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

Optimizing Stochastic Temporal Manufacturing Processes with Inventories: An Efficient Heuristic Algorithm 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-2015-8

Abstract

This paper deals with stochastic temporal manufacturing processes with work-in-process inventories in which multiple products are produced from raw materials and parts. The processes may be composed of subprocesses, which, in turn may be either composite or atomic, i.e., a machine on a manufacturing floor. We assume that machines' throughput is stochastic and so are work-in-process inventories and costs. We consider the problem of optimizing the process, that is, finding throughput expectation setting for each machine at each time point over the time horizon as to minimize the total cost of production subject to satisfying the production demand with a requested probability. To address this problem, we propose an efficient iterative heuristic algorithms that is based on (1) producing high quality candidate machine settings based on a deterministic approximation of the stochastic problem, and (2) running stochastic simulations to find the best machine setting out of the produced candidates using optimal simulation budget allocation methods. We conduct an experimental study that shows that our algorithm significantly outperforms four popular simulation-based optimization algorithms.

1 Introduction

In the past few years, there has been significant technological advancements in different areas of process analysis and optimization. Examples of processes include manufacturing processes, such as assembly lines, and supply chain management. These processes often involve physical or virtual inventories of products, parts and materials that are used to anticipate uncertainties on supply or throughputs of machines. Over time, the state of the machines, inventories and the whole process changes until process completion. We use the term Buffered Temporal Flow Processes (BTFP) to describe these types of processes. BTFP can be found in many different areas of manufacturing and supply chain. A particularly important BTFP occurs in the area of discrete manufacturing such as in vehicles, furniture, smartphones, airplanes and toys.

Due to increased global competition, manufacturing companies look toward ways to reduce their cost and increase efficiency of operations. Thus, there is a greater need for analysis and optimization of the operation results on the manufacturing floor while taking into account sustainability metrics. In addition to this, the metrics in the manufacturing model may contain noise that makes them stochastic. To support analysis and optimization of BTFP with stochastic variables, there is a need to accurately model machines, systems and processes so that they can provide accurate results in a stochastic environment. These models need to capture (a) stochastic control variables; (b) metrics of machines (such as cost, energy consumption, and emission) as a function of these control variables; (c) process routing that describes the flow of materials through the manufacturing floor; and (d) intermediate material storage and distribution (work-in-progress) for inventories. In BTFP, this needs to be modeled over a temporal sequence and include stochasticity of machine throughput and supply.

Using these models, it is desirable to allow manufacturing and process operators to perform a variety of analysis and optimization tasks including what-if prediction and optimization. For example, as a prediction question, a process engineer may ask: given a particular planned machine's expected throughput and load-distribution among the machines, what would be the production output and work-in-progress inventories for each time interval over the time horizon, as well as overall manufacturing key performance indicators (KPIs) such as cost,

efficiency and carbon emissions? Or, as an optimization question, a process engineer may ask: given the process design (which includes the flow of work pieces through various stages of processing), which machines should be on and off, and how to set up the controls of every operational machine, and distribute processing load among the machines, as to minimize the total production cost, while satisfying the demand for every time interval over a planning horizon, and within a limitation on the capacity of work-in-progress inventories?

There has been extensive research on analysis and optimization of BTFP-like processes (e.g., see [1], [2], [3] for an overview). Prior work can be classified into three broad categories: (1) customized domain-specific solutions for optimization of manufacturing processes; (2) simulation-based solutions; and (3) use of optimization solvers based on mathematical programming (MP) and constraint programming (CP).

Customized domain-specific solutions for BTFP are designed for a specific, limited setting of a manufacturing process, and would typically provide a graphical user interface. Examples include [4] and [5]. The implementation of domain specific solutions may use optimization tools based on mathematical programming, and integrate them with other systems such as Enterprise Resource Planning (ERP). However, while these solutions may be both efficient (in terms of optimality of results and computational time), they are typically not extensible to additional aspects of machines, processes and metrics (e.g., stochastic variables), and perform a "silo" optimization, which would not achieve the optimal outcome if the extended system is to be optimized as a whole, as opposed to optimizing a series of "silo" subproblems.

In order to accurately model a system and its inner workings, a simulation-based system can be used. Such systems are typically object-oriented, modular, extensible, and reusable. Furthermore, many simulation tools provide an easy-to-use graphical user interface. Tools like SIMULINK [6] and Modelica-based ones [7] like JModelica [8], Dymola [9], and MapleSim [10] allow users to simulate models of complex systems in mechanical, hydraulic, thermal, control, and electrical power. For example, Modelica comes with over 1000 generic model components that can all be reused. In addition, tools like OMOptim [11], Efficient Traceable Model-Based Dynamic Optimization (EDOp) [12], and jMetal [13] use simulation models to heuristically-guide a trial and error search for the optimal answer. However, this search approach does not utilize the mathematical structure of the underlying problem in the way MP/CP methods do. Due to this, simulation-based optimization tools are significantly inferior to optimization solutions based on MP/CP in terms of optimality of results and computational complexity for problems that can be expressed using MP/CP formulation.

Optimization solvers and modeling languages based on MP and CP are often the technology of choice, when optimality and computational complexity are the priority. Many classes of MP, such as linear programming (LP), mixed integer linear programming (MILP), and non-linear programming (NLP), have been very successful in solving real-world large-scale optimization problems. CP, on the other hand, has been broadly used for combinatorial optimization problems like scheduling and planning. To use these tools, one would have to use an algebraic modeling language such as A Modeling Language for Mathematical Programming (AMPL) [14], Optimization Programming Language (OPL) [15], General Algebraic Modeling System (GAMS) [16], or Advanced Interactive Multidimensional Modeling System (AIMMS) [17]. However, MP and CP present a significant challenge for engineers and business analysts. It requires an OR expert to model a problem and express it in an algebraic modeling language like the ones mentioned. Additionally, these formal models are typically difficult to modify, extend, or reuse. This is comparable to "spaghetti" code versus an object-oriented approach.

Temporal Manufacturing Query Language (tMQL) [18] was recently proposed to target modularity and reusability of manufacturing floor components for deterministic BTFP processes. In turn, tMQL is based on the Process Analytics Formalism [19][20], by extending it to BTFP process components. In turn, SPAF is based on the ideas of modular representation of constraints and reductions to formal optimization models from [21], [22], [23], [24], [25], and [26]. These models can be reused to perform analysis and optimization tasks that use declarative computation and optimization queries. Computation queries are reduced to simulation models, while optimization queries are reduced to Mixed Integer Linear Programming (MILP) model and solved using a commercial solver. However, tMQL in [18] is limited to a deterministic setting, whereas process variables and metrics are often stochastic. While extending the definition of the tMQL process optimization problem to a 1-stage stochastic programming problem is easy, developing efficient algorithms to solve it is not. This is exactly the focus of this paper.

We are concerned with BTFP that can be described with tMQL, extended with stochasticity of machines' throughput. That is, we consider stochastic temporal manufacturing processes with work-in-process inventories in which multiple products are produced from raw materials and parts. The processes may be composed of subprocesses, which, in turn may be either composite or atomic, i.e., a machine on a manufacturing floor. We assume that machines' throughput is stochastic and so are work-in-process inventories and costs. We consider the problem of optimizing the process, that is finding throughput expectation setting for each machine at each time point over the time horizon as to minimize the total cost of production subject to satisfying the production demand with a requested probability.

The key technical contributions of this paper are twofold. First, we propose an efficient heuristic algorithm, called Iterative Heuristic Optimization Simulation (IHOS) based on (1) producing a set of high quality candidate machine settings based on deterministic approximations of the given stochastic problem, and (2) running stochastic simulations to find the best machine setting out of the candidate set, using optimal simulation budget allocation methods. Second, we conduct an initial experimental study to compare the proposed algorithm with four popular simulation-based optimization algorithms: Nondominated Sorting Genetic Algorithm 2 (NGSA2) [27], Indicator Based Evolutionary Algorithm (IBEA) [28], Strength Pareto Evolutionary Algorithm 2 (SPEA2) [29], and Fast Pareto Genetic algorithm (FastPGA) [30]. The experimental study demonstrates that IHOS significantly outperforms the other algorithms in terms of optimality of results and computation time; in particular, in 64 seconds the cost achieved by IHOS is 5% of the cost achieved by competing algorithms, and is not matched by them in 1500 seconds (total run time), while the cost achieved by IHOS in 1500 seconds is 80% of the competing algorithms and 87% of the cost reached by IHOS in 64 seconds.

The rest of this paper is organized as follows: Section 2 motivates the reader on the need for solving models on manufacturing processes with stochastic variables. Section 3 gives details about the IHOS algorithm. In Section 4 the experimental setup and results of the comparison between IHOS and other metaheuristic algorithms is explained. Finally, Section 5 concludes and discusses further work.

2 Motivating Problem

Manufacturing processes can be complicated to model and may contain very complex structures. Additionally, the parameters in the manufacturing model may be stochastic. We explain one such manufacturing process in this section and illustrate the challenges in obtaining meaningful answers from the floor. Consider the sand and cut manufacturing processes shown in Figure 1 that takes plywood as its input. In this example, part of the plywood goes to the *sand1* machine and the remaining goes to the *sand2* machine. The sanded plywood is then buffered and redistributed among the *cut1* and *cut2* machines. The sanded plywood is cut in these machines and finally, the cut plywood is collected and provided as output from the sand and cut manufacturing processes. We assume that time is divided into time intervals of duration Δt, where time intervals start and end at time points (t). We assume without loss of generality that $\Delta t = 1$. A time interval (also known as a period) is denoted by $p_{i+1} = (t_i, t_{i+1})$.

In order to compose the sand and cut manufacturing processes in tMQL [18], six modular components will be initialized to map them to the physical entities of the floor. The first component is the input quantity aggregators (IQA) such as $a1$, that get the items from the input and distributes them among the $sand1$ and $sand2$ processes. The second component is the base process that maps to the machines on the floor. This process has controls for the throughput or speed of the machines as well as Key Performance Indicator (KPI) variables such as cost and energy consumed. In our example, the base processes are the $sand1$, $sand2$, $cut1$ and $cut2$ components. The third component is the inventory aggregators (IA) such as $a2$ that provide the analytical knowledge for a work-in-progress inventory. The fourth component is the output quantity aggregators (OQA) such as $a3$, that collect items from the $cut1$ and $cut2$ machines and dispense them from the floor. The fifth component are the item flows that carry items from the input or output or between the processes and aggregators. Finally, the sixth component is that of the composite process that contains one or more of the six components mentioned here to represent the manufacturing process as a whole. The composite process also provides global metrics such as total cost and total energy consumed by the entire floor. The total cost and total energy of the machines are functions that take the throughput of the machines as parameters. These modules and model composition are explained in more detail in [18].

In most manufacturing settings, the model parameters are stochastic variables. Thus, these machines will run at different throughputs at each time interval, with some added noise that follows a distribution around the expected throughput. Since variables such as total cost incurred and the number of items that flow through the aggregators are a direct result of the throughput of the machines, it becomes very difficult for process operators to get meaningful answers from these models for queries such as: (a) what is the total cost of operating the machines for the required demand; (b) what are the maximum number of items stored in the inventory across all time periods; (c) what-if the noise level is increased or decreased in one of the machines; and (d) how is cost affected if one machine is replaced by a faster machine?

In order to obtain meaningful answers for such queries from the model, it is required that the stochastic nature of the variables be taken into consideration. In tMQL, the deterministic optimization problem was defined as finding the decision variables within an atomic or composite process at each time point over the time horizon as to minimize or maximize the objective function subject to satisfying all the constraints within the component and any sub-components. The mathematical formulation of this problem is given in [18]. We extend this definition to include the stochastic variables within the components. Thus, the problem of stochastic optimization can be cast as finding the throughput expectation setting for each base process component at each time point over the time horizon as to minimize the total expected cost

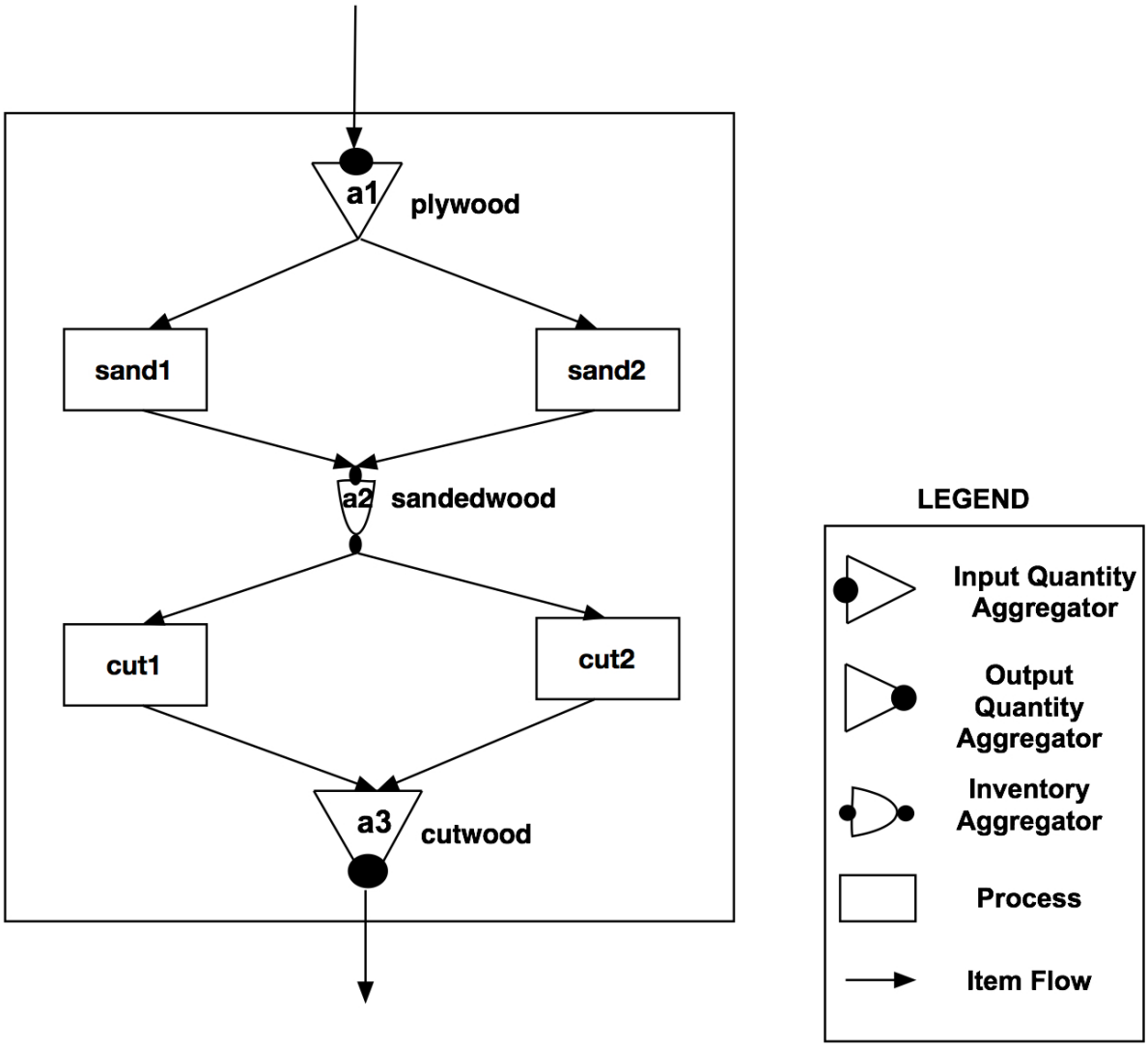

Figure 1: A graphical notation for the sand and cut manufacturing processes

of production subject to satisfying the production demand with a requested probability. More generally, this task may involve finding decision variables within any atomic or composite process as to minimize or maximize the expected value of the objective subject to satisfying constraints with a requested probability. For the sake of completeness, an OPL-like formulation of this one stage stochastic optimization problem is given in the Appendix.

To solve such a problem, an approximate cost needs to be computed such that it takes the approximate value of the machine's throughputs into consideration. In order to do this, in addition to the expected throughputs, the machines should also be provided with the actual throughput. At each time point, the actual machine throughputs can then be calculated by adding the noise to the expected throughput. The expected throughput mentioned here is the optimal value of the throughput found in a deterministic setting. It is now possible to apply some heuristics on top of the model to obtain meaningful answers for questions that depend on the stochastic variables. The next section gives details of our algorithm to perform this approximation.

3 Iterative Heuristic Optimization-simulation Algorithm

This section describes the iterative heuristic optimization-simulation (IHOS) algorithm. IHOS uses the powerful tMQL domain specific components that can be reused to accurately model a system and its inner workings and can be used in optimization solutions based on MP/CP where optimality of results and computational complexity are a priority. To model and query a manufacturing process for the real world, it is necessary to take into consideration the stochastic nature of the variables of the model. As

described in Section 2, we account for the stochastic throughput of the machines by adding noise to the expected throughput parameter. Using these metrics, the algorithms described in this section approximate the total cost incurred of running the entire composite process when the actual throughput of the machines is stochastic. The main idea behind these algorithms is to run multiple iterations of optimization and stochastic simulation. The optimization solver solves the problem of finding the expected throughputs of the machines such that the demand is satisfied at each period while minimizing the cost. The stochastic simulation then adds noise to these mean throughputs to check whether the probability of satisfying the demand at each time period has a certain confidence. A heuristic is then applied on the demands so that they can be increased or decreased such that the optimizer can give more realistic mean throughputs in the next iteration. A number of candidate mean throughputs are collected and then simulations are run on these candidates to make sure that the probability of demand satisfied indeed has the desired confidence and the cost is approximately minimum. In this way, the algorithm uses the model knowledge in both optimization and stochastic simulation to provide an optimal setting for the throughputs of the machines that the process operator can use in a stochastic environment.

The pseudo code for the IHOS algorithm is shown in Algorithm 1. This algorithm works in two phases. The first phase (lines 5 - 43) is the heuristic optimization-simulation phase. In this phase, the algorithm tries to collect candidates that could potentially result in the minimum expected cost such that the demands (*actualDemand*) are satisfied in a stochastic environment. In order to do this for the stochastic throughputs, a number of iterations of optimization and stochastic simulation are run that are bound by a certain budget (*totalIterations*) or by the number of desired candidates (*storeSize*). This algorithm first tries to solve a deterministic optimization problem where the throughput of the machines are decision variables, the demand on the number of items to be produced at each time period (*currentDemand*) is one of the constraints and the total cost is minimized subject to constraints in each individual component. These decision variables will be the optimal throughput of the machines for the *currentDemand* in a deterministic environment and are returned back as *throughputExp* in line 6. Then, the problem is run in a stochastic simulation environment that takes these throughputs as expected throughputs and adds noise to them. This noise may follow any distribution such that its parameters (e.g., σ) are set by the process operator in the simulation. A call to the stochastic simulation is made in line 7 whereas the pseudo-code for the simulation is given in Algorithm 2.

The simulation function runs multiple (*noSimulations*) Monte Carlo simulations on the model for the manufacturing processes (Algorithm 2, lines 1 - 3). Then, the *avgCost* is computed as the average of the total costs obtained from all simulations (Algorithm 2, line 4). In order to check the number of times the demand was satisfied, this algorithm then counts the number of times the produced items (*producedItems*) were greater than the demand at each time point. The probability of the demand being satisfied (*demandSetProbs*) is the ratio between the number of times the demand was satisfied at each period and *noSimulations* (Algorithm 2, line 5 - 12).

After the stochastic simulations returns (Algorithm 1, line 7), the iteration proceeds to check whether the confidence that the *demandSetProbs* is greater than the *probabilityBound* for each period is at least *iterationConfidence* (lines 8 - 11). If this is the case, this iteration is stored as a candidate (lines 12). Then, the iteration proceeds to check whether this confidence is also at least equal to *finalConfidence*. The *finalConfidence* is a higher confidence level than the *iterationConfidence* and is the desired confidence level for the demands at which a *avgCost* is accepted. If the *finalConfidence* is obtained, the global *minCost* metric is updated if such a minimum *avgCost* is achieved (lines 15 - 19). If the *finalConfidence* cannot be obtained from the current run of stochastic simulations, a greater number of stochastic simulations (bound by *maxExtraSimulationIterations*) are run to check whether the *finalConfidence* can be obtained and the cost can be further minimized or whether the candidate can be refuted with a lower *refuteConfidence* so that no more simulations are wasted on this candidate (lines 20 - 39).

The last part of the iteration is the call to the heuristic function demand adjust (line 41). The pseudo-code for this heuristic function is shown in Algorithm 3. This function uses the difference in the probabilities of the demand satisfaction and an exponential function with parameter λ to either increase or decrease the value of the demand at each period. This increase (or decrease) in demand is decided by an exponential distribution with the parameter equal to the difference in the probability of demand satisfaction obtained from the simulation runs (*demandSetProbs*) and the *probabilityBound*. This ensures that if the percentage of the number of times the demand was satisfied for each period was very low (or very high), this heuristic will increase (or decrease) the demand for that period exponentially so that the optimization solver can adjust the mean throughputs accordingly in the next iteration. Thus, the *demandSetProbs* would increase (or decrease) in the next iteration thereby resulting in a greater likelihood of obtaining the desired confidence of demand satisfaction.

The second phase of the IHOS algorithm is simulation-based refinement using an extended Optimal Computing Budget Allocation (OCBA) algorithm [31]. OCBA is a budget allocation algorithm, which ensures that the resulting probability of the current selection (i.e., the current top candidate is indeed the best) be maximized. An application of this budget allocation algorithm can be found in [32]. The objective for extended OCBA in

Algorithm 1: Iterative heuristic optimization-simulation

Input : storeSize, totalIterations, actualDemand, σ, noSimulations, lastTP, probabilityBound, iterationConfidence, finalConfidence, refuteConfidence maxExtraSimulationIterations, λ, budgetDelta, budgetThreshold
Output: throughputWithMinCost, minCost

```
1  noCandRuns := 0
2  currentDemand := actualDemand
3  noIterations := 0
4  minCost := ∞
5  repeat
6      throughputExp := DetOpt (currentDemand)
7      (demandSetProbs, avgCost) := Simulate (throughputExp, noSimulations, actualDemand, lastTP)
8      foreach tp← 1 to lastTP do
9          confidencePerTP[tp] := Confidence (demandSetProb[tp] >= probabilityBound)
10     end
11     if ConfidenceSatisfied (confidencePerTP >= iterationConfidence) then
12         storedRuns.add(<avgCost, throughputExp, confidencePerTP>)
13         candSimulations.add(noSimulations)
14         noCandRuns := noCandRuns + 1
15         if ConfidenceSatisfied ( confidencePerTP >= finalConfidence) then
16             if avgCost < minCost then
17                 minCost := avgCost
18             end
19         else
20             c := 1  // c is a counter that counts the number of extra simulation iterations performed
21             while c <= maxExtraSimulationIterations do
22                 (demandSetProbs, avgCost) := Simulate (throughputExp, noSimulations*c, actualDemand, lastTP)
23                 foreach tp← 1 to lastTP do // lastTP is the last time point
24                     confidencePerTP[tp] := Confidence (demandSetProb[tp] >= probabilityBound)
                       // refute probabilityBound is less than the actual probabilityBound by ϵ, e.g.  ϵ = 0.15
25                     refuteConfidencePerTP[tp] := Confidence (demandSetProb[tp] <= (probabilityBound - ϵ))
26                 end
27                 candSimulations.add(noSimulations*c)
28                 if ConfidenceSatisfied (confidencePerTP >= finalConfidence) then
29                     if avgCost < minCost then
30                         minCost := avgCost
31                     end
32                 else
33                     if ConfidenceRefuted (refuteConfidencePerTP >= refuteConfidence) then
34                         Remove candidate from the storedRuns
35                     end
36                 end
37                 c := c+1
38             end
39         end
40     end
41     currentDemand := DemandAdjust (currentDemand, demandSetProbs, probabilityBound, λ, lastTP)
42     noIterations := noIterations + 1
43 until noIterations > totalIterations or noCandRuns > storeSize
44 minCost := ExtendedOCBA (storedRuns, actualDemand, probabilityBound, finalConfidence, minCost, candSimulations,
   budgetDelta, budgetThreshold, maxExtraSimulationIterations, refuteConfidence, lastTP)
45 throughputWithMinCost := storedRuns.get(avgCost = minCost)
46 return throughputWithMinCost, minCost
```

this case is minimum cost among the candidates whose confidence of demand satisfaction is above the threshold. The call to the extended OCBA is shown in line 44 in Algorithm 1, whereas the pseudo-code for the extended OCBA algorithm is shown in Algorithm 4. Extended OCBA has an initialization phase and an iterative phase. The data obtained from the first phase of the IHOS algorithm can be reused in the initialization phase of extended OCBA. Therefore, Algorithm 4 directly starts with the iterative phase of extended OCBA. In this phase, the average cost is sorted in ascending order into *sortedCost*. In order to find the boundary of the top candidate of interest, simulations are run until the confidence of demand satisfaction is achieved on

Algorithm 2: Simulate

```
Input  : throughputExp, noSimulations, actualDemand,
         lastTP
Output: demandSetProbs, avgCost
1  foreach run ← 1 to noSimulations do
2      (totalCost[run], producedItems[run]) :=
       MonteCarloSimulation (throughputExp)
3  end
4  avgCost := average of totalCost
5  foreach tp ← 1 to lastTP do
6      foreach run ← 1 to noSimulations do
7          if producedItems[run][tp] >= actualDemand[tp]
           then
8              demandSetProbs[tp] = demandSetProbs[tp] + 1
9          end
10     end
11     demandSetProbs[tp] =
       demandSetProbs[tp]/noSimulation
12 end
13 return demandSetProbs, avgCost
```

Algorithm 3: DemandAdjust

```
Input  : currentDemand, demandSetProbs,
         probabilityBound,
         λ, lastTP
Output: currentDemand
1  foreach tp ← 1 to lastTP do
2      if probabilityBound > demandSetProbs[tp]) then
3          x := probabilityBound - demandSetProbs[tp]
4          increment :=
           currentDemand[tp] * exponentialDensity (λ, x)
5          currentDemand[tp] = currentDemand[tp] +
           increment
6      else
7          x := demandSetProbs[tp] - probabilityBound
8          decrement :=
           currentDemand[tp] * exponentialDensity (λ, x)
9          currentDemand[tp] = currentDemand[tp] - decrement
10     end
11 end
12 return currentDemand
```

a top candidate. (Algorithm 4, lines 3 - 11). Then, the number of simulations is computed using OCBA such that a larger portion of the *budgetDelta* is allocated to candidates that are closer to the boundary. (Algorithm 4, lines 12 - 13). After this, the iterative phase of extended OCBA follows very similarly the first phase of the IHOS algorithm where after running the initial number of simulations, more simulations are run in an effort to decrease the cost and increase the confidence of demand satisfaction (Algorithm 4, lines 14 - 46). The iterative phase of extended OCBA continues until the number of stochastic simulations (*budget*) crosses the total number of allowed simulations (*budgetThreshold*).

4 Experimental Results

This section compares the IHOS algorithm with other metaheuristic algorithms for multi-objective optimization. As explained in Section 3, IHOS performs iterations of optimization and simulation tasks on the same tMQL model whose components map the entities of the manufacturing processes. In order to compare IHOS with other metaheuristic algorithms, the tMQL model for the sand and cut manufacturing process from Section 2 was encoded into the deterministic optimization solver of Optimizing Programming Language (OPL) [15]. For the simulation part, the calculation of the metrics for each period was coded in Java. The metrics in the simulation problem were subject to the same bounds as those set in the optimization problem. For the comparison algorithms, we used the jMetal package, which is an object-oriented Java-based framework for multi-objective optimization with metaheuristics [13]. The algorithms chosen for comparison were the Nondominated Sorting Genetic Algorithm 2 (NGSA2) [27], Indicator Based Evolutionary Algorithm (IBEA) [28], Strength Pareto Evolutionary Algorithm 2 (SPEA2) [29], and Fast Pareto Genetic algorithm (FastPGA) [30]. All these algorithms decide the throughput variables and call the same simulation model of the sand and cut manufacturing process multiple times as a black box. The simulation model computes the estimated average costs and the confidence of the demand satisfaction, just as in the IHOS algorithm. This cost and demand satisfaction information is then used by the jMetal algorithms to further increase or decrease the throughput variable using heuristics. The jMetal package is in core Java and hence it was run with the same JVM as the IHOS algorithm. All the experiments were conducted on a 2.3 GHz quad-core machine with 4 GB of RAM that ran the Ubuntu Linux operating system.

The simulations for IHOS were run with the same bounds as the constraints set in the optimization model in OPL. First, the IHOS algorithm was run with two different settings. The first setting considered 100 candidates collected from phase 1 of IHOS (noCands = 100) and the second setting considered 500 candidates. Table 1 provides the input parameters to the algorithms described in Section 3 and the bounds used for experimentation of IHOS in these two settings. If the parameter is specifically used in phase 1 or phase 2 of the algorithm, then this is shown as a subscript on the parameters in the table. The total demand (*actualDemand*) from the sand and cut manufacturing process was set to be 50 sanded and cut wood and the cumulative sum of the demand required to be produced for each time period is shown in Table 2. The comparison algorithms were run with the same (relevant) parameters from Table 1 and the cumulative *actualDemand* was set to be same as the IHOS algorithm (Table 2). The cost function that computes the *totalCost* for each simulation was set to be a piecewise linear function with the stochastic throughput variable as its parameter such that *sand*1 and *cut*1 were the faster

Algorithm 4: ExtendedOCBA

Input : storedRuns, actualDemand, probabilityBound, finalConfidence, minCost, candSimulations, budgetDelta, budgetThreshold, maxExtraSimulationIterations, refuteConfidence, lastTP
Output: minCost

budget := 0
repeat
  sortedCost := Sort storedRuns.avgCost in ascending order
  foreach cand ∈ sortedCost **do**
    if `ConfidenceSatisfied` (confidencePerTP >= finalConfidence) **then**
      Let firstCandAvg be the cand's average cost and break out of the loop
    else
      Run a number of simulations on the cand such that either the confidence is satisfied or it is refuted. If confidence is refuted, then discard the candidate from storedRuns and sortedCost, otherwise store cand's average cost as firstCandAvg and break out of the loop
    end
  end
  secondCandAvg := Get the second lowest average cost from sortedCost
  foreach run ∈ storedRuns **do**
    noSimulations[run] := Allocate budgetDelta to each stored run proportional to $\frac{\sigma^2_{run}/candSimulations}{(b-mean_{run})^2}$ where:
    σ_{run} = standard deviation of candidate, run
    $mean_{run}$ = mean of candidate, run
    $candSimulations$ = Number of simulations performed to get $mean_{run}$ and σ_{run}
    $b = \frac{firstCandAvg+secondCandAvg}{2}$
    (demandSetProbs, avgCost) := `Simulate` (run.throughputExp, noSimulations[run], actualDemand, lastTP)
    candSimulations.add(noSimulations[run])
    budget := budget + noSimulations[run]
    foreach tp ← 1 **to** lastTP **do** `// lastTP is the last time point`
      confidencePerTP[tp] := `Confidence` (demandSetProb[tp] >= probabilityBound)
      `// refute probabilityBound is less than the actual probabilityBound by` ϵ`, e.g.  ` ϵ `= 0.15`
      refuteConfidencePerTP[tp] := `Confidence` (demandSetProb[tp] <= (probabilityBound - ϵ))
    end
    if `ConfidenceSatisfied` (confidencePerTP >= finalConfidence) **then**
      if avgCost < minCost **then**
        minCost := avgCost
      end
    else
      c := 1 `// c is a counter that counts the number of extra simulation iterations performed`
      while c <= maxExtraSimulationIterations **do**
        (demandSetProbs, avgCost) := `Simulate` (run.throughputExp, noSimulations[run]*c, actualDemand, lastTP)
        candSimulations.add(noSimulations[run]*c)
        budget := budget + noSimulations[run]*c
        foreach tp ← 1 **to** lastTP **do** `// lastTP is the last time point`
          confidencePerTP[tp] := `Confidence` (demandSetProb[tp] >= probabilityBound)
        end
        if `ConfidenceSatisfied` (confidencePerTP >= finalConfidence) **then**
          if avgCost < minCost **then**
            minCost := avgCost
          end
        else
          if `ConfidenceRefuted` (refuteConfidencePerTP >= refuteConfidence) **then**
            Remove candidate from the storedRuns
          end
        end
        c := c+1
      end
    end
  end
until budget > budgetThreshold
return minCost

Table 1: Overview of input parameters

Variable name	noCands	Value
$storeSize_{phase1}$	100	100
$storeSize_{phase1}$	500	500
$totalIterations_{phase1}$	100	1000
$totalIterations_{phase1}$	500	1500
$\sigma_{sand1}(noise)$	100/500	1.4
$\sigma_{sand2}(noise)$	100/500	0.9
$\sigma_{cut1}(noise)$	100/500	1.6
$\sigma_{cut2}(noise)$	100/500	1.0
$noSimulations$	100/500	100
$lastTP$	100/500	20
$probabilityBound$	100/500	0.95
$iterationConfidence_{phase1}$	100/500	0.7
$finalConfidence$	100/500	0.95
$refuteConfidence$	100/500	0.8
$maxExtraSimulationIterations$	100/500	5
$\lambda_{DemandAdjust}$	100/500	0.5
$budgetDelta_{phase2}$	100/500	1000
$budgetThreshold_{phase2}$	100	2000000
$budgetThreshold_{phase2}$	500	4000000

Table 2: Cumulative *actualDemand* values for all time periods

period	Cumulative demand
1	0
2	0
3	1
4	2
5	3
6	5
7	7
8	9
9	12
10	15
11	18
12	21
13	24
14	27
15	30
16	33
17	37
18	41
19	45
20	50

machines and *sand*2 and *cut*2 were the slower machines.

The data collected from the experiments included the estimated average costs achieved at different elapsed time points performed for IHOS with the two settings and all the comparison algorithms. Figure 2 shows the estimated average costs achieved at different elapsed time points including 95% confidence bars around the mean at each elapsed time point. It can be observed that both settings of IHOS perform better than the comparison algorithms initially. The reason for this is that IHOS uses deterministic optimization to set the mean throughputs of the machines whereas the other algorithms start at a random point in the search space. As time progresses, the average cost estimated by the IHOS algorithm plateaus (between 8 and 512 sec) and then the average cost further decreases after this point. We think that during this time, the iterative candidate collection phase of IHOS (phase 1) has successfully found the minimum cost among the collected candidates and after the plateau, the extended OCBA algorithm (phase 2) is able to decrease the cost further. In comparison, the other algorithms progressively decrease the estimated average costs and after some point they all plateau. As shown in the figure, the average cost found by the IHOS algorithm was 20% better than the nearest comparison algorithm (IBEA) when the experiment ended. After this, the IHOS algorithm was run without the extended OCBA where the *budgetDelta* was equally distributed among the candidates in phase 2. The IHOS algorithm without the extended OCBA was then compared to the one with extended OCBA as shown in Figure 2. The IHOS setting of noCands = 500 was used for this comparison. It is clear from the figure, that by using extended OCBA, the algorithm is able to converge faster and at the end of the experiment, IHOS with extended OCBA is able to achieve a better estimate of average cost by about 8%. We also ran the Tukey-Kramer procedure for pairwise comparisons of correlated means on all six algorithm types for each time point. By doing so, we confirmed that the IHOS algorithm was indeed better than the other algorithms when the experiment ended.

The IHOS algorithm was also performed for a longer period of time and it was compared with the comparison algorithms run for the same time period. In this case, the number of candidates collected from phase 1 of the IHOS algorithm were 5000 (noCands = 5000). The parameters from Table 1 that were changed for noCands = 5000 were: (a)$storeSize_{phase1}$ = 5000; (b)$totalIterations_{phase1}$ = 8000; and (c)$budgetThreshold_{phase2}$ = 10000000. The IHOS algorithm and the four comparison algorithms were run with the same *actualDemand* values shown in Table 2. Figure 3 shows the estimated average costs achieved at different elapsed time points for this scenario. It can be seen that after about 2500 seconds, all the comparison algorithms stop improving. On the other hand, the IHOS algorithm continues to improve beyond this time point and gives a 16% improvement in the estimated average cost at the end of two hours.

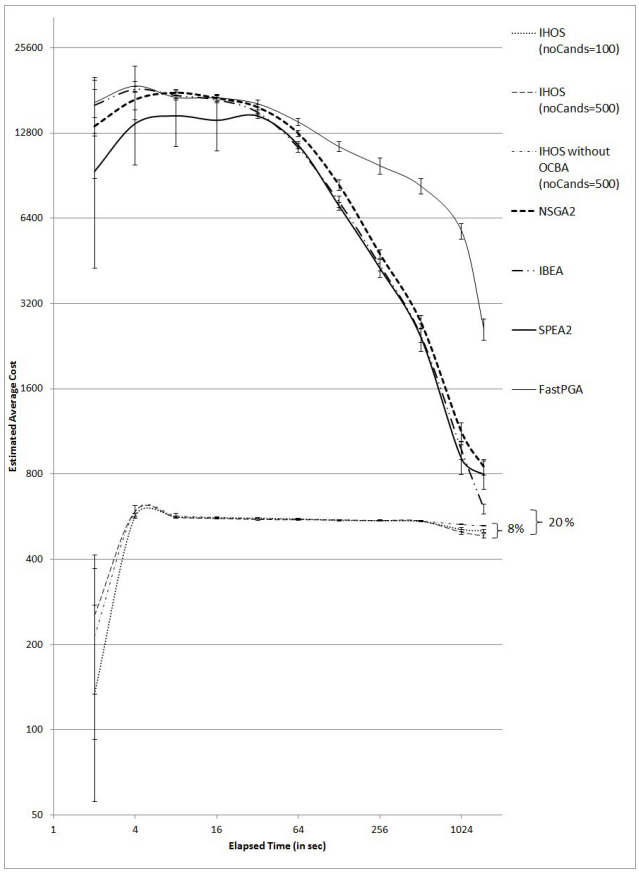

Figure 2: Estimated average cost for the elapsed time (max runtime = 25 minutes) (both axis in log-scale)

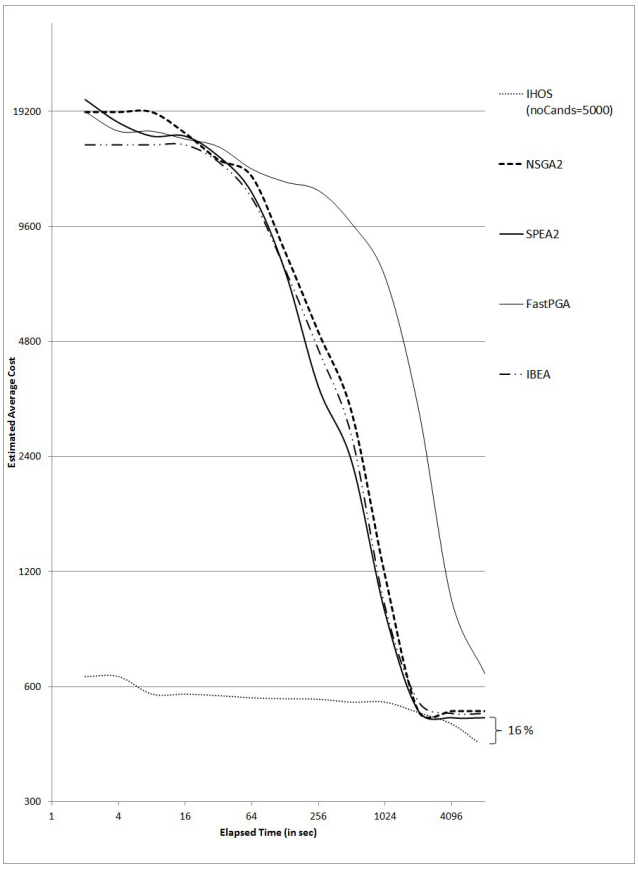

Figure 3: Estimated average cost for the elapsed time (max runtime = 120 minutes) (both axis in log-scale)

5 Conclusion

This paper demonstrated an iterative procedure to deal with stochastic variables in the manufacturing process. The IHOS algorithm iteratively performs deterministic optimization and simulation on the manufacturing process tMQL models to collect the most promising candidates of machine throughputs that will yield the minimum cost and satisfy customer demands in a stochastic environment. The idea of generating promising candidates from an approximated solutions is very central to the SimQL query language [33][34], [35][32]. However, SimQL algorithm extracts approximations by regression analysis, whereas IHOS extracts approximations directly from the stochastic model. The solution of the deterministic problem gives the throughputs of the machines such that the cost is minimized and the demand is satisfied. The stochastic simulation will then confirm that these throughputs yield the same results in a stochastic environment and if not, a simple heuristic is applied on the demand so as to get better results in the next iteration. The IHOS algorithm then allocates the computing budget of simulations in a smart way with the help of the extended OCBA algorithm to select the top candidate with the best cost and confidence of demand satisfaction.

This paper also compares the IHOS algorithm with four metaheuristic multi objective optimization algorithms. The IHOS algorithm is initially run in two settings with the number of candidates collected in each setting equal to 100 and 500, respectively. Then, the IHOS algorithm is run for a longer time with 5,000 candidates collected. The results show that the IHOS algorithm not only converges faster, but also outperforms all other algorithms with regards to the objective cost computed. This is the case because solving the deterministic optimization problem is very strong when optimality of results and computational complexity are a priority. Also, the allocation of budget performed with the help of the extended OCBA algorithm further helps to improve the cost. This is especially evident from the experiment where the original IHOS algorithm is compared with the case where instead of using the extended OCBA, the budget is equally distributed among the candidates. In addition, the IHOS algorithm runs the optimization and simulation as a white box with the help of the tQML models. These models are reusable, flexible and expressive of the manufacturing process components. Using these models greatly increases the opportunities presented to the IHOS algorithm to be more aware of the problem. This results in a more directed approach of

finding the minimum cost in the search space.

We are currently investigating: (a) dynamically executing phases 1 and 2 of the IHOS algorithm to improve the exploration of the search space; (b) extending the IHOS algorithm for multiple objectives; (c) adding stronger heuristics to IHOS; (d) incorporating sustainability metric functions into IHOS; (e) developing graphical user interfaces for modeling and querying with IHOS; and (f) developing specialized algorithms that can utilize preprocessing of stored (and therefore, static) components to speed up optimization, generalizing the results in [36][37][38].

Acknowledgements

This work is partially supported by NIST grant No. 70NANB12H277.

Appendices

A OPL-like formulation of the one-stage stochastic optimization problem

```
/* ============= Global computation =============*/
int noPeriods = ...;
int lastTP = noPeriods;
range timeRange = 0..lastTP;
float periodLength = ...;
float epsilon = 0.000001;
float probabilityBround = ...;

tuple idFlowPair {
   string id;
   string flow;
}

 /* ============= itemFlow Computation ============= */
{string} itemFlowIds = ...;
string itemFlow_mt[itemFlowIds] = ...;
dvar int+ itemFlow_tpAlloc[itemFlowIds][0..lastTP];
dvar int+ itemFlow_periodQty[itemFlowIds][1..noPeriods];

 /* ============= IA Computation ============= */
{string} IAids = ...;
string IA_mt[IAids] = ...;
{string}  IA_I[IAids] = ...;
{string}  IA_O[IAids] = ...;
dvar int+ IA_totalQty[IAids];
int IA_invQty[IAids][0..lastTP];
int IA_initInv[IAids] = ...;
int IA_capacity[IAids] =...;
{idFlowPair} IA_IdInputFlowPairs = { <id,i>    | id in IAids, i in IA_I[id] };
{idFlowPair} IA_IdOutputFlowPairs = { <id,i>    | id in IAids, i in IA_O[id] };
float IA_inAllocRatio[IA_IdInputFlowPairs] = ...;
float IA_outAllocRatio[IA_IdOutputFlowPairs] = ...;

forall(id in IAids){
    dexpr IA_invQty[id][0] = IA_initInv[id];
    forall(t in 1..lastTP){
        dexpr IA_invQty[id][t] = IA_invQty[id][t-1] +
            sum(i in IA_I[id]) itemFlow_periodQty[i][t] -
            sum(o in IA_O[id]) itemFlow_periodQty[o][t];
    }
    forall(t in 0..lastTP){
        forall(i in IA_I[id]){
            dexpr itemFlow_tpAlloc[i][t] =
               IA_inAllocRatio[<id,i>]*(IA_totalQty[id] - IA_invQty[id][t]);
        }
        forall(o in IA_O[id]){
            dexpr itemFlow_tpAlloc[o][t] =
                IA_outAllocRatio[<id,o>]*(IA_invQty[id][t]);
        }
    }
}
```

```
/* ============= IQA Computation ============= */
{string} IQAids = ...;
string IQA_mt[IQAids] = ...;
string  IQA_I = ...;
{string}  IQA_O[IQAids] = ...;
{idFlowPair} IQA_IdOutputFlowPairs = { <id,o> | id in IQAids, o in IQA_O[id] };
float IQA_outAllocRatio[IQA_IdOutputFlowPairs] = ...;
int IQA_totalTPAlloc[IQAids][0..lastTP];

forall(id in IQAids){
    forall(t in 0..lastTP){
        dexpr IQA_totalTPAlloc[id][t] = itemFlow_tpAlloc[IQA_I][t];
    }
    forall(t in 0..lastTP, o in IQA_O[id]){
        dexpr itemFlow_tpAlloc[o][t] =
            IQA_outAllocRatio[<id,o>]*IQA_totalTPAlloc[id][t];
    }
    forall(p in 1..noPeriods){
        dexpr itemFlow_periodQty[IQA_I][p]) =
            sum (o in IQA_O[id])(itemFlow_periodQty[o][p]);
    }
}

 /* ============= OQA Computation ============= */
{string} OQAids = ...;
string OQA_mt[OQAids] = ...;
{string}  OQA_I[OQAids] = ...;
string  OQA_O = ...;
{idFlowPair} OQA_IdInputFlowPairs = { <id,i>    | id in OQAids, i in OQA_I[id] };
float OQA_inAllocRatio[OQA_IdInputFlowPairs] = ...;
int OQA_totalTPAlloc[OQAids][0..lastTP];

forall(id in OQAids){
     forall(t in 0..lastTP){
         dexpr OQA_totalTPAlloc[id][t] = itemFlow_tpAlloc[OQA_O][t];
    }
    forall(t in 0..lastTP, i in OQA_I[id]){
        dexpr itemFlow_tpAlloc[i][t] =
            OQA_inAllocRatio[<id,i>] *(OQA_totalTPAlloc[id][t]);
    }
    forall(p in 1..noPeriods){
        dexpr itemFlow_periodQty[OQA_O][p] =
            sum (i in OQA_I[id])(itemFlow_periodQty[i][p]) ==
    }
}

 /* ============= baseProcess Computation ============= */
float machineSigma = ...;
{string} GBPids = ...;
{string} GBP_I[GBPids] =...;
{string} GBP_O[GBPids] = ...;
string GBP_o[gbpid in GBPids] = first(GBP_O[gbpid]);
dvar float+ GBP_throughputExp[GBPids][1..noPeriods];
float GBP_throughputControl[GBPids][1..noPeriods];
float GBP_capacity[GBPids] = ...;
{idFlowPair} GBP_IdInputFlowPairs = { <id,i>    | id in GBPids, i in GBP_I[id] };
```

```
int GBP_inputPerOutput[GBP_IdInputFlowPairs] = ...;
float GBP_accumAmt[GBPids][1..noPeriods];
float GBP_leftOver[GBPids][0..lastTP];
float GBP_slope1[GBPids] = ...;
float GBP_slope2[GBPids] = ...;
pwlFunction costFunction[id in GBPids] =
   piecewise{GBP_slope1[id]->10; GBP_slope2[id] -> 40; 7.5}(0.5,100);
dexpr float GBP_cost[id in GBPids][p in 1..noPeriods] =
   costFunction[id](GBP_throughputControl[id][p]);
dexpr float GBP_totalCost[id in GBPids] = sum(p in 1..noPeriods) GBP_cost[id][p];

forall(id in GBPids){
    forall(p in 1..noPeriods){
        dexpr GBP_throughputControl[id][p] =
            GBP_throughputExp[id][p] + Gaussian({exp: 0.0, sigma:machineSigma});
        dexpr GBP_accumAmt[id][p] = GBP_leftOver[id][p-1] +
            GBP_throughputControl[id][p] * periodLength;
        dexpr GBP_leftOver[id][p] = GBP_accumAmt[id][p] -
            itemFlow_periodQty[GBP_o[id]][p];
        forall(i in GBP_I[id]){
            dexpr itemFlow_periodQty[i][p] = itemFlow_periodQty[GBP_o[id]][p] *
                GBP_inputPerOutput[<id,i>];
        }
    }
}

/* ============= CompositeProcess Computation ============= */
{string} outputId = ...;
int demand[1..noPeriods] = ...;
dexpr float totalCost = sum(id in GBPids) GBP_totalCost[id];

minimize E(totalCost);
constraints{
    /* ============= itemFlow Constraints ============= */
    forall(id in itemFlowIds){
        forall(p in 1..noPeriods){
            Prob(itemFlow_periodQty[id][p] <= itemFlow_tpAlloc[id][p-1])>=
                probabilityBound;
        }
    }

    /* ============= IA Constraints ============= */
    forall(id in IAids){
        Prob(IA_totalQty[id] <= IA_capacity[id]) >= probabilityBround;
        sum (p in IA_IdInputFlowPairs : p.id == id) IA_inAllocRatio[p] == 1;
        sum (p in IA_IdOutputFlowPairs : p.id == id) IA_outAllocRatio[p] == 1;
     }

     /* ============= IQA Constraints ============= */
    forall(id in IQAids){
        sum (p in IQA_IdOutputFlowPairs : p.id == id) IQA_outAllocRatio[p] == 1;
    }

    /* ============= OQA Constraints ============= */
    forall(id in OQAids){
        sum (p in OQA_IdInputFlowPairs : p.id == id) OQA_inAllocRatio[p] == 1;
    }
```

```
    /* ============= baseProcess Constraints ============= */
    forall(id in GBPids){
        GBP_leftOver[id][0] == 0.0;
        forall(p in 1..noPeriods){
            Prob(GBP_throughputControl[id][p] <= GBP_capacity[id]) >= probabilitBound;
            Prob(itemFlow_periodQty[GBP_o[id]][p] <= GBP_accumAmt[id][p]) >=
                probabilityBound;
    }

    /* ============= CompositeProcess Constraints ============= */
    /*cumilitive demand constraint*/
        forall(o in outputId, period in 1..noPeriods){
            Prob(sum(p in 1..period) itemFlow_periodQty[o][p]>=demand[period])>=
                probabilityBound;
    }
}
```

References

[1] M. C. Fu, F. W. Glover, and J. April, "Simulation optimization: a review, new developments, and applications," in *Proceedings of the Winter Simulation Conference, 2005*, pp. 13 – 95, Dec 2005.

[2] M. C. Fu, C. H. Chen, and L. Shi, "Some topics for simulation optimization," in *Proceedings of the Winter Simulation Conference, 2008*, pp. 27–38, 2008.

[3] R. V. Rao, *Advanced Modeling and Optimization of Manufacturing Processes*. Springer Series in Advanced Manufacturing, Springer, London, 2011.

[4] S. Melouk, N. Freeman, D. Miller, and M. Dunning, "Simulation optimization-based decision support tool for steel manufacturing," *International Journal of Production Economics*, vol. 141, pp. 269–276, Jan. 2013.

[5] P. Raska and Z. Ulrych, "Simulation optimization in manufacturing systems," *Annals of DAAAM for 2012 & Proceedings of the 23rd International DAAAM Symposium*, vol. 23, no. 1, pp. 221 – 224, 2012.

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

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

[8] J. Åkesson, M. Gäfvert, and T. Tummescheit, "Jmodelica-an open source platform for optimization of modelica models," in *Proceedings of 6th International Conference on Mathematical Modeling (MATHMOD)*, 2009.

[9] D. Brück, H. Elmqvist, S. E. Mattsson, and H. Olsson, "Dymola for multi-engineering modeling and simulation," in *Proceedings of Modelica*, 2002.

[10] J. Hřebíček and M. Řezáč, "Modelling with maple and maplesim," in *Proceedings of the 22nd European Conference on Modelling and Simulation, 2008*, pp. 60–66, 2008.

[11] 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.

[12] OpenModelica, *Efficient Traceable Model-Based Dynamic Optimization - EDOp*, 2009 (accessed September 12, 2014).

[13] 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.

[14] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: a modeling language for mathematical programming*. Cengage Learning, 2002.

[15] V. H. Pascal, *The OPL optimization programming language*. Cambridge, MA, USA: MIT Press, 1999.

[16] R. F. Boisvert, S. E. Howe, and D. K. Kahaner, "Gams: A framework for the management of scientific software," *ACM transactions on mathematical software*, vol. 11, no. 4, pp. 313–355, 1985.

[17] J. Bisschop and R. Entriken, *AIMMS: The modeling system*. Paragon Decision Technology, 1993.

[18] M. Krishnamoorthy, A. Brodsky, and D. A. Menascé, "Temporal manufacturing query language (tMQL) for domain specific composition, what-if analysis, and optimization of manufacturing processes with inventories," Tech. Rep. GMU-CS-TR-2014-3, Department of Computer Science, George Mason University, Fairfax, VA, USA, 2014.

[19] A. Brodsky, G. Shao, and F. Riddick, "Process analytics formalism for decision guidance in sustainable manufacturing," *Journal of Intelligent Manufacturing*, pp. 1–20, 2013.

[20] G. Shao, A. Brodsky, S. J. Shin, and D. Kim, "Decision guidance methodology for sustainable manufacturing using process analytics formalism," *Journal of Intelligent Manufacturing*, pp. 1–18, 2014.

[21] A. Brodsky, N. E. Egge, and X. S. Wang, "Supporting agile organizations with a decision guidance query language," *J. Management Information Systems*, vol. 28, no. 4, pp. 39–68, 2012.

[22] A. Brodsky, M. M. Bhot, M. Chandrashekar, N. E. Egge, and X. S. Wang, "A decisions query language (DQL): High-level abstraction for mathematical programming over databases," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, (New York, NY, USA), pp. 1059–1062, ACM, 2009.

[23] A. Brodsky, "Constraint databases: Promising technology or just intellectual exercise?," *Constraints J.*, vol. 2, no. 1, pp. 35–44, 1997.

[24] A. Brodsky and V. E. Segal, "The C3 constraint object-oriented database system: An overview," in *Second International Workshop on Constraint Database Systems, Constraint Databases and Their Applications*, (London, UK, UK), pp. 134–159, Springer-Verlag, 1997.

[25] A. Brodsky and H. Nash, "Cojava: Optimization modeling by nondeterministic simulation," in *Principles and Practice of Constraint Programming-CP 2006*, pp. 91–106, Springer, 2006.

[26] A. Brodsky and H. Nash, "CoJava: A unified language for simulation and optimization," in *Principles and Practice of Constraint Programming - CP 2005* (P. van Beek, ed.), vol. 3709 of *Lecture Notes in Computer Science*, pp. 877–877, Springer Berlin Heidelberg, 2005.

[27] 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.

[28] 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.

[29] 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.

[30] H. Eskandari, C. D. Geiger, and G. B. Lamont, "FastPGA: A dynamic population sizing approach for solving expensive multiobjective optimization problems," in *4th International Conference on Evolutionary Multi-Criterion Optimization, 2007*, vol. 4403 of *Lecture Notes in Computer Science*, pp. 141–155, Springer, 2007.

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

[32] S. Farley, A. Brodsky, and C. H. Chen, "A regression dependent iterative algorithm for optimizing top-k selection in simulation query language," *International Journal of Decision Support System Technology*, vol. 4, pp. 12–24, July 2012.

[33] S. Farley, A. Brodsky, and C. H. Chen, "A simulation query language for defining and analyzing uncertain data," in *The 15th International Association of Science and Technology for Development Conference on Software Engineering and Applications*, 2011.

[34] S. Farley, A. Brodsky, N. Egge, and J. McDowall, "SimQL: Simulation-based decision modeling over stochastic databases," in *The 15th International Federation for Information Processing Conference WG8*, 2010.

[35] S. Farley, A. Brodsky, and C. H. Chen, "Regression based algorithm for optimizing top-k selection in simulation query language," *30th IEEE International Conference on Data Engineering*, vol. 1, pp. 103–110, 2012.

[36] 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.

[37] 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.

[38] 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.