Department of Computer Science
George Mason University Technical Reports

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/ 703-993-1530

Stochastic Optimization for Steady State Production Processes based on Deterministic Approximations

Mohan Krishnamoorthy mkrishn4@gmu.edu
Alexander Brodsky brodsky@gmu.edu
Daniel A. Menascé menasce@gmu.edu

Technical Report GMU-CS-TR-2017-3

Abstract

We consider steady-state production processes that produce a product and have feasibility constraints and metrics of cost and throughput that are stochastic functions of process controls. We propose an efficient stochastic optimization algorithm for the problem of finding process controls that minimize the expectation of cost while satisfying deterministic feasibility constraints and stochastic steady state demand for the output product 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. We demonstrate the proposed algorithm on a use case of a real-world heat-sink production process that involves contract suppliers and manufacturers as well as unit manufacturing processes of shearing, milling, drilling, and machining, and conduct an experimental study that shows that the proposed algorithm significantly outperforms four popular simulation-based stochastic optimization algorithms.

1 Introduction

This paper considers steady-state production processes that produce a product and have feasibility constraints and metrics of cost and throughput that are stochastic functions of process controls. We are 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 deterministic feasibility constraints and stochastic steady state demand for the output product with a given high probability. These problems are prevalent in manufacturing processes, such as machining, assembly lines, and supply chain management. 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 [1] and [2] for a review of such techniques). Tools like SIMULINK [3] and Modelica [4, 5] allow users to run stochastic simulations on models of complex systems in mechanical, hydraulic, thermal, control, and electrical power. Tools like OMOptim [6], Efficient Traceable Model-Based Dynamic Optimization (EDOp) [7], and jMetal [8] 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., [1] and [9]). For this reason, a number of approaches have been developed to bridge the gap between stochastic simulation and MP. For instance, [10] 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 [11] 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 [12] 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; more details are provided in section 7. However, extraction of the mathematical structure through sampling using a black-box simulation is computationally expensive, especially for real-world production processes composed of complex process networks.

Instead of extracting the mathematical structure using black-box simulation, in [13], 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 [13] is limited to processes described using piece-wise linear arithmetic. Whereas, many production processes, particularly in manufacturing, have models based on physics-based equations with non-linear arithmetic. This paper tries to close the gap for stochastic optimization problems for steady-state production processes described using non-linear arithmetic.

More specifically, the contributions of this paper are three-fold: First, we propose a heuristic algorithm called Stochastic Optimization Algorithm based on Deterministic Approximations (SODA) to solve the problem of finding production process controls that minimize the expectation of cost while satisfying the deterministic process feasibility constraints and stochastic steady state demand for the output product 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 [14], [15]). Second, we demonstrate the proposed algorithm on a use case of a real-world heat-sink production process that involves contract suppliers and manufacturers as well as unit manufacturing processes of shearing, milling, drilling, and machining with models from the literature that use non-linear physics-based equations. Third, we conduct an initial experimental study using the heat-sink production process to compare the proposed algorithm with four popular simulation-based stochastic optimization algorithms viz., Nondominated Sorting Genetic Algorithm 2 (NGSA2) [16], Indicator Based Evolutionary Algorithm (IBEA) [17], Strength Pareto Evolutionary Algorithm 2 (SPEA2) [18], and Speed-constrained Multi-objective Particle swarm optimization (SMPSO) [19]. The experimental study demonstrates that SODA significantly outperforms the other algorithms in terms of optimality of results and computation time. In particular, running over a 12-process problem using a 8-core server with 16GB RAM, in 40 minutes, SODA achieves a production cost lower than that of competing algorithms by 61%; in 16 hours SODA achieves 29% better cost; and, in 3 days it achieves 7% better cost.

The rest of this paper is organized as follows. Section 2 formally describes the stochastic optimization problem over steady-state production processes. SODA, including deterministic approximations, is presented in section 3. Section 4 describes the model of a real world manufacturing use case of a heat-sink service network, which is used in the experimental study presented in section 5. Key observations and extensions are discussed in section 6. Section 7 further discusses related work. Finally, section 8 concludes with some future research directions.

2 Optimization of Stochastic Production Processes with Closed-form Non-linear Arithmetic

The stochastic optimization problem for steady state production processes considered in this paper 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 deterministic process constraints and steady state demand for the output product with a given probability. More formally, the stochastic optimization problem is of the form:

$$\begin{aligned} \underset{\vec{X}\in\vec{D}}{\text{minimize}} \quad & E(cost(\vec{X})) \\ \text{subject to} \quad & C(\vec{X}) \wedge \\ & P(thru(\vec{X}) \geq \theta) \geq \alpha \end{aligned} \tag{1}$$

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 $\vec{X}$

$thru(\vec{X})$ is a random variable defined in terms of $\vec{X}$

$C(\vec{X})$ is a deterministic constraint in $\vec{X}$ i.e., a function $C : \vec{D} \rightarrow \{true, false\}$

$\theta \in \mathbb{R}$ is a throughput threshold

$\alpha \in [0, 1]$ is a probability threshold, and

$P(thru(\vec{X}) \geq \theta)$ is the probability that $thru(\vec{X})$ is greater than or equal to θ

Note in this problem that upon increasing θ to some θ', the size of the space of alternatives that satisfy the stochastic demand 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'. We assume that the random variables, *cost*($\vec{X}$) and *thru*($\vec{X}$) as well as the deterministic constraint $C(\vec{X})$ are expressed by an SCFA simulation $\mathbb{S}$ that computes the stochastic arithmetic variable *(cost, thru)* $\in \vec{\mathbb{R}}^2$ as well as the deterministic boolean variable $C \in \{true, false\}$.

While we exemplified the SCFA simulation using a very simple example in this section, many complex real-world processes can be formulated as SCFA simulation such as the use case described in section 4.

3 Stochastic Optimization Algorithm based on Deterministic Approximations

This section presents the Stochastic Optimization Algorithm based on Deterministic Approximations (SODA). The problem of optimizing stochastic production processes can be solved using simulation-based optimization approaches by initializing the control settings and performing simulations to check whether the throughput satisfies the demand 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 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 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 performs deterministic approximations of the original stochastic problem. SODA 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. The deterministic approximations $cost_0(\vec{X})$ and $thru_0(\vec{X})$ of $cost(\vec{X})$ and $thru(\vec{X})$, 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 can be described as follows:

$$\begin{aligned} \underset{\vec{X}\in\vec{D}}{\text{minimize}} \quad & cost_0(\vec{X}) \\ \text{subject to} \quad & C(\vec{X}) \wedge \\ & thru_0(\vec{X}) \geq \theta' \end{aligned} \tag{2}$$

where $\theta' \geq \theta$ is a conservative approximation of θ.

This deterministic approximation is performed iteratively such that the control settings found in the current iteration are more likely to generate throughputs that satisfy demand 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.

The *inflate* phase is intuitively trying to increase the throughput to satisfy the demand with the desired probability. When the current candidate control settings do not generate the throughput that satisfies the original user-defined demand with a desired probability, the demand parameter itself is exponentially inflated. This inflation yields higher controls for the machines and

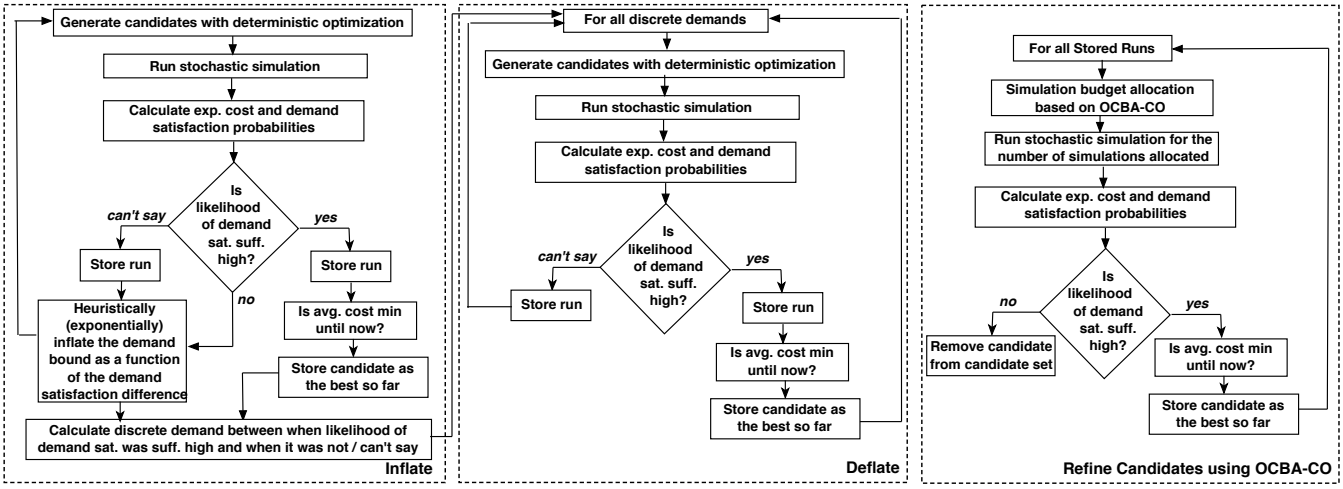

Figure 1: Overview of SODA

Algorithm 1: Stochastic Optimization of Steady-state NL Process

Input : actualDemand, $\vec{\sigma}, \overrightarrow{min}, \overrightarrow{max}, \lambda$
ConfigParams: storeSize, totalIterations, $\delta_{cost}, \delta_{restart}$, noSimulations, probabilityBound, finalConfidence, refuteConfidence, maxAccRejBudget, budgetDelta, budgetThreshold
Output : cand_{best}, bestCost

1 bestCost, candSet_1, $\text{canSet}_2 \leftarrow$ `InflateDeflate` (actualDemand, $\vec{\sigma}$, $\overrightarrow{min}, \overrightarrow{max}, \lambda$, storeSize, totalIterations, $\delta_{cost}, \delta_{restart}$, noSimulations, probabilityBound, finalConfidence, refuteConfidence, maxAccRejBudget) `// Algorithm 2`
2 cand_{best}, bestCost $\leftarrow$ `RefineCandidates` (candSet_1, canSet_2, actualDemand, $\vec{\sigma}$, bestCost, noSimulations, probabilityBound, finalConfidence, refuteConfidence, budgetDelta, budgetThreshold, maxAccRejBudget) `// Algorithm 5`
3 **return** cand_{best}, bestCost

thus increases the overall throughput. However, this may result in the throughput overshooting the demand in the stochastic setting, which degrades the objective cost.

To overcome this, in the *deflate* phase, SODA scales the demand back by splitting it into a number of points, separated by a small epsilon, in the interval between the inflated demand where throughput overshot the original demand and the previous demand at which the last inflation occurred. Deterministic approximations are run for this lower demand to check whether the throughput still satisfies the demand with desired probability while yielding a better objective cost. In this way, the *inflate* and *deflate* phases find a more optimum demand threshold for which it can get the right balance between optimum cost and demand 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 uses the model knowledge in *inflate*, *deflate*, and *refineCandidates* phases to provide optimal control settings of the non-linear processes that a process operator can use on the manufacturing floor in a stochastic environment. The phases of SODA 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. The deterministic approximation by first inflating the demand to find the accepted candidate is performed in lines 8-25. In this part, first a number of candidate control settings with different objective costs are obtained by running a deterministic optimization with a constraint on the current demand (θ') in the *GenerateCandidates* procedure (line 9). Then, the confidence of demand satisfaction is obtained into the *result* variable in the *AcceptRejectCandidate* procedure by running stochastic simulations on the candidate that yielded the least cost among the generated candidates (line 10). If the confidence is higher than the threshold (α), i.e., *result* is *accept* or if the confidence is high enough so that the candidate is not refuted, i.e., *result* is *not-reject*, then the candidate control settings that yielded the least cost is put into $candSet_1$ and the other candidates

Algorithm 2: InflateDeflate

Input : actualDemand, $\vec{\sigma}$, $\overrightarrow{min}, \overrightarrow{max}$

ConfigParams: λ, storeSize, totalIterations, $\delta_{cost}, \delta_{restart}$, noSimulations, probabilityBound, finalConfidence, refuteConfidence, maxAccRejBudget

Output : bestCost, candSet$_1$ = {cand$_1$, ..., cand$_p$}, candSet$_2$ = $\{\overrightarrow{X}_1, \ldots, \overrightarrow{X}_q\}$, p $\leq$ storeSize; where $\forall_{i \in \{1,\ldots,p\}}$ cand$_i$ = ($\overrightarrow{X}_i$, candExpCost$_i$, candCostStddev$_i$ candProb$_i$, candProbStddev$_i$, candNoSims$_i$, conf$_i$)

```
noCandidates ← 1
noIterations ← 1
bestCost ← ∞
repeat
    currentDemand ← actualDemand + Z ~ N(0, ψ) // ψ < 0.5
    lastInflateDemand ← currentDemand
    result ← unknown
    repeat
        // Inflate from currentDemand
        ((X_1→, cost_1), ..., (X_k→, cost_k)) ← GenerateCandidates (currentDemand, min→, max→, δ_cost, δ_restart) // Algorithm 3
        // Algorithm 4
        result, (candExpCost, candCostStddev, candProb, candProbStddev, candNoSims, conf) ← AcceptRejectCandidate
            (X_1→, actualDemand, σ→, noSimulations, probabilityBound, finalConfidence, refuteConfidence, maxAccRejBudget)
        if result is accept or not-reject then
            candSet_1 ← candSet_1 ∪ {(X_1→, candExpCost, candCostStddev, candProb, candProbStddev, candNoSims, conf)}
            candSet_2 ← candSet_2 ∪ {X_2→, ..., X_k→}
            noCandidates ← noCandidates + k
            if result is accept and expCost ≤ bestCost then
                bestCost ← expCost
            end
        end
        if result is reject or not-reject then
            probDiff ← probabilityBound - candProb
            increment ← currDemand * expDensity (λ, probDiff)
            currDemand ← currDemand + increment
            lastInflateDemand ← currentDemand
        end
    until result is accept
    // Result is accept.  So deflate from currentDemand to lastInflateDemand
    discreteDemandSeq ← Calculate discrete demands in ⌊lastInflateDemand, currentDemand⌋ separated by a small ε
    foreach demand_i from discreteDemandSeq do
        // Algorithm 3
        ((X_1→, cost_1), ..., (X_k→, cost_k)) ← GenerateCandidates (demand_i, min→, max→, δ_cost, δ_restart) // Algorithm 3
        // Algorithm 4
        result, (candExpCost, candCostStddev, candProb, candProbStddev, candNoSims, conf) ← AcceptRejectCandidate
            (X_1→, actualDemand, σ→, noSimulations, probabilityBound, finalConfidence, refuteConfidence, maxAccRejBudget)
        if result is accept or not-reject then
            candSet_1 ← candSet_1 ∪ {(X_1→, candExpCost, candCostStddev, candProb, candProbStddev, candNoSims, conf)}
            candSet_2 ← candSet_2 ∪ {X_2→, ..., X_k→}
            noCandidates ← noCandidates + k
            if result is accept and expCost ≤ bestCost then
                bestCost ← expCost
            end
        end
    end
    noIterations ← noIterations + 1
until noIterations > totalIterations or noCandidates > storeSize
return bestCost, candSet_1, candSet_2
```

generated by *GenerateCandidates* are put into $candSet_2$. Also, the optimum cost (*bestCost*) is updated if the accepted candidate yielded a lower cost (lines 12-17). On the other hand, if the candidate is refuted, i.e., *result* is *reject* or even if the confidence is high enough so that the candidate is not refuted, i.e., *result* is *not-reject*, then the demand setting θ' is heuristically inflated (lines 19-24). The inflation amount is decided by an exponential distribution with parameter λ for the difference in the probability of demand satisfaction. This ensures that if the probability of demand satisfaction was very low, this heuristic will increase the demand exponentially so that the deterministic optimization solver can adjust the control settings of the processes accordingly in the next iteration. This would then increase the probability of demand satisfaction in the next iteration thereby resulting in a greater likelihood of satisfying the demand in the stochastic setting.

The deterministic approximation performed while deflating the demand is shown in lines 26-38. After the first accepted candidate is obtained during inflation, the interval between the last non-accepted candidate's demand (*lastInflateDemand*) and the accepted candidate's demand (*currentDemand*) is discretized, separated by a small ϵ (line 26). Then, each discrete demand setting is considered in descending order for the deterministic approximation, similar to that performed during inflation (lines 27-38). The idea here is that since the demand setting at *lastInflateDemand* was not accepted and that at *currentDemand* was accepted during inflation, the sweet spot of the deflated demand for finding the candidate with minimum expected cost with sufficient probability of demand satisfaction lies between these two demands. A number of deterministic approximations may need to be performed here especially if ϵ is very small. But the heuristics used during inflation allows for significant reduction of the search space to a set of close to optimal solutions, i.e., those obtained by performing a deterministic approximation for the demand interval between *lastInflateDemand* and *currentDemand* that give a better solution with fast convergence.

To generate candidate control settings on the current demand, *InflateDeflate* uses the *GenerateCandidates* procedure. The pseudo-code for this procedure is shown in Algorithm 3. Since the processes considered here have non-linear objectives that may not necessarily be convex, the iterative procedure chooses a random starting point for all the decision variables (line 6). Then, the problem is abstracted into a deterministic model such as the one shown in equation 2 by ignoring its stochastic components and then this model is optimized using a deterministic non-linear optimization solver (line 7). Since the optimization is being run using different starting points in each iteration, it is possible that the deterministic non-linear optimization solvers give different controls with the same or different objective cost in some iterations. If the obtained objective cost is within the accepted limit, the different control setting obtained is collected into the candidate set (lines 8-15). This procedure continues until there is little or no change in the objective cost for a specified number of iterations (line 17).

To check the confidence of demand satisfaction in the stochastic setting for the candidates that were generated in *GenerateCandidates*, *InflateDeflate* uses the *AcceptRejectCandidate* procedure. The pseudo-code for this procedure is shown in Algorithm 4. In this procedure, first a number of simulations are run on the candidate to obtain some statistics for this candidate in the stochastic setting (line 5). Using these statistics, the confidence of demand satisfaction with sufficient probability is obtained (lines 6-7). Finally, the *result* variable is created for this candidate and this value assumes *accept* for confidence of demand satisfaction being above the *finalConfidence* threshold, *reject* for the candidate being refuted and *not-reject* for any candidate that is neither accepted nor rejected (lines 11-17). Thus the *result* variable is used to evaluate the quality of the generated candidate in the stochastic setting.

3.2 Candidate Refinement Phase

After generating the candidate sets using deterministic approximations, SODA 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 demand satisfaction, this phase uses the Optimal Computing Budget Allocation method for Constraint Optimization (OCBA-CO) [15] to allocate the budget among the candidates so that the greatest fraction of the budget is allocated to the most promising candidates in the candidate set. This phase is performed iteratively for some delta budget that is allocated among the candidates using OCBA-CO and in each iteration, the additional simulations on the candidates can yield a better objective cost or a higher confidence of demand satisfaction. 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 5. Since the confidence of demand satisfaction is only obtained for the candidates from $candSet_1$ in the *InflateDeflate* phase, the *RefineCandidates* procedure first finds the confidence of demand satisfaction into the *result* variable using the *AcceptRejectCandidate* procedure for the candidates in $candSet_2$ (line 4). Then, similar to the code in *InflateDeflate*, the candidate is either removed from the set or it is stored along with its base statistics if the *result* is *reject* or not, respectively (lines 5-12). Then, all the candidates from $candSet_1$ and $candSet_2$ are combined into a single *candSet* (line 14). Then, this phase uses the *ExtendedOCBA* procedure, which is based on OCBA-CO, to iteratively allocate a fraction of

Algorithm 3: GenerateCandidates

Input : currentDemand, $\overrightarrow{min}, \overrightarrow{max}$
ConfigParams: δ_{cost}, $\delta_{restart}$
Output : $((\overrightarrow{X_1}, cost_1), \ldots, (\overrightarrow{X_k}, cost_k))$ ordered by increasing order of cost

noOfRestarts ← 1
bestCost ← ∞
outputSeq ← ()
$\epsilon \leftarrow 0.0001$
repeat
  $\overrightarrow{S}$ ← random vector uniformly chosen between $\overrightarrow{min}$ and $\overrightarrow{max}$ inclusive
  $(\vec{X}, cost) \leftarrow$ `DetOpt` ($\vec{S}$, currentDemand)
  if cost <= bestCost or (cost - δ_{cost}) < bestCost **then**
    if cost < bestCost **then**
      bestCost ← cost
    end
    if outputSeq does not already contain $(\vec{X}, cost)$ **then**
      outputSeq ← outputSeq + $(\vec{X}, cost)$
    end
  end
  noOfRestarts ← noOfRestarts +1
until bestCost$_{noOfRestarts}$ - bestCost$_{noOfRestarts-\delta_{restart}} < \epsilon$
return outputSeq

Algorithm 4: AcceptRejectCandidate

Input : $\vec{X}$, actualDemand, $\vec{\sigma}$
ConfigParams: noSimulations, probabilityBound, finalConfidence, refuteConfidence, maxAccRejBudget
Output : result ∈ {accept, reject, not-reject}, (candExpCost, candCostStddev candProb, candProbStddev, candNoSims, conf); where candExpCost is the candidate's expected cost, candCostStddev is the standard deviation of the cost, candProb is its probability of demand satisfaction, candProbStddev is the standard deviation of demand satisfaction, candNoSims is the total number of simulations run on the candidate, and conf is the confidence on candProb.

conf ← 0
refConf ← 0
budget ← 1
repeat
  expCost, costStddev, probabilityFound, probabilityStddev ← `MonteCarloSimulation` ($\vec{X}$, actualDemand, $\vec{\sigma}$, noSimulations)
  conf ← `Confidence` (candProb >= probabilityBound)
  refConf ← `Confidence` (p <= (probabilityBound - ϵ)) `// ` $\epsilon \ll probabilityBound$. `e.g.,`$\epsilon$ `= 0.15`
  candExpCost, candCostStddev, candProb, candProbStddev, candNoSims ← `UpdateStats` (expCost, costStddev, probabilityFound, probabilityStddev, noSimulations)
  budget ← budget + noSimulations
until conf > finalConfidence or refConf > refuteConfidence or budget > maxAccRejBudget
if conf > finalConfidence **then**
  result ← accept
else if refConf > refuteConfidence **then**
  result ← reject
else
  result ← not-reject
end
return result, (candExpCost, candCostStddev, candProb, candProbStddev, candNoSims, conf)

Algorithm 5: RefineCandidates

Input : $\text{candSet}_1 = \{\text{cand}_1, \ldots, \text{cand}_p\}$, $\text{candSet}_2 = \{\text{X}_1, \ldots, \text{X}_q\}$, actualDemand, $\vec{\sigma}$, bestCostUntilNow; where $\forall_{i \in \{1,\ldots,p\}}\ \text{cand}_i = (\overrightarrow{X}_i, \text{candExpCost}_i, \text{candCostStddev}_i, \text{candProb}_i, \text{candProbStddev}_i, \text{candNoSims}_i, \text{conf}_i)$, and k = p + q

ConfigParams: noSimulations, probabilityBound, finalConfidence, refuteConfidence, budgetDelta, budgetThreshold, maxAccRejBudget

Output : cand_{best}, bestCost

$\text{newCandSet}_2 \leftarrow \{\}$
bestCost ← bestCostUntilNow
foreach c ∈ candSet_2 **do**
  `// Algorithm 4`
  result, (candExpCost, candCostStddev, candProb, candProbStddev, candNoSims, conf) ← `AcceptRejectCandidate` ($\overrightarrow{X}_c$, actualDemand, $\vec{\sigma}$, noSimulations, probabilityBound, finalConfidence, refuteConfidence, maxAccRejBudget)
  if result is reject **then**
    Remove c from candSet_2
  else
    $\text{newCandSet}_2 \leftarrow \text{newCandSet}_2 \cup \{\overrightarrow{X}_c, \text{candExpCost}, \text{candCostStddev}, \text{candProb}, \text{candProbStddev}, \text{noSimulations}, \text{conf}\}$
    if result is accept and candExpCost < bestCost **then**
      bestCost ← candExpCost
    end
  end
end
$\text{candSet} \leftarrow \text{candSet}_1 \cup \text{newCandSet}_2$
budget ← 1
iterationNo ← 1
repeat
  $\overrightarrow{N}^{iterationNo} \leftarrow$ `ExtendedOCBA`($\overrightarrow{N}^{iterationNo-1}$, iterationNo, candSet, noSimulations, budgetDelta) `// Algorithm 6`
  foreach c ∈ candSet **do**
    `// Algorithm 7`
    expCost, costStddev, probabilityFound, probabilityStddev ← `MonteCarloSimulation` ($\vec{X}_c$, actualDemand, $\vec{\sigma}$, $N_c^{iterationNo}$)
    candExpCost_c, candCostStddev_c, candProb_c, candProbStddev_c, candNoSims_c ← `UpdateStats` (expCost, costStddev, probabilityFound, probabilityStddev, $N_c^{iterationNo}$)
    conf_c ← `Confidence` ($candProb_c$ >= probabilityBound)
    refConf ← `Confidence` ($candProb_c$ <= (probabilityBound - ϵ)) `//` $\epsilon \ll probabilityBound$. `e.g.,`$\epsilon$ `= 0.15`
    if conf_c >= finalConfidence and $candExpCost_c$ < bestCost **then**
      bestCost ← $candExpCost_c$
    else if refConf > refuteConfidence **then**
      Remove c from candSet
    end
  end
  budget ← budget + budgetDelta
  iterationNo ← iterationNo + 1
until budget > budgetThreshold
$\text{cand}_{best} \leftarrow \{\text{c} \in \text{canSet} \mid \text{candExpCost}_c = \text{bestCost}\}$
return cand_{best}, bestCost

the delta budget among the candidates in the *candSet* (line 18). Then, Monte Carlo simulations are run for the allocated number of simulations for each candidate (line 20). Using these statistics, the updated confidence of demand satisfaction with sufficient probability is obtained (lines 22-23). If the updated confidence of demand satisfaction is above the *finalConfidence* and the cost is further minimized, then the best cost and best candidate are updated (lines 24-25). On the other hand, if the candidate can be refuted, then it is removed from the *candSet* (lines 26-27). This iteration of budget allocation and simulation refinement is repeated until some maximum budget number of simulations is depleted.

The pseudo-code for the *ExtendedOCBA* based on OCBA-CO procedure is given in Algorithm 6. 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. This procedure does this while considering the demand constraint, i.e., the best candidate is the one with the best cost with sufficient confidence of demand satisfaction (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 demand satisfaction 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 in proportion to the noise-to-signal ratio (lines 5-10). More details about the OCBA-CO algorithm can be obtained from [15].

4 Stochastic Closed-form Arithmetic Simulation of a Real World Production Process

While the formulation of the problem is abstract, we wanted to check whether it is possible to apply SODA on a use case of a real-world production process and to conduct an experimental study to evaluate the performance of SODA. These real world production processes can be described using the SCFA simulation that was introduced in section 2. In this section, the SCFA simulation of a real-world Heat-Sink Production Process (HSPP) is provided whereas the experimental study to evaluate SODA using HSPP is described in section 5.

The graphical representation for the HSPP is shown in Fig. 2. The production process can be described as a composition of three sub-processes of *supply*, *production*, and *demand*. The *supply* sub-process further contains the suppliers of the raw materials for the heat-sink part, which includes suppliers of *aluminum*, *heat-sink case* and *accessories* such as screws, and bolts. The production sub-process contains processes that will transform the input raw materials into the output heat-sink part. To do this, the raw material of *aluminum* is first cut using a *cutting/shearing* process. Then, the cut aluminum is sent to a *CNC machining* process that uses the *milling* and *drilling* sub-processes to mill and drill on the cut aluminum part to produce the raw heat-sink part. Then, this raw part goes through the *washing and finishing* process, after which it is inspected for defects and accuracy in the *quality inspection* process. Finally, this inspected part is assembled together with the *heat-sink case* using the *accessories* in the *assembly* process. The last sub-process is a virtual process for the *demand* of heat-sink part production that dictates the minimum number of heat-sinks to be produced so as to satisfy either the customer demand or foretasted sales.

If one looks at the HSPP top-down, it is a composed of the processes of *supply*, *production*, and *demand*. The *supply* process is itself composed of atomic sub-processes such as the suppliers for *aluminum*, *heat-sink case*, and *accessories*. Similarly, the *production* process is itself composed of atomic sub-processes such as *cutting/shearing*, *machining*, and *assembly*. As mentioned in section 2, the HSPP and all of its sub-process are described using a SCFA simulation that transforms the input control variables and other fixed parameters of the process to stochastic metric variables of the cost and throughput per item as well as the deterministic constraint variables for the corresponding process. For instance, the SCFA simulation for the machining process has inputs of control variables such as cutting speed and feed rate, and fixed parameters such as number of holes and depth of cut. The output from this simulation may then contain values for the metrics of interest such as production cost, cycle time, and energy use, as well as constraints such as satisfaction of the production demand, and other feasibility constraints.

Consider P_H to be a set of the sub-processes of the HSPP i.e., P_H contains the processes of *supply*, *production*, and *demand*. Similarly, consider P_S and P_R to be the sub-processes of *supply* and *production*, respectively. By running the SCFA simulation of each process within P_S and P_R, it is possible to compute the cost of the corresponding process and then compute the total cost of the *supply* and *production* processes as the sum of the cost of its respective sub-processes as described in equation 3.

$$\begin{aligned} cost_S &= \sum_{p \in P_S} cost_p \\ cost_R &= \sum_{p \in P_R} cost_p \end{aligned} \tag{3}$$

Also, the SCFA simulation of each process within P_S computes the throughput of each item supplied, i.e, it computes the throughput of the supplied aluminum ($thru_{al}$), heat-sink case ($thru_{hsc}$), and accessories ($thru_a$). Similarly, the SCFA simulation of each process within P_R computes the throughput of the respective item produced and finally the throughput of the completed heat-sink part ($thru_h$) is determined from the SCFA simulation of the

Algorithm 6: ExtendedOCBA

Input : $\overrightarrow{N}^{prev}$, iterationNo, candSet = $\{\text{cand}_1, \ldots, \text{cand}_k\}$; where $\forall_{i\in\{1,\ldots,k\}}$ $\text{cand}_i = (\overrightarrow{X}_i, \text{candExpCost}_i, \text{candCostStddev}_i, \text{candProb}_i, \text{candProbStddev}_i, \text{candNoSims}_i, \text{conf}_i)$

ConfigParams: noSimulations, budgetDelta

Output : $\overrightarrow{N}$, a vector of number of simulations allocated using OCBA-CO for each candidate

bestCand $\leftarrow \arg\min_{i\in candSet}$ candExpCost$_i$ s.t. conf$_i \geq$ finalConfidence

$\Theta_O \leftarrow \left\{ i | i \in candSet, i \neq bestCand, \frac{(candProb_i - probabilityBound)}{candProbStddev_i} \leq \frac{(candExpCost_i - candExpCost_{bestCand})}{candCostStddev_i} \right\}$

$\Theta_F \leftarrow \left\{ i | i \in candSet, i \neq bestCand, \frac{(candProb_i - probabilityBound)}{candProbStddev_i} > \frac{(candExpCost_i - candExpCost_{bestCand})}{candCostStddev_i} \right\}$

$$\forall_{i\in candSet}\ \eta_i \leftarrow \begin{cases} \frac{candProbStddev_i}{(candProb_i - probabilityBound)} & \text{if } i \in \Theta_F \\ \frac{candCostStddev_i}{(candExpCost_i - candExpCost_{bestCand})} & \text{if } i \in \Theta_O \\ \frac{candProbStddev_{bestCand}}{(candProb_{bestCand} - probabilityBound)} & \text{if } i = bestCand \\ 0 & \text{otherwise} \end{cases}$$

$\forall_{i\in candSet\setminus\{bestCand\}}\ \alpha_i \leftarrow$ proportional to η_i, i.e., $(\alpha_i/\alpha_j) = (\eta_i/\eta_j)^2$ for all i $\neq$ j $\neq$ bestCand

$\forall_{i\in candSet\setminus\{bestCand\}}\ N_i \leftarrow \alpha_i(k \times noSimulations + iterationNo \times budgetDelta)$

$\alpha_O \leftarrow candCostStddev_{bestCand}\sqrt{\sum_{i\in\Theta_O}(\alpha_i/candCostStddev_i)^2}$

$\alpha_F \leftarrow$ proportional to $\eta_{bestCand}$, i.e., $(\alpha_F/\alpha_i) = (\eta_{bestCand}/\eta_i)^2$ for all i $\neq$ bestCand

$\alpha_{bestCand} \leftarrow \max(\alpha_F, \alpha_O)$

$\forall_{i\in candSet}\ N_i \leftarrow$ Adjust the allocation for each candidate accordingly so that $\sum_{i=1}^{k} max(0, (N_i - N_i^{prev})) = budgetDelta$

return $\overrightarrow{N}$

assembly process.

The SCFA simulation of each sub-process described above also computes boolean constraint variables of $C_p, p \in \{P_S, P_R\}$, such as those that satisfy the zero-sum constraints and feasibility constraints. The zero sum constraint states that for every item, the total input of this item coming as outputs from the sub-processes must be greater or equal to the total output that is distributed among inputs of sub-processes. On the other hand, the feasibility constraints include bounds on the operating capacity of the sub-processes and bounds on their control variables.

The *demand* process is a virtual process that specifies the user-defined demand for the completed heat-sink part via its throughput ($thru_{hd}$) and whose cost is 0, i.e., $cost_D = 0$. The SCFA simulation of HSPP computes all the metrics and constraints of its sub-processes recursively and then computes the total cost ($cost_H$) and throughput ($thru_H$) of the entire HSPP composite process as shown in equation 4.

$$\begin{aligned} cost_H &= \sum_{p\in P_H} cost_p = cost_S + cost_R + cost_D \\ thru_H &= thru_R = thru_h \end{aligned} \tag{4}$$

Finally, the constraints of HSPP (see equation 5) are computed as the satisfaction of all of its sub-process constraints recursively. This includes the satisfaction of the demand constraint, i.e, the number of parts produced by HSPP ($thru_h$) is greater or equal to a user-defined demand parameter ($thru_{hd}$).

$$\begin{aligned} C_H : \wedge_{p\in P_H} C_p : C_S \wedge C_R \wedge C_D \\ \text{where } C_D : thru_h \geq thru_{hd} \end{aligned} \tag{5}$$

In this section, we only provide an overview of the SCFA simulation for HSPP. In fact, the input and output of the SCFA simulation of HSPP is actually described as JavaScript Object Notation (JSON) structures and the SCFA simulation itself is described using JSONiq, which is a query and processing language specifically designed for JSON data models [20]. Also, to perform the deterministic approximations for SODA as described in section 3, we use a system that automatically converts the SCFA simulation of HSPP into an equivalent deterministic optimization problem as shown in equation 2. But discussing the code of the SCFA simulation for HSPP and the system that performs its automatic conversion is outside the scope of this paper. But more details about an initial realization of these can be found in [21]. To evaluate SODA against the state of the art algorithms, experiments were run on the SCFA simulation of HSPP to solve the stochastic optimization problem considered in this paper. Results of this experimental study are described in section 5.

5 Experimental Results

This section provides the experimental results that evaluate SODA by comparing the quality of objective cost

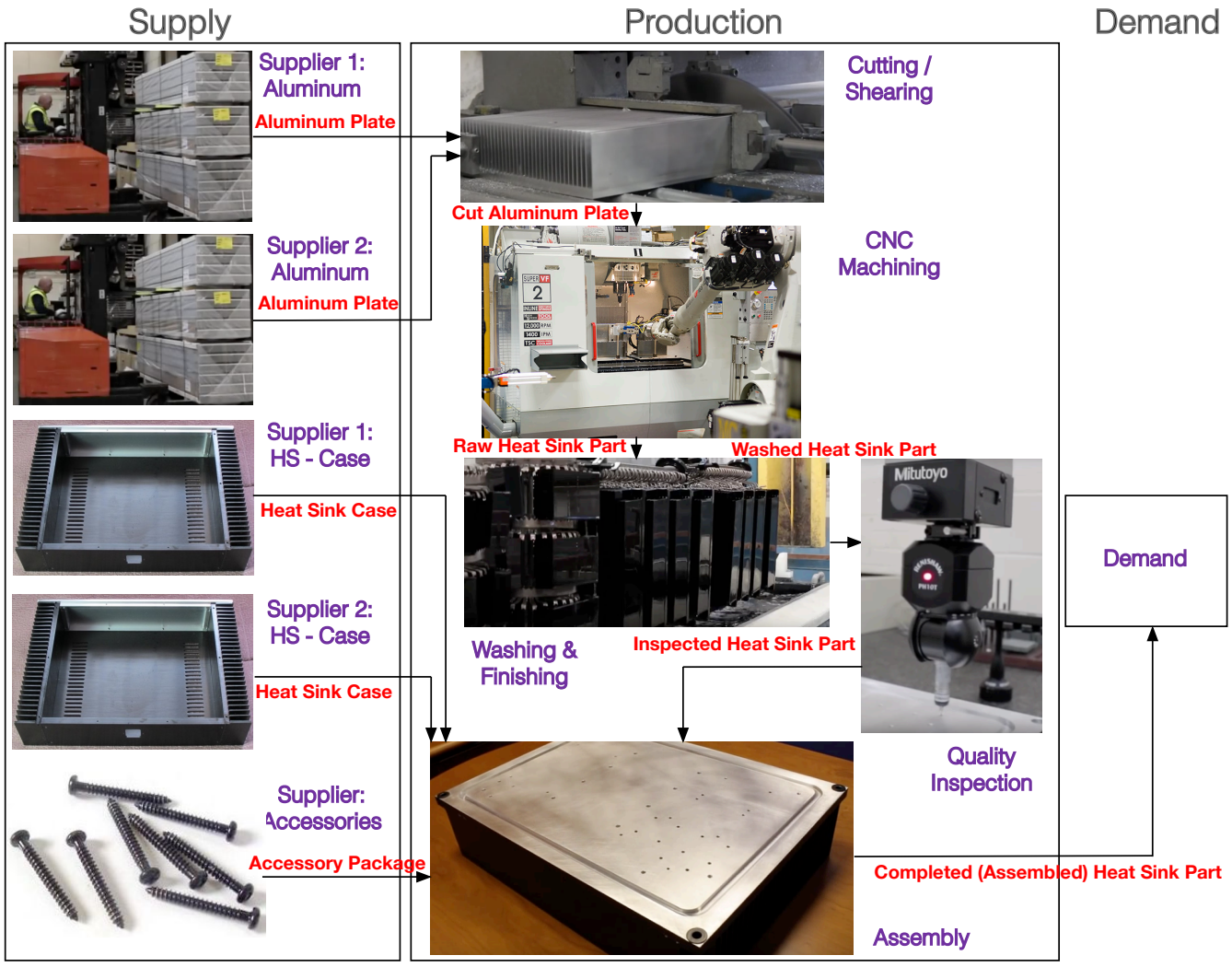

Figure 2: Graphical representation of the Heat-Sink Production Process (HSPP)

and rate of convergence obtained from SODA with other metaheuristic simulation-based optimization algorithms. In order to evaluate SODA, the SCFA simulation of the HSPP described in section 4 is used. The SCFA simulation of HSPP is written in JSONiq whereas SODA itself is written in Java. The deterministic approximations for SODA are performed using a system that automatically converts the SCFA simulation of HSPP into a deterministic optimization problem, which is a deterministic abstraction of the original SCFA simulation.

For the comparison algorithms, we used the jMetal package, which is an object-oriented Java-based framework for multi-objective optimization with metaheuristics [8]. The algorithms chosen for comparison include Nondominated Sorting Genetic Algorithm 2 (NGSA2) [16], Indicator Based Evolutionary Algorithm (IBEA) [17], Strength Pareto Evolutionary Algorithm 2 (SPEA2) [18], and Speed-constrained Multi-objective Particle swarm optimization (SMPSO) [19]. All these algorithms use the SCFA of HSPP to perform the computation of the cost, throughput and feasibility constraints. The computation of the demand satisfaction information from expected throughput is also similar to that of SODA. The cost and demand satisfaction information is then used by the jMetal algorithms to further increase or decrease the control settings using heuristics.

SODA was run with two different settings. In the first setting, SODA was run for 16 hours and the number of candidates collected from the *InflateDeflate* phase was 2,000 (*storeSize = 2,000*). In the second setting, SODA was run for approximately 72 hours (3 days) with *storeSize = 10,000*. Other configuration parameters used include *probabilityBound = 0.95*, *finalConfidence = 99%*, and *refuteConfidence = 70%*. Additionally, all production processes in HSPP are stochastic due to noise added to the controls. This noise has a mean of 0 and standard deviation between 1.4 to 2.4. Finally, the demand (*actualDemand*) from the HSPP was set to be 4. The comparison algorithms were also run with the same (relevant) parameters.

The data collected from the experiments include the estimated average costs achieved at different elapsed time points for SODA in two settings and all the comparison algorithms. Fig. 3 shows the estimated average costs achieved after 16 hours whereas Fig. 4 shows the estimated average costs achieved after about three days. Each experiment was run multiple times and 95% confidence bars are included around the mean at certain elapsed time point in both figures.

It can be observed that both settings of SODA perform better than the comparison algorithms initially. This is because SODA uses deterministic approximation to reduce the search space of potential candidates quickly whereas the competing algorithms start at a random point in the search space and need a number of additional iterations (and time) to find candidates closer to those found by SODA.

As time progresses, SODA in both settings is able to achieve much better expected cost in the *inflateDeflate* phase than the other algorithms. Hence, we claim that SODA is able to converge quicker toward the points close to the (near) optimal solution found at the end of the algorithm. This is because SODA uses heuristics in the *inflate* and *deflate* phases that reduce the search space quickly, which allows SODA to quickly converge toward a more promising candidate.

Also, the solutions found by SODA at the end of the experiment are much better than those found by the competing algorithms. After 16 hours, the expected cost found by SODA was 29% better than the nearest comparison algorithm (SMPSO) and 49% better than the second best comparison algorithm (NSGA2) (see Fig. 3). After three days though, the advantage of SODA over the nearest comparison algorithm (SMPSO) reduces to 7% and that to the second best algorithm (NSGA2) reduces to 17% (see Fig. 4). This is not surprising since SMPSO and NSGA2 use strong meta-heuristics and given enough time such algorithms will eventually converge toward the (near) optimal solution found by SODA. Although, it should be noted from Fig. 4 that SODA reaches close to the optimal solution found at the end much more quickly than its competition. Also, since the x-axis in Fig. 4 is in log scale, it should be noted that out of the total experiment time of about 259,200 sec, the other algorithms stop improving at around 180,911 sec whereas SODA continues to improve for a longer time (until about 210,562 sec) due to a good collection of candidates from the *InflateDeflate* phase and performing simulation refinements using an optimal budget allocation scheme of OCBA-CO in *RefineCandidates* phase. We also ran the Tukey-Kramer procedure for pairwise comparisons of correlated means on all six algorithm types. By doing so, we confirmed that SODA was indeed better than the other algorithms when the experiment ended.

In Fig. 5, we compare different approaches used within *GenerateCandidates*. At the end of the *InflateDeflate* phase, the approach that uses global optimization algorithm of Lipschitz Global Optimization (LGO) [22] within AMPL has a very similar expected cost as the approach that uses the local optimization algorithm of Modular In-core Nonlinear Optimization System (MINOS) [23] with multiple random restarts. However, the approach that uses MINOS without random restarts performs the worst at all times and at the end of the experiment the estimated expected cost is about 10% worse than the other two approaches. This experiment shows that generating candidates within *GenerateCandidates* using a local optimization algorithm with multiple random restarts or a global optimization algorithm results in obtaining better quality candidates than using just a single local optimization approach.

Running *RefineCandidates* with and without OCBA-CO is also evaluated and the results are shown in Fig. 6. In *RefineCandidates* without OCBA-CO, the delta simulation

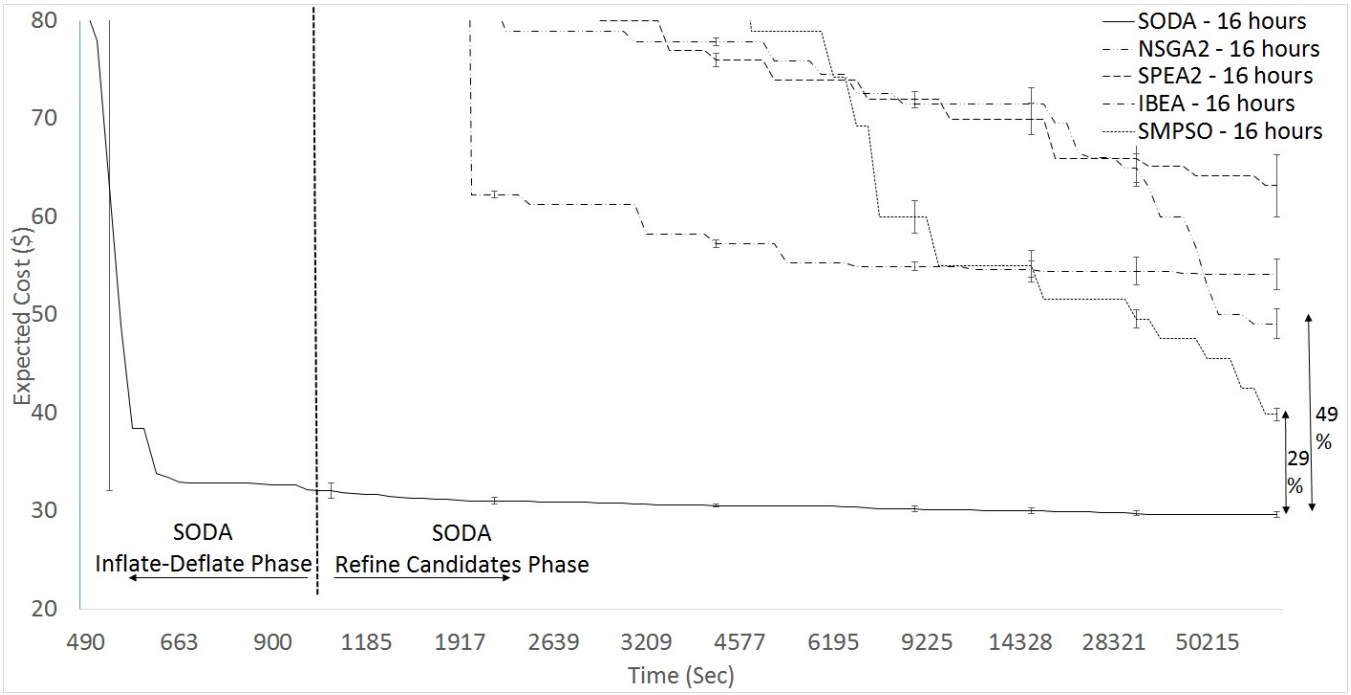

Figure 3: Estimated average cost for the elapsed time (max runtime = 16 hours) (x-axis in log-scale)

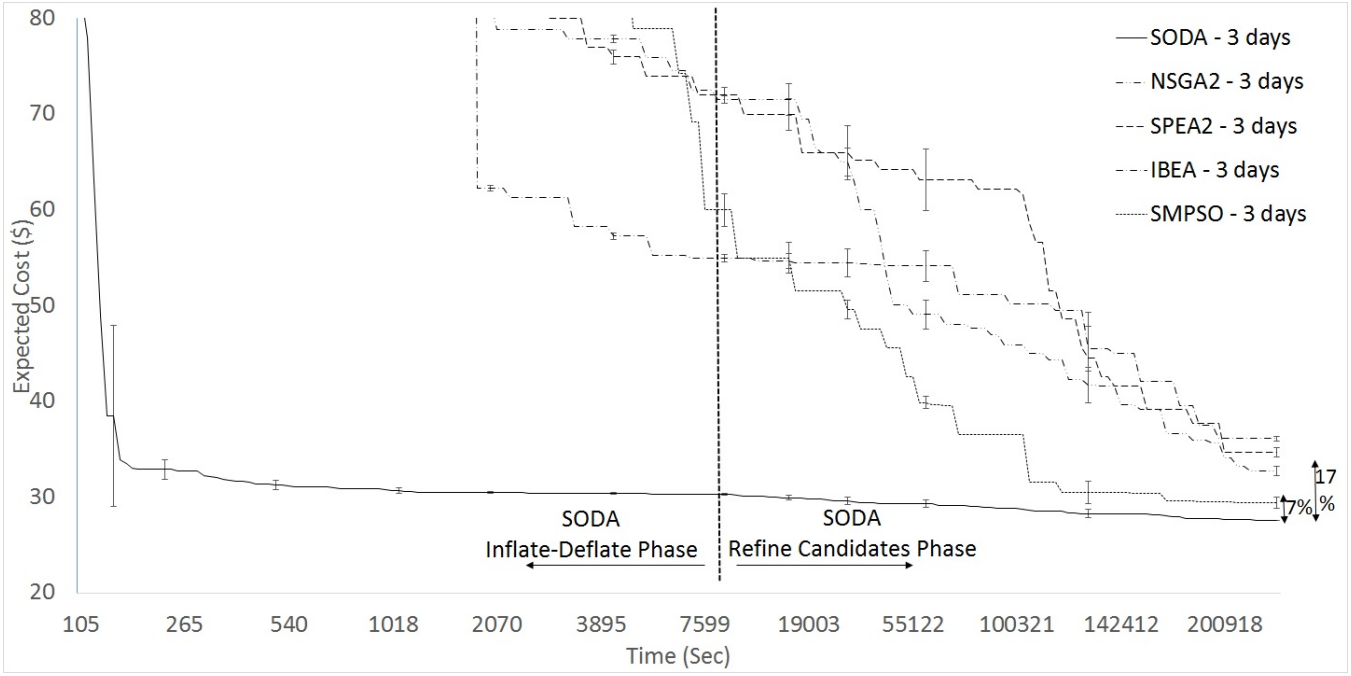

Figure 4: Estimated average cost for the elapsed time (max runtime = 3 days) (x-axis in log-scale)

budget for each iteration is equally distributed among all the candidates. It is clear from the figure, that when using OCBA-CO, the algorithm is able to converge faster and at the end of the experiment, the approach using OCBA-CO is able to achieve a better estimate of expected cost by about 4%.

6 Discussion

The experimental study discussed in section 5 showed that SODA converges much faster toward the optimal point found at the end of the algorithm and that for any given time, the optimal point found by SODA is better than those found by any of the competing algorithms considered in this study. If SODA were allowed

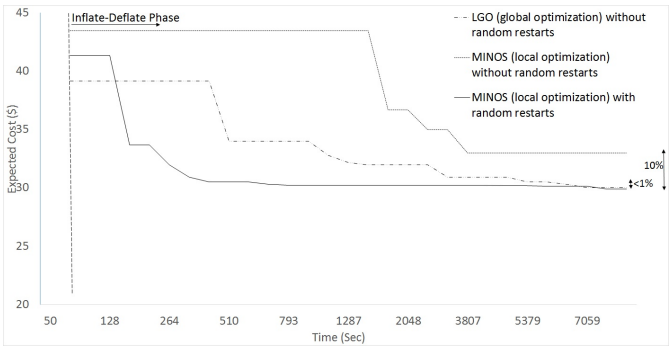

Figure 5: Comparison of different local and global algorithms used during the deterministic optimization in GenerateCandidates

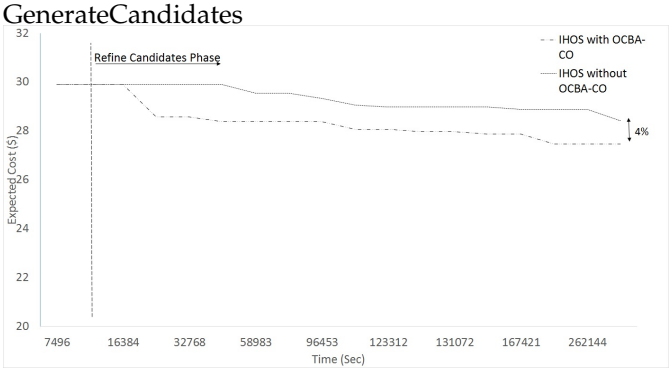

Figure 6: Comparison of RefineCandidates using OCBA-CO and without OCBA

to run for a much longer time, it would result in wasted computing resources since in the *RefineCandidates* phase, more simulation budget would be used to achieve very minor improvements in the optimal point among the fixed number of candidates collected in the *InflateDeflate* phase. Instead, these resources can be used to go back to the *InflateDeflate* phase to perform another series of deterministic approximations to find some other candidates that might yield better improvements in the optimal point. By repeating this process multiple times and given enough time to do so, SODA will eventually converge towards the global optima as guaranteed by the competing algorithms considered in the experimental study.

The experimental study compares SODA's performance against four metaheuristic simulation-based optimization algorithms. These algorithms are popular and are used to solve stochastic optimization problems in a number of complex domains. However, it may be useful to compare SODA's performance to methods that solve the same problem using techniques that typically extract the mathematical structure of the problem using black box simulations and utilize this structure to perform deterministic approximations. Such a comparison may result in a better understanding of how SODA performs against similar but known to be less efficient approaches than SODA. But such a comparative study is outside the scope of this paper and we defer this study to our future work.

7 Related Work

A number of past works have focused on solving the stochastic optimization problem for complex domains such as production processes. This is a typically challenging problem due to the uncertainty present in the problem. Uncertainty is defined as the difference between the amount of information required to perform a task and the amount of information that is actually present. Among others, researchers have looked into two types of uncertainties, namely demand uncertainty and yield uncertainty [24]. Demand uncertainty refers to the difficulty of accurately projecting customer demand in the future. This poses a significant challenge because it makes inventory hard to control and manage [25]. On the other hand, yield uncertainty could be due to the randomness in capacity or where the effects of the control settings are stochastic. In this case, the output from production the minimum of the input and the random yield [26]. In the literature, approaches for stochastic optimization in the presence of demand uncertainty include [27], [28], and [29] and those in the presence of yield uncertainty include [30], [31], and [26].

Stochastic optimization has typically been performed using simulation-based optimization techniques. For instance, [32] present a technique to solve large scale stochastic programming models with a stochastic simulated annealing based on Hammersley sequence sampling technique to improve computational efficiency. Here, the uncertainties are sampled and then using their statistical outcomes as well as error penalties, the simulated annealing approach tries to find the optimal variables for the problem iteratively. [33] try to optimize the production system with a number of plants so as to minimize the expected cost that includes the production cost and inventory holding costs. To overcome the complexities of solving this problem, they propose an approximated approach (AA) to evaluate the objective function. Then, they propose two algorithms to solve the production planning problem, one of which is based on Particle Swam Optimization (PSO) combining with the AA and the other is a hybrid PSO algorithm integrating the AA, neural network (NN) and PSO. Another approach that is used to decide production quantities in planning problems with demand uncertainties is genetic programming. In this work, the Genocop genetic algorithm is used to initialize the population and search for the optimal solutions effectively in a multi-objective production planning problem [34].

Deterministic approximation techniques have been used for stochastic optimization to bridge the gap between stochastic simulation formulation and the use of MP. [27] consider a production planning problem with

uncertain demand. They characterize demand by standard probability distributions and propose deterministic approximations with the mean demand and compute bounds on the optimal value. They show that, for the most commonly used probability distributions of demands, the relative error bounds are very small. They extend these results to obtain a priori upper bounds for a general class of production planning problems in [35].

Another approach used for deterministic approximation is sample average approximations (SAA). One such work tries to solve a discrete optimization problem by generating a random sample of the uncertain variable and then approximating the expected value function using the corresponding sample average function. The obtained sample average optimization problem is solved deterministically, and the procedure is repeated several times until a stopping criterion is satisfied [36]. Although SAA has been used in a number of other works (see [37] for an overview), it is usually better suited for problems where the objective function is easily computable (as a black-box) given the sample values, the expected objective function cannot be written in a closed form, and/or its parameter values cannot be easily calculated [36, 38].

Deterministic approximations have also been used for multistage stochastic optimization. One such approach is called the stochastic linear program with recourse. In this, the objective of the stochastic model is represented as a function of the expected value of the recourse function. The recourse function is itself represented as a deterministic linear model. To evaluate the expected value, multiple recourse function values need to be computed by solving the linear model and then the integral or sum of these values has to be evaluated [39]. Solving this problem of finding multiple (optimal or near-optimal) values of the recourse functions has been performed using decomposition techniques. A number of them use Benders decomposition see e.g., [39], [40], [41], [42]. Some authors like [43] also used decomposition based branch-and-bound strategy to solve a stochastic integer program with recourse.

Another deterministic approximation approach used for multistage stochastic optimization is scenario analysis. [42] model the uncertainty on the stochastic controls via a set of scenarios. To deal with uncertainties the controls are obtained for each scenario, while satisfying the constraints in that scenario. Each scenario is associated with a probability level representing the decision makers expectation of the occurrence of that particular scenario. These different scenarios are then incorporated into a model that is similar to the stochastic program with recourse discussed above to find the expectation of the recourse function based on the scenarios . Other examples of work that use scenario analysis approach include [44], [45], and [46].

A number of approaches also take advantage of the problem or model structure to perform multistage stochastic optimization. Hierarchical control is an approach that tries to simplify the complexity of a large stochastic problem by considering the hierarchy or stages of the problem. [47] propose a hierarchical production planning approach for a problem with demand uncertainties where the first stage of the system produces a set of semi-finished products having relatively stable demands, and the second stage produces finished products having highly variable weekly demand. The first stage is described using an aggregate model whereas the second stage is described as an operational model. The idea here is to build a deterministically limiting model from the aggregate level that is computationally more tractable and then based on this aggregate plan, production planning is performed at the operational level with uncertainties. Model Predictive Control (MPC) is another approach that utilizes the structure of the problem to optimize a dynamic production problem over a rolling horizon. This is akin to dynamic programming problems with multiple decision stages. [46] assume that a model of the actual supply chain is available to the decision maker and that the problem is to minimize the long-term costs of operation, while satisfying the capacity and service level constraints. This problem is solved by using MPC in conjunction with the scenario modeling approach to sample scenarios and handle uncertainties from various sources such as demand, lead times, and yield.

Whilst the existing literature includes a wide variety of simulation-based optimization approaches that use deterministic approximation techniques, they are based on using samples of black-box simulation to either traverse the search space or to extract the mathematical structure of the problem. Thus, there is a shortage of methods that capture the uncertainty explicitly and use this uncertainty to guide the deterministic approximation using the white-box of the model in a one stage stochastic optimization problem. It is well known that models that capture uncertainty explicitly and use this information in the optimization generally perform better than a deterministic-equivalent approach. This is because the uncertainty model conveys important information about the variability of uncertain quantities, as compared to substituting them by their expected or the most likely values [46]. Also, past works do not provide an efficient approach to perform one-stage stochastic optimization over complex production processes with non-linear arithmetic. To bridge this gap, in this paper, we proposed SODA that solves a one stage stochastic optimization problem over a complex production process with non-linear arithmetic. SODA solves this problem by performing a series of deterministic approximation by extracting the mathematical structure from a white-box simulation code analysis. The experimental results show that this approach converges quickly and yields more optimal results as compared to simulation-based stochastic optimization algorithms for a real-world use

case of the production process problem.

8 Conclusion and Future Work

This paper presents an efficient stochastic optimization algorithm called SODA for the problem of finding process controls of a steady-state production processes that minimize the expectation of cost while satisfying the deterministic feasibility constraints and stochastic steady state demand for the output product with a given high probability. SODA is based on performing (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.

This paper also demonstrates SODA on a use case of a real-world heat-sink production process that involves contract suppliers and manufacturers as well as unit manufacturing processes of shearing, milling, drilling, and machining. Finally, this paper conducts an experimental study that shows that SODA significantly outperforms four popular simulation-based stochastic optimization algorithms. In particular, the study shows that SODA performs better than the best competing algorithm by 29% after 16 hours and by 7% after three days.

Future research directions include: (a) dynamically executing the inflate deflate phase and candidate refinement phase of SODA to improve the exploration of the search space; (b) generalizing SODA to handle production process that produce multiple products; (c) adding stronger heuristics to SODA; (d) comparing SODA with an existing stochastic optimization algorithm based on deterministic approximations; and (e) developing specialized algorithms that can utilize preprocessing of stored (and therefore, static) components to speed up optimization, generalizing the results in [48, 49, 50].

References

[1] S. Amaran, N. V. Sahinidis, B. Sharda, and S. J. Bury, "Simulation optimization: a review of algorithms and applications," *Annals of Operations Research*, vol. 240, no. 1, pp. 351–380, 2016.

[2] A.-T. Nguyen, S. Reiter, and P. Rigo, "A review on simulation-based optimization methods applied to building performance analysis," *Applied Energy*, vol. 113, pp. 1043–1058, jan 2014.

[3] J. B. Dabney and T. L. Harman, *Mastering SIMULINK 4*. Upper Saddle River, NJ, USA: Prentice Hall, 1st ed., 2001.

[4] P. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 2.1*. Piscataway, NJ, USA: Wiley-IEEE Press, 2004.

[5] G. Provan and A. Venturini, "Stochastic simulation and inference using modelica," in *Proceedings of the 9th International Modelica Conference*, pp. 829–837, Sep 2012.

[6] H. Thieriota, M. Nemera, M. Torabzadeh-Tarib, P. Fritzson, R. Singh, and J. J. Kocherry, "Towards design optimization with openmodelica emphasizing parameter optimization with genetic algorithms," in *Modellica Conference*, Modelica Association, 2011.

[7] OpenModelica, *Efficient Traceable Model-Based Dynamic Optimization - EDOp*. OpenModelica, 2009.

[8] J. J. Durillo and A. J. Nebro, "jMetal: A Java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, no. 10, pp. 760 – 771, 2011.

[9] A. Klemmt, S. Horn, G. Weigert, and K.-J. Wolter, "Simulation-based optimization vs. mathematical programming: A hybrid approach for optimizing scheduling problems," *Robotics and Computer-Integrated Manufacturing*, vol. 25, no. 6, pp. 917–925, 2009.

[10] S. D. Thompson and W. J. Davis, "An integrated approach for modeling uncertainty in aggregate production planning," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, pp. 1000–1012, Sept. 1990.

[11] D. Paraskevopoulos, E. Karakitsos, and B. Rustem, "Robust Capacity Planning under Uncertainty," *Management Science*, vol. 37, no. 7, pp. 787–800, 1991.

[12] J. Xu, S. Zhang, E. Huang, C. H. Chen, L. H. Lee, and N. Celik, "An ordinal transformation framework for multi-fidelity simulation optimization," in *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 385–390, Aug 2014.

[13] M. Krishnamoorthy, A. Brodsky, and D. Menascé, "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, 2015.

[14] C. H. Chen and L. H. Lee, *Stochastic Simulation Optimization: An Optimal Computing Budget Allocation*. Hackensack, NJ, USA: World Scientific Publishing Company, 2011.

[15] L. H. Lee, N. A. Pujowidianto, L. W. Li, C. H. Chen, and C. M. Yap, "Approximate simulation budget allocation for selecting the best design in the presence of stochastic constraints," *IEEE Transactions on Automatic Control*, vol. 57, pp. 2940–2945, Nov 2012.

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[17] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature, 2004*, pp. 832–842, Springer, 2004.

[18] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," Tech. Rep. 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 2001.

[19] A. Nebro, J. Durillo, J. García-Nieto, C. Coello Coello, F. Luna, and E. Alba, "Smpso: A new pso-based metaheuristic for multi-objective optimization," in *2009 IEEE Symposium on Computational Intelligence in Multicriteria Decision-Making (MCDM 2009)*, pp. 66–73, IEEE Press, 2009.

[20] J. Robie, G. Fourny, M. Brantner, D. Florescu, T. Westmann, and M. Zaharioudakis, "Jsoniq the complete reference," 2015.

[21] A. Brodsky, M. Krishnamoorthy, W. Z. Bernstein, and M. O. Nachawati, "A system and architecture for reusable abstractions of manufacturing processes," in *2016 IEEE International Conference on Big Data (Big Data)*, pp. 2004–2013, Dec 2016.

[22] J. D. Pintér, *Global optimization in action: continuous and Lipschitz optimization: algorithms, implementations and applications*, vol. 6. Springer Science & Business Media, 2013.

[23] B. A. Murtagh and M. A. Saunders, "Minos. a large-scale nonlinear programming system (for problems

with linear constraints). user's guide.," tech. rep., STANFORD UNIV CALIF SYSTEMS OPTIMIZATION LAB, 1977.

[24] J. Mula, R. Poler, J. P. Garca-Sabater, and F. C. Lario, "Models for production planning under uncertainty: A review," *International Journal of Production Economics*, vol. 103, pp. 271–285, Sept. 2006.

[25] M. Fisher and A. Raman, "Reducing the Cost of Demand Uncertainty through Accurate Response to Early Sales," *Operations Research*, vol. 44, no. 1, pp. 87–99, 1996.

[26] C. H. Glock, E. H. Grosse, and J. M. Ries, "The lot sizing problem: A tertiary study," *International Journal of Production Economics*, vol. 155, pp. 39–51, Sept. 2014.

[27] G. R. Bitran and H. H. Yanasse, "Deterministic Approximations to Stochastic Production Problems," *Operations Research*, vol. 32, no. 5, pp. 999–1018, 1984.

[28] R. W. Grubbstrm and Z. Wang, "A stochastic model of multi-level/multi-stage capacity-constrained productioninventory systems," *International Journal of Production Economics*, vol. 8182, pp. 483–494, Jan. 2003.

[29] A. Gupta and C. D. Maranas, "Managing demand uncertainty in supply chain planning," *Computers & Chemical Engineering*, vol. 27, pp. 1219–1227, Sept. 2003.

[30] C. A. Yano and H. L. Lee, "Lot Sizing with Random Yields: A Review," *Operations Research*, vol. 43, no. 2, pp. 311–334, 1995.

[31] A. Grosfeld-Nir and Y. Gerchak, "Multiple Lotsizing in Production to Order with Random Yields: Review of Recent Advances," *Annals of Operations Research*, vol. 126, pp. 43–69, Feb. 2004.

[32] K.-J. Kim and U. M. Diwekar, "Hammersley stochastic annealing: efficiency improvement for combinatorial optimization under uncertainty," *IIE Transactions*, vol. 34, pp. 761–777, Sept. 2002.

[33] Y.-F. Lan, Y.-K. Liu, and G.-J. Sun, "Modeling fuzzy multi-period production planning and sourcing problem with credibility service levels," *Journal of Computational and Applied Mathematics*, vol. 231, pp. 208–221, Sept. 2009.

[34] H. B. Chen, N. Zhao, and G. Q. Sun, "An Approach for Production Planning Optimization Under Correlated Uncertain Demand," in *2007 International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 3736–3739, Sept. 2007.

[35] G. R. Bitran and D. Sarkar, "On upper bounds of sequential stochastic production planning problems," *European Journal of Operational Research*, vol. 34, pp. 191–207, Mar. 1988.

[36] A. Kleywegt, A. Shapiro, and T. Homem-de Mello, "The Sample Average Approximation Method for Stochastic Discrete Optimization," *SIAM Journal on Optimization*, vol. 12, pp. 479–502, Jan. 2002.

[37] S. Kim, R. Pasupathy, and S. G. Henderson, "A Guide to Sample Average Approximation," in *Handbook of Simulation Optimization* (M. C. Fu, ed.), no. 216 in International Series in Operations Research & Management Science, pp. 207–243, Springer New York, 2015. DOI: 10.1007/978-1-4939-1384-8_8.

[38] A. Shapiro, "Sample Average Approximation," in *Encyclopedia of Operations Research and Management Science* (S. I. Gass and M. C. Fu, eds.), pp. 1350–1355, Springer US, 2013. DOI: 10.1007/978-1-4419-1153-7_1154.

[39] G. S. U. Infanger, "Planning Under Uncertainty Solving Large-Scale Stochastic Linear Programs," Tech. Rep. SOL-92-8, Stanford Univ., CA (United States). Systems Optimization Lab., Dec. 1992.

[40] E. Schweitzer, *Multi-Stage Mathematical Programming under Uncertainty*. PhD thesis, Israel Institute of Technology, Haifa, Heshvan, 5755, Oct. 1994.

[41] S. A. MirHassani, C. Lucas, G. Mitra, E. Messina, and C. A. Poojari, "Computational solution of capacity planning models under uncertainty," *Parallel Computing*, vol. 26, pp. 511–538, Mar. 2000.

[42] A. Alonso-Ayuso, L. F. Escudero, A. Garn, M. T. Ortuo, and G. Prez, "An Approach for Strategic Supply Chain Planning under Uncertainty based on Stochastic 0-1 Programming," *Journal of Global Optimization*, vol. 26, pp. 97–124, May 2003.

[43] S. Ahmed and R. Garcia, "Dynamic Capacity Acquisition and Assignment under Uncertainty," *Annals of Operations Research*, vol. 124, pp. 267–283, Nov. 2003.

[44] T. Santoso, S. Ahmed, M. Goetschalckx, and A. Shapiro, "A stochastic programming approach for supply chain network design under uncertainty," *European Journal of Operational Research*, vol. 167, pp. 96–115, Nov. 2005.

[45] J. P. Watson, D. L. Woodruff, and W. E. Hart, "PySP: Modeling and solving stochastic programs in Python," *Mathematical Programming Computation*, vol. 4, no. 2, pp. 109–149, 2012.

[46] G. Schildbach and M. Morari, "Scenario-based model predictive control for multi-echelon supply chain management," *European Journal of Operational Research*, vol. 252, pp. 540–549, July 2016.

[47] E.-H. Aghezzaf, C. Sitompul, and F. Van den Broecke, "A robust hierarchical production planning for a capacitated two-stage production system," *Computers & Industrial Engineering*, vol. 60, pp. 361–372, Mar. 2011.

[48] N. Egge, A. Brodsky, and I. Griva, "Online optimization through preprocessing for multi-stage production decision guidance queries," *30th IEEE International Conference on Data Engineering Workshops*, vol. 1, pp. 41–48, 2012.

[49] N. Egge, A. Brodsky, and I. Griva, "Distributed manufacturing networks: Optimization via preprocessing in decision guidance query language," *International Journal of Decision Support System Technology*, vol. 4, pp. 25–42, July 2012.

[50] N. Egge, A. Brodsky, and I. Griva, "An efficient preprocessing algorithm to speed-up multistage production decision optimization problems," in *46th Hawaii International Conference on System Sciences (HICSS), 2013*, pp. 1124–1133, Jan 2013.