SODA-m *(to update)*[1]

Mohan Krishnamoorthy[a,2], Alexander Brodsky[a], and Daniel A. Menascé[a]

[a] *Department of Computer Science, George Mason University, Fairfax, Virginia 22030, USA.*

[a] {mkrishn4, brodsky, menasce}@gmu.edu

Acknowledgement
This work was partially supported by NIST Grant No. 70NANB12H277.

[1] Word Count: 6319 *(to update)*
[2] Corresponding Author

SODA-m *(to update)*

Abstract

We consider a process with feasibility constraints and metrics of interest including a cost function where the metrics are stochastic functions of the process controls. We propose an efficient stochastic optimization algorithm for the problem of finding process controls that minimize the expectation of cost while satisfying multiple deterministic and stochastic feasibility constraints with a given high probability. The proposed algorithm is based on (1) a series of deterministic approximations to produce a candidate set of near-optimal control settings for the production process, and (2) stochastic simulations on the candidate set using optimal simulation budget allocation methods. In an experimental study, we demonstrate the proposed algorithm on a use case of a real-world heat-sink service network that involves contract suppliers and manufacturers as well as unit manufacturing processes of shearing, milling, drilling, and machining. The experimental study shows that the proposed algorithm significantly outperforms four popular simulation-based stochastic optimization algorithms.

Keywords: decision support; decision guidance; deterministic approximations; stochastic simulation optimization; heuristic algorithm

1 Introduction

This paper considers a process with feasibility constraints and metrics of interest including a cost function where the metrics are stochastic functions of the process controls. This paper is concerned with the development of a one-stage stochastic optimization algorithm for the problem of finding process controls that minimize the expectation of cost while satisfying multiple deterministic and stochastic feasibility constraints with a given high probability. These problems are prevalent in manufacturing processes, such as assembly lines and supply chain management where the goal is to find the process controls that minimize the expected cost subject to satisfying deterministic control bound constraints and stochastic steady state demand for the multiple output products with a given high probability. There is an increasing need for process analysis and optimization to solve this problem efficiently as companies want to be competitive and need to reduce their cost and improve efficiency of operations in the face of increased global competition.

Stochastic optimization have typically been performed using simulation-based optimization techniques (see (Amaran, Sahinidis, Sharda, & Bury, 2016) and (Nguyen, Reiter, & Rigo, 2014) for a review of such techniques). Tools like SIMULINK (Dabney & Harman, 2001) and Modelica (Fritzson, 2004; Provan & Venturini, 2012) allow users to run stochastic simulations on models of complex systems in mechanical, hydraulic, thermal, control, and electrical power. Tools like OMOptim (Thieriota et al., 2011), Efficient Traceable Model-Based Dynamic Optimization (EDOp) (OpenModelica, 2009), and jMetal (Durillo & Nebro, 2011) use simulation models to heuristically-guide a trial and error search for the optimal answer. However, the general limitation of simulation-based approaches is that simulation is used as a black box, and the underlying mathematical structure is not utilized.

From the work on Mathematical Programming (MP), we know that, for deterministic problems, utilizing the mathematical structure can lead to significantly better results in terms of optimality of results and computational complexity compared to simulation-based approaches (see e.g., (Amaran et al., 2016) and (Klemmt, Horn, Weigert, & Wolter, 2009)). For this reason, a number of approaches have been developed to bridge the gap between stochastic

simulation and MP. For instance, (Thompson & Davis, 1990) propose an integrated approach that combines simulation with MP where the MP problem is constructed from the original stochastic problem with uncertainties being resolved to their mean values by using a sample of black-box simulations. This strategy of extracting an MP from the original problem is also used by (Paraskevopoulos, Karakitsos, & Rustem, 1991) to solve the optimal capacity planning problem by incorporating the original objective function augmented with a penalty on the sensitivity of the objective function to various types of uncertainty. The authors of (Xu et al., 2014) propose an ordinal transformation framework, consisting of a two-stage optimization framework that first extracts a low fidelity model using simulation or a queuing network model using assumptions that simplify the original problem and then uses this model to reduce the search space over which high fidelity simulations are run to find the optimal solution to the original problem. Other stochastic optimization approaches in the literature try to extract the mathematical structure of the original problem using similar techniques. However, extraction of the mathematical structure through sampling using a black-box simulation is computationally expensive, especially for real-world processes composed of complex process networks.

Instead of extracting the mathematical structure using black-box simulation, in (Krishnamoorthy, Brodsky, & Menascé, 2015), we used the extraction of mathematical structure from a white-box simulation code analysis as part of a heuristic algorithm to solve a stochastic optimization problem of finding controls for temporal production processes with inventories as to minimize the total cost while satisfying the stochastic demand with a predefined probability. Similar to the previous approaches, the mathematical structure was used for approximating a candidate set of solutions by solving a series of deterministic MP problems that approximate the stochastic simulation. However, the class of problems considered in (Krishnamoorthy et al., 2015) is limited to processes described using piece-wise linear arithmetic. Whereas, many processes have models based on physics-based equations with non-linear arithmetic.

To close this gap, we extended the heuristic algorithm from (Krishnamoorthy et al., 2015) to an algorithm called Stochastic Optimization Algorithm based on Deterministic Approximations (SODA) to solve the stochastic optimization problem over a composite service network that involve processes described using non-linear arithmetic (Krishnamoorthy, Brodsky, & Menascé, 2017). However, SODA was only designed for problems with processes that involved stochastic constraint over a single output metric. Whereas, many processes, particularly in manufacturing, have multiple outputs and the stochastic optimization problem needs to consider the satisfaction of the stochastic constraint over all these output metrics. Hence, in this paper, we generalize SODA to close the gap for stochastic optimization problems for processes that have feasibility constraints over multiple stochastic metrics while being described using non-linear arithmetic.

More specifically, the contributions of this paper are two-fold: First, we propose a heuristic algorithm called Stochastic Optimization Algorithm based on Deterministic Approximations with multiple constraints (SODA-m) to solve the problem of finding process controls that minimize the expectation of cost while satisfying multiple deterministic and stochastic feasibility constraints with a given high probability. The proposed algorithm is based on (1) a series of deterministic approximations to produce a candidate set of near-optimal control settings for the production process, and (2) stochastic simulations on the candidate set using optimal simulation budget allocation methods (e.g., see (Chen & Lee, 2011), (Lee, Pujowidianto, Li, Chen, & Yap, 2012)). Second, we conduct an initial experimental study over a real world use case of heat-sink service network to compare the proposed algorithm with four popular simulation-based stochastic optimization algorithms viz., Nondominated Sorting Genetic Algorithm 2 (NGSA2) (Deb, Pratap, Agarwal, & Meyarivan, 2002), Indicator Based Evolutionary Algorithm (IBEA) (Zitzler & Künzli, 2004), Strength Pareto Evolutionary Algorithm 2 (SPEA2) (Zitzler, Laumanns, &

Thiele, 2001), and Speed-constrained Multi-objective Particle swarm optimization (SMPSO) (Nebro et al., 2009). The experimental study demonstrates that SODA-m ***(complete this after running experiments)***

write organization of the paper

2 Stochastic Optimization over Processes with Multiple Stochastic Feasibility Constraints and Closed-form Non-linear Arithmetic

We now borrow the stochastic optimization problem from (Krishnamoorthy et al., 2017) and extend it for processes that have feasibility constraints over multiple stochastic metrics and are described using non-linear arithmetic. The stochastic optimization problem for such processes assumes a stochastic closed-form arithmetic (SCFA) simulation of the following form. A SCFA simulation on input variable $\vec{X}$ is a sequence $y_1 = expr_1, \ldots, y_n = expr_n$
where $expr_i, 1 \leq i \leq n$ is either

(a) An arithmetic or boolean expression in terms of a subset of the elements of $\vec{X}$ and/or $y_1, \ldots, y_{i-1}$. We say that y_i is arithmetic or boolean if the $expr_i$ is arithmetic or boolean correspondingly.

(b) An expression invoking $PD(\vec{P})$, which is a function that draws from a probability distribution (e.g., gaussian, exponential, uniform) using parameters $\vec{P}$ that are a subset of the elements of $\vec{X}$ and/or $y_1, \ldots, y_{i-1}$.

We say that $y_i, 1 \leq i \leq n$ is stochastic if, recursively,

(a) $expr_i$ invokes $PD(\vec{P})$, or

(b) $expr_i$ uses at least one stochastic variable $y_j, 1 \leq j < i$

If y_i is not stochastic, we say that it is deterministic. Also, we say that a SCFA simulation $\mathbb{S}$ computes a variable v if $v = y_i$, for some $1 \leq i \leq n$.
To clarify, consider a simple SCFA simulation example that consists of the following sequence of expressions:

```
1: stochSpeed := speed + 𝒩(0,σ)
2: stochTime := f(stochSpeed)
3: throughput := 1/stochTime
4: cost := throughput × pricePerUnit
5: C := lb ≤ speed ≤ ub
```

In this example, the SCFA simulation computes the stochastic arithmetic variables of *cost* and *throughput* as well as the deterministic boolean variable *C*. The variable *speed* is deterministic (e.g., machine speed) and it should be bounded within some lower bound (*lb*) and upper bound (*ub*). The boolean variable *C* describes whether *speed* is bounded. Also, the effects of *speed* is stochastic (*stochSpeed*) due to normally distributed random noise $\mathcal{N}(0, \sigma)$. For the sake of brevity, say that the stochastic time to produce one item (*stochTime*) is obtained from a function described in terms of *stochSpeed*. Then, *throughput* is computed as the inverse of *stochTime* and *cost* is computed as the product of *throughput* and a fixed parameter of price to produce one unit of item (*pricePerUnit*).

This paper considers the stochastic optimization problem of finding process controls that minimize the cost expectation while satisfying multiple deterministic and stochastic feasibility

constraints with a given probability. More formally, the stochastic optimization problem is of the form:

$$\begin{aligned} \underset{\vec{X}\in\vec{D}}{\text{minimize}} \quad & E(cost(\vec{X})) \\ \text{subject to} \quad & C(\vec{X}) \wedge \\ & \forall_{k\in\{1,\ldots,|\mathcal{M}|\}} P(m_k(\vec{X}) \geq \theta_k) \geq \alpha_k \end{aligned} \tag{1}$$

where $\vec{D} = D_1 \times \cdots \times D_n$ is the domain for decision variables $\vec{X}$
$\vec{X}$ is a vector of decision variables that range over $\vec{D}$
$cost(\vec{X})$ is a random variable defined in terms of $\vec{X}$
$C(\vec{X})$ is a deterministic constraint in $\vec{X}$ i.e., a function $C : \vec{D} \rightarrow \{true, false\}$
$\mathcal{M}$ is a set of random variables, all defined in terms of $\vec{X}$ and $m_k(\vec{X}) \in \mathcal{M}$
$\theta_k \in \mathbb{R}$ is the threshold for m_k
$\alpha_k \in [0, 1]$ is the k^{th} probability threshold, and
$P(m_k(\vec{X}) \geq \theta_k)$ is the probability that $m_k(\vec{X})$ is greater than or equal to θ_k

Note in this problem that upon increasing any θ_k to some θ'_k, the size of the space of alternatives that satisfy the respective stochastic constraint in equation 1 increases and hence it can be said that the best solution, i.e., the minimum expected cost is monotonically improving in θ'_k.

We assume that the random variables, $cost(\vec{X})$, $m_1(\vec{X}), \ldots, m_k(\vec{X})$ as well as the deterministic constraint $C(\vec{X})$ are expressed by an SCFA simulation $\mathbb{S}$ that computes the stochastic arithmetic variable $cost(\vec{X})$, $m_1(\vec{X}), \ldots, m_k(\vec{X}) \in \vec{\mathbb{R}}^{k+1}$ as well as the deterministic boolean variable $C \in \{true, false\}$.

3 Stochastic Optimization Algorithm based on Deterministic Approximations with multiple constraints

This section presents the Stochastic Optimization Algorithm based on Deterministic Approximations with multiple constraints (SODA-m). The problem of stochastic optimization considered in this paper can be solved using simulation-based optimization approaches by initializing the control settings and performing simulations to check whether the stochastic constraints are satisfied with sufficient probability. But such an approach is inefficient because the stochastic space of this problem is very large and hence this approach will typically converge very slowly to the optimum solution. So, the key idea of SODA-m is that instead of working with a large number of choices in the stochastic space, we use deterministic approximations to generate a small set of candidate control settings and then validate these control settings in the stochastic space using simulations.

An overview of SODA-m is shown in Fig. 1 and a corresponding pseudo code is given in Algorithm 1. To generate a small set of candidate control settings, SODA-m performs deterministic approximations of the original stochastic problem. SODA-m achieves this by defining a deterministic computation $\mathbb{S}_0$ from the SCFA simulation $\mathbb{S}$ described in section 2 by replacing every expression that uses a probability distribution $PD(\vec{P})$ with the expectation of that distribution. Thus the deterministic approximations $cost_0(\vec{X})$, $m_{0,1}(\vec{X}), \ldots, m_{0,k}(\vec{X})$ of $cost(\vec{X})$, $m_1(\vec{X}), \ldots, m_k(\vec{X})$, respectively can be expressed using $\mathbb{S}_0$. To optimize this reduced problem, a deterministic optimization problem that approximates the stochastic optimization problem shown in equation 1 is used as a heuristic. This deterministic optimization problem

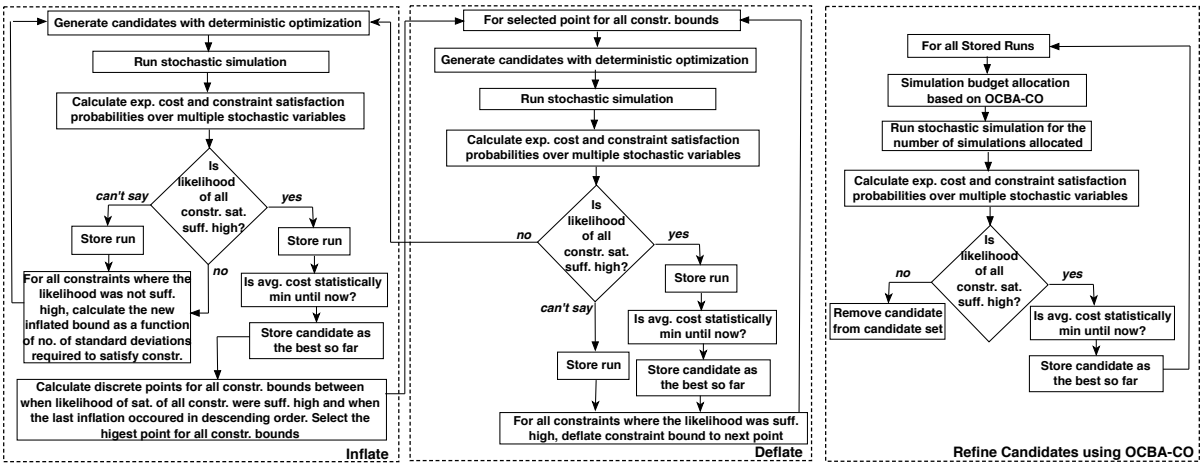

Figure 1: Overview of SODA-m

Algorithm 1: Stochastic Optimization of NL Process with multiple constraints

Input : $\vec{B}, \vec{\sigma}, \overrightarrow{min}, \overrightarrow{max}, \mathcal{D}$
ConfigParams: $\delta_{cost}, \delta_{restart}$, noSimulations, $\overrightarrow{PB}$, CB, $\vec{\alpha}$, $\vec{\beta}$, totalIterations, storeSize, maxSimBudget, budgetDelta, budgetThreshold
Output : bestCandidate, $\hat{O}.m$

1 $acceptedCandSet_1$, $acceptedCandSet_2$, $\hat{O}.m \leftarrow$ `InflateDeflate` ($\vec{B}, \vec{\sigma}, \overrightarrow{min}, \overrightarrow{max}, \mathcal{D}, \delta_{cost}, \delta_{restart}$, noSimulations, $\overrightarrow{PB}$, CB, $\vec{\alpha}$, $\vec{\beta}$, totalIterations, storeSize, maxSimBudget) `// Algorithm 2`
2 bestCandidate, $\hat{O}.m \leftarrow$ `RefineCandidates` ($acceptedCandSet_1$, $acceptedCandSet_2$, $\vec{B}$, $\vec{\sigma}$, $\hat{O}.m$, $\mathcal{D}$, noSimulations, $\overrightarrow{PB}$, CB, maxSimBudget, budgetDelta, budgetThreshold) `// Algorithm 7`
3 **return** bestCandidate, $\hat{O}.m$

Algorithm 2: InflateDeflate

Input : $\vec{B}, \vec{\sigma}, \overrightarrow{min}, \overrightarrow{max}, \mathcal{D}$
ConfigParams: $\delta_{cost}, \delta_{restart}$, noSimulations, $\overrightarrow{PB}$, CB, $\vec{\alpha}$, $\vec{\beta}$, totalIterations, storeSize, maxSimBudget
Output : $acceptedCandSet_1$, $acceptedCandSet_2$, $\hat{O}.m$

1 $\overrightarrow{CIB} \leftarrow \vec{B}$
2 $\hat{O}.m \leftarrow \infty$ `// Best expected objective cost till now`
3 noCandidates $\leftarrow 1$
4 noIterations $\leftarrow 1$
5 **repeat**
6 `// Algorithm 3`
7 ($acceptedCandSet_1$, $acceptedCandSet_2$, noCandidates, $\hat{O}.m, \overrightarrow{LIB}, \overrightarrow{CIB}$) $\leftarrow$ `PerformInflations` ($\vec{B}, \overrightarrow{CIB}, \vec{\sigma}, \overrightarrow{min}, \overrightarrow{max}, \mathcal{D}$, $acceptedCandSet_1$, $acceptedCandSet_2$, noCandidates, $\hat{O}.m, \delta_{cost}, \delta_{restart}$, noSimulations, maxSimBudget, $\overrightarrow{PB}$, CB, $\vec{\alpha}$) `// Algorithm 5`
8 ($acceptedCandSet_1$, $acceptedCandSet_2$, noCandidates, $\hat{O}.m, \overrightarrow{CIB}$) $\leftarrow$ `PerformDeflations`($\vec{B}, \overrightarrow{LIB}, \overrightarrow{CIB}, \vec{\sigma}, \overrightarrow{min}, \overrightarrow{max}$, $\mathcal{D}$,$acceptedCandSet_1$, $acceptedCandSet_2$, noCandidates, $\hat{O}.m, \delta_{cost}, \delta_{restart}$, noSimulations, maxSimBudget, $\overrightarrow{PB}$, CB, $\vec{\beta}$)
9 noIterations $\leftarrow$ noIterations + 1
10 **until** noIterations > totalIterations or noCandidates > storeSize or $\hat{O}.m_{noIterations} - \hat{O}.m_{noIterations-\delta} = 0$
11 **return** $acceptedCandSet_1$, $acceptedCandSet_2$, $\hat{O}.m$

can be described as follows:

$$\begin{aligned} \underset{\vec{X} \in \vec{D}}{\text{minimize}} \quad & cost_0(\vec{X}) \\ \text{subject to} \quad & C(\vec{X}) \wedge \\ & m_{0,1}(\vec{X}) \geq \theta_1{}' \wedge \cdots \wedge m_{0,k}(\vec{X}) \geq \theta_k{}' \end{aligned} \tag{2}$$

where $\theta_1{}' \geq \theta_1, \ldots, \theta_k{}' \geq \theta_k$ are conservative approximations of $\theta_1, \ldots, \theta_k$.

This deterministic approximation is performed iteratively such that the control settings found in the current iteration are more likely to satisfy the stochastic constraints with the desired probability than in the previous iterations. This is possible because of the *inflate* phase (left box of Fig. 1) and the *deflate* phase (middle box of Fig. 1) of SODA-m.

The *inflate* phase is intuitively trying to increase the bounds to satisfy the stochastic constraints with the desired probability. When the current candidate control settings do not generate any metric that satisfies its respective user-defined bound with a desired probability, this bound parameter itself is artificially inflated as a function of the number of standard deviations required so that the metric satisfies the bound with desired probability. This inflation very quickly yields

Algorithm 3: PerformInflations

Input : $\vec{B}, \overrightarrow{CIB}, \vec{\sigma}, \overrightarrow{min}, \overrightarrow{max}, \mathcal{D}$, *acceptedCandSet*$_1$, *acceptedCandSet*$_2$, noCandidates, $\hat{O}.m$
ConfigParams: $\delta_{cost}, \delta_{restart}$, noSimulations, maxSimBudget, $\overrightarrow{PB}$, CB, $\vec{\alpha}$
Output : *acceptedCandSet*$_1$, *acceptedCandSet*$_2$, noCandidates, $\hat{O}.m, \overrightarrow{LIB}, \overrightarrow{CIB}$

$\overrightarrow{LIB} \leftarrow \overrightarrow{CIB}$
repeat
  (result, $\overrightarrow{NIB}, X_1, \{X_2, \ldots, X_k\}, O.m, O.sd, \overrightarrow{SC.m}, \overrightarrow{SC.sd}$, N) ← `Inflate` ($\vec{B}, \overrightarrow{CIB}, \vec{\sigma}, \overrightarrow{min}, \overrightarrow{max}, \mathcal{D}, \delta_{cost}, \delta_{restart}$, noSimulations, maxSimBudget, $\overrightarrow{PB}$, CB, $\vec{\alpha}$) `// Algorithm 4`
  if result is accept or not-reject **then**
    acceptedCandSet$_2$ ← acceptedCandSet$_2$∪ $\{\overrightarrow{X_2}, \ldots, \overrightarrow{X_k}\}$
    noCandidates ← noCandidates + (k - 1)
    if result is accept and O.m statistically better than $\hat{O}$.m with at least CB confidence **then**
      acceptedCandSet$_1$ ← acceptedCandSet$_1$∪ $\{(\overrightarrow{X_1}, O.m, O.sd, \overrightarrow{SC.m}, \overrightarrow{SC.sd}$, N)$\}$
      $\hat{O}$.m ← O.m
      noCandidates ← noCandidates + 1
    end
  else
    $\overrightarrow{LIB} \leftarrow \overrightarrow{CIB}$
    $\overrightarrow{CIB} \leftarrow \overrightarrow{NIB}$
  end
until result is accept
return *acceptedCandSet*$_1$, *acceptedCandSet*$_2$, noCandidates, $\hat{O}.m, \overrightarrow{LIB}, \overrightarrow{CIB}$

Algorithm 4: Inflate

Input : $\vec{B}, \overrightarrow{CIB}, \vec{\sigma}, \overrightarrow{min}, \overrightarrow{max}, \mathcal{D}$
ConfigParams: $\delta_{cost}, \delta_{restart}$, noSimulations, maxSimBudget, $\overrightarrow{PB}$, CB, $\vec{\alpha}$
Output : result, $\overrightarrow{NIB}, X_1, \{X_2, \ldots, X_k\}, O.m, O.sd, \overrightarrow{SC.m}, \overrightarrow{SC.sd}$, N

$((\overrightarrow{X_1}, O_1), \ldots, (\overrightarrow{X_k}, O_k))$ ← `GenerateCandidates` ($\overrightarrow{CIB}, \overrightarrow{min}, \overrightarrow{max}, \delta_{cost}, \delta_{restart}$)
(result, $O.m, O.sd, \overrightarrow{SC.m}, \overrightarrow{SC.sd}$, N) ← `PerformStochasticSimulations` ($\overrightarrow{X}_1, \vec{B}, \vec{\sigma}, \mathcal{D}$, noSimulations, maxSimBudget, $\overrightarrow{PB}$, CB) `// Algorithm 9`
$\overrightarrow{NIB} \leftarrow [\,]$
if result is reject or not-reject **then**
  for $i \in \mathcal{D}$ **do**
    if $P(SC.m_i \geq PB_i) \not\geq CB$ **then**
      $b_i \leftarrow SC.m_i - \frac{\text{StdNormInvCDF}((1-PB_i)+\epsilon) \times SC.sd_i}{\sqrt{N}}$
      $\Delta \leftarrow b_i - CIB_i$
      $NIB_i \leftarrow CIB_i + (\alpha_i \times \Delta)$
    else
      $NIB_i \leftarrow CIB_i$
    end
  end
end
return result, $\overrightarrow{NIB}, X_1, \{X_2, \ldots, X_k\}, O.m, O.sd, \overrightarrow{SC.m}, \overrightarrow{SC.sd}$, N

Algorithm 5: PerformDeflations

Input : $\vec{B}, \overrightarrow{LIB}, \overrightarrow{CIB}, \vec{\sigma}, \overrightarrow{min}, \overrightarrow{max}, \mathcal{D}$, *acceptedCandSet*$_1$, *acceptedCandSet*$_2$, noCandidates, $\hat{O}.m$
ConfigParams: $\delta_{cost}, \delta_{restart}$, noSimulations, maxSimBudget, $\overrightarrow{PB}$, CB, $\vec{\beta}$
Output : *acceptedCandSet*$_1$, *acceptedCandSet*$_2$, noCandidates, $\hat{O}.m, \overrightarrow{CIB}$

repeat
  `// Algorithm 6`
  (result, $\overrightarrow{NIB}, X_1, \{X_2, \ldots, X_k\}, O.m, O.sd, \overrightarrow{SC.m}, \overrightarrow{SC.sd}$, N) ← `Deflate` ($\vec{B}, \overrightarrow{LIB}, \overrightarrow{CIB}, \vec{\sigma}, \overrightarrow{min}, \overrightarrow{max}, \mathcal{D}, \delta_{cost}, \delta_{restart}$, noSimulations, $\overrightarrow{PB}$, CB, $\vec{\beta}$)
  if result is accept or not-reject **then**
    acceptedCandSet$_2$ ← acceptedCandSet$_2$∪ $\{\overrightarrow{X_2}, \ldots, \overrightarrow{X_k}\}$
    noCandidates ← noCandidates + (k - 1)
    if result is accept and O.m statistically better than $\hat{O}$.m with at least CB confidence **then**
      acceptedCandSet$_1$ ← acceptedCandSet$_1$∪ $\{(\overrightarrow{X_1}, O.m, O.sd, \overrightarrow{SC.m}, \overrightarrow{SC.sd}$, N)$\}$
      $\hat{O}$.m ← O.m
      noCandidates ← noCandidates + 1
    end
  end
  $\overrightarrow{CIB} \leftarrow \overrightarrow{NIB}$
until result is reject
return *acceptedCandSet*$_1$, *acceptedCandSet*$_2$, noCandidates, $\hat{O}.m, \overrightarrow{CIB}$

Algorithm 6: Deflate

Input : $\vec{B}, \overrightarrow{LIB}, \overrightarrow{CIB}, \vec{\sigma}, \overrightarrow{min}, \overrightarrow{max}, \mathcal{D}$
ConfigParams: $\delta_{cost}, \delta_{restart}$, noSimulations, $\overrightarrow{PB}$, CB, $\vec{\beta}$
Output : result, $\overrightarrow{NIB}, X_1, \{X_2, \ldots, X_k\}, O.m, O.sd, \overrightarrow{SC.m}, \overrightarrow{SC.sd}$, N

$((\overrightarrow{X_1}, O_1), \ldots, (\overrightarrow{X_k}, O_k))$ ← `GenerateCandidates` ($\overrightarrow{CIB}, \overrightarrow{min}, \overrightarrow{max}, \delta_{cost}, \delta_{restart}$)
`// Algorithm 9`
(result, $O.m, O.sd, \overrightarrow{SC.m}, \overrightarrow{SC.sd}$, N) ← `PerformStochasticSimulations` ($\overrightarrow{X}_1, \vec{B}, \vec{\sigma}, \mathcal{D}$, noSimulations, maxSimBudget, $\overrightarrow{PB}$, CB)
$\overrightarrow{NIB} \leftarrow [\,]$
if result is accept **then**
  for $i \in \mathcal{D}$ **do**
    $\Delta \leftarrow CIB_i - LIB_i$
    if $(\beta_i \times \Delta) \geq \tau$ **then**
      $NIB_i \leftarrow CIB_i - (\beta_i \times \Delta)$
    else
      $NIB_i \leftarrow CIB_i$
    end
  end
end
return result, $\overrightarrow{NIB}, X_1, \{X_2, \ldots, X_k\}, O.m, O.sd, \overrightarrow{SC.m}, \overrightarrow{SC.sd}$, N

metrics that satisfy its respective user-defined bound. However, this may result in the metric overshooting the bound, which degrades the objective cost.

To overcome this, in the *deflate* phase, SODA-m scales the inflated bounds back by calculating the discrete points for all bounds between when the constraint was satisfied with sufficient probability and when the last inflation occurred. Deterministic approximations are run for this lower bound to check whether the updated metric still satisfies the user-defined bound with desired probability while yielding a better objective cost. In this way, the *inflate* and *deflate* phases find a more optimum bound threshold for all stochastic constraints for which it can get the right balance between optimum cost and stochastic constraint satisfaction with desired probability.

After the iterative *inflate* and *deflate* procedure, more simulations may be needed to check if a promising candidate that currently is not a top candidate could be the optimal solution or to choose an optimal solution from multiple candidates. To resolve this, these candidates are further refined in the *refineCandidates* phase (right box in Fig. 1) where a number of simulations are allocated to each candidate as obtained from an optimal simulation budget allocation method. These simulations are run on these candidates to check if this produces a new optimal solution.

In this way, SODA-m uses model knowledge from the white box simulation code in *inflate*, and *deflate* phases, and optimal simulation budget allocation in *refineCandidates* phase to provide optimal control settings of the non-linear processes with multiple stochastic constraints that a process operator can use on the manufacturing floor in a stochastic environment. The phases of SODA-m are explained in greater detail with the help of their pseudo-code in the subsections below.

3.1 Inflate and Deflate Phase

The pseudo-code for the *InflateDeflate* procedure is shown in Algorithm 2 and it iteratively calls *PerformInflations* and *PerformDeflations* to perform deterministic approximation by inflating and deflating the bound parameters for a user defined budget or until there is no improvement in

the objective cost ($\hat{O}.m$) for a certain number of iterations. The uniqueness of this procedure is that it is iterative and hence not only will the inflated bounds that satisfied the constraints with sufficient probability be the starting point for deflation but the bounds at which the deflation fails to satisfy any constraint with sufficient probability will be used to inflate from, thus yielding optimal solutions very early in the running of SODA-m.

The *PerformInflations* procedure as shown in Algorithm 3 tries to iteratively find the inflated bounds for multiple constraints into $\overrightarrow{CIB}$ that can be used for deterministic approximation. The goal is that these inflated bounds will yield control settings when the problem in equation 2 is solved and when these controls are used in the stochastic setting, the original constraints from the stochastic problem will be satisfied with sufficient probability. To achieve this, the *PerformInflations* procedure calls *Inflate* that performs one iteration of inflation as shown in Algorithm 4.

In the *Inflate* procedure, first a number of candidate control settings with different objective costs are obtained by running a deterministic optimization with a constraint on the current bounds from $\overrightarrow{CIB}$ in the *GenerateCandidates* procedure (Algorithm 4 line 1). We adopt *GenerateCandidates* procedure as is from (Krishnamoorthy et al., 2017) where random starting point are chosen for all the decision variables since the processes considered here have non-linear objectives that may not necessarily be convex. Then, the deterministic problem with the current bounds and different starting points are solved to generate different candidates with different control setting and objective costs. More information and pseudo-code of this procedure can be found in (Krishnamoorthy et al., 2017).

The *PerformInflations* procedure then calls the *PerformStochasticSimulations* procedure as shown in Algorithm 9 that runs stochastic simulations on the candidate that yielded the least cost among the generated candidates and calculates the confidence of satisfaction of all the constraints as well as a verdict for this candidate that is returned back through the *result* variable (Algorithm 4 line 2). The *result* variable can be *accept*, when the probability of satisfaction of all the stochastic constraints are sufficiently high, *reject*, when the probability of satisfaction of at least one of the stochastic constraint is not sufficiently high, and *not-reject*, when *accept* or *reject* could not be decided for the number of simulations performed.

For all the constraints where the probability of satisfaction of the stochastic constraint is not sufficiently high, a new inflated bound is computed as a function of number of standard deviations required to satisfy the constraint with desired probability. The assumption here is that the population is normally distributed and the standard deviation approaximately remains the same for the current bound and the inflated bound. Hence the inflated bound for the constraint is calculated by rearranging the terms in one-sample t-test (Algorithm 4 lines 4-14). Finally, the *result* variable, the new bounds ($\overrightarrow{NIB}$) and the generated candidates from the current iteration are returned from the *Inflate* procedure (line 15). This returned inormation is then used in the *PerformInflations* procedure to store the candidates and update the objective cost (Algorithm 3 lines 4-11). This inflation process continues with the new bounds calculated by the *Inflate* procedure until the *result* is *accept* and inflation of the bound parameters are no longer necessary (line 12-16).

The *PerformDeflations* procedure as shown in Algorithm 5 tries to iteratively deflate the bounds that were inflated. These deflated bounds can be used for deterministic approximation. The goal here is that using these deflated bounds in the deterministic problem (equation 2) will yield control settings that will not only satisfy the original constraints from the stochastic problem with sufficient probability but also do so with a better objective cost. To achieve this, the *PerformDeflations* procedure calls the *Deflate* procedure that performs one iteration of

Algorithm 7: RefineCandidates

Input : $acceptedCandSet_1$, $acceptedCandSet_2$, $\vec{B}$, $\vec{\sigma}$, $\hat{O}.m$, $\mathcal{D}$
ConfigParams: noSimulations, $\overrightarrow{PB}$, CB, maxSimBudget, budgetDelta, budgetThreshold
Output : bestCandidate, $\hat{O}.m$

1 newCandSet$_2$ ← Perform simulations and store candidates from $acceptedCandSet_2$ that return the *result* as *accept* from `PerformStochasticSimulations` (Algorithm 9) and whose O.m is statistically better than $\hat{O}.m$ with at least CB confidence
2 acceptedCandSet ← acceptedCandSet$_1$∪ newCandSet$_2$
3 budget ← 1
4 iterationNo ← 1
5 **repeat**
6 | $\overrightarrow{N}^{iterNo}$ ← `ExtendedOCBA`($\overrightarrow{N}^{iterNo-1}$, iterationNo, $\mathcal{D}$, acceptedCandSet, noSimulations, budgetDelta) `// Algorithm 8`
7 | **for** $c \in acceptedCandSet$ **do**
8 | | (result, $O.m, O.sd, \overrightarrow{SC.m}, \overrightarrow{SC.sd}$, N) ← `PerformStochasticSimulations` ($\overrightarrow{X}_c, \vec{B}, \vec{\sigma}, \mathcal{D}$, N_c^{iterNo}, N_c^{iterNo}, $\overrightarrow{PB}$, CB) `// Algorithm 9`
9 | | **if** result is accept and $O.m < \hat{O}.m$ **then**
10 | | | $\hat{O}$.m ← O.m
11 | | **else if** result is reject **then**
12 | | | Remove c from acceptedCandSet
13 | **end**
14 | budget ← budget + budgetDelta
15 | iterationNo ← iterationNo + 1
16 **until** budget > budgetThreshold
17 bestCandidate ← {c ∈ acceptedCandSet | $O.m_c = \hat{O}.m$}
18 **return** bestCandidate, $\hat{O}.m$

Algorithm 8: ExtendedOCBA

Input : $\overrightarrow{N}^{prev}$, iterationNo, $\mathcal{D}$, acceptedCandSet = {cand$_1$, ..., cand$_k$}; where $\forall_{i\in\{1,...,k\}}$ cand$_i$ = ($\vec{X}_i$, O.m$_i$, O.sd$_i$, $\overrightarrow{SC.m_i}$, $\overrightarrow{SC.sd_i}$, N$_i$)
ConfigParams: noSimulations, budgetDelta
Output : $\overrightarrow{N}$, a vector of number of simulations allocated for each candidate

1 bestCand ← arg min$_{i\in acceptedCandSet}$ O.m$_i$ s.t. $\forall_{i\in\mathcal{D}} P(SC.m_i \geq PB_i) \geq CB$
2 $\Theta_O \leftarrow \left\{ i | i \in acceptedCandSet, i \neq bestCand, \frac{(SC.m_{s,i} - PB_{s,i})}{SC.sd_{s,i}} \leq \frac{(O.m_i - O.m_{bestCand})}{O.sd_i} \right\}$
3 $\Theta_F \leftarrow \left\{ i | i \in acceptedCandSet, i \neq bestCand, \frac{(SC.m_{s,i} - PB_{s,i})}{SC.sd_{s,i}} > \frac{(O.m_i - O.m_{bestCand})}{O.sd_i} \right\}$
4 $\forall_{i\in acceptedCandSet}\ \eta_i \leftarrow \begin{cases} \frac{SC.sd_{s,i}}{(SC.m_{s,i} - PB_{s,i})} & \text{if } i \in \Theta_F \\ \frac{O.sd_i}{(O.m_i - O.m_{bestCand})} & \text{if } i \in \Theta_O \\ \frac{SC.sd_{s,bestCand}}{(SC.m_{s,bestCand} - PB_{s,bestCand})} & \text{if } i = bestCand \\ 0 & \text{otherwise} \end{cases}$
5 $\forall_{i\in acceptedCandSet\setminus\{bestCand\}}\ \alpha_i$ ← proportional to η_i, i.e., $(\alpha_i/\alpha_j) = (\eta_i/\eta_j)^2$ for all i ≠ j ≠ bestCand
6 $\forall_{i\in acceptedCandSet\setminus\{bestCand\}}\ N_i \leftarrow \alpha_i(k\times\ noSimulations\ +\ iterationNo\ \times\ budgetDelta)$
7 $\alpha_O \leftarrow O.sd_{bestCand}\sqrt{\sum_{i\in\Theta_O}(\alpha_i/O.sd_i)^2}$
8 α_F ← proportional to $\eta_{bestCand}$, i.e., $(\alpha_F/\alpha_i) = (\eta_{bestCand}/\eta_i)^2$ for all i ≠ bestCand
9 $\alpha_{bestCand} \leftarrow \max(\alpha_F, \alpha_O)$
10 $\forall_{i\in acceptedCandSet}\ N_i$ ← Adjust the allocation for each candidate accordingly so that $\sum_{i=1}^{k} max(0, (N_i - N_i^{prev})) = budgetDelta$
11 **return** $\overrightarrow{N}$

Algorithm 9: PerformStochasticSimulations

Input : $\vec{X}, \vec{B}, \vec{\sigma}, \mathcal{D}$
ConfigParams: noSimulations, maxSimBudget, $\overrightarrow{PB}$, CB
Output : result, $O.m, O.sd, \overrightarrow{SC.m}, \overrightarrow{SC.sd}$, N

1 N ← 0
2 **repeat**
3 | cost.m, cost.sd, $\overrightarrow{p.m}, \overrightarrow{p.sd}$ ← `MonteCarloSimulation` ($\vec{X}, \vec{B}, \vec{\sigma}, \mathcal{D}$, noSimulations)
4 | $O.m, O.sd, \overrightarrow{SC.m}, \overrightarrow{SC.sd}$, N ← `UpdateCandidateStats` (cost.m, cost.sd, $\overrightarrow{p.m}$, $\overrightarrow{p.sd}$, noSimulations)
5 | $\overrightarrow{conf}$ ← `Confidence` ($\overrightarrow{SC.m} \geq \overrightarrow{PB}$)
| // $\epsilon \ll probabilityBound$. `e.g.,`$\epsilon$ `= 0.15`
6 | $\overrightarrow{refConf}$ ← `Confidence` ($\overrightarrow{SC.m} \leq \overrightarrow{PB} - \epsilon$)
7 | N ← N + noSimulations
8 **until** $\forall_{i\in\mathcal{D}}$ $conf_i \geq CB$ *or* $\exists_{i\in\mathcal{D}}$ $refConf_i \geq CB$ *or* $budget \geq maxSimBudget$
9 **if** $\forall_{i\in\mathcal{D}}$ $conf_i \geq CB$ **then**
10 | result ← accept
11 **else if** $\exists_{i\in\mathcal{D}}$ $refConf_i \geq CB$ **then**
12 | result ← reject
13 **else if** $budget \geq maxSimBudget$ **then**
14 | result ← not-reject
15 **end**
16 **return** result, $O.m, O.sd, \overrightarrow{SC.m}, \overrightarrow{SC.sd}$, N

deflation as shown in Algorithm 6. In the *Deflate* procedure, similar to the *Inflate* procedure, the candidates are generated using the current bounds from $\overrightarrow{CIB}$ and stochastic simulations are run on these candidates to calculate the confidence of satisfaction of all the constraints and a verdict for the candidate in the *result* variable (Algorithm 6 lines 1-2). If the *result* is *accept* the interval between the current bound and the bound at which the last inflation occurred is divided by a constant amount from $\vec{\beta}$ (lines 4-13). Finally, the *result* variable, the new bounds ($\overrightarrow{NIB}$) and the generated candidates from the current iteration are returned from the *Deflate* procedure (line 14). This returned inormation is then used in the *PerformedDeflations* procedure to store the candidates and update the objective cost (Algorithm 5 lines 3-9). This deflation process continues with the new bounds calculated by the *Deflate* procedure until the *result* is *reject* and deflation of bounds is no longer useful to generate better candidates (line 12-13).

3.2 Candidate Refinement Phase

After generating the candidate sets using deterministic approximations, SODA-m refines these candidates by running Monte Carlo simulations on them in this phase. To maximize the likelihood of selecting the best candidate, i.e., candidate with the least cost and sufficient probability of constraint satisfaction, this phase uses the Optimal Computing Budget Allocation method for Constraint Optimization (OCBA-CO) (Lee et al., 2012) to allocate some delta budget among the candidates so that the greatest fraction of the budget is allocated to the most promising candidates in the candidate set and in each iteration of this phase, this allocated simulation budget is run on the candidates. In each subsequent iteration, OCBA-CO can use the updated estimates to find new promising candidates and allocate a greater fraction of the delta budget to them to check if these candidates can be further refined.

The pseudo-code for the *RefineCandidates* procedure is given in Algorithm 7. In this procedure, first, simulations are run on any candidates for which there are no base statistic, i.e, candidates on which simulations was not run in the inflate and deflate phase. Those candidates that return the result of *accept* and whose objective cost is statistically better than the best objective cost are stored for further consideration (Algorithm 7 lines 1-2). Then, this phase uses the *ExtendedOCBA* procedure, which is based on OCBA-CO, to iteratively allocate a fraction of the delta budget among the candidates (line 6). Then, stochastic simulations are run for the allocated number of simulations for each candidate (line 8). If the calculated verdict for the candidate (*result*) is *accept* and if the cost is further minimized, then the best cost is updated (lines 9-10). On the other hand, if the *result* is *reject*, then the candidate is removed from further consideration (lines 11-12). This iteration of budget allocation and simulation refinement continues until some maximum budget number of simulation budget is depleted.

The pseudo-code for the *ExtendedOCBA* based on OCBA-CO procedure is given in Algorithm 8. This procedure allocates a greater fraction of the delta budget to candidates that have the highest likelihood of being the best candidate or near the best candidate. In this case, the best candidate is the one with the best cost with sufficient confidence of satisfaction of all the stochastic constraints (line 1). This procedure creates two sets for the non-best candidates: θ_O where optimality of cost objective is a more dominant feature among the candidates and θ_F where probability of a significant constraint is a more dominant feature among the candidates (lines 2-3). Then, the noise-to-signal ratio of each candidate is obtained (line 4). Finally, the number of allocated simulations is obtained proportional to their noise-to-signal ratio (lines 5-10). More details about the OCBA-CO algorithm can be obtained from (Lee et al., 2012).

References

Amaran, S., Sahinidis, N. V., Sharda, B., & Bury, S. J. (2016). Simulation optimization: a review of algorithms and applications. *Annals of Operations Research*, *240*(1), 351–380.

Chen, C. H., & Lee, L. H. (2011). *Stochastic simulation optimization: An optimal computing budget allocation*. Hackensack, NJ, USA: World Scientific Publishing Company.

Dabney, J. B., & Harman, T. L. (2001). *Mastering simulink 4* (1st ed.). Upper Saddle River, NJ, USA: Prentice Hall.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182-197.

Durillo, J. J., & Nebro, A. J. (2011). jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, *42*(10), 760 - 771. Retrieved from `http://jmetal.sourceforge.net`

Fritzson, P. (2004). *Principles of object-oriented modeling and simulation with modelica 2.1*. Piscataway, NJ, USA: Wiley-IEEE Press.

Klemmt, A., Horn, S., Weigert, G., & Wolter, K.-J. (2009). Simulation-based optimization vs. mathematical programming: A hybrid approach for optimizing scheduling problems. *Robotics and Computer-Integrated Manufacturing*, *25*(6), 917–925.

Krishnamoorthy, M., Brodsky, A., & Menascé, D. (2015). Optimizing stochastic temporal manufacturing processes with inventories: An efficient heuristic algorithm based on deterministic approximations. In *Proceedings of the 14th informs computing society conference* (pp. 30–46).

Krishnamoorthy, M., Brodsky, A., & Menascé, D. A. (2017). *Stochastic optimization for steady state production processes based on deterministic approximations* (Tech. Rep. No. GMU-

CS-TR-2017-3). 4400 University Drive MSN 4A5, Fairfax, VA 22030-4444 USA: Department of Computer Science, George Mason University. Available at http://cs.gmu.edu.

Lee, L. H., Pujowidianto, N. A., Li, L. W., Chen, C. H., & Yap, C. M. (2012, Nov). Approximate simulation budget allocation for selecting the best design in the presence of stochastic constraints. *IEEE Transactions on Automatic Control*, *57*(11), 2940-2945.

Nebro, A., Durillo, J., García-Nieto, J., Coello Coello, C., Luna, F., & Alba, E. (2009). Smpso: A new pso-based metaheuristic for multi-objective optimization. In *2009 ieee symposium on computational intelligence in multicriteria decision-making (mcdm 2009)* (p. 66-73). IEEE Press.

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

OpenModelica. (2009). Efficient traceable model-based dynamic optimization - edop, `https://openmodelica.org/research/omoptim/edop` [Computer software manual].

Paraskevopoulos, D., Karakitsos, E., & Rustem, B. (1991). Robust Capacity Planning under Uncertainty. *Management Science*, *37*(7), 787–800. Retrieved 2017-05-17, from `http://www.jstor.org/stable/2632535`

Provan, G., & Venturini, A. (2012, Sep). Stochastic simulation and inference using modelica. In *Proceedings of the 9th international modelica conference* (p. 829-837).

Thieriota, H., Nemera, M., Torabzadeh-Tarib, M., Fritzson, P., Singh, R., & Kocherry, J. J. (2011). Towards design optimization with openmodelica emphasizing parameter optimization with genetic algorithms. In *Modellica conference*. Modelica Association. Retrieved from `https://openmodelica.org/research/omoptim`

Thompson, S. D., & Davis, W. J. (1990, September). An integrated approach for modeling uncertainty in aggregate production planning. *IEEE Transactions on Systems, Man, and Cybernetics*, *20*(5), 1000–1012. doi: 10.1109/21.59965

Xu, J., Zhang, S., Huang, E., Chen, C. H., Lee, L. H., & Celik, N. (2014, Aug). An ordinal transformation framework for multi-fidelity simulation optimization. In *2014 ieee international conference on automation science and engineering (case)* (p. 385-390).

Zitzler, E., & Künzli, S. (2004). Indicator-based selection in multiobjective search. In *Proceedings of the 8th international conference on parallel problem solving from nature, 2004* (pp. 832–842). Springer.

Zitzler, E., Laumanns, M., & Thiele, L. (2001). *SPEA2: Improving the strength pareto evolutionary algorithm* (Tech. Rep. No. 103). Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.