, formulating and solving single-objective problems within a trust region around the isolated point, and incorporating the single-objective solutions into the estimated local efficient set if appropriate.

3. AN EPSILON-CONSTRAINT METHOD FOR INTEGER-ORDERED BI-OBJECTIVE SIMULATION OPTIMIZATION

Recall that we consider the context of multi-objective simulation optimization (MOSO) on integer lattices, that is, nonlinear optimization in which two or more simultaneous objectives can only be observed with error, and each decision variable can only take on integer values. The solution to a MOSO problem is the set of feasible decision points for which no other feasible decision point is at least as good on all objectives and strictly better on at least one objective. Recall that we refer to this set, and the decision points therein, as *efficient*; the image of this set and the points therein are *Pareto*.

We propose a new efficient and provably-convergent algorithm called Retrospective Partitioned Epsilon-constraint with Relaxed Local Enumeration (R-PεRLE, written as R-PERLE when special characters are not allowed) to solve bi-objective SO problems on integer lattices. Our algorithm is a competitor to MO-COMPASS for MOSO with exactly two objectives. R-PεRLE employs three key concepts: a version of SAA called retrospective approximation (RA) for overall algorithmic efficiency (see, e.g., Pasupathy and Ghosh, 2013), the ε-constraint method (see, e.g., Miettinen, 1999) which enables us to find sample-path local efficient points using a pseudo-gradient-based single-objective solver, and relaxed local enumeration (RLE) to certify the solution returned in each RA iteration is, in some sense, sample-path optimal.

To explore each of these key concepts in turn, first, consider RA. Recall from §2.1 that RA is an algorithmic framework that requires solving a sequence of sample-path problems (formulated in §3.3.1) at increasing sample sizes. One *RA iteration* comprises solving a sample-path problem at one sample size. To ensure sampling

efficiency, the solution from the previous RA iteration is used as a warm start in the next RA iteration, which has a higher sample size. Thus large sample sizes are not wasted on suboptimal points in early RA iterations. Instead, they are saved for later RA iterations in which the warm start likely is close to the true solution.

Within each RA iteration, we propose the (deterministic) PεRLE algorithm to solve the sample-path bi-objective problem on an integer lattice. PεRLE comprises two sub-algorithms with the primary goals of pseudogradient-based search and certification, respectively: Pε ('P epsilon') and RLE. First, the Pε algorithm partitions the objective space and solves a set of sample-path ε-constraint problems (defined in §3.5) at carefully-chosen constraint values, denoted by ε. This technique is the ε-constraint method discussed in §2.2.1, is a standard scalarization method for solving multi-objective optimization problems (Miettinen, 1999); we choose the values of ε to make infeasible all regions of the decision space that map to objective vectors closer than a function of the standard error away from known sample-path local Pareto points. We use the single-objective, pseudo-gradient-based SPLINE algorithm (Wang et al., 2013, listed in §3.5.3) to solve each ε-constraint problem. Then, the set of local efficient points found by Pε is sent to the RLE algorithm, which enumerates the neighborhood of the set and crawls to new local efficient points as required, up until it can certify that the set is a sample-path approximate local efficient set (defined in §3.3.2). For sampling efficiency in our RA framework, "completeness" of the local efficient set in RLE also depends on the standard errors of the objective values of the points in the set.

3.1 Contributions

We view the specific contributions of this work as follows:

1. R-PεRLE can solve a wide class of bi-objective SO problems having integer-valued decision variables and deterministic constraints. The importance of developing algorithms for this class of problems is demonstrated by the abundance

of so-called integer-ordered problems on the `simopt.org` website (Henderson and Pasupathy, 2018), and by the wide variety of application areas in which bi-objective SO problems arise (Hunter et al., 2019).

2. R-PεRLE, which shows promising numerical performance, adapts the ε-constraint method for use with bi-objective SO problems in an algorithmically efficient way. First, since an RA framework prescribes obtaining the same number of simulation replications at every point within an RA iteration, the required simulation replications can be obtained in parallel with common random numbers (CRN, see, e.g., Law, 2015). Solving sample-path ε-constraint problems inside Pε can also be completed in parallel, and in the limit, the number of ε-constraint problems solved in each RA iteration corresponds to the cardinality of the local efficient set, which we assume is finite. Further, the Pε algorithm employs the pseudo-gradient-based SPLINE algorithm to quickly locate local efficient points as the solution to each ε-constraint problem. Finally, both Pε and RLE employ relaxations to ensure we only solve each sample-path problem to an error tolerance commensurate with our sampling error. The relaxations inside Pε ensure that at the end of each RA iteration, we return an "even" approximation of the local Pareto set, where the granularity is a function of the standard errors of the sample-path local Pareto points found.

3. While we propose R-PεRLE for mainstream use, we also propose and discuss the convergence properties of two other RA algorithms, R-Pε and R-MinRLE.

   (a) The R-Pε algorithm is identical to R-PεRLE, except that RLE is never invoked. To show that Pε usually provides good starting points to RLE, we demonstrate that R-Pε converges under regularity conditions that are more restrictive than those required for the convergence of R-PεRLE.

   (b) The R-MinRLE algorithm is identical to R-PεRLE, except that the Pε algorithm is replaced by the GetMin algorithm. The GetMin algorithm brings up the sample sizes of all estimated efficient points from the last

RA iteration, updates the sample-path local minimizers on each objective, and removes sample-path dominated points. Then, it invokes RLE. The R-MinRLE algorithm can solve problems with two or more objectives. We view R-MinRLE as a somewhat naïve, provably convergent, pseudogradient-based benchmark algorithm for MOSO problems on integer lattices with two or more objectives. Since the proofs of convergence for R-PεRLE and R-MinRLE rely only on having invoked RLE, Pε can be considered an "accelerator" for RLE in two objectives that out-performs the naïve accelerator, the GetMin algorithm. In the future, other accelerators can be developed for RLE, where the convergence guarantee is provided by this paper.

3.2 Problem Context: Preliminaries for MOSO on Integer Lattices

Recall that we formulate the MOSO problem on integer lattices with d simultaneous objectives as

Problem M_d:

$$\text{minimize}_{\mathbf{x}\in\mathcal{X}} \ \{\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_d(\mathbf{x})) := (\mathbb{E}[G_1(\mathbf{x}, \xi)], \ldots, \mathbb{E}[G_d(\mathbf{x}, \xi)])\},$$

where $\mathbf{g}\colon \mathcal{X} \to \mathbb{R}^d$ is an unknown vector-valued function and the nonempty feasible set $\mathcal{X} \subseteq \mathbb{Z}^q$ is a subset of an integer lattice. Any deterministic constraints present in Problem M_d are defined through the feasible set $\mathcal{X}$. In what follows, we define optimality concepts for Problem M_d and provide a formal problem statement. Although the focus of R-PεRLE is the bi-objective case of $d = 2$, we retain the generality of d objectives whenever possible because our benchmark algorithm, R-MinRLE, is defined for $d \geq 2$ objectives.

3.2.1 Optimality Concepts

Since our focus is on local optimality, we define a flexible neighborhood structure on which our local optimality holds. For $\mathbf{x} \in \mathcal{D} \subseteq \mathbb{Z}^q$ and neighborhood size parameter $a \in \overline{\mathbb{R}}$, $a \geq 1$, define the $\mathcal{N}_a$-neighborhood of a point $\mathbf{x}$ as $\mathcal{N}_a(\mathbf{x}) := \{\mathbf{x}' \in \mathbb{Z}^q : d(\mathbf{x}, \mathbf{x}') \leq a\}$. Further define the $\mathcal{N}_a$-neighborhood of a set as the union of the $\mathcal{N}_a$-neighborhoods of all the points belonging to the set. That is, for $\mathcal{S} \subset \mathcal{D} \subseteq \mathbb{Z}^q$, the $\mathcal{N}_a$-neighborhood of set $\mathcal{S}$ is $\mathcal{N}_a(\mathcal{S}) := \cup_{\mathbf{x}\in\mathcal{S}} \mathcal{N}_a(\mathbf{x})$. Then for any set $\mathcal{A}$, define $\mathcal{N}'_a(\mathcal{A}) := \mathcal{N}_a(\mathcal{A}) \setminus \mathcal{A}$ as the *deleted neighborhood* of $\mathcal{A}$.

Minimizers and Efficient Points

Following Wang et al. (2013), for each objective $k \in \{1, \ldots, d\}$, we define local minimizers of the kth objective function as follows.

Definition 6 (Wang et al., 2013)**.** *Given an objective function $g_k : \mathcal{X} \to \mathbb{R}$, a point $\mathbf{x}_k^{\min} \in \mathcal{X}$ is an* $\mathcal{N}_a$-local minimizer *of g_k if $g_k(\mathbf{x}_k^{\min}) \leq g_k(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{N}_a(\mathbf{x}_k^{\min}) \cap \mathcal{X}$.*

Given our definition of an $\mathcal{N}_a$-neighborhood, for consistency with our notation in this chapter, we re-define and expand upon our previous definitions of local efficient and local weakly efficient points that appear in §1.2.2.

Definition 7. *A point $\mathbf{x}^{\mathrm{w}} \in \mathcal{X}$ is an* $\mathcal{N}_a$-local weakly efficient point *(LWEP) if*

1. *$\nexists \mathbf{x} \in \mathcal{N}_a(\mathbf{x}^{\mathrm{w}}) \cap \mathcal{X}$ such that $\mathbf{g}(\mathbf{x}) < \mathbf{g}(\mathbf{x}^{\mathrm{w}})$; or equivalently, if*
2. *$\forall \mathbf{x} \in \mathcal{N}_a(\mathbf{x}^{\mathrm{w}}) \cap \mathcal{X}$, $\mathbf{g}(\mathbf{x}) \not< \mathbf{g}(\mathbf{x}^{\mathrm{w}})$, that is, $\exists k \in \{1, \ldots, d\}$ such that $g_k(\mathbf{x}^{\mathrm{w}}) \leq g_k(\mathbf{x})$.*

Definition 8. *A point $\mathbf{x}^* \in \mathcal{X}$ is an* $\mathcal{N}_a$-local efficient point *(LEP) if*

1. *$\nexists \mathbf{x} \in \mathcal{N}_a(\mathbf{x}^*) \cap \mathcal{X}$ such that $\mathbf{g}(\mathbf{x}) \leq \mathbf{g}(\mathbf{x}^*)$; or, equivalently, if*
2. *$\forall \mathbf{x} \in \mathcal{N}_a(\mathbf{x}^*) \cap \mathcal{X}$, $\mathbf{g}(\mathbf{x}) \not\leq \mathbf{g}(\mathbf{x}^*)$, that is, one of the following holds: (a) $\exists k \in \{1, \ldots, d\}$ such that $g_k(\mathbf{x}^*) < g_k(\mathbf{x})$, or (b) $\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{x}^*)$.*

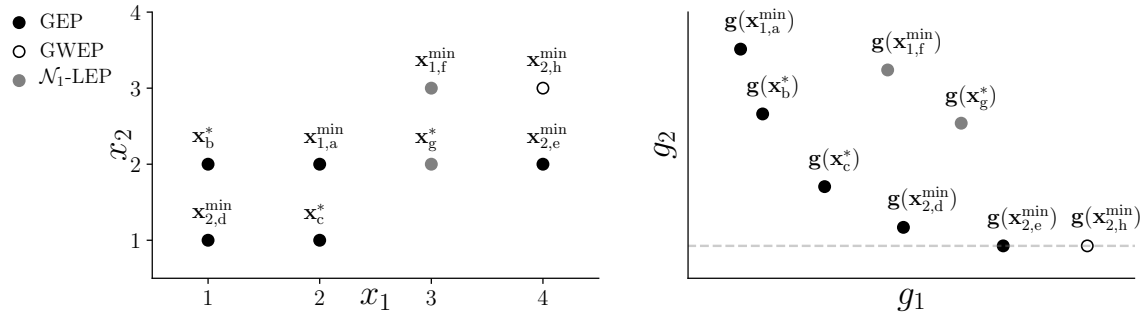

Figure 3.1. The figure shows an example feasible space (left) and the image of the feasible space (right). $\mathcal{N}_1$-local minimizers are denoted by the superscript min. The set $\mathcal{L}_1 = \{\mathbf{x}_{1,\mathrm{a}}^{\min}, \mathbf{x}_{\mathrm{b}}^*, \mathbf{x}_{\mathrm{c}}^*, \mathbf{x}_{2,\mathrm{d}}^{\min}\}$ is an $\mathcal{N}_1$-LES. The set $\mathcal{E} = \{\mathbf{x}_{1,\mathrm{a}}^{\min}, \mathbf{x}_{\mathrm{b}}^*, \mathbf{x}_{\mathrm{c}}^*, \mathbf{x}_{2,\mathrm{d}}^{\min}, \mathbf{x}_{2,\mathrm{e}}^{\min}\}$ is the GES. The set $\mathcal{E}^{\mathrm{w}} = \mathcal{E} \cup \{\mathbf{x}_{2,\mathrm{h}}^{\min}\}$ is the GWES. The point $\mathbf{x}_{\mathrm{g}}^*$ is an $\mathcal{N}_1$-LEP that does not belong to an $\mathcal{N}_1$-LES or an $\mathcal{N}_1$-LWES.

Notice that every $\mathcal{N}_a$-local minimizer of some objective g_k, $k \in \{1, \ldots, d\}$, is an $\mathcal{N}_a$-LWEP, and if the $\mathcal{N}_a$-local minimizer is the unique minimum in its neighborhood, then it is also an $\mathcal{N}_a$-LEP. Further, all $\mathcal{N}_a$-LEP's are $\mathcal{N}_a$-LWEP's. We define a *global minimizer*, a *global weakly efficient point* (GWEP), and a *global efficient point* (GEP) as an $\mathcal{N}_a$-local minimizer, $\mathcal{N}_a$-LWEP, and $\mathcal{N}_a$-LEP, respectively, in which we set the neighborhood size parameter $a = \infty$.

Efficient and Pareto Sets

We collect the various types of efficient points defined in the previous section into various types of efficient sets, updated from §1.2.2 using our notation in this chapter..

Definition 9. *A set* $\mathcal{L}^{\mathrm{w}} \subseteq \mathcal{X}, |\mathcal{L}^{\mathrm{w}}| \geq 1$, *is an* $\mathcal{N}_a$-local weakly efficient set *(LWES) if (a)* $\forall \mathbf{x}^{\mathrm{w}} \in \mathcal{L}^{\mathrm{w}}$, $\mathbf{x}^{\mathrm{w}}$ *is an* $\mathcal{N}_a$*-LWEP, and (b) no points in* $\mathbf{g}(\mathcal{L}^{\mathrm{w}})$ *strictly dominate other points in* $\mathbf{g}(\mathcal{L}^{\mathrm{w}})$*, and (c)* $\forall \mathbf{x} \in \mathcal{N}_a'(\mathcal{L}^{\mathrm{w}}) \cap \mathcal{X}$, $\exists \mathbf{x}^{\mathrm{w}} \in \mathcal{L}^{\mathrm{w}}$ *such that* $\mathbf{g}(\mathbf{x}^{\mathrm{w}}) \leqq \mathbf{g}(\mathbf{x})$.

Definition 10. *A set* $\mathcal{L} \subseteq \mathcal{X}, |\mathcal{L}| \geq 1$, *is an* $\mathcal{N}_a$-local efficient set *(LES) if (a)* $\forall \mathbf{x}^* \in \mathcal{L}$, $\mathbf{x}^*$ *is an* $\mathcal{N}_a$*-LEP, (b) no points in* $\mathbf{g}(\mathcal{L})$ *dominate other points in* $\mathbf{g}(\mathcal{L})$*, and (c)* $\forall \mathbf{x} \in \mathcal{N}_a'(\mathcal{L}) \cap \mathcal{X}$, $\exists \mathbf{x}^* \in \mathcal{L}$ *such that* $\mathbf{g}(\mathbf{x}^*) \leq \mathbf{g}(\mathbf{x})$.

Notice that every $\mathcal{N}_a$-LES is also an $\mathcal{N}_a$-LWES. Finally, we define the *global weakly efficient set* (GWES), denoted $\mathcal{E}^{\mathrm{w}}$, and the *global efficient set* (GES), denoted $\mathcal{E}$, as an $\mathcal{N}_a$-LWES and $\mathcal{N}_a$-LES, respectively, in which the neighborhood size is $a = \infty$. Although our definitions exist primarily in the decision space so far, we also define a $\mathcal{N}_a$-local Pareto set (LPS) as the image of an $\mathcal{N}_a$-LES, $\mathbf{g}(\mathcal{L})$.

We remark here that under our definitions, there may exist $\mathcal{N}_a$-LWEP's that do not belong to an $\mathcal{N}_a$-LWES. To see an example of such a case, consider Figure 3.1, and notice that $\mathbf{g}(\mathbf{x}^*_{\mathrm{g}})$ is not dominated by the image of any points in the $\mathcal{N}_1$-neighborhood of $\mathbf{x}^*_{\mathrm{g}}$, which are the points $\mathbf{g}(\mathbf{x}^{\min}_{1,\mathrm{a}}), \mathbf{g}(\mathbf{x}^{\min}_{2,\mathrm{e}})$, and $\mathbf{g}(\mathbf{x}^{\min}_{1,\mathrm{f}})$. Therefore, $\mathbf{x}^*_{\mathrm{g}}$ is an $\mathcal{N}_1$-LWEP. (It is also an $\mathcal{N}_1$-LEP.) However, $\{\mathbf{x}^*_{\mathrm{g}}\}$ is not an $\mathcal{N}_1$-LWES because $\mathbf{g}(\mathbf{x}^*_{\mathrm{g}})$ does not dominate $\mathbf{g}(\mathbf{x}^{\min}_{1,\mathrm{a}}), \mathbf{g}(\mathbf{x}^{\min}_{2,\mathrm{e}})$, or $\mathbf{g}(\mathbf{x}^{\min}_{1,\mathrm{f}})$. In this example, it is not possible to construct an $\mathcal{N}_1$-LWES using only the points $\mathbf{x}^{\min}_{1,\mathrm{f}}, \mathbf{x}^*_{\mathrm{g}}$, and $\mathbf{x}^{\min}_{2,\mathrm{h}}$ because the images of these points do not dominate the images of any other feasible points, and therefore cannot dominate the images of the points in their deleted $\mathcal{N}_1$-neighborhood. Thus any $\mathcal{N}_1$-LWES including $\mathbf{x}^*_{\mathrm{g}}$ must also include a member of the GES whose image does not dominate its image, such as $\mathbf{x}^{\min}_{1,\mathrm{a}}, \mathbf{x}^*_{\mathrm{b}}$, or $\mathbf{x}^{\min}_{2,\mathrm{e}}$. But, including any of these points in the candidate $\mathcal{N}_1$-LWES with $\mathbf{x}^*_{\mathrm{g}}$ implies that there exist neighborhood points that violate the definition of an $\mathcal{N}_1$-LWES. Thus $\mathbf{x}^*_{\mathrm{g}}$ does not belong to an $\mathcal{N}_1$-LWES.

Also, to help the reader, we provide acronyms in Table 3.2.1.

3.2.2 Problem Statement

Using the optimality concepts defined in the previous section, we consider the following problem statement: Given a neighborhood size a and an oracle capable of producing estimators $\bar{\boldsymbol{G}}_n(\mathbf{x})$ of $\mathbf{g}(\mathbf{x})$ such that $\bar{\boldsymbol{G}}_n(\mathbf{x}) \to \mathbf{g}(\mathbf{x})$ w.p.1 as the sampling effort $n \to \infty$ for each $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{Z}^q$, find a local solution to Problem M_d, which is an $\mathcal{N}_a$-LES.

Table 3.1.
The table lists acronyms used throughout the chapter (Cooper et al., 2018).

Acronym	Description	Defined in Chapter
SO	Simulation Optimization	1
MOSO	Multi-Objective Simulation Optimization	1
SAA	Sample Average Approximation	2.1, 3.3
RA	Retrospective Approximation	2.1, 3.3
CRN	Common Random Numbers	3.1
Pε	Partitioned Epsilon (algorithm)	3.5
RLE	Relaxed Local Enumeration (algorithm)	3.6
LWEP / LEP	Local Weakly Efficient Point / Local Efficient Point	3.2.1
GWEP / GEP	Global Weakly Efficient Point / Global Efficient Point	3.2.1
LWES / LES	Local Weakly Efficient Set / Local Efficient Set	3.2.1
GWES / GES	Global Weakly Efficient Set / Global Efficient Set	3.2.1
LPS	Local Pareto Set	3.2.1
ALES	sample-path Approximate $\mathcal{N}_a$-Local Efficient Set	3.3.2
NCN	sample-path Non-Conforming Neighborhood	3.6.1

3.3 Solution Context: Retrospective Approximation

To address our problem statement, we employ an RA framework, which is a version of SAA. First, we discuss the sample-path problem and solution in the context of SAA, as well as its use inside an RA framework. Then, we discuss an approximate sample-path solution designed to enhance the efficiency of our RA framework for MOSO on integer lattices.

3.3.1 The Sample-Path Problem and Solution

Recall from §2.1 SAA is a solution framework for Problem M_d defined by the sample-path problem

Problem $\overline{M}_{d,n}$:

$$\text{minimize}_{\mathbf{x}\in\mathcal{X}} \left\{\bar{\boldsymbol{G}}_n(\mathbf{x}) = (\bar{G}_{1,n}(\mathbf{x}), \ldots, \bar{G}_{d,n}(\mathbf{x})) := \left(\tfrac{1}{n}\sum_{i=1}^{n} G_1(\mathbf{x},\xi_i), \ldots, \tfrac{1}{n}\sum_{i=1}^{n} G_d(\mathbf{x},\xi_i)\right)\right\},$$

where $\bar{\boldsymbol{G}}_n(\mathbf{x})$ is an estimator of $\mathbf{g}(\mathbf{x})$, and at each feasible point $\mathbf{x} \in \mathcal{X}$ the oracle generates n copies of the random objective vector $\boldsymbol{G}(\mathbf{x},\xi_i) := (G_1(\mathbf{x},\xi_i), \ldots, G_d(\mathbf{x},\xi_i))$ for all $i = 1, \ldots, n$. For fixed values of the random variables ξ_i, $i = 1, \ldots, n$, solving the sample-path Problem $\overline{M}_{d,n}$ is a deterministic optimization problem. Thus we define *sample-path* versions of all optimality concepts in §3.2.1 by replacing the objective function values $\mathbf{g}(\mathbf{x})$ and $g_k(\mathbf{x})$ with $\bar{\boldsymbol{G}}_n(\mathbf{x})$ and $\bar{G}_{k,n}(\mathbf{x})$, respectively, for all $k \in \{1, \ldots, d\}$. We denote sample-path $\mathcal{N}_a$-local minimizers, sample-path $\mathcal{N}_a$-LWEP's, and sample-path $\mathcal{N}_a$-LEP's as $\boldsymbol{X}^{\min}_{k,n}$, $\boldsymbol{X}^{\mathrm{w}}_n$, and $\boldsymbol{X}^{*}_n$, respectively. A local solution to Problem $\overline{M}_{d,n}$ is a sample-path $\mathcal{N}_a$-LES.

Recall that in RA, instead of solving Problem $\overline{M}_{d,n}$ at a static sample size n, we solve a sequence of sample-path problems. These sample-path problems are denoted Problem $\overline{M}_{d,m_\nu}$ at sample size m_ν, where $\{m_\nu, \nu = 1, 2, \ldots\}$ is a sequence of increasing sample sizes and ν is the RA iteration number. The solution to Problem $\overline{M}_{d,m_{\nu-1}}$ is used as a warm start to find the solution to Problem $\overline{M}_{d,m_\nu}$. (Henceforth, within an RA iteration ν, we usually denote the sample size as $n = m_\nu$ to reduce our use of double subscripts.)

3.3.2 An Approximate Sample-Path Solution

Assuming the sample-path Problem $\overline{M}_{d,n}$ solved within an RA iteration is well-behaved, one can solve it to optimality by locating a complete sample-path $\mathcal{N}_a$-LES. However, when sample sizes are small, there may be a relatively large error in estimating the objective vectors. Thus it may be inefficient to chase down all members

of a sample-path $\mathcal{N}_a$-LES if the points are unlikely to be true $\mathcal{N}_a$-LES members — for a convergent algorithm, non-$\mathcal{N}_a$-LES members eventually will be eliminated from consideration in a future RA iteration. Instead, imagine that we solve the sample-path problem only to within a certain error tolerance that is commensurate with the amount of error we have in estimating the objective function values (see, e.g., Pasupathy, 2010, for similar concepts in the stochastic root-finding context). To employ such a concept, we would like a relaxed definition of a sample-path $\mathcal{N}_a$-LES that will enable us to stop our search for a sample-path $\mathcal{N}_a$-LES within an RA iteration early. To this end, we define an approximate version of local optimality for Problem $\overline{M}_{d,n}$, as follows.

Definition 11. *A set $\mathcal{A} \subseteq \mathcal{X}$ is a* sample-path approximate $\mathcal{N}_a$-LES *(ALES) for the sample-path Problem $\overline{M}_{d,n}$ if no points in $\bar{\boldsymbol{G}}_n(\mathcal{A})$ dominate other points in $\bar{\boldsymbol{G}}_n(\mathcal{A})$ and, given a vector-valued* completeness function $\boldsymbol{\delta}\colon \mathcal{X} \to \overline{\mathbb{R}}^d$ *such that $\mathbf{0}_d \leqq \boldsymbol{\delta}(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$,*

(a) $\forall \mathbf{x}^{\mathrm{w}} \in \mathcal{A}$, $\mathbf{x}^{\mathrm{w}}$ is a sample-path $\mathcal{N}_a$-LWEP ($\forall \mathbf{x} \in \mathcal{N}_a(\mathbf{x}^{\mathrm{w}}) \cap \mathcal{X}$, $\bar{\boldsymbol{G}}_n(\mathbf{x}) \not\leq \bar{\boldsymbol{G}}_n(\mathbf{x}^{\mathrm{w}})$), and

(b) for each $\mathbf{x} \in \mathcal{N}_a'(\mathcal{A}) \cap \mathcal{X}$, (i) $\exists \mathbf{x}^{\mathrm{w}} \in \mathcal{A}$ such that $\bar{\boldsymbol{G}}_n(\mathbf{x}^{\mathrm{w}}) \leqq \bar{\boldsymbol{G}}_n(\mathbf{x})$, or (ii) $\exists \mathbf{x}^{\mathrm{w}} \in \mathcal{A}$ such that ($\bar{\boldsymbol{G}}_n(\mathbf{x}) \leq \bar{\boldsymbol{G}}_n(\mathbf{x}^{\mathrm{w}})$ and $\bar{\boldsymbol{G}}_n(\mathbf{x}^{\mathrm{w}}) - \boldsymbol{\delta}(\mathbf{x}^{\mathrm{w}}) \leqq \bar{\boldsymbol{G}}_n(\mathbf{x}) + \boldsymbol{\delta}(\mathbf{x})$), or (iii) $\forall \mathbf{x}^{\mathrm{w}} \in \mathcal{A}$, $\bar{\boldsymbol{G}}_n(\mathbf{x}) \not\leqq \bar{\boldsymbol{G}}_n(\mathbf{x}^{\mathrm{w}})$, and $\exists \tilde{\mathbf{x}}^{\mathrm{w}} \in \mathcal{A}$ such that $\bar{\boldsymbol{G}}_n(\tilde{\mathbf{x}}^{\mathrm{w}}) - \boldsymbol{\delta}(\tilde{\mathbf{x}}^{\mathrm{w}}) \leqq \bar{\boldsymbol{G}}_n(\mathbf{x}) + \boldsymbol{\delta}(\mathbf{x})$ or $\bar{\boldsymbol{G}}_n(\mathbf{x}) - \boldsymbol{\delta}(\mathbf{x}) \leqq \bar{\boldsymbol{G}}_n(\tilde{\mathbf{x}}^{\mathrm{w}}) + \boldsymbol{\delta}(\tilde{\mathbf{x}}^{\mathrm{w}})$.

Definition 11 is similar to the definition of a sample-path $\mathcal{N}_a$-LWES, except for Part (b). Definition 11(b) requires that all feasible points in the deleted neighborhood of the ALES are either (i) weakly dominated by a point in the set, or (ii) dominate a point in the set by less than a certain amount, or (iii) do not weakly dominate any points in the set, and would either weakly dominate or be weakly dominated by a point in the set if both were moved by a certain amount. The "certain amount" is specified by the function $\boldsymbol{\delta}$. We call $\boldsymbol{\delta}(\cdot) = (\delta_1(\cdot), \ldots, \delta_d(\cdot))$ the completeness function because it allows the ALES to have neighborhood points that violate the definition of a sample-path $\mathcal{N}_a$-LWES, and bigger values of $\boldsymbol{\delta}$ result in a "less-complete" ALES.

If $\delta_k(\mathbf{x}) = \infty$ for all $\mathbf{x} \in \mathcal{X}, k \in \{1, \ldots, d\}$, the ALES is a collection of sample-path $\mathcal{N}_a$-LWEP's that may or may not belong to a sample-path $\mathcal{N}_a$-LWES. If $\boldsymbol{\delta}(\mathbf{x}) = \mathbf{0}_d$ for all $\mathbf{x} \in \mathcal{X}$, the ALES is a sample-path $\mathcal{N}_a$-LWES.

We set the completeness function using the estimated standard errors of the objective function values $\bar{G}_{k,n}(\mathbf{x})$ for all $k \in \{1, \ldots, d\}$, $\mathbf{x} \in \mathcal{X}$. Thus we require the following definitions. For all $\mathbf{x} \in \mathcal{X}$, $k \in \{1, \ldots, d\}$, define the variance $\sigma_k^2(\mathbf{x}) := \mathbb{V}(G_k(\mathbf{x}, \xi)) < \infty$. (The assumption of finite variances is made formal in §3.8.) Then, let the estimated standard deviation of the kth objective value at $\mathbf{x} \in \mathcal{X}$ be $\hat{\sigma}_{k,n}(\mathbf{x}) := \sqrt{(n-1)^{-1} \sum_{i=1}^{n} (G_k(\mathbf{x}, \xi_i) - \bar{G}_{k,n}(\mathbf{x}))^2}$, and let the standard error of the kth estimated objective value be $\widehat{\text{s.e.}}(\bar{G}_{k,n}(\mathbf{x})) := \hat{\sigma}_{k,n}(\mathbf{x})/n^{1/2}$. Further, for each objective $k \in \{1, \ldots, d\}$ and for all $\mathbf{x} \in \mathcal{X}$, $\beta \in (0, \infty]$, define the function $\hat{f}_k(\mathbf{x}, \beta) := \widehat{\text{s.e.}}(\bar{G}_{k,n}(\mathbf{x}))(n^{1/2-\beta})$ if $\beta \in (0, \infty)$ and $\hat{f}_k(\mathbf{x}, \infty) := 0$. Let $\hat{\mathbf{f}}(\mathbf{x}, \beta) := (\hat{f}_1(\mathbf{x}, \beta), \ldots, \hat{f}_d(\mathbf{x}, \beta))$.

For the remainder of this thesis, we consider the ALES completeness function specified by $\hat{\mathbf{f}}$. Since the value of this function is random for each $\mathbf{x} \in \mathcal{X}$, henceforth, we denote the completeness function as $\hat{\boldsymbol{\delta}}(\mathbf{x}) := \hat{\mathbf{f}}(\mathbf{x}, \beta_\delta)$ for all $\mathbf{x} \in \mathcal{X}$, where $\beta_\delta \in (0, \infty]$ is the *completeness parameter*. Smaller values of β_δ correspond to larger values of $\hat{\boldsymbol{\delta}}(\mathbf{x})$, thus specifying a less-complete ALES. The value $\beta_\delta = \infty$ implies that the ALES is a sample-path $\mathcal{N}_a$-LWES. Since the completeness function is a function of the standard error, assuming the variances are finite, the value $\hat{\boldsymbol{\delta}}(\mathbf{x}) \to 0$ w.p.1 for each $\mathbf{x} \in \mathcal{X}$ as the sampling effort increases.

3.4 The Main Algorithm: R-PεRLE for Two Objectives

We are now ready to provide an overview of our main RA algorithm, R-PεRLE, which is listed in Algorithm 2. As previously discussed, R-PεRLE employs an RA framework that solves the sequence of sample-path Problems $\overline{M}_{2,m_\nu}$ at increasing sample sizes $\{m_\nu, \nu = 1, 2, \ldots\}$, where ν is the RA iteration number. Within an RA iteration, we propose the deterministic PεRLE algorithm as the sample-path solver.

For compactness, we implicitly define the PεRLE algorithm as calling Pε followed by RLE within one RA iteration ν (Algorithm 2 Steps 3 and 4). The solution to Problem $\overline{M}_{2,m_\nu}$ found by PεRLE on the νth RA iteration, denoted $\hat{\mathcal{A}}_\nu$ in Algorithm 2, is guaranteed by RLE to be an ALES. For efficiency, R-PεRLE uses the sample-path solution from the preceding RA iteration, $\hat{\mathcal{A}}_{\nu-1}$ which is an ALES for Problem $\overline{M}_{2,m_{\nu-1}}$, as an initial set of points for finding an ALES that solves Problem $\overline{M}_{2,m_\nu}$. Given the amount of detail inherent in the algorithms Pε and RLE, we address these algorithms separately in §3.5 and §3.6, respectively.

Algorithm 2: The R-PεRLE Algorithm for $d = 2$

Input: initial point $\mathbf{x}_0 \in \mathcal{X}$; sequence of sample sizes to expend at each visited point, $\{m_\nu\}$; sequence of limits on oracle calls during search, $\{b_\nu\}$; ε-placement and ALES parameters, $\boldsymbol{\beta} = (\beta_\varepsilon, \beta_\delta)$

1 Initialize: $\hat{\mathcal{A}}_0 = \{\mathbf{x}_0\}$ and set $\mathbf{x}_0$ as a global variable
2 **for** $\nu = 1, 2, \ldots$ *with CRN* **do**
3 $\quad \hat{\mathcal{A}}_\nu = \text{P}\varepsilon(\hat{\mathcal{A}}_{\nu-1}, m_\nu, b_\nu, \beta_\varepsilon)$ `/partition and solve ε-constraint problems`
4 $\quad \hat{\mathcal{A}}_\nu = \text{RLE}(\hat{\mathcal{A}}_\nu, m_\nu, b_\nu, \beta_\delta)$ `/guarantee the returned set is an ALES`

R-PεRLE requires a few input parameters, in addition to an initial feasible point $\mathbf{x}_0 \in \mathcal{X}$ and a sequence of sample sizes $\{m_\nu, \nu = 1, 2, \ldots\}$. First, R-P$\varepsilon$RLE requires a sequence of limits on oracle calls during search, $\{b_\nu, \nu = 1, 2, \ldots\}$, that prevents chase-offs in the case of "bad" sample-path realizations of Problem $\overline{M}_{2,m_\nu}$. We set this sequence so that for large enough RA iteration numbers ν, search "time outs" due to binding b_ν do not occur w.p.1. (Such a sequence is also required by SPLINE in §3.5.3.) R-PεRLE also requires parameters $\boldsymbol{\beta} = (\beta_\varepsilon, \beta_\delta)$, which essentially control the completeness of the ALES and are discussed in the sections that follow. We suppress the choice of neighborhood size, which is $a = 1$ by default. The convergence properties of R-PεRLE under different parameter values is discussed in §3.8, and we specify default settings for these parameters in §3.9. *Under the default settings, the initial feasible point $\mathbf{x}_0 \in \mathcal{X}$ is the only required user-specified input parameter for* R-PεRLE.

3.5 The Pε Algorithm for Two Objectives

The Pε algorithm is the first algorithm in our proposed two-part sample-path solver, PεRLE. In RA iteration ν, the Pε algorithm uses the ε-constraint method to find a collection of points that are sample-path $\mathcal{N}_a$-LWEP's for Problem $\overline{M}_{2,m_\nu}$; recall that $n = m_\nu$ is the current sample size inside RA iteration ν. Using the ε-constraint method, we re-formulate the sample-path Problem $\overline{M}_{2,n}$ into a set of constrained single-objective problems. Given an objective to minimize $k^* \in \{1, 2\}$ and an ε value, the sample-path ε-constraint problem is

Problem $\bar{S}_{2,n}(k^*, \varepsilon)$:

$$\text{minimize}_{\mathbf{x}\in\mathcal{X}}\ \bar{G}_{k^*,n}(\mathbf{x}) \quad \text{s.t.} \quad \bar{G}_{k^{\text{con}},n}(\mathbf{x}) \leq \varepsilon \text{ for } k^{\text{con}} \in \{1, 2\}, k^{\text{con}} \neq k^*.$$

On an integer lattice, the sample-path $\mathcal{N}_a$-local minimizer for Problem $\bar{S}_{2,n}(k^*, \varepsilon)$ is guaranteed to be an $\mathcal{N}_a$-LWEP for the original sample-path Problem $\overline{M}_{2,n}$ (see, e.g., Miettinen, 1999, for the continuous context). Except in pathological cases, varying the ε values and solving each resulting sample-path ε-constraint problem results in locating multiple sample-path $\mathcal{N}_a$-LWEP's for Problem $\overline{M}_{2,n}$.

The Pε algorithm operates as follows. First, it finds sample-path $\mathcal{N}_a$-local minimizers on each objective to bound the space where ε values are placed. Then it selects an objective k^* to minimize and solves a collection of Problems $\bar{S}_{2,n}(k^*, \varepsilon)$ at a set of carefully-placed ε values. We use the $\hat{\mathbf{f}}$ function defined in §3.3.2 to place the ε values a function of the standard error away from the images of known sample-path $\mathcal{N}_a$-LWEP's. Such known sample-path $\mathcal{N}_a$-LWEP's may have been carried forward from the previous RA iteration, or may have been found during the current RA iteration. Thus Pε yields a collection of sample-path $\mathcal{N}_a$-LWEP's that includes a local minimizer on each objective and the sample-path $\mathcal{N}_a$-LWEP's that result from solving Problem $\bar{S}_{2,n}(k^*, \varepsilon)$ for each chosen ε.

Algorithm 3: $\hat{\mathcal{A}}_{\text{new}} = \text{P}\varepsilon(\hat{\mathcal{A}}_{\text{old}}, n, b, \beta_\varepsilon)$

Input: estimated efficient set from the last RA iteration, $\hat{\mathcal{A}}_{\text{old}} \subseteq \mathcal{X}$; sample size, n; limit on oracle calls during search, b; epsilon placement parameter, β_ε

Output: $\hat{\mathcal{A}}_{\text{new}} \subseteq \mathcal{X}$, a collection of sample-path $\mathcal{N}_a$-LWEP's

1 $\hat{\mathcal{A}}_n^0 = \text{GetMin}(\hat{\mathcal{A}}_{\text{old}} \cup \{\mathbf{x}_0\}, n, b)$ /search: update sample-path $\mathcal{N}_a$-local minimizers

2 $[\sim, \hat{\mathcal{A}}_n^{\text{w}}, \sim] = \text{RemoveNonLWEP}(\hat{\mathcal{A}}_n^0)$ /get the set of sample-path $\mathcal{N}_a$-LWEP's in $\hat{\mathcal{A}}_n^0$

3 **if** $\hat{\mathcal{A}}_n^{\text{w}} = \emptyset$ **then** $\hat{\mathcal{A}}_n^{\text{w}} \leftarrow \hat{\mathcal{A}}_n^0$ /GetMin update timed out, no other sample-path $\mathcal{N}_a$-LWEP's exist

4 Initialize: $c^0 \leftarrow |\hat{\mathcal{A}}_n^{\text{w}}|$ /set ε's using sample-path $\mathcal{N}_a$-LWEP's

5 **for** $k^* = 1, 2$ **do** /determine objective to minimize, k^*

6 Initialize: $k^{\text{con}} \leftarrow k$ for $k \in \{1, 2\}$ such that $k \neq k^*$

7 Sort $\bar{\boldsymbol{G}}_n(\hat{\mathcal{A}}_n^{\text{w}})$ on k^{con} to get $(\boldsymbol{X}_{(1)}^{\text{w}}, \ldots, \boldsymbol{X}_{(c^0)}^{\text{w}})$ where $\bar{G}_{k^{\text{con}},n}(\boldsymbol{X}_{(1)}^{\text{w}}) \leq \ldots \leq \bar{G}_{k^{\text{con}},n}(\boldsymbol{X}_{(c^0)}^{\text{w}})$

8 Set constraint lower bound $L_{k^*} \leftarrow \bar{G}_{k^{\text{con}},n}(\boldsymbol{X}_{(1)}^{\text{w}}) + \hat{f}_{k^{\text{con}}}(\boldsymbol{X}_{(1)}^{\text{w}}, \beta_\varepsilon)$

9 **for** $i = 2, \ldots, c^0$ **do**

10 $\varepsilon_{k^*}^{\text{L}}(i) \leftarrow \bar{G}_{k^{\text{con}},n}(\boldsymbol{X}_{(i)}^{\text{w}}) - \hat{f}_{k^{\text{con}}}(\boldsymbol{X}_{(i)}^{\text{w}}, \beta_\varepsilon)$ and $\varepsilon_{k^*}^{\text{U}}(i) \leftarrow \bar{G}_{k^{\text{con}},n}(\boldsymbol{X}_{(i)}^{\text{w}}) + \hat{f}_{k^{\text{con}}}(\boldsymbol{X}_{(i)}^{\text{w}}, \beta_\varepsilon)$

11 $\Xi_{k^*} \leftarrow \{\varepsilon_{k^*}^{\text{L}}(i) \colon i \in \{2, \ldots, c^0\},\ L_{k^*} < \varepsilon_{k^*}^{\text{L}}(i),\ \varepsilon_{k^*}^{\text{L}}(i) \notin (\varepsilon_{k^*}^{\text{L}}(i'), \varepsilon_{k^*}^{\text{U}}(i'))$ for all $i' \in \{2, \ldots, c^0\}\}$

12 $C_{k^*} \leftarrow |\Xi_{k^*}|$

13 $K^* \leftarrow \text{argmin}\{C_{k^*} \colon k^* \in \{1, 2\}\}$ /choose *least* ε-constraints; break ties randomly

14 $C \leftarrow C_{K^*}$ and $K^{\text{con}} \leftarrow k$ for $k \in \{1, 2\}$ such that $k \neq K^*$

15 **if** $C > 0$ **then**

16 Sort Ξ_{K^*} in ascending order to get the ordered list $(\varepsilon_1, \ldots, \varepsilon_C)$

17 **for** $j = 1, 2, \ldots, C$ **do** /partition space

18 $\varepsilon_j^{\text{L}} \leftarrow \max(L_{K^*}, \max\{\varepsilon_{K^*}^{\text{U}}(i) \colon \varepsilon_{K^*}^{\text{U}}(i) < \varepsilon_j, i \in \{2, \ldots, c^0\}\})$ /get traceback lower bound

19 Initialize: $\varepsilon_{\text{new}} \leftarrow \varepsilon_j$, $\hat{\mathcal{A}}_j \leftarrow \emptyset$, $\mathcal{T} \leftarrow \emptyset$

20 **while** $\varepsilon_j^{\text{L}} < \varepsilon_{\text{new}}$ **do** /find new sample-path $\mathcal{N}_a$-LWEP's

21 $\boldsymbol{X}^0 \leftarrow \text{argmin}\{\bar{G}_{K^*,n}(\boldsymbol{X}) \colon \boldsymbol{X} \in \mathcal{T} \cup \hat{\mathcal{A}}_n^0,\ \bar{G}_{K^{\text{con}},n}(\boldsymbol{X}) \leq \varepsilon_{\text{new}}\}$

22 $[\boldsymbol{X}_n^{\text{w}}, \mathcal{T}', \mathcal{N}(\boldsymbol{X}_n^{\text{w}})] = \text{SPLINE}(K^*, \boldsymbol{X}^0, \mathcal{X}_{K^*}(\varepsilon_{\text{new}}), n, b)$ /search: solve ε-constrained

23 $\hat{\mathcal{A}}_j \leftarrow \hat{\mathcal{A}}_j \cup \{\boldsymbol{X}_n^{\text{w}}\}$ and $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}'$

24 $\varepsilon_{\text{new}} \leftarrow \bar{G}_{K^{\text{con}},n}(\boldsymbol{X}_n^{\text{w}}) - f_{K^{\text{con}}}(\boldsymbol{X}_n^{\text{w}}, \beta_\varepsilon)$ /traceback: set new ε to disqualify

25 $\hat{\mathcal{A}}_{\text{LWEP}} = \cup_{j=1}^{C} \hat{\mathcal{A}}_j$ /collect new sample-path $\mathcal{N}_a$-LWEP's

26 $\hat{\mathcal{A}}_{\text{new}} = \text{RemoveDominated}(\bar{\boldsymbol{G}}_n(\hat{\mathcal{A}}_n^{\text{w}} \cup \hat{\mathcal{A}}_{\text{LWEP}} \cup \{\mathbf{x}_0\}))$ /remove dominated, do no harm

3.5.1 Pε Algorithm Listing

We now discuss the Pε algorithm (Algorithm 3) in more detail. First, to ensure that all sample-path ε-constraint problems have a non-empty feasible set, in Step 1, Pε obtains bounds on the objective values of a sample-path $\mathcal{N}_a$-LWES by obtaining a set, $\hat{\mathcal{A}}_n^0$, containing updated sample-path $\mathcal{N}_a$-local minimizers on each objective. The GetMin algorithm called in Step 1 is listed in Algorithm 4 and discussed further in §3.5.2. The sample-path $\mathcal{N}_a$-local minimizers at current sample size n ensure that for the constrained objective k^{con}, no ε values are placed outside of the interval $(\bar{G}_{k^{\text{con}},n}(\boldsymbol{X}_{k^{\text{con}},n}^{\min}), \bar{G}_{k^{\text{con}},n}(\boldsymbol{X}_{k^*,n}^{\min})]$ for $k^{\text{con}} \neq k^*$. Thus for every ε-constraint problem posed, there exists a sample-path feasible point in the set $\hat{\mathcal{A}}_n^0$.

To select an objective $k^* \in \{1, 2\}$ to minimize, in Steps 2 through 14, Pε places constraints a function of the standard error away from the images of known sample-path $\mathcal{N}_a$-LWEP's on both objectives, and ultimately selects the objective that results in solving the *least* number of ε-constraint problems. This strategy ensures that we do not constrain an objective with relatively small standard errors, thus wasting simulation effort attempting to order many points on a high standard error objective. To determine the set of ε-constraint values for each objective, first, in Step 2, Pε creates an initial set of known sample-path $\mathcal{N}_a$-LWEP's, $\hat{\mathcal{A}}_n^{\text{w}}$, from the points in $\hat{\mathcal{A}}_n^0$ using the RemoveNonLWEP function (discussed in §3.6.2). If there are no such points in $\hat{\mathcal{A}}_n^0$, the search budget value b must have been binding in the GetMin algorithm. Then in Step 3, we set ε values based on $\hat{\mathcal{A}}_n^0$. Since b is non-binding in the limit, for large enough RA iteration number ν, all ε values will be set based on known sample-path $\mathcal{N}_a$-LWEP's. In Steps 5 through 12, Pε sorts the initial points and partitions the objective space by placing ε values in each part of the objective space where the standard error intervals of the initial points do not overlap. The standard error intervals in Steps 5 through 12 are defined using the function $\hat{\mathbf{f}}$ from §3.3.2. For all known sample-path $\mathcal{N}_a$-LWEP's, denoted $\boldsymbol{X}_n^{\text{w}}$, the P$\varepsilon$ algorithm does not place any new ε values in the interval $(\bar{G}_{k^{\text{con}},n}(\boldsymbol{X}_n^{\text{w}}) - \hat{f}_{k^{\text{con}}}(\boldsymbol{X}_n^{\text{w}}, \beta_\varepsilon), \bar{G}_{k^{\text{con}},n}(\boldsymbol{X}_n^{\text{w}}) + \hat{f}_{k^{\text{con}}}(\boldsymbol{X}_n^{\text{w}}, \beta_\varepsilon))$, where $\beta_\varepsilon \in (0, \infty)$ is a parameter setting. The β_ε parameter controls how large the standard error intervals are; notice that larger standard error values and smaller β_ε values both result in wider intervals. Once both sets of ε-constraint values have been determined, Steps 13 and 14 select the objective to minimize, where ties are broken randomly.

If the chosen objective to minimize results in solving one or more ε-constraint problems, these problems are solved in Steps 15 through 25. First, Pε partitions the objective space based on the ε values. Then, each ε-constraint problem is solved using the single-objective pseudo-gradient-based SPLINE algorithm, further discussed in §3.5.3. Within a partition of the objective space, once a new sample-path $\mathcal{N}_a$-LWEP is found as the solution to the initial ε-constraint problem, Pε performs what we call

a *traceback* in Step 24 by attempting to place a new ε value that is both within the current partition and that sets as infeasible the newly-found sample-path $\mathcal{N}_a$-LWEP. If the new ε value is outside the partition, the search ends; otherwise, the new ε-constraint problem is solved, and this process repeats until no more ε values can be placed in the current partition. The new sample-path $\mathcal{N}_a$-LWEP's found across all partitions are collected into a set of sample-path non-dominated $\mathcal{N}_a$-LWEP's for Problem $\overline{M}_{2,n}$ and returned by Pε in Step 26. We remark here that as the RA iteration number $\nu \to \infty$ in R-PεRLE, under appropriate regularity conditions, the Pε algorithm solves as many ε-constraint problems as there are points in the $\mathcal{N}_a$-LES to which the algorithm converges (see §3.8).

3.5.2 The GetMin Algorithm for Many Objectives

The GetMin algorithm for $d \geq 2$ objectives, called in Pε Step 1 and listed in Algorithm 4, is a relatively simple algorithm that takes in any set of feasible points, brings up the sample sizes, updates the sample-path $\mathcal{N}_a$-local minimizers on each objective, and removes any points whose images are sample-path dominated. The resulting set of sample-path non-dominated points and sample-path $\mathcal{N}_a$-local minimizers are returned as the set $\hat{\mathcal{A}}_n^*$.

Algorithm 4: $\hat{\mathcal{A}}_n^* = \text{GetMin}(\hat{\mathcal{A}}_{\text{old}}, n, b)$

Input: a set of feasible points $\hat{\mathcal{A}}_{\text{old}} \subset \mathcal{X}$; sample size, n; limit on SPLINE calls, b
Output: $\hat{\mathcal{A}}_n^*$, a candidate EALES with updated sample-path local minimizers at sample size n

1 **for** $k = 1, 2, \ldots, d$ **do**
2 $\quad \boldsymbol{X}_{\text{k,old}}^{\min} \leftarrow \text{argmin}\{\bar{G}_{k,n}(\boldsymbol{X})\colon \boldsymbol{X} \in \hat{\mathcal{A}}_{\text{old}}\}$
3 $\quad [\boldsymbol{X}_{k,n}^{\min}, \sim] = \text{SPLINE}(k, \boldsymbol{X}_{\text{k,old}}^{\min}, \mathcal{X}, n, b)$ `/search: sample-path` $\mathcal{N}_a$ `-local minimizer`
4 $\bar{\mathcal{M}}_n \leftarrow \cup_{k=1}^{d} \{\boldsymbol{X}_{k,n}^{\min}\}$ `/points in` $\bar{\boldsymbol{G}}_n(\bar{\mathcal{M}}_n)$ `may dominate other points in` $\bar{\boldsymbol{G}}_n(\bar{\mathcal{M}}_n)$
5 $\hat{\mathcal{A}}_n^* = \text{RemoveDominated}(\bar{\boldsymbol{G}}_n(\hat{\mathcal{A}}_{\text{old}} \cup \bar{\mathcal{M}}_n \cup \{\mathbf{x}_0\}))$

3.5.3 The SPLINE Algorithm for One Objective

The SPLINE algorithm of Wang et al. (2013) is the engine that underlies all of our deterministic single-objective searches, that is, solving the ε-constraint problems in the Pε algorithm, Step 22, and finding the sample-path $\mathcal{N}_a$-local minimizers in the GetMin algorithm, Step 3. The SPLINE algorithm (Algorithm 5) finds a sample-path $\mathcal{N}_a$-local minimizer of objective k on a feasible set $\mathcal{X}$ using sample size n. Our version of the SPLINE algorithm contains minor modifications that allow us to input an objective to minimize, input the feasible space, and output the search trajectory for later use in our algorithms.

The SPLINE algorithm consists of two primary steps: SPLI and NE, which are called iteratively until a sample-path $\mathcal{N}_a$-local minimizer is found, or the search times out. The SPLI algorithm conducts a pseudogradient-based line search with piecewise linear interpolation. The NE algorithm performs neighborhood enumeration to either move to a better neighborhood point, or to certify a local minimum has been found. We refer the reader to Wang et al. (2013) for detailed listings and further explanations of SPLI and NE.

Algorithm 5: $[\boldsymbol{X}^*, \mathcal{T}, \mathcal{N}(\boldsymbol{X}^*)] = \text{SPLINE}(k, \boldsymbol{X}_0, \mathcal{X}, n, b)$

Input: objective k; initial point $\boldsymbol{X}_0 \in \mathcal{X}$; feasible set $\mathcal{X}$; sample size n; limit on search oracle calls, b
Output: local solution $\boldsymbol{X}^*$ on $\mathcal{X}$; sample-path search set, $\mathcal{T}$; neighborhood points, $\mathcal{N}(\boldsymbol{X}^*)$

1 Initialize: search oracle calls spent so far $b^* \leftarrow 0$, $\boldsymbol{X}_{\text{NE}} \leftarrow \boldsymbol{X}_0$, and $\mathcal{T} \leftarrow \{\boldsymbol{X}_0\}$
2 **repeat**
3 $\quad [b', \boldsymbol{X}_{\text{SPLI}}, \bar{\boldsymbol{G}}_n(\boldsymbol{X}_{\text{SPLI}})] = \text{SPLI}(k, \boldsymbol{X}_{\text{NE}}, \mathcal{X}, n, b)$ `/search: line search with interpolation`
4 $\quad$ **if** $\bar{G}_{k,n}(\boldsymbol{X}_{\text{SPLI}}) > \bar{G}_{k,n}(\boldsymbol{X}_{\text{NE}})$ **then** $\boldsymbol{X}_{\text{SPLI}} \leftarrow \boldsymbol{X}_{\text{NE}}$ `/SPLI cannot cause harm`
5 $\quad [b'', \boldsymbol{X}_{\text{NE}}, \bar{\boldsymbol{G}}_n(\boldsymbol{X}_{\text{NE}}), \mathcal{N}(\boldsymbol{X}_{\text{NE}})] = \text{NE}(k, \boldsymbol{X}_{\text{SPLI}}, \mathcal{X}, n)$ `/neighborhood enumeration`
6 $\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{\boldsymbol{X}_{\text{SPLI}}, \boldsymbol{X}_{\text{NE}}\}$ `/update trajectory`
7 $\quad b^* \leftarrow b^* + b' + b''$ `/update oracle calls expended`
8 **until** $\bar{G}_{k,n}(\boldsymbol{X}_{\text{NE}}) = \bar{G}_{k,n}(\boldsymbol{X}_{\text{SPLI}})$ *or* $b^* > b$ `/find a local solution or time out`

We remark here that (Liuzzi et al., 2018) also provide a line search algorithm for integer lattices. Certainly, other algorithms could be used in place of SPLINE such as those contained in Audet and Hare (2017) and Conn et al. (2009); we select SPLINE because of its impressive performance on single-objective SO problems in (Wang et al., 2013).

3.6 The RLE Algorithm for Many Objectives

The RLE algorithm is the second algorithm in our proposed two-part sample-path solver, PεRLE. The collection of sample-path $\mathcal{N}_a$-LWEP's found by the Pε algorithm is sent to RLE to certify that this collection of points is indeed an ALES, or to create an ALES using this collection of points as an initial set. Without RLE to certify an ALES, an algorithm like Pε that relies only on collecting sample-path $\mathcal{N}_a$-LWEP's may "get stuck" by returning points that do not belong to the same, or to any, sample-path $\mathcal{N}_a$-LWES. For example, in Figure 3.1, the Pε algorithm may return $\mathcal{S} = \{\mathbf{x}_{1,\mathrm{f}}^{\min}, \mathbf{x}_{\mathrm{g}}^{*}, \mathbf{x}_{2,\mathrm{e}}^{\min}\}$, which is a set of $\mathcal{N}_a$-LWEP's containing a minimum on each objective but that is not an $\mathcal{N}_a$-LWES. The RLE algorithm is designed to crawl out of the sample-path version of this scenario when the completeness function is small enough. In what follows, we first define a key concept used in RLE called the sample-path non-conforming neighborhood. Then, we discuss the RLE algorithm in detail.

3.6.1 The Sample-Path Non-Conforming Neighborhood

Suppose we are given a set of feasible points $\mathcal{S}$ such that none of the estimated images of points in $\mathcal{S}$ dominate the estimated images of other points in $\mathcal{S}$. The non-conforming neighborhood of $\mathcal{S}$, defined in Definition 12, is the set of points in the deleted neighborhood of $\mathcal{S}$ that prevent it from being an ALES.

Definition 12. *Let $\mathcal{S} \subseteq \mathcal{X}$ be a collection of feasible points such that no points in $\bar{\boldsymbol{G}}_n(\mathcal{S})$ dominate other points in $\bar{\boldsymbol{G}}_n(\mathcal{S})$. Then given a completeness function $\boldsymbol{\delta}\colon \mathcal{X} \to \overline{\mathbb{R}}^d$ such that $\mathbf{0}_d \leqq \boldsymbol{\delta}(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$, define the* sample-path non-conforming neighborhood *(NCN) of $\mathcal{S}$, $\mathcal{N}_a^{\mathrm{nc}}(\mathcal{S})$, as all feasible points in the deleted neighborhood of $\mathcal{S}$, $\mathbf{x} \in \mathcal{N}_a'(\mathcal{S}) \cap \mathcal{X}$, such that*

(a) $\exists \tilde{\mathbf{x}} \in \mathcal{S}$ such that $\mathbf{x} \in \mathcal{N}_a(\tilde{\mathbf{x}})$ and $\bar{\boldsymbol{G}}_n(\mathbf{x}) < \bar{\boldsymbol{G}}_n(\tilde{\mathbf{x}})$, or

(b) (i) $\nexists \tilde{\mathbf{x}} \in \mathcal{S}$ such that $\bar{\boldsymbol{G}}_n(\tilde{\mathbf{x}}) \leqq \bar{\boldsymbol{G}}_n(\mathbf{x})$, and (ii) $\nexists \tilde{\mathbf{x}} \in \mathcal{S}$ such that ($\bar{\boldsymbol{G}}_n(\mathbf{x}) \leq \bar{\boldsymbol{G}}_n(\tilde{\mathbf{x}})$ and $\bar{\boldsymbol{G}}_n(\tilde{\mathbf{x}}) - \boldsymbol{\delta}(\tilde{\mathbf{x}}) \leqq \bar{\boldsymbol{G}}_n(\mathbf{x}) + \boldsymbol{\delta}(\mathbf{x})$), and (iii) $\exists \tilde{\mathbf{x}} \in \mathcal{S}$ such that $\bar{\boldsymbol{G}}_n(\mathbf{x}) \leqq$

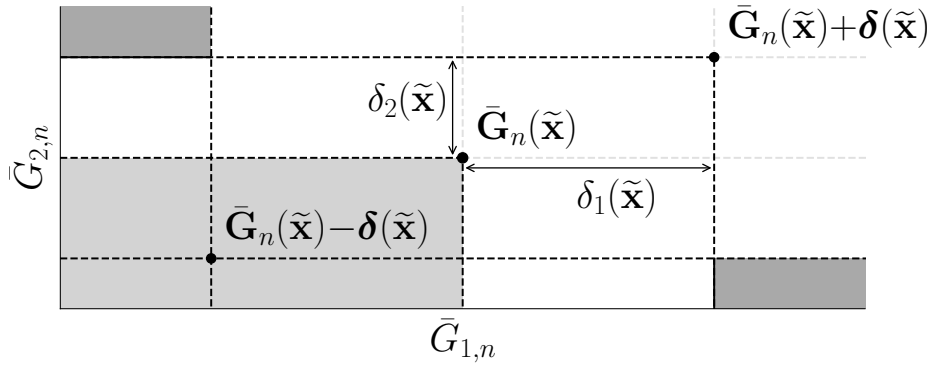

Figure 3.2. Let $\mathcal{S} = \{\tilde{\mathbf{x}}\}$ with two objectives, and let $\mathbf{x} \in \mathcal{N}_a'(\mathcal{S}) \cap \mathcal{X}$ be in the deleted neighborhood of $\mathcal{S}$. Def. 12(a) adds $\mathbf{x}$ to the NCN if $\mathbf{x} \in \mathcal{N}_a(\tilde{\mathbf{x}})$ and $\bar{\boldsymbol{G}}_n(\mathbf{x})$ is in the light gray area. If $\mathbf{x}$ does not satisfy Def. 12(a), Def. 12(b) adds $\mathbf{x}$ to the NCN if its $\boldsymbol{\delta}$ box, defined by corners $\bar{\boldsymbol{G}}_n(\mathbf{x}) \pm \boldsymbol{\delta}(\mathbf{x})$, is contained in the dark gray area.

$\bar{\boldsymbol{G}}_n(\tilde{\mathbf{x}})$, *or* $\nexists \tilde{\mathbf{x}} \in \mathcal{S}$ *such that* $\bar{\boldsymbol{G}}_n(\tilde{\mathbf{x}}) - \boldsymbol{\delta}(\tilde{\mathbf{x}}) \leqq \bar{\boldsymbol{G}}_n(\mathbf{x}) + \boldsymbol{\delta}(\mathbf{x})$ *or* $\bar{\boldsymbol{G}}_n(\mathbf{x}) - \boldsymbol{\delta}(\mathbf{x}) \leqq \bar{\boldsymbol{G}}_n(\tilde{\mathbf{x}}) + \boldsymbol{\delta}(\tilde{\mathbf{x}})$.

First, Definition 12(a) adds $\mathbf{x} \in \mathcal{N}_a'(\mathcal{S}) \cap \mathcal{X}$ to the NCN if it prevents a point in $\mathcal{S}$ from being a sample-path $\mathcal{N}_a$-LWEP. Definition 12(b) also adds a feasible deleted-neighborhood point to the NCN if it violates the conditions of Definition 11(b), that is, the point (i) is not weakly dominated by any points in $\mathcal{S}$, and (ii) does not dominate any points in $\mathcal{S}$ by less than a certain amount, and (iii) weakly dominates a point in $\mathcal{S}$, or would not either weakly dominate or be weakly dominated by a point in $\mathcal{S}$ if both were moved by a certain amount.

Given a singleton feasible set $\mathcal{S} = \{\tilde{\mathbf{x}}\}$, Figure 3.2 shows regions of the objective space that correspond to a feasible point in the deleted neighborhood of $\mathcal{S}$, $\mathbf{x} \in \mathcal{N}_a'(\mathcal{S}) \cap \mathcal{X}$, being declared a member of the NCN. Definition 12(a) implies $\mathbf{x}$ is in the NCN if it is an $\mathcal{N}_a$-neighbor of $\tilde{\mathbf{x}}$ and $\bar{\boldsymbol{G}}_n(\mathbf{x})$ is in the light gray region of Figure 3.2. Excluding points that meet the requirements of Definition 12(a), Definition 12(b) implies that $\mathbf{x}$ is in the NCN if its entire "$\boldsymbol{\delta}$ box," defined by the corners $\bar{\boldsymbol{G}}_n(\mathbf{x}) \pm \boldsymbol{\delta}(\mathbf{x})$, is completely contained in the dark gray shaded region of Figure 3.2. Thus there is

Algorithm 6: $\hat{\mathcal{A}}_{\text{ALES}} = \text{RLE}(\mathcal{S}, n, b, \beta_\delta)$

Input: set of points $\mathcal{S} \subset \mathcal{X}$; limit on search oracle calls, b; ALES completeness parameter β_δ
Output: $\hat{\mathcal{A}}_{\text{ALES}}$, which is an ALES for the sample-path problem with sample size n

1 $\mathcal{S} = \text{RemoveDominated}(\bar{\boldsymbol{G}}_n(\mathcal{S} \cup \{\mathbf{x}_0\}))$
2 $\mathcal{N}^{\text{nc}} = \text{GetNCN}(\mathcal{S}, \beta_\delta)$ /get non-conforming neighborhood
3 Initialize: outer search oracle calls spent so far $b^* \leftarrow 0$
4 **while** $b^* \leq b$ *and* $\mathcal{N}^{\text{nc}} \neq \emptyset$ **do** /search: traverse sample-path $\mathcal{N}_a$-LWEP chains
5 $[b', \mathcal{N}^{\text{w}}, \mathcal{N}_2^*] = \text{RemoveNonLWEP}(\mathcal{N}^{\text{nc}})$
6 $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{N}^{\text{w}}$ /add neighborhood sample-path $\mathcal{N}_a$-LWEP's to $\mathcal{S}$
7 **if** $\mathcal{N}^{\text{w}} = \emptyset$ **then** /the neighbors are dominated by their neighbors: $\mathcal{N}_2^* \neq \emptyset$
8 Initialize: $\mathcal{X}^{\text{new}} \leftarrow \mathcal{N}_2^*$, $\mathcal{X}^{\text{w}} \leftarrow \emptyset$, and inner search oracle calls spent so far $b^{**} \leftarrow 0$
9 **while** $b^{**} \leq b$ *and* $\mathcal{X}^{\text{w}} = \emptyset$ **do** /search: traverse dominating chains
10 $[b''', \mathcal{X}^{\text{w}}, \mathcal{N}_2^*] = \text{RemoveNonLWEP}(\mathcal{X}^{\text{new}})$
11 $\mathcal{X}^{\text{new}} \leftarrow \mathcal{N}_2^*$ and $b^{**} \leftarrow b^{**} + b'''$
12 $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{X}^{\text{w}}$
13 **if** $\mathcal{X}^{\text{w}} = \emptyset$ **then** $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{X}^{\text{new}}$ /keep progress if search times out
14 $\mathcal{S} = \text{RemoveDominated}(\bar{\boldsymbol{G}}_n(\mathcal{S} \cup \{\mathbf{x}_0\}))$
15 $[b'', \mathcal{N}^{\text{nc}}] = \text{GetNCN}(\mathcal{S}, \beta_\delta)$ /get non-conforming neighborhood
16 $b^* \leftarrow b^* + b' + b''$
17 **return** $\hat{\mathcal{A}}_{\text{ALES}} \leftarrow \mathcal{S}$

no overlap between the $\boldsymbol{\delta}$ boxes of $\mathbf{x}$ and $\tilde{\mathbf{x}}$ on any objective. Henceforth, we use $\hat{\boldsymbol{\delta}}(\mathbf{x}) = \hat{\mathbf{f}}(\mathbf{x}, \beta_\delta)$ for all $\mathbf{x} \in \mathcal{X}$ (see §3.3.2) as the NCN completeness function.

3.6.2 RLE Algorithm Listing

We now discuss the RLE algorithm (Algorithm 6) in detail. To guarantee an ALES, in Step 1, RLE first removes any points in $\mathcal{S}$ whose estimated objective vectors are dominated by the estimated objective vectors of other points in $\mathcal{S}$, since these points cannot be members of an ALES. Then, in Step 2, RLE calculates the NCN of the remaining points using the function GetNCN. If the NCN is empty in Step 2, then RLE certifies it has found an ALES and the algorithm terminates; otherwise, RLE enters a search phase.

The "outer" RLE search phase, which begins in Step 4, checks to see if any members of the NCN are also sample-path $\mathcal{N}_a$-LWEP's using RemoveNonLWEP. The RemoveNonLWEP function takes an input set of feasible points and outputs three quantities: (a) the number of simulation replications expended, (b) the set of sample-path $\mathcal{N}_a$-LWEP's in the input set, and (c) a set of points in the neighborhood of the

input set whose images sample-path dominate the images of the members of the input set. Thus when RemoveNonLWEP is passed a non-empty NCN, it enumerates the neighborhood of the NCN; these points are neighbors of neighbors of the original set. If the NCN contains sample-path $\mathcal{N}_a$-LWEP's in Step 5, RLE adds them to $\mathcal{S}$ in Step 6. If no members of the NCN are sample-path $\mathcal{N}_a$-LWEP's in Step 7, there must exist neighbors of the NCN that dominate points in the NCN, denoted as $\mathcal{N}_2^*$ in Steps 5 and 8. If this is the case, RLE enters an "inner" search phase.

The "inner" RLE search phase, which begins in Step 9, allows the algorithm to find new sample-path $\mathcal{N}_a$-LWEP's by traversing points whose estimated images are sample-path dominated. Once a sample-path $\mathcal{N}_a$-LWEP is found or the inner search times out, the new points are added to $\mathcal{S}$ in Steps 12 and 13. After removing points whose images are sample-path dominated by the images of other points in $\mathcal{S}$, in Step 15, RLE checks the new set $\mathcal{S}$ to see if it is an ALES. If not, this process repeats until a complete ALES is found, or a total outer search budget has been exhausted. Since the search budget sequence is non-binding in the limit, for large enough RA iteration number ν, RLE guarantees an ALES.

3.7 Other Algorithms: R-Pε and R-MinRLE

To assess and understand the performance of R-PεRLE, we find it helpful to define and analyze two other RA algorithms: R-Pε and R-MinRLE, which we discuss in this section. We discuss R-Pε for two objectives first, followed by R-MinRLE for many objectives.

First, we define the R-Pε algorithm for two objectives as identical to R-PεRLE (Algorithm 2), except without the call to RLE in Step 4. To show that the Pε algorithm delivers "good points" to the RLE algorithm, in §3.8, we prove convergence of the R-Pε algorithm under a set of fairly restrictive assumptions; we remark that these assumptions are violated in the example in Figure 3.1. We also find the R-Pε algorithm useful for numerically analyzing the settings of the β_ε parameter in §3.10.2.

Algorithm 7: The R-MinRLE Algorithm for $d \geq 2$

Input: initial point $\mathbf{x}_0 \in \mathcal{X}$; sequence of sample sizes to expend at each visited point, $\{m_\nu\}$; sequence of limits on oracle calls during search, $\{b_\nu\}$; ALES completeness parameter, β_δ

Initialize: $\hat{\mathcal{A}}_0 = \{\mathbf{x}_0\}$ and set $\mathbf{x}_0$ as a global variable
for $\nu = 1, 2, \ldots$ *with CRN* **do**
  $\hat{\mathcal{A}}_{\min} = \text{GetMin}(\hat{\mathcal{A}}_{\nu-1}, m_\nu, b_\nu)$ `/update the sample-path` $\mathcal{N}_a$ `-local minimizers`
  $\hat{\mathcal{A}}_\nu = \text{RLE}(\hat{\mathcal{A}}_{\min}, m_\nu, b_\nu, \beta_\delta)$ `/guarantee the returned set is an ALES`

We emphasize here that we do not recommend R-Pε for implementation. Unless the decision-maker has knowledge of special structure in the objective functions, one should always choose R-PεRLE over R-Pε for bi-objective SO.

The R-MinRLE algorithm, listed in Algorithm 7, is arguably the most general algorithm we propose, since it is defined for $d \geq 2$ objectives. R-MinRLE is like R-PεRLE except that instead of obtaining a set of sample-path $\mathcal{N}_a$-LWEP's from Pε on each RA iteration, R-MinRLE uses the GetMin algorithm (Algorithm 4) to update the sample-path $\mathcal{N}_a$-local minimizer on each objective before invoking RLE. We define the sample-path solver MinRLE as calling GetMin followed by RLE within an RA iteration; the MinRLE algorithm locates an ALES for sample-path Problem $\overline{M}_{d,m_\nu}$ for $d \geq 2$ and each $\nu = 1, 2, \ldots$.

We view the R-MinRLE algorithm as a naïve pseudo-gradient-based benchmark algorithm for many objectives. Loosely speaking, for two objectives, notice that R-MinRLE is likely to exhibit "outside-in" convergence behavior. That is, because R-MinRLE only guarantees locating the sample-path $\mathcal{N}_a$-local minimizers on each RA iteration, if the completeness parameter β_δ is "small" so that RLE crawls less, MinRLE locates the sample-path $\mathcal{N}_a$-local minimizers and perhaps a few points nearby to complete an ALES. All externalities being equal, this ALES is less likely to contain points that map to the "center" of a sample-path $\mathcal{N}_a$-LPS than a corresponding ALES located by PεRLE. Therefore in our numerical experiments, comparisons with R-MinRLE demonstrate the usefulness of using the Pε algorithm as a precursor to RLE within an RA iteration, as opposed to the naïve GetMin algorithm.

3.8 Asymptotic Behavior

We now study the asymptotic behavior of our RA algorithms and show that, under appropriate regularity conditions, R-PεRLE, R-Pε, and R-MinRLE converge to an $\mathcal{N}_a$-LES w.p.1. In the sections that follow, first, we discuss the assumptions required for our results to hold. Then, we prove the convergence of algorithms that rely on RLE, that is, R-PεRLE and R-MinRLE. The proof of convergence for algorithms that rely on RLE is general in the sense that, under our regularity conditions, the proof provides a convergence result for any RA algorithm designed to solve Problem M_d whose sample-path solver invokes RLE last with the completeness function $\hat{\boldsymbol{\delta}}$ and $\beta_\delta \in (0, \infty]$. We also prove the convergence of R-Pε under fairly restrictive assumptions. Finally, in §3.8.4, we provide sampling efficiency results. Throughout this section, we assume the search budget sequence $\{b_\nu, \nu = 1, 2, \ldots\}$ is non-binding w.p.1 for all ν large enough, under our regularity conditions. Therefore we ignore issues related to binding budget sequences for small ν. We specify the default budget sequence in §3.9.

3.8.1 Preliminaries and Assumptions

We require several regularity conditions on both the true, unknown objective functions and the sample-path objective functions. We require Assumptions 1–3 in all of our results.

Assumption 1. *(Wang et al., 2013, p. 12) For each $\mathbf{x} \in \mathcal{X}$ and $k \in \{1, \ldots, d\}$, there exists $\alpha_k > 0$, dependent on $\mathbf{x}$, such that $\mathcal{S}_k(\mathbf{x}, \alpha_k) := \{\mathbf{x}' \in \mathcal{X} \colon g_k(\mathbf{x}') \leq g_k(\mathbf{x}) + \alpha_k\}$ is finite.*

Assumption 2. *(Wang et al., 2013, p. 12) For each $k \in \{1, \ldots, d\}$, we assume the following. Let $\mathcal{S}_k(\mathbf{x}, \alpha_k)$ be as in Assumption 1, and define $\hat{\mathcal{S}}_{k,\nu}(\mathbf{x}) := \{\mathbf{x}' \in \mathcal{X} \colon \bar{G}_{k,m_\nu}(\mathbf{x}') \leq \bar{G}_{k,m_\nu}(\mathbf{x})\}$ for all $\nu = 1, 2, \ldots$. Given $\mathbf{x} \in \mathcal{X}$, there*

exists a sequence $\{p_\nu\}_{\nu=1}^{\infty}$ *such that* $\mathbb{P}\{\tilde{\mathbf{x}} \in \hat{\mathcal{S}}_{k,\nu}(\mathbf{x})\} \leq p_\nu$ *and* $\sum_{\nu=1}^{\infty} p_\nu < \infty$ *for all* $\tilde{\mathbf{x}} \in \mathcal{X} \setminus \mathcal{S}_k(\mathbf{x}, \alpha_k)$.

Assumption 3. *All variances are finite, that is,* $\max_{k\in\{1,\dots,d\}} \sigma_k^2(\mathbf{x}) < \infty$ *for all* $\mathbf{x} \in \mathcal{X}$.

First, Assumption 1 implies that at each feasible point $\mathbf{x} \in \mathcal{X}$ and for every objective k, there exists a constant such that the level set created by adding the constant to $g_k(\mathbf{x})$ is finite. Since this property holds for every objective k, then under Assumption 1, the union of the level sets over the objectives k is also finite. That is, for each $\mathbf{x} \in \mathcal{X}$, define $\boldsymbol{\alpha} := (\alpha_1, \dots, \alpha_d)$, where $\boldsymbol{\alpha}$ also depends on $\mathbf{x}$. Define the set $\mathcal{S}(\mathbf{x}, \boldsymbol{\alpha}) := \cup_{k=1}^{d} \mathcal{S}_k(\mathbf{x}, \alpha_k)$ as the set of all feasible points that map to objective values that are not strictly dominated by the point $\mathbf{g}(\mathbf{x}) + \boldsymbol{\alpha}$; notice that $\mathbf{g}(\mathcal{S}^c(\mathbf{x}, \boldsymbol{\alpha})) = \{\mathbf{g}(\mathbf{x}) + \boldsymbol{\alpha}\} + \{\mathbf{y} \in \mathbb{R}^d \colon \mathbf{0}_d < \mathbf{y}\}$ is the set of points that $\mathbf{g}(\mathbf{x}) + \boldsymbol{\alpha}$ strictly dominates for all $\mathbf{x} \in \mathcal{X}$. Then under Assumption 1, $\mathcal{S}(\mathbf{x}, \boldsymbol{\alpha})$ is finite for all $\mathbf{x} \in \mathcal{X}$.

Assumption 1 further implies the nonempty GWES exists and all $\mathcal{N}_a$-LWES's are finite. This result is presented without proof in Lemma 1, where we first define the following notation. Given a neighborhood size parameter $a \in (0, \infty]$, let $\mathfrak{L}^{\mathrm{w}}$ and $\mathfrak{L}$ be the collection of all possible $\mathcal{N}_a$-LWES's and $\mathcal{N}_a$-LES's for Problem M_d, respectively, where $\mathfrak{L} \subseteq \mathfrak{L}^{\mathrm{w}}$. Since the GWES is also an $\mathcal{N}_a$-LWES for $a \geq 1$, notice that $|\mathfrak{L}^{\mathrm{w}}| \geq 1$ if the GWES exists.

Lemma 1. *Under Assumption 1, given* $a \geq 1$*, the following hold:*

(a) The GWES, $\mathcal{E}^{\mathrm{w}} \subseteq \mathcal{X}$*, exists and is nonempty.*

(b) All $\mathcal{N}_a$*-LWES's are finite; that is,* $1 \leq |\mathcal{L}^{\mathrm{w}}| < \infty$ *for all* $\mathcal{L}^{\mathrm{w}} \in \mathfrak{L}^{\mathrm{w}}$.

Proof. Proof of Lemma 1 Part (a). Under Assumption 1, global minimizers exist for each objective. Since each global minimizer is a GWEP that must belong to the GWES, the GWES exists and is nonempty. □

Proof. Proof of Lemma 1 Part (b).For a contradiction, suppose there exists $\mathcal{L}^{\mathrm{w}} \in \mathfrak{L}^{\mathrm{w}}$ such that $|\mathcal{L}^{\mathrm{w}}| = \infty$. Let $\mathbf{x}^{\mathrm{w}} \in \mathcal{L}^{\mathrm{w}}$, which is an $\mathcal{N}_a$-LWEP by definition. Under

Assumption 1, there exists $\boldsymbol{\alpha} > \mathbf{0}_d$ such that $\mathcal{S}(\mathbf{x}^{\mathrm{w}}, \boldsymbol{\alpha})$ is finite. But under Definition 9, no points in $\mathcal{S}^{\mathrm{c}}(\mathbf{x}^{\mathrm{w}}, \boldsymbol{\alpha})$ are eligible to belong to $\mathcal{L}^{\mathrm{w}}$, because they map to points that are strictly dominated by $\mathbf{g}(\mathbf{x}^{\mathrm{w}})$. Thus all points in $\mathcal{L}^{\mathrm{w}}$ must belong to $\mathcal{S}(\mathbf{x}^{\mathrm{w}}, \boldsymbol{\alpha})$, which is finite. □

Now, let us turn our attention to Assumption 2, which is a condition defined by (Wang et al., 2013) ensuring that the probability of incorrectly estimating a level set decays sufficiently fast. (Wang et al., 2013) provide a detailed discussion of the conditions under which Assumption 2 holds. For completeness, we include the conditions below as Lemma 2; recall that the variance $\sigma_k^2(\mathbf{x})$ is $\mathbb{V}(G_k(\mathbf{x}, \xi))$ for each $\mathbf{x} \in \mathcal{X}$ and objective $k \in \{1, \ldots, d\}$. Essentially, Lemma 2 implies that Assumption 2 holds under a large-deviations regime or under the conditions of the Central Limit Theorem, whenever the sample size sequence increases at a sufficiently fast rate. For compactness, we refer the reader to (Wang et al., 2013) for additional explanation of Assumption 2 and Lemma 2.

Lemma 2. *(see Wang et al., 2013, p. 13–14) Assumption 2 holds if one of the following two sets of conditions holds:*

C1. (a) for all $k \in \{1, \ldots, d\}$, the sequence of random variables $\{\bar{G}_{k,m_\nu}(\mathbf{x}) - g_k(\mathbf{x})\}$ is governed by a large-deviation principle with rate function $I_{k,\mathbf{x}}(s)$ (Dembo and Zeitouni, 1998);

(b) each $I_{k,\mathbf{x}}(s)$ is such that for any $\epsilon > 0$, $\inf_{\mathbf{x}\in\mathcal{X},\, k\in\{1,\ldots,d\}} \min(I_{k,\mathbf{x}}(\epsilon), I_{k,\mathbf{x}}(-\epsilon)) = \eta > 0$; and

(c) the sequence of sample sizes $\{m_\nu\}$ increases faster than logarithmically, that is, $\limsup_{\nu\to\infty}(m_\nu)^{-1}(\log \nu)^{1+\Delta_1} = 0$ for some $\Delta_1 > 0$.

C2. (a) for all $k \in \{1, \ldots, d\}$, a central limit theorem holds on the sequence of random variables $\{\bar{G}_{k,m_\nu}(\mathbf{x})\}$ for each $\mathbf{x} \in \mathcal{X}$, that is, $\sqrt{m_\nu}(\sigma_k(\mathbf{x}))^{-1}(\bar{G}_{k,m_\nu}(\mathbf{x}) - g_k(\mathbf{x})) \Rightarrow Z$, where $\sigma_k(\mathbf{x}) > 0$ satisfies $\sup_{\mathbf{x}\in\mathcal{X}} \sigma_k^2(\mathbf{x}) < \infty$, and

(b) as $\nu \to \infty$, $\sup_y |F_{k,\mathbf{x},m_\nu}(y) - \Phi(y)| = O(1/\sqrt{m_\nu})$ for all $\mathbf{x} \in \mathcal{X}$, where $F_{k,\mathbf{x},m_\nu}(\cdot)$ denotes the cumulative distribution function of the random variable

$\sqrt{m_\nu}(\sigma_k(\mathbf{x}))^{-1}(\bar{G}_{k,m_\nu}(\mathbf{x}) - g_k(\mathbf{x}))$, *and*

(c) the sequences of sample sizes $\{m_\nu\}$ *satisfies* $\limsup_{\nu\to\infty}(m_\nu)^{-1}\nu^{2+\Delta_2} = 0$ *for some* $\Delta_2 > 0$.

The primary implication of Assumptions 1 and 2 is the convergence of the estimated level sets into the true level sets, as described in the following Lemma 3. Before we present the lemma, recall that $\mathcal{S}(\mathbf{x}, \boldsymbol{\alpha}) = \cup_{k=1}^{d}\mathcal{S}_k(\mathbf{x}, \alpha_k)$ is finite, and define $\hat{\mathcal{S}}_\nu(\mathbf{x}) := \cup_{k=1}^{d}\hat{\mathcal{S}}_{k,\nu}(\mathbf{x})$ as the set of all decision points estimated as being at least as good as $\mathbf{x}$ on at least one objective.

Lemma 3. *Under Assumptions 1 and 2,*

(a) (see Wang et al., 2013, p. 15) for each $k \in \{1, \ldots, d\}$ *and any* $\mathbf{x} \in \mathcal{X}$*, the sets* $\hat{\mathcal{S}}_{k,\nu}(\mathbf{x})$ *converge almost surely into the set* $\mathcal{S}_k(\mathbf{x}, \alpha_k)$*, that is,* $\mathbb{P}\{\hat{\mathcal{S}}_{k,\nu}(\mathbf{x}) \not\subseteq \mathcal{S}_k(\mathbf{x}, \alpha_k)$ *i.o.*$\} = 0$;

(b) the sets $\hat{\mathcal{S}}_\nu(\mathbf{x})$ *converge almost surely into the sets* $\mathcal{S}(\mathbf{x}, \boldsymbol{\alpha})$ *for any* $\mathbf{x} \in \mathcal{X}$*, that is,* $\mathbb{P}\{\hat{\mathcal{S}}_\nu(\mathbf{x}) \not\subseteq \mathcal{S}(\mathbf{x}, \boldsymbol{\alpha})$ *i.o.*$\} = 0$.

Proof Sketch. The proof of Lemma 3 Part (a) is provided in (Wang et al., 2013, p. 15) and follows from the first Borel-Cantelli lemma (Billingsley, 1995, p. 59). By Lemma 3 Part (a), for each objective $k \in \{1, \ldots, d\}$, there exists $\tilde{\nu}_k$, dependent on $\boldsymbol{\alpha}$, $\mathbf{x}$, and the random realization, such that for all $\nu \geq \tilde{\nu}_k$, $\hat{\mathcal{S}}_{k,\nu}(\mathbf{x}) \subseteq \mathcal{S}_k(\mathbf{x}, \alpha_k)$ w.p.1. Let $\tilde{\nu} := \max_k\{\tilde{\nu}_k\}$, so that for all $\nu \geq \tilde{\nu}$, $\hat{\mathcal{S}}_\nu(\mathbf{x}) \subseteq \mathcal{S}(\mathbf{x}, \boldsymbol{\alpha})$ w.p.1.

Finally, we need our last required assumption, Assumption 3, because our ε-placement and ALES completeness parameters rely on the estimated standard errors of the objective function values. Notice that Assumption 3 is implied under the conditions of Lemma 2.

In addition to Assumptions 1–3 discussed above, some of our results require additional structure on the true, unknown objective functions in Problem M_d. We present these assumptions as Assumptions 4–6, and then we discuss their implications.

Assumption 4. *For all* $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$*, if* $\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{x}')$*, then* $\mathbf{x} = \mathbf{x}'$.

Assumption 5. *There exists $\kappa > 0$ such that*

$\min_{k\in\{1,\dots,d\}} \inf\{|g_k(\mathbf{x}) - g_k(\mathbf{x}')| \colon \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \mathbf{x} \neq \mathbf{x}'\} > \kappa$.

Assumption 6. *Given $a \in [1, \infty)$, all $\mathcal{N}_a$-LWEP's are GEP's and there exists exactly one $\mathcal{N}_a$-LES that solves Problem M_d which is also the GES, $\mathcal{E}$.*

Assumption 4 ensures that two or more decision points in the feasible space do not map to the same point in the objective space. Under this assumption, the following Lemma 4 holds regarding the existence of an $\mathcal{N}_a$-LES within each $\mathcal{N}_a$-LWES. We present the result without proof; intuitively, it follows because Assumption 4 prevents the points in the $\mathcal{N}_a$-LWES from having identical objective vector values. Thus the set must contain $\mathcal{N}_a$-LEP's, from which the $\mathcal{N}_a$-LES can be constructed.

Lemma 4. *Under Assumptions 1 and 4, given $a \in (0, \infty]$, all $\mathcal{N}_a$-LWES's contain an $\mathcal{N}_a$-LES; that is, for all $\mathcal{L}^{\mathrm{w}} \in \mathfrak{L}^{\mathrm{w}}$, there exists $\mathcal{L}^* \in \mathfrak{L}$ such that $\mathcal{L}^{\mathrm{w}} \supseteq \mathcal{L}^*$.*

Proof. Let $\mathcal{L}^{\mathrm{w}} \in \mathfrak{L}^{\mathrm{w}}$ be an $\mathcal{N}_a$-LWES; by definition, $|\mathcal{L}^{\mathrm{w}}| \geq 1$. First, we show that there must exist at least one $\mathcal{N}_a$-LEP in $\mathcal{L}^{\mathrm{w}}$. For a contradiction, suppose that all points in $\mathcal{L}^{\mathrm{w}}$ are $\mathcal{N}_a$-LWEP's and not $\mathcal{N}_a$-LEP's. Let $\mathbf{x}^{\mathrm{w}} \in \mathcal{L}^{\mathrm{w}}$ and define $\tilde{\mathcal{N}}(\mathbf{x}^{\mathrm{w}}) := \{\tilde{\mathbf{x}} \in \mathcal{N}_a(\mathcal{L}^{\mathrm{w}}) \cap \mathcal{X} \colon \mathbf{g}(\tilde{\mathbf{x}}) \leq \mathbf{g}(\mathbf{x}^{\mathrm{w}})\}$, which is nonempty and finite since $\mathbf{x}^{\mathrm{w}}$ is not an $\mathcal{N}_a$-LEP. Thus the set $\tilde{\mathcal{N}}^*(\mathbf{x}^{\mathrm{w}}) := \{\mathbf{x}^* \in \tilde{\mathcal{N}}(\mathbf{x}^{\mathrm{w}}) \colon \nexists \tilde{\mathbf{x}} \in \tilde{\mathcal{N}}(\mathbf{x}^{\mathrm{w}}) \ni \mathbf{g}(\tilde{\mathbf{x}}) \leq \mathbf{g}(\mathbf{x}^*)\}$ is also nonempty, where $\tilde{\mathcal{N}}^*(\mathbf{x}^{\mathrm{w}}) \subseteq \tilde{\mathcal{N}}(\mathbf{x}^{\mathrm{w}}) \subseteq \mathcal{N}_a(\mathcal{L}^{\mathrm{w}}) \cap \mathcal{X}$. Now consider $\mathbf{x}^* \in \tilde{\mathcal{N}}^*(\mathbf{x}^{\mathrm{w}})$. Then by Definition 9, $\exists \mathbf{x}' \in \mathcal{L}^{\mathrm{w}}$ such that $\mathbf{g}(\mathbf{x}') \leqq \mathbf{g}(\mathbf{x}^*) \leq \mathbf{g}(\mathbf{x}^{\mathrm{w}})$, which implies $\mathbf{x}' \in \mathcal{L}^{\mathrm{w}} \cap \tilde{\mathcal{N}}(\mathbf{x}^{\mathrm{w}})$. If $\mathbf{g}(\mathbf{x}') \neq \mathbf{g}(\mathbf{x}^*)$, then we have a contradiction. Thus $\mathbf{g}(\mathbf{x}') = \mathbf{g}(\mathbf{x}^*)$, which implies $\mathbf{x}' = \mathbf{x}^*$ under Assumption 4. Then $\mathbf{x}^* \in \mathcal{L}^{\mathrm{w}} \cap \tilde{\mathcal{N}}^*(\mathbf{x}^{\mathrm{w}})$, which contradicts the fact that $\mathbf{x}^*$ is not an $\mathcal{N}_a$-LEP. Now consider the nonempty set $\mathcal{A} = \{\mathbf{x} \in \mathcal{L}^{\mathrm{w}} \colon \mathbf{x} \text{ is an } \mathcal{N}_a\text{-LEP}\}$; from this set, create the nonempty set $\mathcal{L}^* := \{\mathbf{x}^* \in \mathcal{A} \colon \nexists \mathbf{x} \in \mathcal{A} \ni \mathbf{g}(\mathbf{x}) \leq \mathbf{g}(\mathbf{x}^*)\}$, where $\mathcal{L}^* \subseteq \mathcal{L}^{\mathrm{w}}$. By construction and Assumption 4, Definition 10 holds for $\mathcal{L}^*$. □

Finally, we remark on Assumptions 5 and 6. Assumption 5, which subsumes Assumption 4, ensures that each feasible point is distinguishable on each objective.

Under this assumption, every $\mathcal{N}_a$-LWEP is an $\mathcal{N}_a$-LEP, and every $\mathcal{N}_a$-LWES is an $\mathcal{N}_a$-LES. Assumption 6 is required for the convergence of R-Pε, and stipulates that every $\mathcal{N}_a$-LWEP is a GEP, and there exists exactly one $\mathcal{N}_a$-LES, which is also the GES. Under this assumption, the R-Pε algorithm cannot "get stuck" by returning parts of different $\mathcal{N}_a$-LES's.

3.8.2 Convergence of R-PεRLE and R-MinRLE

We now consider the convergence of the algorithms R-PεRLE and R-MinRLE under the regularity conditions discussed in the previous section. These algorithms invoke RLE to guarantee that the set of points returned at the end of each RA iteration is an ALES. Theorem 1 and its proof are presented for $d \geq 2$ since R-MinRLE converges for two or more objectives. Indeed, given appropriate parameter values, the proof of convergence of Theorem 1 holds for any RA algorithm that invokes RLE as the last step in the sample-path solver.

Theorem 1. *Let Assumptions 1–3 hold. For any neighborhood size $a \in [1, \infty)$, initial point $\mathbf{x}_0 \in \mathcal{X}$, ε-placement rule, and completeness parameter $\beta_\delta \in (0, \infty]$,* R-P$\varepsilon$RLE *($d = 2$) and* R-MinRLE *($d \geq 2$) generate a sequence of estimated solutions $\{\hat{\mathcal{A}}_\nu\}$ such that*

(a) $\{\hat{\mathcal{A}}_\nu\}$ converges into an $\mathcal{N}_a$-LWES almost surely, that is, $\exists \mathcal{L}^{\mathrm{w}} \in \mathfrak{L}^{\mathrm{w}}$ such that $\mathbb{P}\{\hat{\mathcal{A}}_\nu \not\subseteq \mathcal{L}^{\mathrm{w}}$ i.o.$\} = 0$;

(b) under Assumption 4, $\{\hat{\mathcal{A}}_\nu\}$ contains an $\mathcal{N}_a$-LES infinitely often almost surely, that is, $\exists \mathcal{L} \in \mathfrak{L}$ such that $\mathbb{P}\{\mathcal{L} \not\subseteq \hat{\mathcal{A}}_\nu$ i.o.$\} = 0$;

(c) under Assumption 5, $\{\hat{\mathcal{A}}_\nu\}$ converges to an $\mathcal{N}_a$-LES almost surely, that is, $\exists \mathcal{L} \in \mathfrak{L}$ such that $\mathbb{P}\{\hat{\mathcal{A}}_\nu \neq \mathcal{L}$ i.o.$\} = 0$.

(d) under Assumptions 5 and 6, $\{\hat{\mathcal{A}}_\nu\}$ converges to the GES almost surely, that is, $\mathbb{P}\{\hat{\mathcal{A}}_\nu \neq \mathcal{E}$ i.o.$\} = 0$.

Theorem 1 presents a series of convergence results that require increasingly stringent assumptions on the underlying Problem M_d. At a minimum, under our required

Assumptions 1–3, R-PεRLE and R-MinRLE converge into an $\mathcal{N}_a$-LWES almost surely. If more restrictive assumptions hold, our algorithms converge to an $\mathcal{N}_a$-LES almost surely.

Proof. Proof of Theorem 1 Part (a). For every ν, R-PεRLE and R-MinRLE return a set $\hat{\mathcal{A}}_\nu$ in finite time. Thus both algorithms produce an infinite sequence of solutions $\{\hat{\mathcal{A}}_\nu\}$. Further, notice that R-PεRLE and R-MinRLE never return a set $\hat{\mathcal{A}}_\nu$ containing a point whose estimated objective vector is dominated by $\bar{\boldsymbol{G}}_{m_\nu}(\mathbf{x}_0)$ (see Algorithm 6, RLE Steps 1 and 14). Now consider the union of the level sets corresponding to the starting point $\mathbf{x}_0$, $\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})$. By Lemma 3, there exists $\tilde{\nu}$ such that for all $\nu \geq \tilde{\nu}$, $\hat{\mathcal{A}}_\nu \subseteq \hat{\mathcal{S}}_\nu(\mathbf{x}_0) \subseteq \mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})$ w.p.1. Since $\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})$ is finite, then any sequence of estimated efficient points $\{\boldsymbol{X}_\nu^* : \boldsymbol{X}_\nu^* \in \hat{\mathcal{A}}_\nu \text{ for all } \nu = 1, 2, \ldots\}$ is bounded w.p.1. Using an argument similar to that in (Wang et al., 2013, Theorem 5.4, p. 15), we now prove that $\{\hat{\mathcal{A}}_\nu\}$ converges into an $\mathcal{N}_a$-LWES w.p.1.

Since $\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})$ is finite and $a \in (0, \infty)$, then $\mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$ is also finite. Thus for all $k \in \{1, \ldots, d\}$, $\bar{G}_{k,m_\nu}(\cdot)$ uniformly converges to $g_k(\cdot)$ w.p.1 as $\nu \to \infty$ on the set $\mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$. Let the set $\mathcal{D}_k := \{(\mathbf{x}, \mathbf{x}') : \mathbf{x}, \mathbf{x}' \in \mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}, g_k(\mathbf{x}') \neq g_k(\mathbf{x})\}$ be the set of all pairs of feasible points in the level set neighborhood that have different true objective function values on objective k, and let $\kappa_1 = \min_{k \in \{1,\ldots,d\}} \inf_{\mathcal{D}_k} |g_k(\mathbf{x}) - g_k(\mathbf{x}')| > 0$ be the smallest difference in objective values across these pairs; Assumption 1 implies $\kappa_1 > 0$. Then w.p.1, there exists ν' (dependent on neighborhood size a, initial point $\mathbf{x}_0$, the constants κ_1 and $\boldsymbol{\alpha}$, and the random realization) such that for all $\nu \geq \nu'$, $\max_{k \in \{1,\ldots,d\}} |\bar{G}_{k,m_\nu}(\mathbf{x}) - g_k(\mathbf{x})| < \kappa_1/4$ for all $\mathbf{x} \in \mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$. Since $\beta_\delta \in (0, \infty]$, the ALES completeness function $\hat{\delta}_k(\cdot) = \hat{f}_k(\cdot, \beta_\delta) = \hat{\sigma}_{k,m_\nu}(\cdot)/m_\nu^{\beta_\delta}$ uniformly converges to zero w.p.1 on the finite set $\mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$ as $\nu \to \infty$ for all $k \in \{1, \ldots, d\}$. Then w.p.1, there exists ν'' (dependent on the same quantities as ν' and dependent on β_δ) such that for all $\nu \geq \nu''$, $\max_{k \in \{1,\ldots,d\}} \hat{\delta}_k(\mathbf{x}) < \kappa_1/4$ for all $\mathbf{x} \in \mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$.

Henceforth, let $\nu \geq \max\{\nu', \nu''\}$. Combining the above results, for all $\mathbf{x} \in \mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$, $\max_{k\in\{1,\ldots,d\}} \big||\bar{G}_{k,m_\nu}(\mathbf{x}) - \hat{\delta}_k(\mathbf{x})| - g_k(\mathbf{x})\big| < \kappa_1/2$ w.p.1. Thus for all $k \in \{1, \ldots, d\}$ and for all $\mathbf{x}, \mathbf{x}' \in \mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$, the following hold:

R1. if $g_k(\mathbf{x}) < g_k(\mathbf{x}')$, then $\bar{G}_{k,m_\nu}(\mathbf{x}) + \hat{\delta}_k(\mathbf{x}) < \bar{G}_{k,m_\nu}(\mathbf{x}') - \hat{\delta}_k(\mathbf{x}')$ w.p.1;

R2. if $\bar{G}_{k,m_\nu}(\mathbf{x}) + \hat{\delta}_k(\mathbf{x}) \leq \bar{G}_{k,m_\nu}(\mathbf{x}') - \hat{\delta}_k(\mathbf{x}')$, then $g_k(\mathbf{x}) \leq g_k(\mathbf{x}')$ w.p.1.

Further, for all $\mathbf{x}, \mathbf{x}' \in \mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$, if $\bar{\boldsymbol{G}}_{m_\nu}(\mathbf{x}) \nleqq \bar{\boldsymbol{G}}_{m_\nu}(\mathbf{x}')$, then $\exists k \in \{1, \ldots, d\}$ such that $\bar{G}_{k,m_\nu}(\mathbf{x}') < \bar{G}_{k,m_\nu}(\mathbf{x})$, implying that $g_k(\mathbf{x}') \leq g_k(\mathbf{x})$ w.p.1. This result, along with results R1 and R2 above, imply that for all $\mathbf{x}, \mathbf{x}' \in \mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$,

R3. if $\bar{\boldsymbol{G}}_{m_\nu}(\mathbf{x}) \nleqq \bar{\boldsymbol{G}}_{m_\nu}(\mathbf{x}')$, then $\mathbf{g}(\mathbf{x}) \not< \mathbf{g}(\mathbf{x}')$ w.p.1;

R4. if $\bar{\boldsymbol{G}}_{m_\nu}(\mathbf{x}) + \hat{\boldsymbol{\delta}}(\mathbf{x}) \leqq \bar{\boldsymbol{G}}_{m_\nu}(\mathbf{x}') - \hat{\boldsymbol{\delta}}(\mathbf{x}')$, then $\mathbf{g}(\mathbf{x}) \leqq \mathbf{g}(\mathbf{x}')$ w.p.1.

Now let $\nu \geq \max\{\tilde{\nu}, \nu', \nu''\}$. Then by Lemma 3, the set of decision points $\hat{\mathcal{A}}_\nu$ returned by each algorithm lie in $\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})$ w.p.1. We now consider all parts of the definition of an ALES (Definition 11). First, Algorithm 6 ensures no points in $\bar{\boldsymbol{G}}_n(\hat{\mathcal{A}}_\nu)$ dominate other points in $\bar{\boldsymbol{G}}_n(\hat{\mathcal{A}}_\nu)$ (RLE Steps 1 and 14). Thus result R3 above implies that no points in $\mathbf{g}(\hat{\mathcal{A}}_\nu)$ strictly dominate other points in $\mathbf{g}(\hat{\mathcal{A}}_\nu)$. Second, Algorithm 6 ensures each point in $\boldsymbol{X}_\nu^{\mathrm{w}} \in \hat{\mathcal{A}}_\nu$ is a sample-path $\mathcal{N}_a$-LWEP (e.g., RLE Steps 2, 5, 10, and 15). Thus result R3 above implies that all points in $\hat{\mathcal{A}}_\nu$ are $\mathcal{N}_a$-LWEP's. Third, Algorithm 6 ensures that the NCN of $\hat{\mathcal{A}}_\nu$ is empty (RLE Steps 2 and 15). Then applying results R3 and R4 above, for all $\boldsymbol{X} \in \mathcal{N}_a'(\hat{\mathcal{A}}_\nu) \cap \mathcal{X}$, (i) $\exists \boldsymbol{X}_\nu^{\mathrm{w}} \in \hat{\mathcal{A}}$ such that $\mathbf{g}(\boldsymbol{X}_\nu^{\mathrm{w}}) \leqq \mathbf{g}(\boldsymbol{X})$ w.p.1, or (ii) $\exists \boldsymbol{X}_\nu^{\mathrm{w}} \in \hat{\mathcal{A}}$ such that $(\mathbf{g}(\boldsymbol{X}) \leqq \mathbf{g}(\boldsymbol{X}_\nu^{\mathrm{w}})$ and $\bar{\boldsymbol{G}}_{m_\nu}(\boldsymbol{X}_\nu^{\mathrm{w}}) - \hat{\boldsymbol{\delta}}(\boldsymbol{X}_\nu^{\mathrm{w}}) \leqq \bar{\boldsymbol{G}}_{m_\nu}(\boldsymbol{X}) + \hat{\boldsymbol{\delta}}(\boldsymbol{X}))$ w.p.1, which happens with probability zero unless $\mathbf{g}(\boldsymbol{X}) = \mathbf{g}(\boldsymbol{X}_\nu^{\mathrm{w}})$, or (iii) employing the complements of the previous two conditions, $\forall \boldsymbol{X}_\nu^{\mathrm{w}} \in \hat{\mathcal{A}}$, $\mathbf{g}(\boldsymbol{X}_\nu^{\mathrm{w}}) \nleqq \mathbf{g}(\boldsymbol{X}) \nleqq \mathbf{g}(\boldsymbol{X}_\nu^{\mathrm{w}})$, and $\exists \tilde{\boldsymbol{X}}_\nu^{\mathrm{w}} \in \hat{\mathcal{A}}$ such that $\bar{\boldsymbol{G}}_{m_\nu}(\tilde{\boldsymbol{X}}_\nu^{\mathrm{w}}) - \hat{\boldsymbol{\delta}}(\tilde{\boldsymbol{X}}_\nu^{\mathrm{w}}) \leqq \bar{\boldsymbol{G}}_{m_\nu}(\boldsymbol{X}) + \hat{\boldsymbol{\delta}}(\boldsymbol{X})$ or $\bar{\boldsymbol{G}}_{m_\nu}(\boldsymbol{X}) - \hat{\boldsymbol{\delta}}(\boldsymbol{X}) \leqq \bar{\boldsymbol{G}}_{m_\nu}(\tilde{\boldsymbol{X}}_\nu^{\mathrm{w}}) + \hat{\boldsymbol{\delta}}(\tilde{\boldsymbol{X}}_\nu^{\mathrm{w}})$ w.p.1, which happens with probability zero. To see this, let $\boldsymbol{X} \in \mathcal{N}_a'(\hat{\mathcal{A}}_\nu) \cap \mathcal{X}$ and notice that the condition $\forall \boldsymbol{X}_\nu^{\mathrm{w}} \in \hat{\mathcal{A}}$, $\mathbf{g}(\boldsymbol{X}_\nu^{\mathrm{w}}) \nleqq \mathbf{g}(\boldsymbol{X}) \nleqq \mathbf{g}(\boldsymbol{X}_\nu^{\mathrm{w}})$ w.p.1 implies that $\forall \boldsymbol{X}_\nu^{\mathrm{w}} \in \hat{\mathcal{A}}$, $\exists k_1, k_2 \in \{1, \ldots, d\}$ such that $g_{k_1}(\boldsymbol{X}_\nu^{\mathrm{w}}) > g_{k_1}(\boldsymbol{X})$ and $g_{k_2}(\boldsymbol{X}_\nu^{\mathrm{w}}) < g_{k_2}(\boldsymbol{X})$ w.p.1, and hence by result R1 above, $\bar{G}_{k_1,m_\nu}(\boldsymbol{X}_\nu^{\mathrm{w}}) - \hat{\delta}_{k_1}(\boldsymbol{X}_\nu^{\mathrm{w}}) > \bar{G}_{k_1,m_\nu}(\boldsymbol{X}) + \hat{\delta}_{k_1}(\boldsymbol{X})$ and $\bar{G}_{k_2,m_\nu}(\boldsymbol{X}) - \hat{\delta}_{k_2}(\boldsymbol{X}) > \bar{G}_{k_2,m_\nu}(\boldsymbol{X}_\nu^{\mathrm{w}}) + \hat{\delta}_{k_2}(\boldsymbol{X}_\nu^{\mathrm{w}})$ w.p.1.

Therefore when $\nu \geq \max\{\tilde{\nu}, \nu', \nu''\}$, R-P$\varepsilon$RLE and R-MinRLE certify that w.p.1, all points in $\hat{\mathcal{A}}_\nu$ are $\mathcal{N}_a$-LWEP's, no points in $\mathbf{g}(\hat{\mathcal{A}}_\nu)$ strictly dominate other points in $\mathbf{g}(\hat{\mathcal{A}}_\nu)$, and for all $\boldsymbol{X} \in \mathcal{N}_a'(\hat{\mathcal{A}}_\nu) \cap \mathcal{X}$, $\exists \boldsymbol{X}_\nu^{\mathrm{w}} \in \hat{\mathcal{A}}$ such that $\mathbf{g}(\boldsymbol{X}_\nu^{\mathrm{w}}) \leqq \mathbf{g}(\boldsymbol{X})$. Thus by Definition 9, $\hat{\mathcal{A}}_\nu$ is an $\mathcal{N}_a$-LWES w.p.1. Further, each $\hat{\mathcal{A}}_\nu$ is such that there does not exist a pair of points $\boldsymbol{X}_{\nu-1}^* \in \hat{\mathcal{A}}_{\nu-1}$ and $\boldsymbol{X}_\nu^* \in \hat{\mathcal{A}}_\nu$ such that $\mathbf{g}(\boldsymbol{X}_{\nu-1}^*) < \mathbf{g}(\boldsymbol{X}_\nu^*)$ w.p.1. (The second part follows because we carry forward the points from $\hat{\mathcal{A}}_{\nu-1}$ into the νth iteration, and we ensure that no points in $\bar{\boldsymbol{G}}_n(\hat{\mathcal{A}}_\nu)$ dominate other points in $\bar{\boldsymbol{G}}_n(\hat{\mathcal{A}}_\nu)$ in Algorithm 6, RLE Steps 1 and 14.) Therefore if $\nu \geq \max\{\tilde{\nu}, \nu', \nu''\}$, there exists $\mathcal{L}^{\mathrm{w}} \in \mathfrak{L}^{\mathrm{w}}$ such that R-PεRLE and R-MinRLE returns $\hat{\mathcal{A}}_\nu \subseteq \mathcal{L}^{\mathrm{w}} \subseteq \mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})$ w.p.1. □

Proof. Proof of Theorem 1 Part (b). Let $\nu \geq \max\{\tilde{\nu}, \nu', \nu''\}$. By the proof of Theorem 1 Part (a), $\hat{\mathcal{A}}_\nu$ is an $\mathcal{N}_a$-LWES w.p.1. Under Assumption 4, by Lemma 4, there exists $\mathcal{L} \in \mathfrak{L}$ such that $\hat{\mathcal{A}}_\nu \supseteq \mathcal{L}$ w.p.1, and the result holds. □

Proof. Proof of Theorem 1 Part (c). Assumption 5 implies that all $\mathcal{N}_a$-LWES's are $\mathcal{N}_a$-LES's. Thus $\mathfrak{L}^{\mathrm{w}} = \mathfrak{L}$, and the result follows from Theorem 1 Parts (a) and (b). □

Proof. Proof of Theorem 1 Part (d). Assumptions 5 and 6 imply that the $\mathcal{N}_a$-LES in Theorem 1 Part (c) is the GES, and the result follows. □

3.8.3 Convergence of R-Pε

We now consider the convergence of R-Pε, which does not rely on RLE to certify that each RA iteration returns an ALES. Since R-Pε is an algorithm designed for exactly two objectives, henceforth in this section, we let $d = 2$.

To show the convergence of R-Pε in Theorem 2, first, we notice that for each objective $k \in \{1, 2\}$, the sequence of sample-path $\mathcal{N}_a$-local minimizers produced by Pε in Step 1 across RA iterations, defined as $\{\bar{\mathcal{M}}_\nu, \nu = 1, 2, \ldots\}$ where $\bar{\mathcal{M}}_\nu = \{\boldsymbol{X}_{1,m_\nu}^{\min}, \boldsymbol{X}_{2,m_\nu}^{\min}\}$ for all $\nu = 1, 2, \ldots$, converges into the set of all true $\mathcal{N}_a$-local minimizers of objective g_k over the feasible set $\mathcal{X}$, $\mathcal{M}_a^* \subseteq \mathcal{X}$, almost surely as $\nu \to \infty$. Since

this result, presented in Lemma 5, follows almost directly from (Wang et al., 2013) under Assumptions 1 and 2, we do not provide a proof. The proof of convergence of R-Pε requires our most restrictive assumptions on the underlying Problem M_2.

Lemma 5. *Under Assumptions 1 and 2, for $d = 2$, any neighborhood size $a \in [1, \infty)$, initial point $\mathbf{x}_0 \in \mathcal{X}$, and ε-placement rule, across RA iterations, Pε Step 1 generates a sequence of sample-path $\mathcal{N}_a$-local minimizers $\{\bar{\mathcal{M}}_\nu\}$ that converges into $\mathcal{M}_a^*$ almost surely, that is, $\mathbb{P}\{\bar{\mathcal{M}}_\nu \not\subseteq \mathcal{M}_a^* \text{ i.o.}\} = 0$.*

Proof. In each iteration ν, R-PεRLE returns a solution $\hat{\mathcal{A}}_\nu$ in finite time. Consequently, R-PεRLE generates an infinite sequence of local solutions $\{\boldsymbol{X}_{k,\nu}^{*\min}\}$ for each objective $k \in \{1, 2\}$. We now prove that for each $k \in \{1, 2\}$, $\{\boldsymbol{X}_{k,\nu}^{*\min}\}$ converges into $\mathcal{M}_k^*(\mathcal{N})$ w.p.1., using a proof similar to that in (Wang et al., 2013, Theorem 5.4, p. 15).

Let $k \in \{1, 2\}$, and notice that R-PεRLE guarantees to return an EALES containing local minima no worse than the given starting point $\mathbf{x}_0$ for every ν. Now consider the level sets corresponding to the initial starting point $\mathbf{x}_0$. Then under Assumptions 1 and 2, by Lemma 3, there exists a value $\tilde{\nu}_k$, dependent on $\mathbf{x}_0$, $\boldsymbol{\alpha}$, and the random realization, such that for all $\nu \geq \tilde{\nu}_k$, $\hat{\mathcal{S}}_{k,\nu}(\mathbf{x}_0) \subseteq \mathcal{S}_k(\mathbf{x}_0, \alpha_k)$ w.p.1. Then for all $\nu \geq \tilde{\nu}_k$, $\boldsymbol{X}_{k,\nu}^{*\min} \in \hat{\mathcal{S}}_{k,\nu}(\mathbf{x}_0) \subseteq \mathcal{S}_k(\mathbf{x}_0, \alpha_k)$ w.p.1. Since the level set $\mathcal{S}_k(\mathbf{x}_0, \alpha_k)$ is finite by Assumption 1, the sequence of minima $\{\boldsymbol{X}_{k,\nu}^{*\min}\}$ is bounded.

We now show that $\{\boldsymbol{X}_{k,\nu}^{*\min}\}$ is non-increasing and is contained in $\mathcal{S}_k(\mathbf{x}_0, \alpha_k)$ w.p.1 as in (Wang et al., 2013, Theorem 5.4, p. 15). Let $\tilde{\nu} = \max(\tilde{\nu}_1, \tilde{\nu}_2)$ and let $\kappa_1 = \min_{k \in \{1,2\}} \inf_{\mathcal{D}_k} |g_k(\mathbf{x}) - g_k(\mathbf{x}')|$, where $\mathcal{D}_k := \{(\mathbf{x}, \mathbf{x}') : \mathbf{x}, \mathbf{x}' \in \mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha}), g_k(\mathbf{x}') \neq g_k(\mathbf{x})\}$ is the set of all pairs of points in the level set that have different true objective function values on objective k. Since $\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})$ is finite, then $\mathcal{N}(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$ is also finite. Thus for all $k \in \{1, 2\}$, $\bar{G}_{k,m_\nu}$ uniformly converges to g_k w.p.1 on the set $\mathcal{N}(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$ as $\nu \to \infty$. Let $\kappa_1 = \min_{k \in \{1,2\}} \inf_{\mathcal{D}_k} |g_k(\mathbf{x}) - g_k(\mathbf{x}')|$, where $\mathcal{D}_k := \{(\mathbf{x}, \mathbf{x}') : \mathbf{x}, \mathbf{x}' \in \mathcal{N}(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}, g_k(\mathbf{x}') \neq g_k(\mathbf{x})\}$ is the set of all pairs of feasible points in the level set neighborhood that have different true objective function values on objective k. Then w.p.1, there exists ν' (dependent on $a, \kappa_1, \mathbf{x}_0, \boldsymbol{\alpha}$, and the

random realization) such that for all $\nu \geq \nu'$, $\max_{k\in\{1,2\}} |\bar{G}_{k,m_\nu}(\mathbf{x}) - g_k(\mathbf{x})| < \kappa_1/2$ for all $\mathbf{x} \in \mathcal{N}(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$. Therefore for all $k \in \{1,2\}$, if $g_k(\mathbf{x}) < g_k(\mathbf{x}')$, then w.p.1 for all $\nu \geq \nu'$, we have $\bar{G}_{k,m_\nu}(\mathbf{x}) < \bar{G}_{k,m_\nu}(\mathbf{x}')$ for all $\mathbf{x}', \mathbf{x} \in \mathcal{N}(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$. Then it also follows that for all $k \in \{1,2\}$, if $\nu \geq \nu'$ and $\bar{G}_{k,m_\nu}(\mathbf{x}) \leq \bar{G}_{k,m_\nu}(\mathbf{x}')$, we have $g_k(\mathbf{x}) \leq g_k(\mathbf{x}')$ for all $\mathbf{x}', \mathbf{x} \in \mathcal{N}(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$ w.p.1. Combining these results across objectives k, if $\nu \geq \nu'$ and $\bar{\boldsymbol{G}}_{m_\nu}(\mathbf{x}) \leq \bar{\boldsymbol{G}}_{m_\nu}(\mathbf{x}')$, we have $\mathbf{g}(\mathbf{x}) \leq \mathbf{g}(\mathbf{x}')$ for all $\mathbf{x},' \mathbf{x} \in \mathcal{N}(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$ w.p.1. Since SPLINE returns a sample-path local minimum, then $\boldsymbol{X}^{*\min}_{k,\nu} \in \mathcal{M}^*_k(\mathcal{N})$ if ν is sufficiently large and $\mathcal{N}'(\boldsymbol{X}^{*\min}_{k,\nu}) \subseteq \mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})$. Thus, for $\nu \geq \max(\tilde{\nu}, \nu')$ every $\boldsymbol{X}^{*\min}_{k,\nu} \in \mathcal{M}^*_k(\mathcal{N})$ w.p.1. □

Theorem 2. *Under Assumptions 1–6, for $d = 2$, any neighborhood size $a \in [1, \infty)$, initial point $\mathbf{x}_0 \in \mathcal{X}$, and ε-placement rule specified by $\beta_\varepsilon \in (0, \infty)$,* R-P$\varepsilon$ *generates a sequence of estimated solutions $\{\hat{\mathcal{A}}_\nu\}$ that converges almost surely to the global efficient set $\mathcal{E}$, in the sense that $\mathbb{P}\{\hat{\mathcal{A}}_\nu \neq \mathcal{E}$ i.o.$\} = 0$.*

Proof. For every ν, R-Pε returns a set $\hat{\mathcal{A}}_\nu$ in finite time, thus producing an infinite sequence of solutions $\{\hat{\mathcal{A}}_\nu\}$. Further, R-P$\varepsilon$ never returns a set $\hat{\mathcal{A}}_\nu$ containing a point whose estimated objective vector is dominated by $\bar{\boldsymbol{G}}_{m_\nu}(\mathbf{x}_0)$ (see Algorithm 3, Pε Step 26). Recall that for all $\nu \geq \tilde{\nu}$, $\hat{\mathcal{A}}_\nu \subseteq \hat{\mathcal{S}}_\nu(\mathbf{x}_0) \subseteq \mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})$ w.p.1, and since $\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})$ is finite, then any sequence of estimated efficient points $\{\boldsymbol{X}^*_\nu : \boldsymbol{X}^*_\nu \in \hat{\mathcal{A}}_\nu$ for all $\nu = 1, 2, \ldots\}$ is bounded w.p.1.

As in the proof of Theorem 1 Part (a), for all $k \in \{1,2\}$, $\bar{G}_{k,m_\nu}(\cdot)$ uniformly converges to $g_k(\cdot)$ w.p.1 as $\nu \to \infty$ on the finite set $\mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$. Since there are only two objectives and $\beta_\varepsilon \in (0, \infty)$, the maximum ε-placement distance $\max_{k^{\text{con}}\in\{1,2\}} \hat{f}_{k^{\text{con}}}(\cdot, \beta_\varepsilon) = \max_{k^{\text{con}}\in\{1,2\}} \hat{\sigma}_{k^{\text{con}},m_\nu}(\cdot)/m_\nu^{\beta_\varepsilon}$ also uniformly converges to zero w.p.1 as $\nu \to \infty$ on $\mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$. Let $\kappa > 0$ be as in Assumption 5. Then w.p.1, there exists $\nu'_{\text{P}\varepsilon}$ (dependent on $a, \mathbf{x}_0, \kappa, \boldsymbol{\alpha}$, and the random realization) such that for all $\nu \geq \nu'_{\text{P}\varepsilon}$ and all $\mathbf{x} \in \mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$, we have $\max_{k\in\{1,2\}} |\bar{G}_{k,m_\nu}(\mathbf{x}) - g_k(\mathbf{x})| < \kappa/4$ w.p.1. Also w.p.1, there exists $\nu''_{\text{P}\varepsilon}$ (dependent on the same quantities as $\nu'_{\text{P}\varepsilon}$ and dependent on β_ε) such that for all $\nu \geq \nu''_{\text{P}\varepsilon}$, $\max_{k^{\text{con}}\in\{1,2\}} \hat{f}_{k^{\text{con}}}(\mathbf{x}, \beta_\varepsilon) < \kappa/4$ for all $\mathbf{x} \in \mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$. Combining the above results, if $\nu \geq \max\{\nu'_{\text{P}\varepsilon}, \nu''_{\text{P}\varepsilon}\}$, then for

all $\mathbf{x} \in \mathcal{N}_a(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$, $\max_{k^{\text{con}} \in \{1,2\}} \left| |\bar{G}_{k^{\text{con}},m_\nu}(\mathbf{x}) - \hat{f}_{k^{\text{con}}}(\mathbf{x}, \beta_\varepsilon)| - g_{k^{\text{con}}}(\mathbf{x}) \right| < \kappa/2$ w.p.1. Henceforth, let $\nu \geq \max\{\nu'_{\text{P}\varepsilon}, \nu''_{\text{P}\varepsilon}\}$. Then for all $\mathbf{x}, \mathbf{x}' \in \mathcal{N}(\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})) \cap \mathcal{X}$, the following hold w.p.1:

R5. $\forall k \in \{1, 2\}$,

$g_k(\mathbf{x}) < g_k(\mathbf{x}')$ if and only if $\bar{G}_{k,m_\nu}(\mathbf{x}) + \hat{f}_k(\mathbf{x}, \beta_\varepsilon) < \bar{G}_{k,m_\nu}(\mathbf{x}') - \hat{f}_k(\mathbf{x}', \beta_\varepsilon)$;

R6. if $\bar{\boldsymbol{G}}_{m_\nu}(\mathbf{x}) \nleq \bar{\boldsymbol{G}}_{m_\nu}(\mathbf{x}')$, then

$\mathbf{g}(\mathbf{x}) \nleqq \mathbf{g}(\mathbf{x}')$, that is, $\exists k \in \{1, 2\}$ such that $g_k(\mathbf{x}') < g_k(\mathbf{x})$.

Under Assumption 1, for any $\mathbf{x}_0 \in \mathcal{X}$, $\mathcal{E} \subseteq \mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})$. Therefore results R5, R6 and Assumptions 5–6 imply that all points in $\mathcal{E}$ are both sample-path $\mathcal{N}_a$-LWEP's and sample-path global efficient points w.p.1. Further, all points in $\mathcal{N}'_a(\mathcal{E})$ are *not* sample-path $\mathcal{N}_a$-LWEP's w.p.1. Let $c_\varepsilon := |\mathcal{E}| \geq 1$, and for any objective $k \in \{1, 2\}$, sort the elements of $\mathcal{E}$ on objective k so that $g_k(\mathbf{x}^*_{k(1)}) < \ldots < g_k(\mathbf{x}^*_{k(c_\varepsilon)})$, where $\mathbf{x}^*_{k(i)}$ denotes the ith ordered element of $\mathcal{E}$ on objective k, $i = 1, \ldots, c_\varepsilon$. If $c_\varepsilon \geq 2$, then result R5 implies that w.p.1 for all $i = 1, \ldots, c_\varepsilon - 1$,

$$\bar{G}_{k^{\text{con}},m_\nu}(\mathbf{x}^*_{k(i)}) + \hat{f}_{k^{\text{con}}}(\mathbf{x}^*_{k(i)}, \beta_\varepsilon) < \bar{G}_{k,m_\nu}(\mathbf{x}^*_{(i+1)}) - \hat{f}_{k^{\text{con}}}(\mathbf{x}^*_{(i+1)}). \tag{3.1}$$

By Lemma 5, w.p.1 there exists ν''' (dependent on the same quantities as $\nu'_{\text{P}\varepsilon}$) such that for all $\nu \geq \nu'''$, the updated sample-path $\mathcal{N}_a$-local minimizers returned as part of Pε Step 1, which we call $\bar{\mathcal{M}}_\nu$, are such that $\bar{\mathcal{M}}_\nu \subseteq \mathcal{M}^*_a$. Under Assumptions 5–6, the set $\mathcal{M}^*_a = \{\mathbf{x}^{\min}_1, \mathbf{x}^{\min}_2\}$ contains the unique global minimizers for each objective $k \in \{1, 2\}$.

Henceforth, let $\nu > \max\{\tilde{\nu}, \nu'_{\text{P}\varepsilon}, \nu''_{\text{P}\varepsilon}, \nu'''\}$, and let $\{k^*_\nu, \nu = 1, 2, \ldots\}$ be any sequence of objectives minimized, where $k^{\text{con}}_\nu \neq k^*_\nu$ for each ν. Then by Lemma 3, the set of decision points $\hat{\mathcal{A}}_\nu$ returned by R-Pε lie in $\mathcal{S}(\mathbf{x}_0, \boldsymbol{\alpha})$ w.p.1, as does the set of points used to set the ε values in Algorithm 3, Pε Step 2, which is a set of sample-path $\mathcal{N}_a$-LWEP's we call $\hat{\mathcal{A}}^{\text{w}}_\nu$. Since all points in $\hat{\mathcal{A}}^{\text{w}}_\nu$ are sample-path $\mathcal{N}_a$-LWEP's, then results R5, R6 and Assumptions 5–6 ensure that $\hat{\mathcal{A}}^{\text{w}}_\nu \subseteq \mathcal{E}$ w.p.1; further, $\bar{\mathcal{M}}_\nu \subseteq \hat{\mathcal{A}}^{\text{w}}_\nu$, where $\bar{\mathcal{M}}_\nu = \{\mathbf{x}^{\min}_1, \mathbf{x}^{\min}_2\}$ w.p.1. If $c_\varepsilon \in \{1, 2\}$, the proof is complete, since $\hat{\mathcal{A}}^{\text{w}}_\nu$ is returned as $\hat{\mathcal{A}}_\nu$ in Algorithm 3, Pε Step 26, and no other points have entered the set w.p.1. Now suppose $c_\varepsilon \geq 3$. All points in $\hat{\mathcal{A}}^{\text{w}}_\nu \cup \mathcal{E}$ can be ordered on k^{con} as in line

(3.1). Points in $\mathcal{E} \setminus \hat{\mathcal{A}}_\nu^{\mathrm{w}}$ are retrieved by Algorithm 3, Pε Steps 15–25, and carried forward to $\hat{\mathcal{A}}_{\nu+1}$; no other points enter the set w.p.1. Then it follows that for all $\nu^* > \nu + 1$, $\hat{\mathcal{A}}_{\nu^*} = \mathcal{E}$ w.p.1, and the result holds. □

3.8.4 Sampling Efficiency

Finally, we provide a result on the sampling efficiency of our algorithms. This result provides insight into how to set the algorithm parameter values in §3.9 to achieve exponential convergence. In Theorem 3, let $\mathcal{X}^{\mathrm{w}}$ denote the set of all $\mathcal{N}_1$-LWEP's for Problem M_d, and let $\bar{\mathcal{X}}_\nu^{\mathrm{w}}$ denote the set of all sample-path $\mathcal{N}_1$-LWEP's on the νth RA iteration. Further, let $\hat{\mathcal{A}}_\nu$ denote the solution returned on the νth RA iteration of R-PεRLE ($d = 2$), R-Pε ($d = 2$), or R-MinRLE ($d \geq 2$) for any $\mathbf{x}_0 \in \mathcal{X}$, ε-placement rule $\beta_\varepsilon \in (0, \infty)$, and completeness parameter $\beta_\delta \in (0, \infty]$.

Theorem 3. *Let the neighborhood size $a = 1$ and suppose the feasible set $\mathcal{X} \subset \mathbb{Z}^q$ is finite with $\max_{k\in\{1,\ldots,d\}} \sup_{\mathbf{x}\in\mathcal{X}} \sigma_k^2(\mathbf{x}) < \infty$. For all objectives $k \in \{1, \ldots, d\}$, let the sequence of random variables $\{\bar{G}_{k,m_\nu}(\mathbf{x}) - g_k(\mathbf{x})\}$ be governed by a large-deviation principle with rate function $I_{k,\mathbf{x}}(s)$, as stipulated in Lemma 2. Then the following hold:*

(a) $\mathbb{P}\{\bar{\mathcal{X}}_\nu^{\mathrm{w}} \not\subseteq \mathcal{X}^{\mathrm{w}}\} = O(e^{-\gamma m_\nu})$ *for some* $\gamma > 0$.

(b) If the sequence of sample sizes increases to infinity at least linearly in R-PεRLE, R-Pε, *and* R-MinRLE, *that is, if* $\limsup_{\nu\to\infty} m_\nu^{-1}\nu < \infty$, *then*

(i) $\mathbb{P}\{\hat{\mathcal{A}}_\nu \not\subseteq \mathcal{X}^{\mathrm{w}}\} = O(e^{-\gamma m_\nu})$ *for some* $\gamma > 0$,

(ii) under Assumption 5 and 6, $\mathbb{P}\{\hat{\mathcal{A}}_\nu \neq \mathcal{E}\} = O(e^{-\gamma m_\nu})$ *for some* $\gamma > 0$.

Proof. Proof of Theorem 3 Part (a). Let $\mathcal{D} \subseteq \mathcal{X}$ be any subset of the feasible set. Since $\mathcal{X}$ is finite, $\mathcal{D}$ is finite. Let $\bar{\mathcal{B}}_{k,\nu}^*(\mathcal{D})$ denote the set of sample-path global minimizers of objective g_k, $k \in \{1, \ldots, d\}$ on the set $\mathcal{D}$, and let $\mathcal{B}_k^*(\mathcal{D})$ denote the corresponding set of true global minimizers on $\mathcal{D}$. Then under our assumptions, by

(Wang et al., 2013, p. 16), for all $k \in \{1, \ldots, d\}$ and all $\mathcal{D} \subseteq \mathcal{X}$, there exists $\eta > 0$ such that for large enough ν,

$$\mathbb{P}\{\bar{\mathcal{B}}^*_{k,\nu}(\mathcal{D}) \not\subseteq \mathcal{B}^*_k(\mathcal{D})\} \leq |\mathcal{D}| e^{-m_\nu \eta}. \tag{3.2}$$

Recall that $\mathcal{X} \subset \mathbb{Z}^q$ and let $\mathbf{x} \in \mathcal{X}$ be a feasible point. Letting $\mathbf{e}_i$ denote a q-dimensional vector of zeros with one in the ith place, divide $\mathcal{N}_1(\mathbf{x})$ into $2q$ sub-neighborhoods that include $\mathbf{x}$ and exactly one other neighborhood point in each direction, $\mathcal{N}_{1,+i}(\mathbf{x}) := \{\mathbf{x}, \mathbf{x} + \mathbf{e}_i\}$ and $\mathcal{N}_{1,-i}(\mathbf{x}) := \{\mathbf{x}, \mathbf{x} - \mathbf{e}_i\}$ for all $i \in \{1, \ldots, q\}$.

For every non-$\mathcal{N}_1$-LWEP $\mathbf{x} \in \mathcal{X} \setminus \mathcal{X}^{\mathrm{w}}$, there must exist $\mathbf{x}' \in \mathcal{N}_1(\mathbf{x}) \cap \mathcal{X}$ such that $\mathbf{g}(\mathbf{x}')$ strictly dominates $\mathbf{g}(\mathbf{x})$. Then for every $\mathbf{x} \in \mathcal{X} \setminus \mathcal{X}^{\mathrm{w}}$, there exists $j \in \{-q, \ldots, -1, 1, \ldots, q\}$ and $\mathbf{x}' \in \mathcal{N}_1(\mathbf{x}) \cap \mathcal{X}$ such that $\mathcal{N}_{1,j}(\mathbf{x}) = \{\mathbf{x}, \mathbf{x}'\}$ and $\mathbf{g}(\mathbf{x}')$ strictly dominates $\mathbf{g}(\mathbf{x})$. Thus $\mathbf{x}$ is not a global minimizer on $\mathcal{N}_{1,j}(\mathbf{x})$ on any objective. If $\mathbf{x} \in \mathcal{X} \setminus \mathcal{X}^{\mathrm{w}}$ is nonetheless estimated as an $\mathcal{N}_1$-LWEP, that is, $\mathbf{x} \in \bar{\mathcal{X}}^{\mathrm{w}}_\nu$ on its $\mathcal{N}_1$-neighborhood, there must exist an objective $k \in \{1, \ldots, d\}$ such that $\bar{\mathcal{B}}^*_{k,\nu}(\mathcal{N}_{1,j}(\mathbf{x})) \not\subseteq \mathcal{B}^*_k(\mathcal{N}_{1,j}(\mathbf{x}))$. Then for large enough ν,

$$\begin{aligned}
\mathbb{P}\{\bar{\mathcal{X}}^{\mathrm{w}}_\nu \not\subseteq \mathcal{X}^{\mathrm{w}}\} &\leq \sum_{\mathbf{x} \in \mathcal{X} \setminus \mathcal{X}^{\mathrm{w}}} \mathbb{P}\{\mathbf{x} \in \bar{\mathcal{X}}^{\mathrm{w}}_\nu\} \\
&\leq \sum_{\mathbf{x} \in \mathcal{X} \setminus \mathcal{X}^{\mathrm{w}}} \sum_{j \in \{-q, \ldots, -1, 1, \ldots, q\}} \sum_{k \in \{1, \ldots, d\}} \mathbb{P}\{\bar{\mathcal{B}}^*_{k,\nu}(\mathcal{N}_{1,j}(\mathbf{x})) \not\subseteq \mathcal{B}^*_k(\mathcal{N}_{1,j}(\mathbf{x}))\} \\
&\leq \sum_{\mathbf{x} \in \mathcal{X} \setminus \mathcal{X}^{\mathrm{w}}} \sum_{j \in \{-q, \ldots, -1, 1, \ldots, q\}} \sum_{k \in \{1, \ldots, d\}} 2e^{-m_\nu \eta} \quad \leq \quad |\mathcal{X}| 4qd e^{-m_\nu \eta},
\end{aligned}$$

where $\eta > 0$ denotes the relevant constant from line (3.2). □

Proof. Proof of Theorem 3 Part (b). We begin by noticing that item (i) follows from Theorem 3 Part (a), along with the assumption that sample sizes increase at least linearly and the fact that our algorithms guarantee $\hat{\mathcal{A}}_\nu$ contains only sample-path $\mathcal{N}_1$-LWEP's.

To prove item (ii), notice that under our assumptions, all $\mathcal{N}_1$-LWEP's are global efficient points. Therefore by item (i), $\mathbb{P}\{\hat{\mathcal{A}}_\nu \not\subseteq \mathcal{E}\} = O(e^{-\gamma m_\nu})$ for some $\gamma > 0$. We now consider $\mathbb{P}\{\mathcal{E} \not\subseteq \hat{\mathcal{A}}_\nu\}$, where $\mathbb{P}\{\mathcal{E} \not\subseteq \hat{\mathcal{A}}_\nu\} \leq \sum_{\mathbf{x} \in \mathcal{E}} \mathbb{P}\{\mathbf{x} \notin \hat{\mathcal{A}}_\nu\}$. Notice that if $\mathbf{x} \in \mathcal{E}$ is not in $\hat{\mathcal{A}}_\nu$, then it must have been incorrectly estimated as dominated by at least

one point in its neighborhood. Then by a proof similar to that of Theorem 3 Part (a), it follows that $\mathbb{P}\{\mathcal{E} \not\subseteq \hat{\mathcal{A}}_\nu\} = O(e^{-\gamma m_\nu})$ for some $\gamma > 0$, which implies the result. $\square$

3.9 Algorithm Parameters and Implementation

We now discuss the details of algorithm implementation and the choice of four algorithm parameters. First, for algorithmic efficiency, everywhere the oracle is called at a point $\mathbf{x}$ with sample size $n = m_\nu$ within an RA iteration, we assume the triple $(\mathbf{x}, \bar{\boldsymbol{G}}_n(\mathbf{x}), \widehat{\text{s.e.}}(\bar{\boldsymbol{G}}_n(\mathbf{x})))$ is stored and is made available to all relevant subroutines within an RA iteration. Thus everywhere a candidate ALES is passed between functions, we assume the estimated objective function values of the neighborhood points are made available to all relevant subroutines, especially to RLE. This practice enhances efficiency by removing the need to re-sample neighborhood points when checking whether a candidate ALES is truly an ALES. All stored points visited and simulation replications obtained is cleared between RA iterations.

While all of our definitions and algorithms allow for a flexible neighborhood specified by the parameter a, as noted by Wang et al. (2013), there exists a tension between the relatively faster convergence enabled by $a = 1$ and the certification of the local solution as optimal in a larger neighborhood. By default, we set $a = 1$. For larger a and for high-dimensional feasible spaces, Wang et al. (2013) remark that the neighborhood enumeration NE routine inside SPLINE may be modified to return only a better point, rather than the best point in the neighborhood. Further, the algorithms GetNCN and RemoveNonLWEP inside RLE can be modified to return only a subset of the non-conforming points encountered in the neighborhood or the first sample-path $\mathcal{N}_a$-LWEP encountered, respectively, rather than all such points. These modifications do not affect the convergence properties of the algorithm.

We now discuss the default sample size sequences and relaxation parameters. By default, we set the monotone-increasing sample size sequence as $m_\nu = \lceil 2 \times 1.1^\nu \rceil$ for all $\nu \geq 1$. This sequence satisfies the requirements of Lemma 2 and Theorem 3 in

§3.8. To ensure every search terminates in finite time, but that for large enough ν, the sample size limit b_ν will not be reached, we set the sequence $b_\nu = \lceil 8 \times 1.2^\nu \rceil$ for all $\nu \geq 1$. Notice that each search we conduct inside Pε and RLE gets a fresh limiting sample size.

We control the placement of the ε values in Pε and the completeness of the ALES returned by RLE using the parameters β_ε and β_δ. Since our algorithm converges for a wide variety of β_ε and β_δ values, there is significant flexibility in setting these parameters. By default, we set $\beta_\varepsilon = \beta_\delta = 0.5$ in all of our algorithms. Thus we search for new sample-path $\mathcal{N}_1$-LEP's that are more than one standard error away from the ones we have in every objective. We numerically explore the effect of these parameters on our algorithm performance in §3.10.

3.10 Numerical Experiments

First, we conduct numerical experiments that compare our main algorithm, R-PεRLE, to the benchmark algorithm R-MinRLE and the current state-of-the-art, MO-COMPASS. Then, we explore the performance of R-PεRLE across a variety of $\boldsymbol{\beta} = (\beta_\varepsilon, \beta_\delta)$ values.

3.10.1 Algorithm Performance with Default Parameter Values

We compare the performances of R-PεRLE, R-MinRLE, and MO-COMPASS on three increasingly complicated test problems. We configure our numerical experiments as follows. In each independent run of an algorithm, we use an initial point $\mathbf{x}_0$ that is generated uniformly from the feasible set $\mathcal{X}$, which is finite in our test problems. Within an algorithm run, we use CRN across points visited. R-PεRLE and R-MinRLE use the default parameter values described in the previous section. We configure MO-COMPASS, including the simulation allocation rule (SAR), as close as possible to the settings used in (Li et al., 2015b, p. 10). In the quantile plots that

follow, the algorithm performance at each value of the total simulation budget t is dependent on its previous performance.

Test Problem A

Our first test problem is a modified version of a problem that appears in Kim and Ryu (2011a). We define Problem T_A as

$$\text{Problem } T_A\text{:} \qquad \text{minimize}_{\mathbf{x}\in\mathcal{X}} \begin{cases} g_1(\mathbf{x}) = \mathbb{E}[(x_1/10 - 2\xi_1)^2 + (x_2/10 - \xi_2)^2] \\ g_2(\mathbf{x}) = \mathbb{E}[x_1^2/100 + (x_2/10 - 2\xi_3)^2] \end{cases}$$

where $\mathcal{X} = \tilde{\mathcal{X}}_{A1} \times \tilde{\mathcal{X}}_{A2}$ and $\tilde{\mathcal{X}}_{A1} = \tilde{\mathcal{X}}_{A2} = \{0, 1, 2, \ldots, 50\}$, $|\mathcal{X}| = 2601$, and ξ_i are independent chi-squared random variables with one degree of freedom so that $\mathbb{E}[\xi_i] = 1$ and $\mathbb{V}(\xi_i) = 2$ for all $i \in \{1, 2, 3\}$. Thus the random objective values returned by the simulation oracle are independent for all $\mathbf{x} \in \mathcal{X}$. A picture of Problem T_A appears in Figure 3.3.

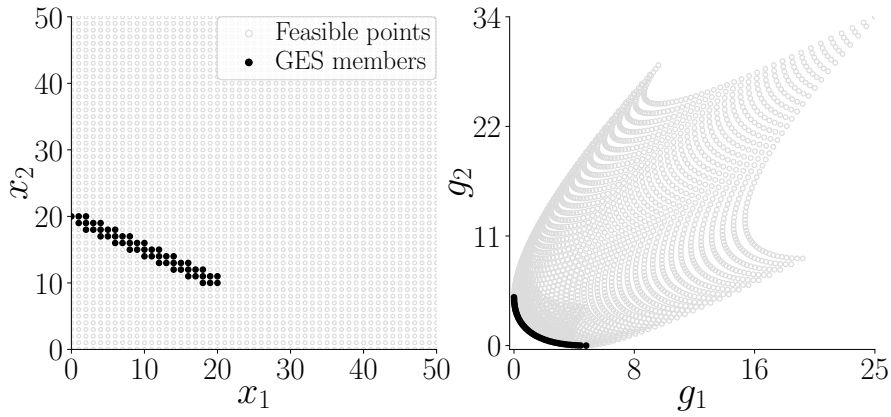

Figure 3.3. Problem T_A: Black circles represent points in the only $\mathcal{N}_1$-LES which is also the GES (left) and their image (right).

Problem T_A satisfies Assumptions 1–6; it has one $\mathcal{N}_1$-LES which equals the GES. Therefore for this problem, by Theorem 1 Part (d), our algorithms converge to the GES as the total simulation work done, denoted by t, goes to infinity. Let $\hat{\mathcal{A}}(t)$ denote the set returned by an algorithm after expending a total of t simulation replications.

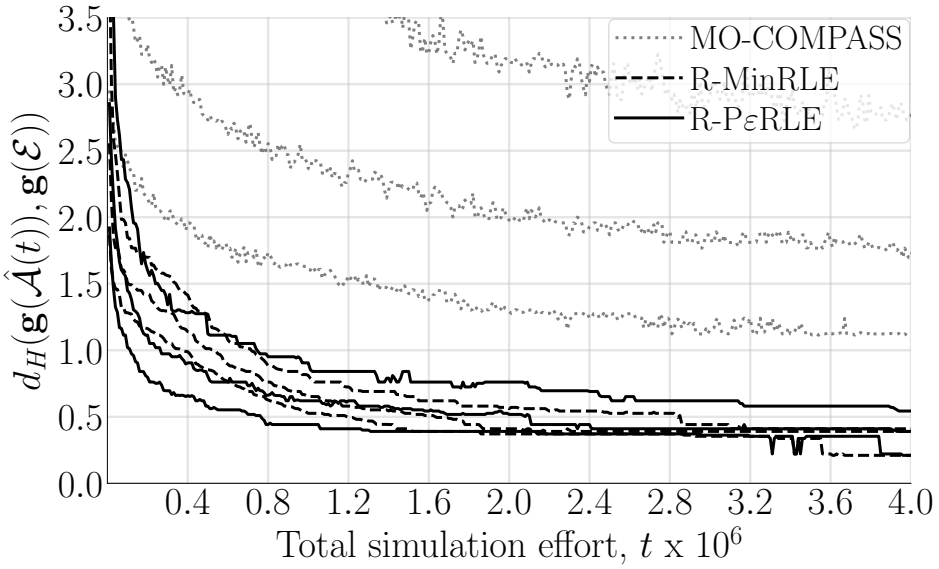

Figure 3.4. Problem T_A: Sample quantiles (0.25, 0.50, 0.75) of the coverage error across 1,000 independent runs.

We measure the solution quality returned by each algorithm using sample quantiles of the coverage error. The coverage error is defined by (Hunter et al., 2019) as the Hausdorff distance between the image of the set returned by the algorithm and the image of the true efficient set as a function of t, $d_H(\mathbf{g}(\hat{\mathcal{A}}(t)), \mathbf{g}(\mathcal{E}))$. Figure 3.4 shows the sample quantiles of the coverage error for 1,000 independent runs each of R-PεRLE, R-MinRLE, and MO-COMPASS on Problem T_A.

Figure 3.4 shows that R-PεRLE and R-MinRLE out-perform MO-COMPASS on Problem T_A. The performances of R-PεRLE and R-MinRLE are similar, with R-PεRLE performing slightly better for lower simulation budgets t.

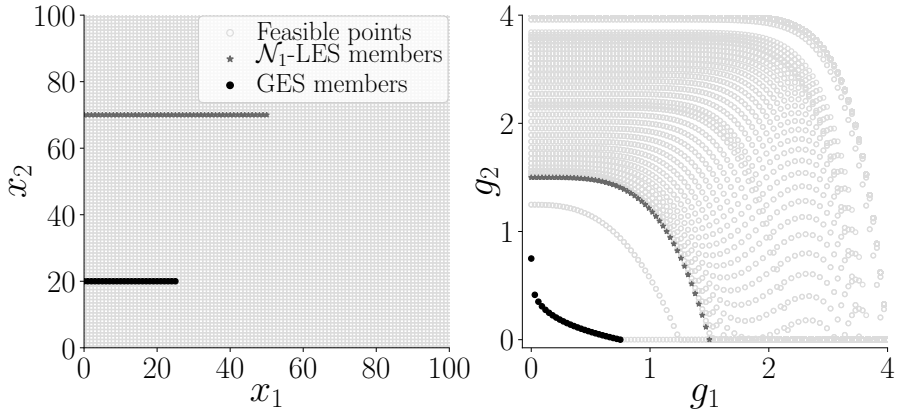

Figure 3.5. Problem T_B: The black circles and gray stars represent points in the GES and the $\mathcal{N}_1$-LES, respectively (left) and their images (right).

Test Problem B

Our second test problem is a modified version of a test problem that appears in Ryu and Kim (2014). We define Problem T_B as

$$\text{Problem } T_B\text{:} \qquad \text{minimize}_{\mathbf{x}\in\mathcal{X}} \begin{cases} g_1(\mathbf{x}) = \mathbb{E}\left[\xi_1 h_1(x_1)\right] \\ g_2(\mathbf{x}) = \mathbb{E}\left[\xi_1\xi_2 f(x_2) h_2(h_1(x_1), f(x_2))\right] \end{cases}$$

where $\mathcal{X} = \tilde{\mathcal{X}}_{B1} \times \tilde{\mathcal{X}}_{B2}$ and $\tilde{\mathcal{X}}_{B1} = \tilde{\mathcal{X}}_{B2} = \{0, 1, \ldots, 100\}$, $|\mathcal{X}| = 10,201$, $h_1(x_1) = 4x_1/100$, and $h_2(h_1, f)$ and $f(x_2)$ are defined as

$$h_2(h_1, f) = \begin{cases} 1 - (h_1/f)^\alpha & \text{if } h_1 \le f, \\ 0 & \text{otherwise;} \end{cases}$$

$$f(x_2) = \begin{cases} 4 - 3\exp\left\{-\left(\frac{x_2-20}{2}\right)^2\right\} & \text{if } 0 \le x_2 \le 40, \\ 4 - 2\exp\left\{-\left(\frac{x_2-70}{20}\right)^2\right\} & \text{if } 40 < x_2 \le 100; \end{cases}$$

and $\alpha = 0.25 + 3.75(f(x_2) - 1)$. As in the previous test problem, ξ_1 and ξ_2 are independent chi-squared random variables with one degree of freedom. Unlike in Problem T_A, Problem T_B has dependence between the random objective function values returned by the simulation oracle. A picture of Problem T_B appears in Figure 3.5.

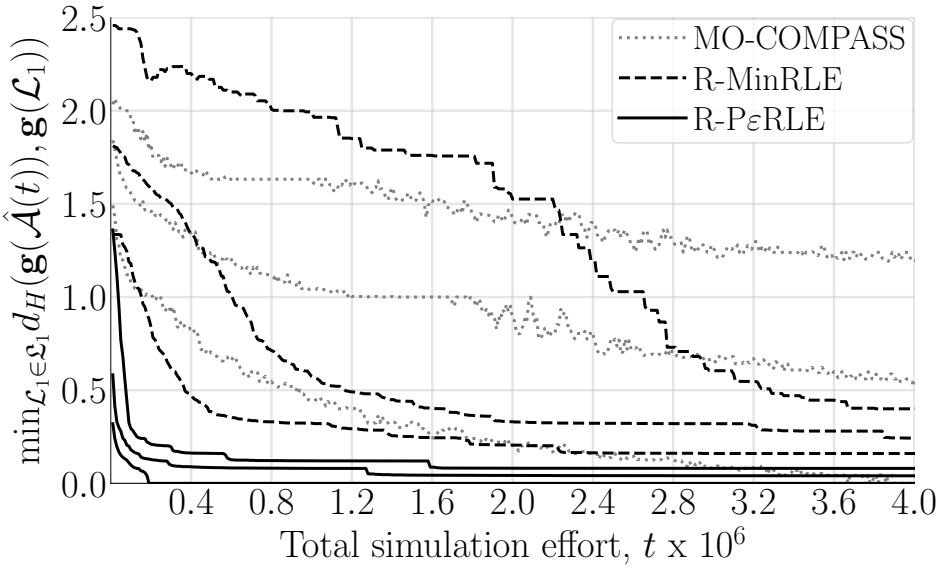

Figure 3.6. Problem T_B: Sample quantiles (0.25, 0.50, 0.75) of the local coverage error across 1,000 independent runs.

Problem T_B has two $\mathcal{N}_1$-LWES's, one of which is the GWES. Since Problem T_B has GWEP's that are not also GEP's, it satisfies only Assumptions 1–4. By Theorem 1 Parts (a) and (b), our algorithms converge into an $\mathcal{N}_1$-LWES almost surely, and contain an $\mathcal{N}_1$-LES almost surely. Nevertheless, for this problem, we use the local coverage error as our solution quality metric (Hunter et al., 2019). In our context, the local coverage error is the Hausdorff distance from the set $\mathbf{g}(\hat{\mathcal{A}}(t))$ to the nearest $\mathcal{N}_1$-LPS as a function of the total simulation work done, $\min_{\mathcal{L}_1 \in \mathfrak{L}_1} d_H(\mathbf{g}(\hat{\mathcal{A}}(t)), \mathbf{g}(\mathcal{L}_1))$. This metric penalizes all algorithms for returning the points that are GWES members but not GES members, which may not be distinguishable with finite sample size. Figure 3.6 shows the sample quantiles of the local coverage error for 1,000 independent runs each of R-PεRLE, R-MinRLE, and MO-COMPASS on Problem T_B.

Figure 3.6 shows that R-PεRLE out-performs both R-MinRLE and MO-COMPASS on Problem T_B. R-MinRLE eventually out-performs MO-COMPASS, but initially suffers from high variance in its performance. We believe this behavior occurs because R-MinRLE crawls from the "outside in," and the guaranteed sample path $\mathcal{N}_1$-local minimizers on each objective may be members of the GWES and not the GES. Also,

R-MinRLE may not retrieve the "middle" of the $\mathcal{N}_1$-LPS until the sample sizes become large enough that the completeness function values are small enough for RLE to crawl there. Thus R-PεRLE's ability to retrieve the middle of the $\mathcal{N}_1$-LPS is likely a crucial aspect of its speedy convergence in Problem T_B.

Test Problem C

Our last test problem, Problem T_C, is also a modified version of a test problem that appears in Ryu and Kim (2014). We define Problem T_C as

$$\text{Problem } T_C\text{:} \qquad \text{minimize}_{\mathbf{x}\in\mathcal{X}} \begin{cases} g_1(\mathbf{x}) = \mathbb{E}\left[\sum_{i=1}^{2} -10\xi_i \exp\left\{-0.2\sqrt{x_i^2 + x_{i+1}^2}\right\}\right] \\ g_2(\mathbf{x}) = \mathbb{E}\left[\sum_{i=1}^{3} \xi_i(|x_i|^{0.8} + 5\sin^3(x_i))\right] \end{cases}$$

where $\mathcal{X} = \tilde{\mathcal{X}}_{C1} \times \tilde{\mathcal{X}}_{C2} \times \tilde{\mathcal{X}}_{C3}$, $\tilde{\mathcal{X}}_{Ci} = \{-5, -4.5, -4.0, -3.5, \ldots, 5\}$ for all $i \in \{1, 2, 3\}$, $|\mathcal{X}| = 9,261$, and ξ_1, ξ_2, and ξ_3 are independent chi-squared random variables with one degree of freedom so that $\mathbb{E}[\xi_i] = 1$ and $\mathbb{V}(\xi_i) = 2$ for all $i \in \{1, 2, 3\}$. Like Problem T_B, Problem T_C has dependence between the random objective function values returned by the simulation oracle. Problem T_C appears in Figure 3.7. We map Problem T_C to an integer lattice so that the $\mathcal{N}_1$-neighborhood corresponds to points within distance 0.5 in the original feasible space.

Problem T_C has multiple feasible points that map to the same objective vector value. Therefore Problem T_C only satisfies Assumptions 1–3. By Theorem 1 Part (a), our algorithm returns a solution that converges into an $\mathcal{N}_1$-LWES w.p.1., with no guarantees on completeness. Nevertheless, we use the local weakly coverage error as our solution quality metric, which we define as $\min_{\mathcal{W}_1 \in \mathfrak{W}_1} d_H(\mathbf{g}(\hat{\mathcal{A}}(t)), \mathbf{g}(\mathcal{W}_1))$. Since all $\mathcal{N}_1$-LES's are also $\mathcal{N}_1$-LWES's, this metric is less stringent than local coverage error. Algorithm performances based on the local weakly coverage error, calculated across a collection of 516 unique $\mathcal{N}_1$-LWES's, appear in Figure 3.8.

Figure 3.8 shows that both R-PεRLE and R-MinRLE out-perform MO-COMPASS on Problem T_C. In many of the $\mathcal{N}_1$-LWES's, the $\mathcal{N}_1$-LWES members are not neighbors. Thus the $\mathcal{N}_1$-LWES members may be far away from each other in the feasible

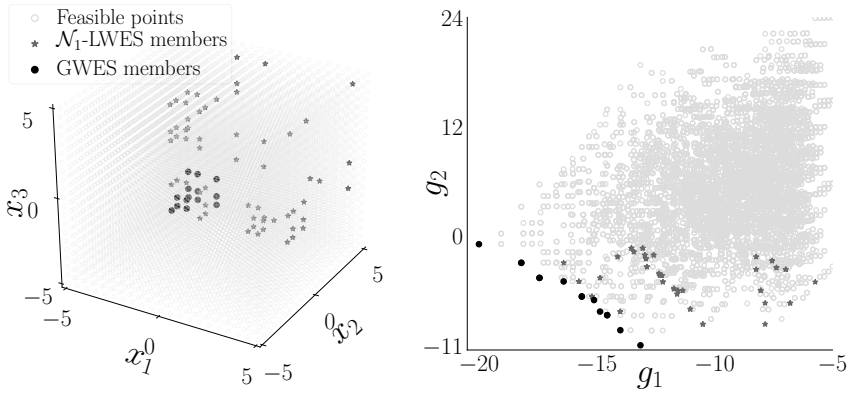

Figure 3.7. Problem T_C: Black circles and gray stars represent points in the GWES and the $\mathcal{N}_1$-LWES members, respectively (left) and their images (right).

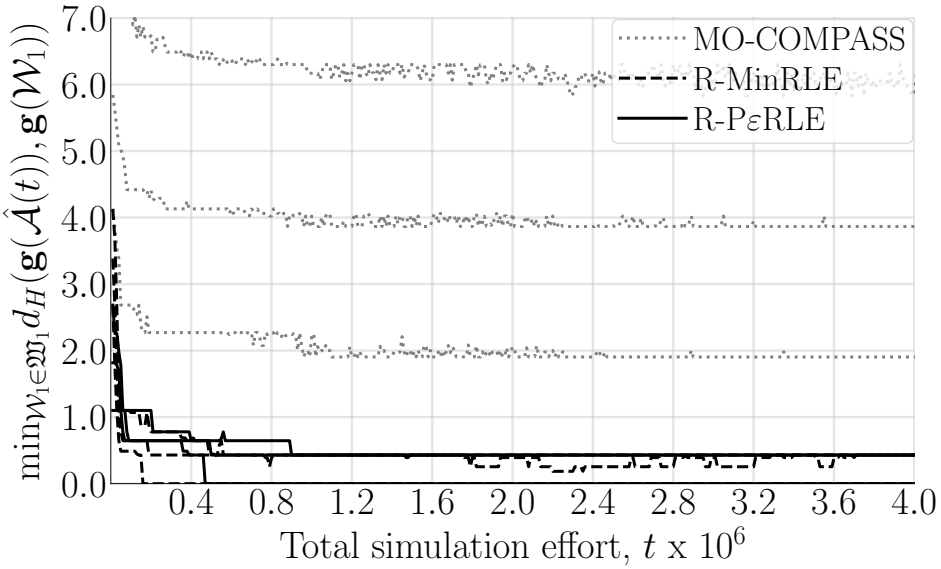

Figure 3.8. Problem T_C: Sample quantiles (0.25, 0.50, 0.75) of the local weakly coverage error across 1,000 independent runs.

space, and often are isolated, as seen in Figure 3.7. We believe the relative efficiency of R-PεRLE and R-MinRLE occurs because RLE crawls to find a sample-path $\mathcal{N}_1$-LWEP that completes the sample-path $\mathcal{N}_1$-LWES, even if the required sample-path $\mathcal{N}_1$-LWEP is far away in the feasible space. MO-COMPASS operates by updating a region of the feasible space called the *most promising region*. We suspect that the

isolated, scattered nature of the $\mathcal{N}_1$-LWES members results in a low probability that these far-away $\mathcal{N}_1$-LWEP's are contained in the most promising region.

To calculate the collection of all possible $\mathcal{N}_1$-LWES's in Problem T_C, we first locate all $\mathcal{N}_1$-LWEP's; we find 512. Then, starting from each $\mathcal{N}_1$-LWEP, we run a program similar to RLE with no relaxation to find the smallest complete $\mathcal{N}_1$-LWES that contains the $\mathcal{N}_1$-LWEP. We refer to these $\mathcal{N}_1$-LWES's as *level-1 $\mathcal{N}_1$-LWES's*; after removing duplicate sets, we find 39. Then, we take all possible unions of two level-1 $\mathcal{N}_1$-LWES's, remove any dominated points, and check if this set is a new, unique $\mathcal{N}_1$-LWES. We refer to all new, unique $\mathcal{N}_1$-LWES's that are found by taking the union of two level-1 $\mathcal{N}_1$-LWES's as *level-2 $\mathcal{N}_1$-LWES's*. We repeat this process for level-3 and so on, up to level-8. We found one level-7 $\mathcal{N}_1$-LWES and no level-8 $\mathcal{N}_1$-LWES's. The total number of unique $\mathcal{N}_1$-LWES's found in this manner, up to level-8, was 516. All together, these 516 $\mathcal{N}_1$-LWES's contain just 73 points; we call these points $\mathcal{N}_1$-LWES members in Figure 3.7. Recall that there are 512 $\mathcal{N}_1$-LWEP's, so not all $\mathcal{N}_1$-LWEP's are members of an $\mathcal{N}_1$-LWES.

3.10.2 R-PεRLE Performance Across a Range of $\boldsymbol{\beta}$ Values

We explore R-PεRLE's performance on our test problems across a variety of $\boldsymbol{\beta} = (\beta_\varepsilon, \beta_\delta)$ parameter values. Recall that for Pε, smaller β_ε values result in solving fewer sample-path ε-constraint problems, and larger β_ε values correspond to solving more sample-path ε-constraint problems. For the RLE algorithm, smaller β_δ implies less crawling and a less-complete ALES, and larger β_δ corresponds to more crawling and a more-complete ALES. Across 1,000 independent runs of R-Pε or R-PεRLE on Problems T_A, T_B, and T_C, Figures 3.9, 3.10, and 3.11 show the sample quantiles of the respective coverage errors at the total simulation budget of $t = 0.4 \times 10^6$ (corresponding to the first t-axis tick mark in Figures 3.4, 3.6, and 3.8) across a variety of parameter settings. Each independent run uses CRN across the $\boldsymbol{\beta}$ values.

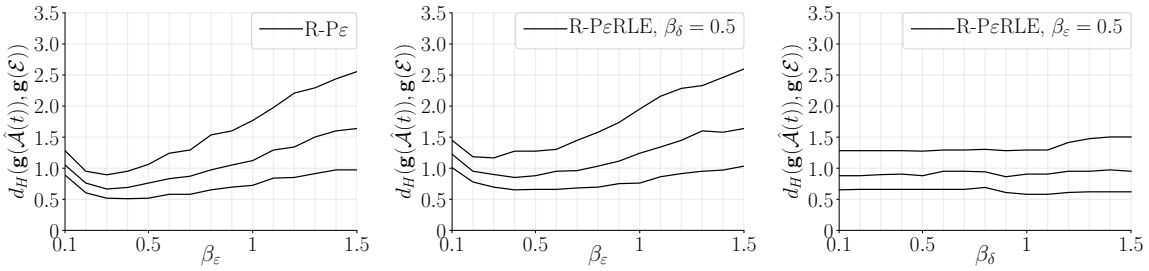

Figure 3.9. Problem T_A: Sample quantiles (0.25, 0.50, 0.75) of the coverage error at $t = 0.4 \times 10^6$ across 1,000 independent runs of R-Pε (left), R-PεRLE, $\beta_\delta = 0.5$ (center), R-PεRLE, $\beta_\varepsilon = 0.5$ (right).

At total simulation budget $t = 0.4 \times 10^6$ on Problem T_A, there seems to be a "sweet spot" for setting β_ε in the interval $(0.2, 0.4)$, as seen in the left and center panels of Figure 3.9. Relative to our sampling error, $\beta_\varepsilon < 0.2$ causes the algorithm to find too few sample-path $\mathcal{N}_1$-LWEP's, while $\beta_\varepsilon > 0.4$ cause the algorithm to find too many. Given that $\beta_\varepsilon = 0.5$, the R-PεRLE performance in the right panel of Figure 3.9 is fairly robust to different values of β_δ. Notice that with $\beta_\varepsilon = 0.5$, for small values of β_δ, R-Pε and R-PεRLE return similar sets.

On Problem T_B, however, solving more ε-constraint problems and crawling more in RLE seems to improve algorithm performance. We suspect that in this problem, correlation between the objectives and using CRN implies each sample-path problem

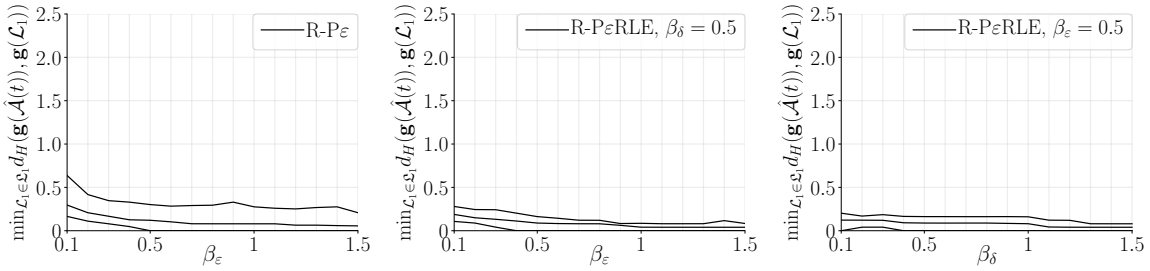

Figure 3.10. Problem T_B: Sample quantiles (0.25, 0.50, 0.75) of the local coverage error at $t = 0.4 \times 10^6$ across 1,000 independent runs of R-Pε (left), R-PεRLE, $\beta_\delta = 0.5$ (center), R-PεRLE, $\beta_\varepsilon = 0.5$ (right).

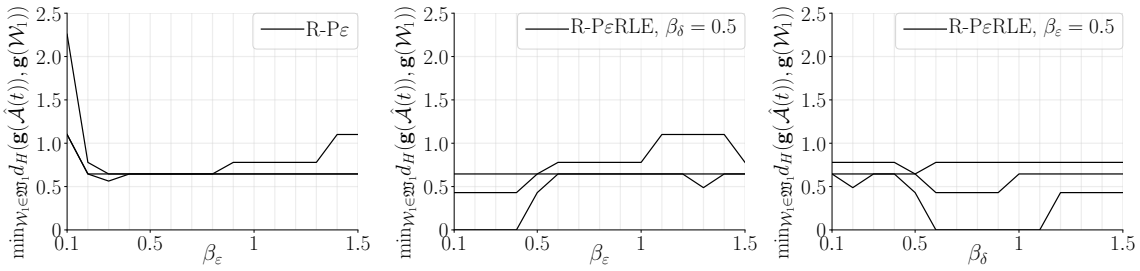

Figure 3.11. Problem T_C: Sample quantiles (0.25, 0.50, 0.75) of the local weakly coverage error at $t = 0.4 \times 10^6$ across 1,000 independent runs of R-Pε (left), R-PεRLE, $\beta_\delta = 0.5$ (center), R-PεRLE, $\beta_\varepsilon = 0.5$ (right).

is similar to the true problem. In this scenario, the ordering of the points in the sample-path problem is similar to the ordering of the points in the true problem with high probability, except among the GWEP's. Thus finding more $\mathcal{N}_1$-LWEP's in Pε and crawling more in RLE is usually better.

Problem T_C is a difficult problem for which R-Pε is not guaranteed to converge. In the left panel of Figure 3.11, like in the left panel of Figure 3.9, R-Pε exhibits u-shaped behavior as a function of β_ε. However, the center and right panels of Figure 3.11 tell an interesting story for R-PεRLE. It seems that in Problem T_C, it is best to solve few ε-constraint problems (smaller β_ε) and let RLE do the work of finding the disconnected $\mathcal{N}_1$-LWES members, with the sweet spot for β_δ shown in the right panel of Figure 3.11. Interestingly, that more effort should be expended in RLE and less effort in Pε explains the good performance of R-MinRLE in Figure 3.8. Finally, we remark that across all the problems, without prior knowledge of the problem structure, our default β_ε and β_δ values seem reasonable.

4. PYMOSO: SOFTWARE FOR MULTI-OBJECTIVE SIMULATION OPTIMIZATION WITH R-PεRLE AND R-MINRLE

:

We present the PyMOSO software package, written in Python, for using, implementing, and testing multi-objective simulation optimization (MOSO) algorithms. PyMOSO currently implements a state-of-the-art algorithm, R-PεRLE, for solving MOSO problems on integer lattices with two objectives, and a competitive benchmark algorithm, R-MinRLE, for solving MOSO problems on integer lattices with many objectives. Since R-PεRLE and R-MinRLE are both pseudo-gradient-based algorithms that rely on the single-objective SPLINE solver by Wang et al. (2013), for convenience, we also include an implementation of R-SPLINE (Wang et al., 2013) for solving single-objective simulation optimization problems on integer lattices.

By the nature of the algorithms currently included in PyMOSO, our focus is on MOSO problems in which the feasible set is a subset of an integer lattice. Recall that MOSO problems on integer lattices arise in many application domains including aviation, healthcare, transportation, and manufacturing (Hunter et al., 2019). For example, in aviation, Li et al. (2015c) solve a bi-objective aircraft spare parts management problem, and Lee et al. (2008) solve a tri-objective inventory control problem for a network of airports. In healthcare, Chen and Wang (2016) solve a bi-objective capacity allocation problem for a hospital's emergency department. In transportation, Zhou et al. (2018) solve a bi-objective problem to reduce congestion in a lighterage terminal. Finally, in manufacturing, Andersson et al. (2007) solve a bi-objective problem to increase the throughput and maintain safety stocks for a camshaft production line. In these applications, the feasible space is an integer lattice,

the objective functions can only be observed with stochastic error though a Monte Carlo simulation, and the goal is to retrieve the entire efficient set as input to the multi-criteria decision-making process.

Though MOSO problems arise in many application areas, few software packages exist to solve Problem M_d for $d \geq 2$ objectives. The software packages that do exist, such as OptQuest (OptTek Systems, Inc., 2018; Thengvall et al., 2016) and PaGMO/PyGMO (Biscani and Izzo, 2018), tend to implement metaheuristics to solve MOSO problems, which do not provide performance guarantees (Hong et al., 2015). We are not aware of any available MOSO software that provides the sampling efficiency and convergence guarantees of R-PεRLE and R-MinRLE, moreover, no software implementing these two algorithms currently exists. Finally, we remark here that an implementation of an as-yet-unpublished algorithm, Multi-objective Partitioned Random Search, is available in a source code repository (Weizhi, 2017).

As in Schmeiser (2008), we adopt the terms 'practitioners' and 'researchers' to describe those who seek a solution to Problem M_d to aid in decision-making, and those who intend to create and compare MOSO algorithms, respectively. We discuss our contributions to each group.

4.1 Contributions

PyMOSO provides practitioners with off-the-shelf access to the state-of-the-art, provably convergent, bi-objective solver R-PεRLE, and to the competitive, provably convergent, multi-objective solver R-MinRLE. PyMOSO can accommodate any Monte Carlo simulation oracle that can be called from Python, including oracles implemented as C, C#, or Java programs. The popular commercial simulation packages AutoMod (Applied Materials, Inc., 2018), Simio (Simio LLC, 2018), and AnyLogic (The AnyLogic Company, 2018) can be interfaced with Python, and thus with PyMOSO, through C, C#, and Java programs, respectively. Further, after the oracle has been implemented, PyMOSO can obtain simulation replications in parallel, which

may reduce runtime. Finally, PyMOSO provides off-the-shelf access to R-SPLINE in the same software framework.

For researchers who intend to create and compare MOSO algorithms, PyMOSO offers two primary benefits, as follows.

1. Researchers designing new algorithms to solve MOSO problems on integer lattices should compare their algorithms with R-PεRLE and R-MinRLE on a variety of test problems. PyMOSO enables researchers to compare algorithms by providing an interface for implementing test problems and calculating user-defined metrics. Then, researchers may use PyMOSO to run many independent sample paths of the algorithms in parallel. PyMOSO provides pseudo-random numbers using `mrg32k3a` (L'Ecuyer, 1999) and random number stream management consistent with L'Ecuyer et al. (2002).

2. PyMOSO provides a framework that enables researchers to create and implement new algorithms. Although any new MOSO algorithm can be implemented in PyMOSO, it is especially easy to implement algorithms that rely on a version of sample average approximation called retrospective approximation (RA); see, e.g., Pasupathy and Ghosh (2013) for an explanation. In particular, researchers can create new RA algorithms for MOSO by writing "accelerator" functions that provide starting points to the naïve search algorithm Relaxed Local Enumeration (RLE) in each RA iteration, as we describe in §4.3.3. Under appropriate regularity conditions, Cooper et al. (2018) prove that such algorithms converge to a local efficient set as the sample size increases.

In what follows, we provide introductions to the current version of PyMOSO for practitioners in §4.2 and for researchers in §4.3. Since all information provided for practitioners is relevant to researchers, we encourage researchers to read both sections. In addition to the online supplement, PyMOSO installation instructions, source code, and the user manual can be found at `https://github.com/HunterResearch/PyMOSO`.

4.2 Practitioners: Using PyMOSO to Solve a Problem

In this section, we discuss using PyMOSO in the practitioner context. Practitioners use PyMOSO in two steps: first, implement the simulation oracle in PyMOSO, which we discuss in §4.2.1, and second, use PyMOSO to solve the problem, which we discuss in §4.2.2.

4.2.1 Structuring an Oracle for Use in PyMOSO

To structure an oracle for use in PyMOSO, a practitioner should modify the Python source code template provided in Figure 4.1. Figure 4.1 implements an oracle named `MyProblem` in a Python file named `myproblem.py`. PyMOSO requires that the problem name matches the file name, although the capitalization does not need to match.

A practitioner implements a MOSO problem by first modifying the number of objectives and the dimension of the feasible points in Lines 8 and 9 of Figure 4.1, re-

```
1  # import the Oracle base class
2  from pymoso.chnbase import Oracle
3
4  class MyProblem(Oracle):
5      '''Example implementation of a user-defined MOSO problem.'''
6      def __init__(self, rng):
7          '''Specify the number of objectives and dimensionality of points.'''
8          self.num_obj = 2
9          self.dim = 1
10         super().__init__(rng)
11
12     def g(self, x, rng):
13         '''Check feasibility and simulate objective values.'''
14         #objective_values = (obj1, obj2), is_feasible = True
15         return is_feasible, objective_values
```

Figure 4.1. The Python file `myproblem.py` is a template PyMOSO oracle. As shown, `g(self, x, rng)` is incomplete.

spectively. Setting the correct values for the practitioner's Problem M_d, and changing nothing else, is sufficient to correctly implement the `__init__(self, rng)` method. Then, the practitioner should replace the comment in Line 14 of Figure 4.1 with valid Python code that, given a Python tuple `x` representing a point $\mathbf{x} \in \mathbb{R}^q$, generates the following to return in Line 15: (a) a boolean indicator denoting whether $\mathbf{x}$ is feasible, and (b) a Python tuple, containing one observation of every objective function at $\mathbf{x}$ if $\mathbf{x}$ is feasible, and a Python tuple containing `None` for every objective if $\mathbf{x}$ is not feasible. In our notation, one observation of every objective function at $\mathbf{x}$ is represented by $(G_1(\mathbf{x}, \boldsymbol{\xi}), \ldots, G_d(\mathbf{x}, \boldsymbol{\xi}))$; alternatively, the practitioner may think of this quantity as one observation of $\bar{\boldsymbol{G}}_n(\mathbf{x})$ where $n = 1$. The function `g(self, x, rng)` may contain any number of lines and may be a wrapper for an external simulation oracle. Returning the feasibility indicator followed by a Python tuple of the objective values is sufficient to correctly implement the `g(self, x, rng)` method.

Optionally, a practitioner may use the PyMOSO object `rng` to generate pseudo-random numbers with `mrg32k3a`, or may use the `mrg32k3a` seed from `rng` as the seed in an external `mrg32k3a` generator. If `MyProblem` implements either approach, PyMOSO ensures that simulation replications obtained in parallel are independent by exploiting the stream and substream capabilities of `mrg32k3a` (L'Ecuyer et al., 2002). Further, the implemented PyMOSO oracle is compatible with PyMOSO's common random number (CRN) framework. Practitioners who ignore `rng` should take care when using PyMOSO's parallel computing and CRN capabilities. To determine when using CRN is appropriate, we refer the reader to Law (2015). More detailed information about `rng` is available in the PyMOSO user manual.

We now list the basic requirements of every `g` implementation.

1. The function `g` must be an instance method of an `Oracle` sub-class, and thus take `self` as its first parameter.

2. The function `g` must take an arbitrarily-named second parameter which is a tuple of length `self.dim` and represents a point. Stylistically, PyMOSO consistently names this parameter `x`.

3. The function `g` must take an arbitrarily-named third parameter which is a modified Python `random.Random` object. Stylistically, PyMOSO consistently names this parameter `rng`.

4. The function `g` must return a boolean first and a tuple of length `self.num_obj` second.

    - The boolean is `True` if `x` is feasible, and `False` otherwise.
    - If `x` is feasible, the tuple contains a single observation of every objective. If `x` is not feasible, each element in the tuple is `None`.

If users already have an implemented simulation oracle, they may find it convenient to implement `g` as wrapper which calls that simulation from Python. As an example, suppose a user has implemented a simulation in C which is compiled to a C library called `mysim.so` and placed in the working directory. Suppose further that the simulation function takes the following as parameters: an array of integers representing a point $\mathbf{x} \in \mathbb{R}$ and an unsigned integer representing the number of observations to take at $\mathbf{x}$. The function output is defined as `struct Simout` with members `feas` set to 0 or 1, `obj` a double array set to the mean of the observed objective values, and `var` a double array set to the sample variance of the observed objective values. Then users can modify the template to wrap the C function `struct Simout c_func(int x, int n)` as in Figure 4.2.

Figure 4.2 is a valid PyMOSO oracle which wraps a C function. However, PyMOSO algorithms cannot enable common random numbers on this oracle. Furthermore, PyMOSO cannot guarantee that observations are independent when taken in parallel. To enable these properties, the external simulation must use `mrg32k3a` as the generator and must accept a user-specified seed.

Suppose the library `mysim.so` also implements the function `set_simseed` which accepts a long array representing an `mrg32k3a` seed. We modify the wrapper in Figure 4.3 for compatibility with common random numbers and to guarantee independence of parallel observations. Figure 4.3 demonstrates using `rng.get_seed()` to return the current `mrg32k3a` seed.

Alternatively, if the number of required pseudo-random numbers is known, users can use `rng.random()` to generate pseudo-random numbers and then pass them to an external simulation if such functionality is supported.

The `rng` object is implemented as a sub-class of Python's `random.Random` class, thus the official Python documentation for `random` applies to `rng` and is found at `https://docs.python.org/3/library/random.html`. In addition to `rng` using `mrg32k3a` as its generator, we also implement `rng.normalvariate` such that it uses the Beasley-Springer-Moro algorithm (Law 2015, p. 458) to approximate the inverse of the standard normal cumulative distribution function.

When using `rng`, to ensure independent sampling of observations, PyMOSO "jumps" forward in the pseudo-random number stream after obtaining every simulation replication. Each jump is of fixed size 2^{76} pseudo-random numbers. Thus, we require that every simulation replication use fewer than 2^{76} pseudo-random numbers. We ensure independence among parallel replications by "giving" each processor a stream (an `rng`), each of which is 2^{127} pseudo-random numbers apart. When using the current PyMOSO algorithms that rely on RA, each RA iteration begins the next available independent stream 2^{127}, where PyMOSO accounts for the possibility of parallel computation within an RA iteration. Thus, in a given RA iteration, a user may simulate 100 million points at a sample size of 1 million, without common random numbers, and easily not reach the limit.

In the remainder of the paper, we assume the practitioner has implemented a PyMOSO oracle called `MyProblem`. So the reader can run our examples, we provide a simple example of `MyProblem` in Figure 4.4, where Problem M_2 is $g_1(x) = x^2 + z_0$ and $g_2(x) = (x-2)^2 + z_1$, $x \in \{-100, -99, ..., 100\}$, and z_0, z_1 are standard normal

random variables. The solution is $\{0, 1, 2\}$. We remark here that Figure 4.4 contains an example of using `rng`.

```
from ctypes import CDLL, c_double, c_uint, c_int, Structure
import os.path
libname = 'mysim.so'
libabspath = os.path.dirname(os.path.abspath(__file__)) + os.path.sep + dll_name
libobj = CDLL(libabspath)

class Simout(Structure):
    _fields_ = [("feas", c_int), ("obj", c_double*2), ("var", c_double*2)]
csimout = libobj.c_func
csimout.restype = Simout

from pymoso.chnbase import Oracle

class MyProblem(Oracle):
    '''Example implementation of a user-defined MOSO problem.'''
    def __init__(self, rng):
        '''Specify the number of objectives and dimensionality of points.'''
        self.num_obj = 2
        self.dim = 1
        super().__init__(rng)

    def g(self, x, rng):
        '''Check feasibility and simulate objective values.'''
        is_feasible = True
        objective_values = (None, None)
        # g takes only one observation so set the c_func parameter to 1
        c_n = c_uint(1)
        # c_func requires is an integer so convert it -- this is a 1D example
        c_x = c_int(x[0])
        # call the C function
        mysimout = csimout(c_x, c_n)
        if not mysimout.feas:
            is_feasible = False
        else:
            is_feasible = True
        if is_feasible:
            objective_values = tuple(mysimout.obj)
        return is_feasible, objective_values
```

Figure 4.2. The `g` function wraps an external simulation written in C.

```
1  from ctypes import CDLL, c_double, c_uint, c_int, Structure, c_long
2  import os.path
3  libname = 'mysim.so'
4  libabspath = os.path.dirname(os.path.abspath(__file__)) + os.path.sep + dll_name
5  libobj = CDLL(libabspath)
6
7  class Simout(Structure):
8      _fields_ = [("feas", c_int), ("obj", c_double*2), ("var", c_double*2)]
9  csimout = libobj.c_func
10 csetseed = libobj.set_simseed
11 csimout.restype = Simout
12
13 from pymoso.chnbase import Oracle
14
15 class MyProblem(Oracle):
16     '''Example implementation of a user-defined MOSO problem.'''
17     def __init__(self, rng):
18         '''Specify the number of objectives and dimensionality of points.'''
19         self.num_obj = 2
20         self.dim = 1
21         super().__init__(rng)
22
23     def g(self, x, rng):
24         '''Check feasibility and simulate objective values.'''
25         is_feasible = True
26         objective_values = (None, None)
27         # get the PyMOSO seed from rng
28         seed = rng.get_seed()
29         # convert the seed to c_long array
30         c_longarr = c_long*6
31         c_seed = c_longarr(seed[0], seed[1], seed[2], seed[3], seed[4], seed[5])
32         # use the library function to set the sim seed
33         csetseed(c_seed)
34         # g takes only one observation so set the c_func parameter to 1
35         c_n = c_uint(1)
36         # c_func requires is an integer so convert it -- this is a 1D example
37         c_x = c_int(x[0])
38         # call the C function
39         mysimout = csimout(c_x, c_n)
40         if not mysimout.feas:
41             is_feasible = False
42         else:
43             is_feasible = True
44         if is_feasible:
45             objective_values = tuple(mysimout.obj)
46         return is_feasible, objective_values
```

Figure 4.3. The g function wraps an external simulation written in C, and maintains compatibility with common random numbers and taking simulation replications in parallel.

```
1  def g(self, x, rng):
2      '''Check feasibility and simulate objective values for MyProblem.'''
3      feas_range = range(-100, 101)
4      obj = []
5      is_feas = False
6      # check that dimensions of x match self.dim
7      if len(x) == self.dim:
8          is_feas = True
9          for i in x:
10             if not i in feas_range:
11                 is_feas = False
12     if is_feas:
13         z_vec = [rng.normalvariate(0, 1) for i in [0, 1]]
14         obj1 = x[0]**2 + z_vec[0]
15         obj2 = (x[0] - 2)**2 + z_vec[1]
16     return is_feas, (obj1, obj2)
```

Figure 4.4. This figure provides an example `g` function, which we use in `MyProblem`.

4.2.2 Solving a MOSO Problem in PyMOSO

Having implemented a PyMOSO oracle called `MyProblem` in §4.2.1, we now discuss using PyMOSO to solve `MyProblem`. Practitioners may use PyMOSO in two modes: as a stand-alone solver invoked from the command line, or as a subroutine in a Python program. The former creates a file containing the output of one run of the selected algorithm, which is an estimated LES. The latter returns the estimated LES as a Python set. In either case, PyMOSO requires the practitioner to specify, using a method we describe, at least the following information: the problem, the algorithm, and an initial feasible point. The method is slightly different depending on the chosen mode. In this section, we only consider the command line mode to solve `MyProblem`. See the user manual for the subroutine mode.

The simplest viable command to solve a problem in command line mode follows the structure `program command problem solver x0`, where `x0` denotes the initial feasible point. The solve command takes options, including the total simulation budget and the number of parallel processors. Practitioners may view the full set of available options by entering `pymoso --help` and the full list of PyMOSO solver names by entering `pymoso listitems`. As an example, the command to solve `MyProblem` using R-PεRLE, starting from the feasible point 97, with a total simulation budget of 10,000 and 4 processors, is below.

```
pymoso solve --budget=10000 --simpar=4 myproblem.py RPERLE 97
```

This invocation requires that `myproblem.py` is in the working directory. Since the feasible points of `MyProblem` are one-dimensional, `x0` is one number. For feasible points in higher dimensions, separate each component with a space, e.g., a three-dimensional point $(97, 23, 18)$ is written as `97 23 18`. After issuing the above command, PyMOSO creates a new subdirectory named `testrun` in the working directory. This subdirectory typically contains two files: one file containing the metadata and one file containing the estimated LES. If PyMOSO detects an error, it may also write an error file.

Similarly, PyMOSO can solve built-in problems, such as `ProbTPA` which has two-dimensional feasible points.

```
pymoso solve ProbTPA RPERLE 40 40
```

Henceforth, we present `solve` examples only for solving `MyProblem`. Since `MyProblem` is bi-objective, we recommend using the R-PεRLE solver. However, for two or more objectives, PyMOSO implements R-MinRLE.

```
pymoso solve myproblem.py RMINRLE 97
```

For a single objective problem, PyMOSO has R-SPLINE. We remark that if given a multi-objective problem, R-SPLINE will simply minimize the first objective. We do not necessarily prohibit such use, but urge that users take care when using R-SPLINE to minimize one objective of a many-objective problem.

```
pymoso solve myproblem.py RSPLNE 97
```

Regardless of the chosen solver, PyMOSO creates a new sub-directory of the working directory containing output. There will be a metadata file, indicating the date, time, solver, problem, and any other specified options. In addition, PyMOSO creates a file containing the solver-generated solution. PyMOSO provides additional options for users solving MOSO problems. We present examples of each option below. First, users can specify the name of the output directory.

```
pymoso solve --odir=OutDirectory myproblem.py RPERLE 45
```

Users can specify the simulation budget, which is currently set to a default of 200.

```
pymoso solve --budget=100000 myproblem.py RPERLE 12
```

Users may specify to take simulation replications in parallel. We only recommend doing so if the user has thought through appropriate pseudo-random number stream control issues (see §4.2.1). Furthermore, due to the overhead of parallelization, we only recommend using the parallel simulation replications feature if observations are sufficiently "expensive" to compute, e.g. the simulation takes a half second or more to generate a single observation. We remark that the run-time complexity of the simulation oracle may not perfectly indicate when it is appropriate to use parallelization; other factors include, e.g., the total simulation budget.

```
pymoso solve --simpar=4 myproblem.py RPERLE 44
```

Currently, all PyMOSO solvers support using common random numbers. Users may enable the functionality using the `crn` option.

```
pymoso solve --crn myproblem.py RMINRLE 62
```

We do not recommend this option unless the oracle is implemented to be compatible, that is, the oracle uses PyMOSO's pseudo-random number generator to generate pseudo-random numbers or to provide a seed to an external `mrg32k3a` generator (see §4.2.1).

Users may specify an initial seed to PyMOSO's `mrg32k3a` pseudo-random number generator. Seeds must be 6 positive integers with spaces. The default is 12345 for each of the 6 components.

```
pymoso solve --seed 11 22 33 44 55 66 myproblem.py RPERLE 23
```

Users may specify algorithm-specific parameters (see the papers in which the algorithms were introduced for detailed explanations of the parameters.) All parameters are specified in the form `--param name value`. For example, the RLE relaxation parameter can be specified and set as `betadel` to a real number. We refer the reader to Table 4.1 for the full list of currently available algorithm-specific parameters.

```
pymoso solve --param betadel 0.2 myproblem.py RPERLE 34
```

Finally, users may specify any number of options in one invocation. However, all options must be specified in after the `solve` command and before the `myproblem.py` argument. Furthermore, any `--param` options must be last. (Note that the \ at the end of the first line continues the command to the second line.)

```
pymoso solve --crn --simpar=4 --budget=10000 --seed 1 2 3 4 5 6 \
  --odir=Exp1 --param mconst 4 --param betadel 0.7 \
       myproblem.py RPERLE 97
```

Users may invoke the `solve` function within a Python program. We provide simple examples below.

Table 4.1.
The table contains the current list of algorithm-specific parameters.

Parameter Name	Default Value	Affected Solvers	Description
mconst	2	R-PεRLE, R-MinRLE, R-Pε, R-SPLINE	Initialize the sample size and subsequent schedule of sample sizes.
bconst	8	R-PεRLE, R-MinRLE, R-Pε, R-SPLINE	Initialize the search sampling limit and subsequent schedule of limits.
radius	1	R-PεRLE, R-MinRLE, R-Pε, R-SPLINE	Set the radius a that determines a point's neighborhood, $\mathcal{N}_a$ (Wang et al., 2013).
betadel	0.5	R-PεRLE, R-MinRLE	Roughly, set how likely RLE is to keep a point in the input set. See Cooper et al. (2018).
betaeps	0.5	R-PεRLE, R-Pε	Roughly, set how likely Pε is to search from a point. See Cooper et al. (2018).

Using `solve` in a Python program is similar to using the CLI `solve` We provide the minimal example here.

```
1  # import the solve function
2  from pymoso.chnutils import solve
3  # import the module containing the RPERLE implementation
4  import pymoso.solvers.rperle as rp
5  # import MyProblem - myproblem.py should usually be in the script directory
6  import myproblem as mp
7
8  # specify an x0. In MyProblem, it is a tuple of length 1
9  x0 = (97,)
10 soln = solve(mp.MyProblem, rp.RPERLE, x0)
11 print(soln)
```

Users can specify options, including algorithm-specific parameters, as shown below.

```
1  # example for specifying budget and seed
2  budget=10000
3  seed = (111, 222, 333, 444, 555, 666)
4  soln1 = solve(mp.MyProblem, rp.RPERLE, x0, budget=budget, seed=seed)
5
6  # specify crn and simpar
7  soln2 = solve(mp.MyProblem, rp.RPERLE, x0, crn=True, simpar=4)
8
9  # specify algorithm specific parameters
10 soln3 = solve(mp.MyProblem, rp.RPERLE, x0, radius=2, betaeps=0.3, betadel=0.4)
11
12 # mix them
13 soln4 = solve(mp.MyProblem, rp.RPERLE, x0, crn=True, seed=seed, radius=5)
```

4.3 Researchers: Testing and Comparing MOSO Algorithms with PyMOSO

In this section, we discuss using PyMOSO in the researcher context. Researchers can use PyMOSO to compare algorithms and to create new algorithms. To compare algorithms, researchers first implement a PyMOSO oracle as in §4.2.1. Then, they create a PyMOSO tester, which we discuss in §4.3.1, and run the tester, which we discuss in §4.3.2. We briefly discuss creating new algorithms in §4.3.3.

4.3.1 Structuring a Test Problem for Use in PyMOSO

After implementing a PyMOSO oracle called `MyProblem` in §4.2.1, researchers must implement a PyMOSO tester for `MyProblem`. We provide an example tester called `MyTester` in Figure 4.5, where the oracle to be tested in Line 27 is specified as `MyProblem`.

Technically, a valid PyMOSO tester may consist of only Lines 24–27 in Figure 4.5. However, Figure 4.5 illustrates two optional features that researchers may find useful. First, researchers can implement a PyMOSO function that generates feasible starting

```
1  import sys, os
2  sys.path.insert(0,os.path.dirname(__file__))
3  # use hausdorff distance (dh) as an example metric
4  from pymoso.chnutils import dh
5  # import the MyProblem oracle
6  from myproblem import MyProblem
7
8  # optionally, define a function to randomly choose a MyProblem feasible x0
9  def get_ranx0(rng):
10     val = rng.choice(range(-100, 101))
11     x0 = (val, )
12     return x0
13
14 # compute the true values of x, for computing the metric
15 def true_g(x):
16     '''Compute the objective values.'''
17     obj1 = x[0]**2
18     obj2 = (x[0] - 2)**2
19     return obj1, obj2
20
21 # define an answer as appropriate for the metric
22 myanswer = {(0, 4), (4, 0), (1, 1)}
23
24 class MyTester(object):
25     '''Example tester implementation for MyProblem.'''
26     def __init__(self):
27         self.ranorc = MyProblem
28         self.answer = myanswer
29         self.true_g = true_g
30         self.get_ranx0 = get_ranx0
31
32     def metric(self, eles):
33         '''Metric to be computed per retrospective iteration.'''
34         epareto = [self.true_g(point) for point in eles]
35         haus = dh(epareto, self.answer)
36         return haus
```

Figure 4.5. The file `mytester.py` implements `MyTester`, a tester for `MyProblem`.

points by setting `self.get_ranx0` to an appropriate function in Line 30. We provide an example function, called `get_ranx0`, that randomly generates a feasible point for

`MyProblem` in Lines 9–12 of Figure 4.5. The function must take `rng` as a parameter and return a Python tuple representing a feasible point. The second feature enables researchers to implement a metric for comparing an estimated solution to the known, true solution. We provide an example metric that calculates the Hausdorff distance from the expected objective values of the points in the estimated LES, `eles`, to the image of the known solution, `self.answer`, in Lines 32–36. To calculate the expected objective values of the points in `eles`, we implement the function `true_g` in Lines 15-19. We also specify `myanswer` in Line 22 and set `self.answer` and `self.true_g` as members of `MyTester` in Lines 28–29. Researchers may replace Lines 34–36 with a metric of their choosing.

As a minimal valid PyMOSO tester, users may do nothing but assign the `MyTester` member `self.ranorc` to a PyMOSO oracle, such as `MyProblem`, in Line 27. However, we expect most users to leverage PyMOSO features by implementing metrics and feasible point generators. The function `get_ranx0` allows the tester to generate feasible points to `MyProblem` and `metric` allows the tester to compute a metric on sets returned by a solver. Researchers may implement any number of additional supporting functions, including members and methods of the tester class. The `true_g` function is an example of such a supporting function, which is used to compute the example metric.

Here, we list the rules for implementing a feasible point generator.

1. The function is arbitrarily named but must be set to the `self.get_ranx0` member of a tester.

2. The function must take a single parameter, an arbitrarily named `random.Random` object we suggest naming `rng`.

3. The function must return a tuple with length corresponding to the `self.dim` member of the `self.ranorc` member of the tester.

Since a researcher's desired metric depends on the algorithm capabilities and problem complexity, PyMOSO allows researchers to implement any metric they choose. We provide three example metrics, but first, we list the implementation rules of the `metric` function.

1. The `metric` function must be an instance method of a tester, and thus take `self` as its first parameter.

2. The second parameter of `metric` is arbitrarily named and is a Python set of tuples.

3. PyMOSO does not enforce the return value of `metric`, but we recommend a scalar real number.

The metric implemented in Figure 4.3.1 is the Hausdorff distance from (a) the true image of an estimated solution returned by an algorithm, to (b) the true solution hard-coded as `myanswer`.

For an example of a different metric, consider a MOSO problem that has more than one local efficient set (LES) and such that each LES contains no members of another LES. Since an algorithm that converges to a LES is may find only one LES, we may define the metric to compute the Hausdorff distance between the true image of the estimated solution and the "closest" true LES, as follows. Let `self.answer` be implemented as a list of sets, and assume a `self.true_g` implementation. Then Figure 4.6 implements the described metric.

For single-objective problems with one correct solution $\mathbf{x}^*$, a simple metric that takes an estimated solution $\boldsymbol{X}$ is $|g(\boldsymbol{X}) - g(\mathbf{x}^*)|$, which we implement in Figure 4.7 assuming an appropriate implementation of `self.answer` and `self.true_g`.

4.3.2 Testing a MOSO Algorithm in PyMOSO

Having implemented both `MyProblem`, in §4.2.1, and its tester, `MyTester` in §4.3.1, in PyMOSO, we now discuss using PyMOSO to test algorithms on `MyProblem`. As

```
1  def metric(self, eles):
2      # use the distance to the closest set.
3      epareto = [self.true_g(point) for point in eles]
4      # self.soln is a list of sets
5      dist_list = [dh(epareto, les) for les in self.answer]
6      return min(dist_list)
```

Figure 4.6. We provide a potentially useful metric for testing MOSO algorithms that converge to a LES on problems with more than one LES, such that none of the LES's have members in common.

```
1  def metric(self, singleton_set):
2      # single objective algorithms still return a set
3      point, = singleton_set
4      # let self.soln be a real number
5      dist = abs(self.true_g(point) - self.answer)
6      return dist
```

Figure 4.7. We provide a potentially useful metric for testing single objective algorithms.

with practitioners solving MOSO problems, researchers can use PyMOSO in two modes for testing algorithms on problems: as a stand-alone solver invoked from the command line, or as a subroutine in a Python program. In this section, we only consider the command line mode to test PyMOSO algorithms. See the user manual for the subroutine mode.

The simplest viable command to test an algorithm in command line mode follows the structure `program command tester solver`. (Researchers may also specify a feasible starting point if the tester is not programmed to generate them.) The `testsolve` command takes options, including the number of independent sample paths of the test problem, the number of processors to use, and whether to compute a metric. As an example, the command to test R-PεRLE by running 16 independent sample paths of `MyProblem` using `MyTester`, on 4 processors and computing a metric, is below.

```
pymoso --isp=16 --proc=4 --metric testsolve mytester.py RPERLE
```

This invocation requires that `myproblem.py` and `mytester.py` are in the working directory. After issuing the above command, PyMOSO creates a new subdirectory named `testrun` in the working directory. This subdirectory contains (a) one metadata file; (b) 16 data files, each containing list of estimated LES's, one for every algorithm iteration; and (c) 16 files containing metric calculations, which are included only when using the `--metric` option. The files containing metric calculations each have data of the form $(\nu,\ w_\nu,\ h_\nu)$, where ν is the RA algorithm iteration number; w_ν is the cumulative work done, measured as the total number of simulations used at the end of iteration ν; and h_ν is the metric computed on the estimated LES at iteration ν. If PyMOSO detects an error, it may also write an error file.

As the first of additional examples, we test R-PεRLE on `MyProblem` using `MyTester`. Since some options are compatible with both `solve` and `testsolve`, we include those options in this example.

```
pymoso testsolve --budget=999 --odir=exp1 \
 --crn --seed 1 2 3 4 5 6 mytester.py RPERLE
```

Users may want to compute some metric on the algorithm-generated solutions. If a metric is defined as part of the tester, such as in `MyTester`, the `testsolve` command can compute the metric on every algorithm iteration using the `--metric` option. `pymoso testsolve --metric mytester.py RPERLE`

The `testsolve` command cannot perform simulation replications in parallel. However, testers can apply the solvers to independent sample paths of the problems. For example, to test R-PεRLE on 100 independent sample paths of `MyProblem`, compute the metrics for each sample path, and use common random numbers in each sample path, use the following command.

```
pymoso testsolve --crn --metric --isp=100 mytester.py RPERLE
```

PyMOSO can perform independent algorithm runs in parallel. Use the `proc` option to specify the number of processes available to PyMOSO.

```
pymoso testsolve --crn --metric --isp=100 \
    --proc=20 mytester.py RPERLE
```

We remark here that, to ensure the algorithm runs remain independent using PyMOSO's pseudo-random number generator (see §4.2.1), researchers should set the total simulation budget so that the included algorithms do not surpass 200 retrospective approximation (RA) iterations. For reference, using the default settings, the sample size at every point in the 200th RA iteration is almost 380 million.

The `testsolve` command creates a results file for each independent sample path. The file contains the solutions generated at every algorithm iteration, such that the solution of iteration 2 is on line 2, iteration 10 on line 10, and so forth. If `--metric` is specified, PyMOSO generates a second file for each independent sample path containing the collection of triples (iteration number, simulations used at end of iteration, metric).

Below, we give examples of how users may invoke the `testsolve` function within a Python program.

Using `testsolve` in a Python program is also similar to using the CLI `testsolve`. Here, we provide an example with options.

```
# import the testsolve functions
from pymoso.chnutils import testsolve
# import the module containing RPERLE
import pymoso.solvers.rperle as rp
# import the MyTester class
from mytester import MyTester

# testsolve needs a "dummy" x0 even if MyTester will generate them
x0 = (1, )
run_data = testsolve(MyTester, rp.RPERLE, x0, isp=100, crn=True, radius=2)
```

When using `testsolve` in a Python program, users must compute their metric. Here, `run_data` is a dictionary of the form described in §4.3.3, in the description of Figure 4.10. In the snippet below, we compute the metric on the 5th algorithm iteration of the 12th independent sample path.

```
iter5_soln = run_data[11]['itersoln'][4]
isp12_iter5_metric = MyTester.metric(iter5_soln)
```

4.3.3 Creating New Algorithms in PyMOSO

Researchers can implement simulation optimization algorithms in the PyMOSO framework. PyMOSO provides support for algorithms in three categories:

1. PyMOSO provides strong support for implementing new MOSO algorithms that rely on RLE in an RA framework.

2. PyMOSO provides strong support for implementing general RA algorithms.

3. PyMOSO provides basic support, such as pseudo-random number control, for implementing other simulation optimization algorithms.

We provide templates of algorithms implemented in each of these three categories, along with example code snippets.

In the first category, we discuss how to implement RA algorithms that invoke an "accelerator" followed by RLE in every RA iteration (see Cooper et al., 2018). For example, in R-PεRLE, "Pε" is the accelerator, and in R-MinRLE, "Min" is the accelerator. Programmers can create new accelerators. We provide an accelerator template in Figure 4.8 so that programmers can use PyMOSO to create new RA algorithms that use RLE for convergence.

```
1  from pymoso.chnbase import RLESolver
2
3  # create a subclass of RLESolver
4  class MyAccel(RLESolver):
5      '''Example implementation of an RLE accelerator.'''
6
7      def accel(self, warm_start):
8          '''Return a collection of points to send to RLE.'''
9          # implement algorithm logic here and return a set
10         return warm_start
```

Figure 4.8. The file `myaccel.py` implements a provably convergent MOSO algorithm by relying on RLE in a RA framework. We encourage MOSO researchers to improve it.

The novel part of these algorithms, created by the user, will be the `accel` function which should collect points to send to RLE for certification. Here, we list the rules for `accel`.

1. The `accel` function must be an instance method of an `RLESolver` object, and thus its first parameter must be `self`.

2. The second parameter is arbitrarily named and is a set of tuples. We recommend naming the parameter `warm_start`, as it represents the sample-path solution of the previous RA iteration.

3. The return value must be a set of tuples representing feasible points; we do not recommend any particular name.

In every RA iteration, PyMOSO will first call `accel(self, warm_start)` and send the returned set to `rle(self, candidate_les)`. The return value must be a set of tuples. The implementer does not need to implement or call RLE, as in Figure 4.8.

Researchers should replace the comment in Line 9 of Figure 4.8 with their own code. The function signature must be `accel(self, warm_start)` and the function must return a Python set. After implementing a PyMOSO algorithm, researchers can test it as in §4.3.2.

```
pymoso --isp=20 --proc=4 --metric testsolve mytester.py myaccel.py
```

In the second category, algorithm designers can quickly implement any RA algorithm by sub-classing `RASolver` and implementing the `spsolve` function, as shown in Figure 4.9. The algorithm can be a single-objective algorithm. PyMOSO cannot guarantee the convergence of such algorithms. Figure 4.9 is technically valid in PyMOSO but is probably not effective.

Though analogous to those of an `RLESolver.accel` method, for completeness, we list the requirements for an `RASolver.spsolve` method.

1. The `spsolve` function must be an instance method of an `RASolver` object, and thus its first parameter must be `self`.

2. The second parameter is arbitrarily named and is a set of tuples. We recommend naming the parameter `warm_start` as it represents the sample-path solution of the previous RA iteration.

3. The return value must be a set of tuples representing feasible points; we do not recommend any particular name.

In the third category, PyMOSO can accommodate any simulation optimization algorithm by implementing the `solve` function of a `MOSOSolver` sub-class as shown in Figure 4.10. It does not have to be a multi-objective algorithm. PyMOSO will require users to send an initial feasible point `x0` whether or not the algorithm needs it. The initial feasible point `x0` is accessed through `self.x0` which is a tuple. We now list the rules for implementing any `MOSOSolver.solve` function.

1. The `solve` function must be an instance method of `MOSOSolver`, and thus take `self` as its first parameter.

2. The second parameter is the simulation budget, a natural number.

3. The `solve` function must return a dictionary (we name it `results` in our example) with at least 3 keys: `'itersoln'`, `'simcalls'`, `'endseed'`. Researchers may track additional data and add it to `results` as desired.

```
1  from pymoso.chnbase import RASolver
2
3  class MyRAAlg(RASolver):
4      '''Template implementation of an RA solver.'''
5
6      def spsolve(self, warm_start):
7          '''Return the sample path solution.'''
8          # implement algorithm logic here and return a set
9          return warm_start
```

Figure 4.9. We provide a template for implementing RA algorithms.

- The `'itersoln'` key itself corresponds to a dictionary with a key for each algorithm iteration labeled $\{0, 1, \ldots\}$. The value at each iteration is a set containing the estimated solution at the end of the iteration.
- The `'simcalls'` key itself corresponds to a dictionary with a key for each algorithm iteration labeled $\{0, 1, \ldots\}$. The value at each iteration is a natural number containing the cumulative number of simulation replications taken at the end of the iteration.
- The `'endseed'` key corresponds to a tuple of length 6, representing an `mrg32k3a` seed. The algorithm programmer should ensure the stream generated by `results['endseed']` is independent of all streams used by the algorithm.

Researchers may use Figure 4.10 to implement new simulation optimization algorithms.

```
1  from pymoso.chnbase import MOSOSolver
2
3  class MyMOSOAlg(MOSOSolver):
4      '''Template implementation of a MOSO solver.'''
5
6      def solve(self, budget):
7          while self.num_calls <= budget:
8              # implement algorithm logic and return the results
9          return results
```

Figure 4.10. We provide a template to implement a simulation optimization algorithm.

For implementing algorithm logic, PyMOSO also provides support for, e.g., obtaining simulation replications from the oracle, computing all points in a set that are non-dominated, and generating neighborhoods of points and sets. For convenience, in the list below, we also provide some example code snippets that we find useful when implementing algorithms in PyMOSO. For reference, §4.4.3 contains a list of most objects accessible to PyMOSO programmers.

- Example code to take simulation replications of a point at some sample size:

```
# pretend x has not yet been visited in this RA iteration and is feasible
x = (1, 1, 1)

# self.m is the sample size of the current RA iteration
m = self.m
# self.num_calls is the cumulative number of simulations used till now
start_num_calls = self.num_calls
# use estimate to sample x and put results in self.gbar and self.sehat
isfeas, fx, se = self.estimate(x)
calls_used = self.num_calls - start_num_calls
print(m == calls_used) # True
print(fx == self.gbar[x]) # True
print(se == self.sehat[x]) # True

# estimate will not simulate again in subsequent visits to a point
start_num_calls = self.num_calls
isfeas, fx, se = self.estimate(x)
calls_used = self.num_calls - start_num_calls
print(calls_used == 0) # True
```

- Example code to retrieve a point's neighbors and take simulation replications:

```
from pymoso.chnutils import get_nbors
r = self.nbor_rad
nbors = get_nbors(x0, r)
self.upsample(nbors)
for n in nbors:
  print(n in self.gbar) # True if n feasible else False
# upsample also returns the feasible subset
nbors = self.upsample(nbors)
```

- Example code to sort points by their observed objective values:

```
# 0 index for first objective
sorted_feas = sorted(nbors | {x}, key=lambda t: self.gbar[t][0])
xmin = sorted_feas[0]
fxmin = self.gbar[x]
```

- Example code to use the built-in SPLINE implementation:

```
# unconstrained minimize the 2nd objective
x0 = (2, 2, 2)
isfeas, fx, sex = self.estimate(x0)
# the suppressed value is the set visited along SPLINE's trajectory
_, xmin, fxmin, sexmin = self.spline(x0, float('inf'), 1, 0)
print(self.gbar[xmin] == fxmin) # True
```

4.4 PyMOSO Technical Details

4.4.1 Installation

Since PyMOSO is programmed in Python, every PyMOSO user must first install Python, which can be downloaded from `https://www.python.org/downloads/`. PyMOSO is compatible with Python versions 3.6 and higher. In the remainder of this section, we assume an appropriate Python version is installed. We discuss three different methods to install PyMOSO: first, from the Python Packaging Index; second, directly from our source code using git; and third, manually installing PyMOSO from our source code.

Install PyMOSO from the Python Packaging Index using `pip`

For ease of distribution, we keep stable, recent releases of PyMOSO on the Python Packaging Index (PyPI). Since the program `pip` is included in Python versions 3.6 and higher, we recommend using `pip` to install PyMOSO. To do so, open a terminal, type the following command, and press enter.

```
pip install pymoso
```

Depending on how users configure their Python installation and how many version of Python they install, they may need to replace `pip` with `pip3`, or other variants of `pip`.

Install PyMOSO from `git` using `pip`

Users with `git` installed can use `pip` to install the most current version of PyMOSO directly from our source code:

```
pip install git+https://github.com/HunterResearch/PyMOSO.git
```

We consider the latest source to be less stable than the fixed releases we upload to PyPI, and thus we recommend most users install PyMOSO as in §4.4.1.

Install PyMOSO Manually from Source Code

Users may follow the steps below to manually install PyMOSO from any version of the source code.

1. Acquire the PyMOSO source code, for example, by downloading it from the repository `https://github.com/HunterResearch/PyMOSO`.
2. Install the `wheel` package, e.g. using the `pip install wheel` command.
3. Open a terminal and navigate into the main project directory which contains the file `setup.py`
4. Build the installable PyMOSO package, called a wheel, using the command `python setup.py bdist_wheel`. As with `pip`, some users may need to replace `python` with `python3` or something similar. The command should create a directory named `dist` containing the PyMOSO wheel.
5. Install the PyMOSO wheel using
   `pip install dist/pymoso-x.x.x-py3-none-any.whl`, where users replace `x.x.x` with the appropriate PyMOSO version.

4.4.2 Command Line Interface (CLI)

PyMOSO users solving MOSO problems and testing MOSO algorithms may do so using the command line interface. First, we show how to access the included help file. Then, we show how to view the lists of solvers, testers, and oracles installed by default with PyMOSO. Finally, we discuss the `solve` and `testsolve` commands.

CLI Help

PyMOSO includes a command line help file. The help file shows syntax templates for every PyMOSO command, the available options, and a selection of example in-

```
Usage:
  pymoso listitems
  pymoso solve [--budget=B] [--odir=D] [--crn] [--simpar=P]
    [(--seed <s> <s> <s> <s> <s> <s>)] [(--param <param> <val>)]...
    <problem> <solver> <x>...
  pymoso testsolve [--budget=B] [--odir=D] [--crn] [--isp=T] [--proc=Q]
    [--metric] [(--seed <s> <s> <s> <s> <s> <s>)] [(--param <param> <val>)]...
    <tester> <solver> [<x>...]
  pymoso -h | --help
  pymoso -v | --version

Options:
  --budget=B      Set the simulation budget [default: 200]
  --odir=D        Set the output file directory name. [default: testrun]
  --crn           Set if common random numbers are desired.
  --simpar=P      Set number of parallel processes for simulation replications.
  --isp=T         Set number of algorithm instances to solve. [default: 1]
  --proc=Q        Set number of parallel processes for the algorithm instances.
  --metric        Set if metric computation is desired.
  --seed          Set the random number seed with 6 spaced integers.
  --param         Specify a solver-specific parameter <param> <val>.
  -h --help       Show this screen.
  -v --version    Show version.

Examples:
  pymoso listitems
  pymoso solve ProbTPA RPERLE 4 14
  pymoso solve --budget=100000 --odir=test1  ProbTPB RMINRLE 3 12
  pymoso solve --seed 12345 32123 5322 2 9543 666666666 ProbTPC RPERLE 31 21 11
  pymoso solve --simpar=4 --param betaeps 0.4 ProbTPA RPERLE 30 30
  pymoso solve --param radius 3 ProbTPA RPERLE 45 45
  pymoso testsolve --isp=16 --proc=4 TPATester RPERLE
  pymoso testsolve --isp=20 --proc=10 --metric --crn TPBTester RMINRLE 9 9
```

Figure 4.11. PyMOSO displays help when users enter the `pymoso --help` invocation.

vocations. The `pymoso --help` invocation prints the file to the terminal. The file is also printed when PyMOSO cannot parse an invocation that begins with `pymoso`. We show the current help file in Figure 4.11.

The `listitems` Command for Viewing Solvers, Testers, and Oracles Included in PyMOSO

The default installation of PyMOSO includes a selection of solvers, testers, and oracles. Users can view the complete lists of included solvers, testers, and oracles using the `pymoso listitems` command. We show the current listing in Figure 4.12. Test problems A, B, and C refer to those in Cooper et al. (2018).

```
Solver                Description
**********************************************************************
RMINRLE               Solver using R-MinRLE for integer-ordered MOSO.
RPE                   Solver using R-Pe for integer-ordered bi-objective MOSO.
RPERLE                Solver using R-PERLE for integer-ordered bi-objective MOSO.
RSPLINE               Solver using R-SPLINE for single objective SO.

Problems              Description                 Test Name (if available)
**********************************************************************
ProbSimpleSO          x^2 + noise.                SimpleSOTester
ProbTPA               Test Problem A              TPATester
ProbTPB               Test Problem B              TPBTester
ProbTPC               Test Problem C              TPCTester
```

Figure 4.12. The `pymoso listitems` invocation shows the lists of built-in solvers, testers, and oracles.

4.4.3 PyMOSO Programming Object List

We describe the object names inside each of the following `pymoso` modules.

`prng.mrg32k3a` The module exposes the pseudo-random number generator and functions to manipulate it.

`MRG32k3a` Sub-class of random.Random, defines all `rng` objects.

`get_next_prnstream(seed)` Return an `rng` object seeded 2^{127} steps from the input seed.

`jump_substream(rng)` Seed the input `rng` object 2^{76} steps forward.

`chnbase` The module implements the base classes for programming oracles and solvers.

`Oracle` Base class for implementing oracles.

`RLESolver` Base class for implementing solvers using RLE.

`RASolver` Base class for implementing RA solvers.

`MOSOSolver` Base class for all solvers.

`chnutils` The module contains generally useful functions for programming or testing algorithms.

`solve(oracle, solver, x0, **kwargs)` See §4.2.2.

`testsolve(tester, solver, x0, **kwargs)` See §4.3.2.

`does_weak_dominate(g, h, relg, relh)` All inputs are tuples of equal length. Returns True if `g` weakly dominates `h` with relaxations.

`does_dominate(g, h, relg, relh)` Returns `True` if `g` dominates `h` with relaxations.

`does_strict_dominate(g, h, relg, relh)` Returns `True` if `g` strictly dominates `h` with relaxations.

`get_nondom(obj_dict)` Input: a dictionary with tuples for keys and values. The keys are feasible points; the values are their objective values. Return: a set of tuples representing non-dominated points.

`get_nbors(x,` r`)` Input: a tuple $\mathbf{x}$, a positive real scalar r indicating the neighborhood radius. Return: Set of tuples, the neighbors.

`get_setnbors(`$\mathcal{S}$`, r)` Input: a set of tuples, and the neighborhood radius. Return: $\cup_{\mathbf{x}\in\mathcal{S}}$ `get_nbors(x,` r`)`.

`dh(A, B)` Returns the Hausdorff distance between set `A` and set `B`.

`edist(x1, x2)` Returns the Euclidean distance between `x1` and `x2`.

`gen_metric(results, tester)` Input: `results` is a dictionary, the output of each sample path of `testsolve`. `tester` must implement `metric`. Returns: The set of triples (iteration, simulation count, metric) for an algorithm run.

`Oracle` When implementing `RAsolver` algorithms, programmers may not need to access `Oracle` objects directly at all. When implementing `MOSOSolver` algorithms, programmers will use (or wrap) `hit` and `crn_advance()`.

`Oracle.num_obj` A positive integer, the number of objectives.

`Oracle.dim` A positive integer, the dimensionality of feasible points.

`Oracle.rng` An instance of `MRG32k3a` internal to the oracle.

`Oracle.hit(x, n)` Take `n` observations of `x`. Return: `True`, and a tuple containing the mean of the observations for each objective `se`, and a tuple containing the standard error for each objective if `x` is feasible. The function handles CRN internally.

`Oracle.set_crnflag(bool)` Turn CRN on (`True`) or off.

`Oracle.set_crnold(state)` Save the `rng` state as the CRN baseline, e.g. for an algorithm iteration.

`Oracle.crn_reset()` Back the oracle `rng` to the CRN baseline.

`Oracle.crn_advance()` If CRN is on, reset, and then jump to the next independent pseudo-random stream and save the new baseline, e.g. before starting a new algorithm iteration.

`Oracle.crn_setobs()` Set an intermediate CRN for individual oracle observations.

`Oracle.crn_nextobs()` Jump the `rng` forward, e.g. after taking an observation, and `set_obs` the seed.

`Oracle.crn_check()` If CRN is on, return to the baseline. Otherwise, use `nextobs` before taking the next observation.

`MOSOSolver` The base class provides a basic structure for implementing new MOSO algorithms in PyMOSO.

`MOSOSolver.orc` The oracle object for the solver to solve.

`MOSOSolver.dim` Number of dimensions of points in the `self.orc`'s feasible points.

`MOSOSolver.num_obj` Similarly, the number of objectives in `self.orc`.

`MOSOSolver.num_calls` A running count of the number of observations taken of `self.orc`.

`MOSOSolver.x0` A feasible starting point. This point is additionally supplied to algorithms that don't need one.

`RASolver` Implements a common structure for all RA algorithms, including: caching of simulation replications, scheduling and updating of sample sizes and limits, and a wrapper to `Oracle.hit`.

`RASolver.sprn` An instance of `MRG32k3a` for the solver to use.

`RASolver.nbor_rad` The neighborhood radius used by solvers seeking local optimality.

`RASolver.gbar` A dictionary where every key and value is a tuple. The keys are feasible points, values are their objective values. `gbar` is "wiped" every retrospective iteration.

`RASolver.sehat` Exactly like `gbar` except the values are standard errors.

`RASolver.m` The sample size of the current iteration.

`RASolver.calc_m(nu)` Compute the sample size of the current iteration. RA algorithms automatically do this every iteration and assign the value to `self.m`.

`RASolver.b` The searching sample limit of the current iteration.

`RASolver.calc_b(nu)` Exactly as `calc_m` but for the searching sample limit.

`RASolver.estimate(x, c, obj)` The `estimate` function is essentially a smart wrapper for `self.orc.hit`. Inputs: tuple `x` to sample, `c` a feasibility constraint, `obj` the objective to constrain. Return: same as `Oracle.hit`. Retrieves or saves the results from/to `gbar` and `sehat` as appropriate. Returns not feasible if the otherwise feasible result is not less than the constraint.

`RASolver.upsample(mcS)` A version of `estimate` for sets. Returns the feasible subset of `mcS`.

`RASolver.spline(x, c, obmin, obcon)` Return a sample path local minimizer. Input: a feasible start, constraint, objective to minimize, objective to constrain. Return: a set of tuples of the trajectory, the minimizer tuple, the minimum tuple, the standard error tuple.

`RLESolver` Builds on `RASolver` to add RLE and its relaxation.

`RLESolver.betadel` Affects the relaxation values computed in RLE.

`RLESolver.calc_delta(se)` Computes the RLE relaxation given a standard error, using `self.m` and `self.betadel`

`RLESolver.rle(candidate_les)` Input: set of tuples, Returns: set of tuples. Finds the LES at sample size `self.m`.

5. CONCLUDING REMARKS

First, we propose a family of algorithms in an RA framework that rely on RLE to converge to a local efficient set. Family members are defined by their accelerator routines, which seek to provide high quality candidate local efficient set members to RLE. We introduce two members: R-PεRLE, a new, provably-convergent algorithm for bi-objective MOSO on integer lattices; and R-MinRLE as a benchmark algorithm for MOSO on integer lattices with two or more objectives. R-PεRLE out-performs both R-MinRLE and the current state- of-the-art algorithm, MO-COMPASS, on our test problems. This work points to a family of RA algorithms for MOSO on integer lattices that employ RLE for sample-path certification of an approximate local efficient set, where the convergence guarantees are provided by Theorem 1.

Second, we propose the PyMOSO software package provides open-source, off-the-shelf access to R-PεRLE and R-MinRLE in an accessible and popular programming language. PyMOSO also provides a framework and useful tools for researchers who wish to compare and create new algorithms, including those that employ RLE for sample-path certification of an approximate local efficient set.

We now discuss several topics of future research.

Random restarts for convergence to the global efficient set Wang et al. (2013) propose R-SPLINE, a RA algorithm which converges to a local minimizer. Nagaraj and Pasupathy (2016) extend R-SPLINE to cgR-SPLINE, an algorithm which performs random restarts of R-SPLINE and, perhaps with stochastic constraints, retrieves a global minimizer. By using a similar scheme for random restarts, we conjecture that a hypothetical gR-PERLE or gR-MinRLE algorithm can retrieve the GES in a MOSO problem.

PyMOSO Algorithms for MOSO on finite sets As with MOSO on continuous sets, PyMOSO is capable of supporting algorithms that solve MOSO problems on finite sets. Candidate algorithms include MOCBA (Lee et al., 2010b) and SCORE (Applegate et al., 2018).

PyMOSO Algorithms for MOSO on continuous sets Though it currently implements no such problems nor algorithms, PyMOSO is capable of supporting problems and algorithms on continuous sets. Implementing problems and algorithms, if only as examples, could encourage PyMOSO use among practitioners and researchers of MOSO on continuous sets. Candidate algorithms include ASTRO-DF (Shashaani et al., 2016) for single objective simulation optimization on continuous sets, and the algorithm of Kim and Ryu (2011b) for bi-objective MOSO on continuous sets.

PyMOSO Testbeds for MOSO on finite, integer-ordered, and continuous sets Though PyMOSO implements some example problems, it does not contain a complete testbed for any class of MOSO problems. Testbeds would allow researchers to easily test their algorithms on a broad class of problems and edge cases to ensure correct algorithm design and implementation.

REFERENCES

Amodeo, L., Prins, C., and Sánchez, D. R. (2009). Comparison of metaheuristic approaches for multi-objective simulation-based optimization in supply chain inventory management. In Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G. A., Ekárt, A., Esparcia-Alcázar, A. I., Farooq, M., Fink, A., Machado, P., McCormack, J., Neri, F., O'Neill, M., Preuss, M., Rothlauf, F., Tarantino, E., and Yang, S., editors, *Applications of Evolutionary Computing*, Lecture Notes in Computer Science, pages 798–807. Springer, Berlin, Germany.

Andersson, M., Grimm, H., Persson, A., and Ng, A. (2007). A web-based simulation optimization system for industrial scheduling. In Henderson, S. G., Biller, B., Hsieh, M.-H., Shortle, J., Tew, J. D., and Barton, R. R., editors, *Proceedings of the 2007 Winter Simulation Conference*, pages 1844–1852, Piscataway, NJ. IEEE.

Applegate, E. A., Feldman, G., Hunter, S. R., and Pasupathy, R. (2018). Multi-objective ranking and selection: Optimal sampling laws and tractable approximations via SCORE. *Optimization Online*.

Applied Materials, Inc. (2018). AutoMod.

Audet, C. and Hare, W. (2017). *Derivative-Free and Blackbox Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, Switzerland.

Baesler, F. F. and Sepulveda, J. A. (2001). Multi-objective simulation optimization for a cancer treatment center. In Peters, B. A., Smith, J. S., Medeiros, D. J., and Rohrer, M. W., editors, *Proceedings of the 2001 Winter Simulation Conference*, pages 1405–1411, Piscataway, NJ. IEEE.

Billingsley, P. (1995). *Probability and Measure*. John Wiley and Sons, New York, 3 edition.

Biscani, F. and Izzo, D. (2018). esa/pagmo2: pagmo 2.9.

Bonnel, H. and Collonge, C. (2014). Stochastic optimization over a Pareto set associated with a stochastic multi-objective optimization problem. *Journal of Optimization Theory and Applications*, 162:405–427.

Branke, J. and Zhang, W. (2015). A new myopic sequential sampling algorithm for multi-objective problems. In Yilmaz, L., Chan, W. K. V., Moon, I., Roeder, T. M. K., Macal, C., and Rossetti, M. D., editors, *Proceedings of the 2015 Winter Simulation Conference*, pages 3589–3598, Piscataway, NJ. IEEE.

Branke, J., Zhang, W., and Tao, Y. (2016). Multiobjective ranking and selection based on hypervolume. In Roeder, T. M. K., Frazier, P. I., Szechtman, R., Zhou, E., Huschka, T., and Chick, S. E., editors, *Proceedings of the 2016 Winter Simulation Conference*, pages 859–870, Piscataway, NJ. IEEE.

Büche, D., Stoll, P., Dornberger, R., and Koumoutsakos, P. (2002). Multiobjective evolutionary algorithm for the optimization of noisy combustion processes. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, 32(4):460–473.

Casella, G. and Berger, R. L. (2002). *Statistical Inference*. Duxbury, Pacific Grove, CA, 2nd edition.

Chan, W. K. V., D'Ambrogio, A., Zacharewicz, G., Mustafee, N., Wainer, G., and Page, E., editors (2017). *Proceedings of the 2017 Winter Simulation Conference*. IEEE, Piscataway, NJ.

Chen, C.-H., Lin, J., Yücesan, E., and Chick, S. E. (2000). Simulation budget allocation for further enhancing the efficiency of ordinal optimization. *Discrete Event Dynamic Systems*, 10(3):251–270.

Chen, H. and Schmeiser, B. W. (2001). Stochastic root finding via retrospective approximation. *IIE Transactions*, 33:259–275.

Chen, T. and Wang, C. (2016). Multi-objective simulation optimization for medical capacity allocation in emergency department. *Journal of Simulation*, 10(1):50–68.

Chew, E. P., Lee, L. H., Teng, S., and Koh, C. H. (2009). Differentiated service inventory optimization using nested partitions and MOCBA. *Computers & Operations Research*, 36(5):1703–1710.

Conn, A. R., Scheinberg, K., and Vicente, L. N. (2009). *Introduction to Derivative-free optimization*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics and Mathetical Programming Society, Philadelphia, PA.

Cooper, K., Hunter, S. R., and Nagaraj, K. (2017). An epsilon-constraint method for integer-ordered bi-objective simulation optimization. In Chan, W. K. V., D'Ambrogio, A., Zacharewicz, G., Mustafee, N., Wainer, G., and Page, E., editors, *Proceedings of the 2017 Winter Simulation Conference*, pages 2303–2314, Piscataway, NJ. IEEE.

Cooper, K., Hunter, S. R., and Nagaraj, K. (2018). Bi-objective simulation optimization on integer lattices using the epsilon-constraint method in a retrospective approximation framework. *Optimization Online*.

Crespo, O., Bergez, J. E., and Garcia, F. (2010). Multiobjective optimization subject to uncertainty: Application to irrigation strategy management. *Computers and Electronics in Agriculture*, 74:145–154.

Deb, K. (1999). Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation*, 7(3):205–230.

Dembo, A. and Zeitouni, O. (1998). *Large Deviations Techniques and Applications*. Springer, New York, 2nd edition.

Ding, H., Benyoucef, L., and Xie, X. (2006). A simulation-based multi-objective genetic algorithm approach for networked enterprises optimization. *Engineering Applications of Artificial Intelligence*, 19:609–623.

Dullinger, C., Struckl, W., and Kozek, M. (2017). Simulation-based multi-objective system optimization of train traction systems. *Simulation Modelling Practice and Theory*, 72:104–117.

Ehrgott, M. and Tenfelde-Podehl, D. (2003). Computation of ideal and nadir values and implications for their use in MCDM methods. *European Journal of Operational Research*, 151:119–139.

Feldman, G. and Hunter, S. R. (2018). SCORE allocations for bi-objective ranking and selection. *ACM Transactions on Modeling and Computer Simulation*, 28(1):7:1–7:28.

Feng, W. H., Kong, N., and Wan, H. (2013). A simulation study of cadaveric liver allocation with a single-score patient prioritization formula. *Journal of Simulation*, 7(2):109–125.

Fliege, J. and Xu, H. (2011). Stochastic multiobjective optimization: sample average approximation and applications. *Journal of Optimization Theory and Applications*, 151:135–162.

Goldsman, D. (2015). A practical guide to ranking and selection methods. In Aleman, D. M. and Thiele, A. C., editors, *TutORials in Operations Research*, chapter 6, pages 89–110. INFORMS, Catonsville, MD.

Henderson, S. G. and Pasupathy, R. (2018). Simulation optimization library.

Hoffenson, S., Arepally, S., and Papalambros, P. Y. (2014). A multi-objective optimization framework for assessing military ground vehicle design for safety. *Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 11(1):33–46.

Hong, L. J. and Nelson, B. L. (2006). Discrete optimization via simulation using COMPASS. *Operations Research*, 54(1):115–129.

Hong, L. J. and Nelson, B. L. (2009). A brief introduction to optimization via simulation. In Rossetti, M. D., Hill, R. R., Johansson, B., Dunkin, A., and Ingalls, R. G., editors, *Proceedings of the 2009 Winter Simulation Conference*, pages 75–85, Piscataway, NJ. IEEE.

Hong, L. J., Nelson, B. L., and Xu, J. (2015). Discrete optimization via simulation. In Fu, M. C., editor, *Handbook of Simulation Optimization*, volume 216 of *International Series in Operations Research & Management Science*, pages 9–44. Springer, New York.

Huang, H. (2016). *Discrete-event simulation and optimization to improve the performance of a healthcare system*. PhD thesis, University of Washington.

Huang, H. and Zabinsky, Z. B. (2014). Multiple objective probabilistic branch and bound for Pareto optimal approximation. In Tolk, A., Diallo, S. Y., Ryzhov, I. O., Yilmaz, L., Buckley, S., and Miller, J. A., editors, *Proceedings of the 2014 Winter Simulation Conference*, pages 3916–3927, Piscataway, NJ. IEEE.

Hunter, S. R., Applegate, E. A., Arora, V., Chong, B., Cooper, K., Rincón-Guevara, O., and Vivas-Valencia, C. (2019). An introduction to multi-objective simulation optimization. *ACM Transactions on Modeling and Computer Simulation*, 29(1).

Hunter, S. R. and McClosky, B. (2016). Maximizing quantitative traits in the mating design problem via simulation-based Pareto estimation. *IIE Transactions*, 48(6):565–578.

Joines, J. A., Gupta, D., Gokce, M. A., King, R. E., and Kay, M. G. (2002). Supply chain multi-objective simulation optimization. In Yücesan, E., Chen, C. H., Snowdon, J. L., and Charnes, J. M., editors, *Proceedings of the 2002 Winter Simulation Conference*, pages 1306–1314, Piscataway, NJ. IEEE.

Kim, S. (2014). A derivative-free trust-region method for biobjective optimization.

Kim, S. and Ryu, J. (2011a). The sample average approximation method for multi-objective stochastic optimization. In Jain, S., Creasey, R. R., Himmelspach, J., White, K. P., and Fu, M., editors, *Proceedings of the 2011 Winter Simulation Conference*, pages 4026–4037, Piscataway, NJ. IEEE.

Kim, S. and Ryu, J. (2011b). A trust-region algorithm for bi-objective stochastic optimization. *Procedia Computer Science*, 4:1422–1430.

Klein, T., Holzkämper, A., Calanca, P., Seppelt, R., and Fuhrer, J. (2013). Adapting agricultural land management to climate change: a regional multi-objective optimization approach. *Landscape Ecology*, 28:2029–2047.

Law, A. M. (2015). *Simulation Modeling and Analysis*. McGraw Hill Education, New York, 5 edition.

L'Ecuyer, P. (1999). Good parameters and implementations for combined multiple recursive random number generators. *Operations Research*, 47(1):159 – 164.

L'Ecuyer, P., Simard, R., Chen, E. J., and Kelton, W. D. (2002). An object-oriented random-number package with many long streams and substreams. *Operations Research*, 50(6):1073–1075.

Lee, L. H., Chew, E. P., Teng, S., and Chen, Y. (2008). Multi-objective simulation-based evolutionary algorithm for an aircraft spare parts allocation problem. *European Journal of Operational Research*, 189(2):476–491.

Lee, L. H., Chew, E. P., Teng, S., and Goldsman, D. (2010a). Finding the non-dominated Pareto set for multi-objective simulation models. *IIE Transactions*, 42:656–674.

Lee, L. H., Chew, E. P., Teng, S., and Goldsman, D. (2010b). Finding the non-dominated Pareto set for multi-objective simulation models. *IIE Transactions*, 42:656–674.

Lee, L. H., Lee, C. U., and Tan, Y. P. (2007). A multi-objective genetic algorithm for robust flight scheduling using simulation. *European Journal of Operational Research*, 177(3):1948–1968.

Li, H., Lee, L. H., Chew, E. P., and Lendermann, P. (2015a). MO-COMPASS: A fast convergent search algorithm for multi-objective discrete optimization via simulation. *IIE Transactions*, 47(11):1153–1169.

Li, H., Lee, L. H., Chew, E. P., and Lendermann, P. (2015b). MO-COMPASS: A fast convergent search algorithm for multi-objective discrete optimization via simulation. *IIE Transactions*, 47(11):1153–1169.

Li, H., Pedrielli, G., Lee, L. H., and Chew, E. P. (2017). Enhancement of supply chain resilience through inter-echelon information sharing. *Flexible Services and Manufacturing Journal*, 29:260–285.

Li, H., Zhu, Y., Chen, Y., Pedrielli, G., Pujowidianto, N. A., and Chen, Y. (2015c). The object-oriented discrete event simulation modeling: a case study on aircraft spare part management. In Yilmaz, L., Chan, W. K. V., Roeder, T. M. K., Macal, C., and Rosetti, M., editors, *Proceedings of the 2015 Winter Simulation Conference*, pages 3514–3525, Piscataway, NJ. IEEE.

Liuzzi, G., Lucidi, S., and Rinaldi, F. (2018). An algorithmic framework based on primitive directions and nonmonotone line searches for black box problems with integer variables. *Optimization Online*.

Lucidi, S., Maurici, M., Paulon, L., Rinaldi, F., and Roma, M. (2016). A simulation-based multi-objective optimization approach for health care service management. *IEEE Transactions on Automation Science and Engineering*, 13(4):1480–1491.

Mattila, V. and Virtanen, K. (2014). Maintenance scheduling of a fleet of fighter aircraft through multi-objective simulation-optimization. *Simulation: Transactions of the Society for Modeling and Simulation International*, 90(9):1023–1040.

Miettinen, K. (1999). *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, Boston.

Nagaraj, K. and Pasupathy, R. (2016). Stochastically constrained simulation optimization on integer-ordered spaces: The cgR-SPLINE algorithm. `http://www.optimization-online.org/DB_HTML/2015/10/5139.html`.

Nelson, B. L. (2010). Optimization via simulation over discrete decision variables. In Hasenbein, J. J., Gray, P., and Greenberg, H. J., editors, *TutORials in Operations Research*, chapter 9, pages 193 – 207. INFORMS, Catonsville, MD.

Nguyen, A., Reiter, S., and Rigo, P. (2014). A review on simulation-based optimization methods applied to building performance analysis. *Applied Energy*, 113:1043–1058.

OptTek Systems, Inc. (2018). OptQuest.

Pasupathy, R. (2010). On choosing parameters in retrospective-approximation algorithms for stochastic root finding and simulation optimization. *Operations Research*, 58(4):889–901.

Pasupathy, R. and Ghosh, S. (2013). Simulation optimization: a concise overview and implementation guide. In Topaloglu, H., editor, *TutORials in Operations Research*, chapter 7, pages 122–150. INFORMS, Catonsville, MD.

Pasupathy, R. and Kim, S. (2011). The stochastic root-finding problem: overview, solutions, and open questions. *ACM Transactions on Modeling and Computer Simulation*, 21(3).

Pasupathy, R. and Schmeiser, B. W. (2009). Retrospective-approximation algorithms for multidimensional stochastic root-finding problems. *ACM Transactions on Modeling and Computer Simulation*, 19(2):5:1–5:36.

Prakash, O., Srinivasan, K., and Sudheer, K. P. (2015). Adaptive multi-objective simulation-optimization framework for dynamic flood control operation in a river-reservoir system. *Hydrology Research*, 46(6):893–911.

Ryu, J. and Kim, S. (2014). A derivative-free trust-region method for biobjective optimization. *SIAM J. Optim.*, 24(1):334–362.

Schmeiser, B. (2008). A practitioner, a vender, and a researcher walk into a bar: trying to explain what researchers do. In Mason, S. J., Hill, R. R., Mönch, L., Rose, O., Jefferson, T., and Fowler, J. W., editors, *Proceedings of the 2008 Winter Simulation Conference*, pages 2–9, Piscataway, NJ. IEEE.

Shashaani, S., Hunter, S. R., and Pasupathy, R. (2016). ASTRO-DF: Adaptive sampling trust-region optimization algorithms, heuristics, and numerical experience. In Roeder, T. M. K., Frazier, P. I., Szechtman, R., and Zhou, E., editors, *Proceedings of the 2016 Winter Simulation Conference*, pages 554–565, Piscataway, NJ. IEEE.

Simio LLC (2018). Simio.

Singh, A. and Minsker, B. S. (2008). Uncertainty-based multiobjective optimization of groundwater remediation design. *Water Resources Research*, 44.

Song, J., Qiu, Y., and Liu, Z. (2016). Integrating optimal simulation budget allocation and genetic algorithm to find the approximate Pareto patient flow distribution. *IEEE Transactions on Automation Science and Engineering*, 13(1):149–159.

Subramanyan, K., Diwekar, U., and Zitney, S. E. (2011). Stochastic modeling and multi-objective optimization for the APECS system. *Computers and Chemical Engineering*, 35:2667–2679.

The AnyLogic Company (2018). AnyLogic.

Thengvall, B., Glover, F., and Davino, D. (2016). Coupling optimization and statistical analysis with simulation models. In Roeder, T. M. K., Frazier, P. I., Szechtman, R., Zhou, E., Huschka, T., and Chick, S. E., editors, *Proceedings of the 2016 Winter Simulation Conference*, Piscataway, NJ. IEEE.

Villarreal-Marroquín, M. G., Svenson, J. D., Sun, F., Santner, T. J., Dean, A., and Castro, J. M. (2013). A comparison of two metamodel-based methodologies for multiple criteria simulation optimization using an injection molding case study. *Journal of Polymer Engineering*, 33(3):193–209.

Wang, H., Pasupathy, R., and Schmeiser, B. W. (2013). Integer-ordered simulation optimization using R-SPLINE: Retrospective Search using Piecewise-Linear Interpolation and Neighborhood Enumeration. *ACM Transactions on Modeling and Computer Simulation*, 23(3).

Wang, W. and Wan, H. (2017). Sequential probability ratio test for multiple-objective ranking and selection. In Chan, W. K. V., D'Ambrogio, A., Zacharewicz, G., Mustafee, N., Wainer, G., and Page, E., editors, *Proceedings of the 2017 Winter Simulation Conference*, pages 1998–2009, Piscataway, NJ. IEEE.

Wang, Y., Lee, L. H., Chew, E. P., Lam, S. S. W., Low, S. K., Ong, M. E. H., and Li, H. (2015). Multi-objective optimization for a hospital inpatient flow process via discrete event simulation. In Yilmaz, L., Chan, W. K. V., Roeder, T. M. K., Macal, C., and Rosetti, M., editors, *Proceedings of the 2015 Winter Simulation Conference*, pages 3622–3631, Piscataway, NJ. IEEE.

Weizhi, L. (2017). PyPRS. GitHub repository.

Wiecek, M. M., Ehrgott, M., and Engau, A. (2016). Continuous multiobjective programming. In Greco, S., Ehrgott, M., and Figueira, J. R., editors, *Multiple Criteria Decision Analysis: State of the Art Surveys*, volume 233 of *International Series in Operations Research & Management Science*, pages 739–815. Springer New York, New York.

Zhang, H. (2008). Multi-objective simulation-optimization for earthmoving operations. *Automation in Construction*, 18:79–86.

Zhou, C., Li, H., Lee, B. K., and Qiu, Z. (2018). A simulation-based vessel-truck coordination strategy for lighterage terminals. *Transportation Research Part C: Emerging Technologies*, 95:149–164.