# Modular Modeling and Optimization of Temporal Manufacturing Processes with Inventories

Mohan Krishnamoorthy, Alexander Brodsky, Daniel A. Menascé
*Department of Computer Science, George Mason University, Fairfax VA, USA, {mkrishn4,brodsky,menasce}@gmu.edu*

***Abstract*—Smart manufacturing requires streamlining operations and optimizing processes at a global and local level. This paper considers temporal manufacturing processes that involve physical or virtual inventories of products, parts and materials that move through a network of subprocesses. The inventory levels vary with time and are a function of the configuration settings of the machines involved in the process. These environments require analysis, e.g., answering what-if questions, and optimization to determine optimal operating settings for the entire process. To address this problem, the paper proposes modular process components that can represent these manufacturing environments at various levels of granularity for performing what-if analysis and decision optimization queries. These components are extensible and reusable against which optimization and what-if questions can be posed. Additionally, the paper describes the steps to translate these complex components and optimization queries into a formal mathematical programming model, which is then solved by a commercial optimization solver.**

***Keywords*-smart manufacturing; decision support; decision guidance; deterministic optimization; decision optimization**

## I. Introduction

Due to increased global competition, manufacturing companies look toward ways to reduce their cost and increase efficiency of operations. To address this need, the NSF Smart Process Manufacturing Operations and Technology Roadmap Full Report [1] envisions that *"Interoperable systems will act with and from the data to support modeling tools and decision support systems that create the right information to ensure safe, optimum process design and execution. Operations will be closed-loop, with automatic sensing of all critical parameters, and will operate under distributed control, ensuring that all parameters that affect the business of the enterprise are maintained within control limits."*. However, the report by the Economist Intelligence Unit [2] found that *"Despite many manufacturers reporting impressive savings from data analysis, 62% are not sure they have been able to keep up with the large volumes of data they collect, and just 50% are sure they can generate useful insights from it, as it comes from too many sources and in a variety of formats and speeds."*.

To generate useful insights from large volumes of collected data, there is a greater need for model-based Decision Support Systems (DSS) involving what-if analysis and optimization while taking into account sustainability metrics. As noted in [3], model-based decision support in manufacturing face challenges such as domain and computational integration issues, taking into account the autonomy of the decision-making entities in face of information asymmetry, modeling the preferences of the decision makers, and efficiently determining robust solutions.

There has been extensive research of model-based DSS of manufacturing and supply chain processes. The approaches for building model-based DSS for what-if analysis and optimization can be broadly classified into three categories: (1) customized domain-specific solutions for optimization of manufacturing processes, (2) simulation-based systems, and (3) optimization solvers and modeling languages based on mathematical programming (MP) and constraint programming (CP). Customized domain-specific optimization solutions, e.g., [4], [5], [6], are easy to use for the end users and amenable to integration with a graphical-user interface. However, they are designed for specific limited settings of manufacturing processes and may not be extensible. Simulation-based systems, e.g., [7], [8], are extensible and reusable, but their solutions are significantly inferior to the optimization solutions and also computationally expensive. Optimization solvers and modeling languages based on MP/CP, e.g., [9], [10], are efficient in providing near-optimal results but they are more difficult to model, modify, and reuse [11]. These models may also require an OR expert to model the problem. Process optimization application, e.g., [12], [13], provide optimization models to solve specific manufacturing problems, but it is difficult to reuse, extend, and modify these models since this approach uses a low level of abstraction of the problem. The related work is described in greater detail in Section II.

This paper focuses on addressing the outlined technical limitation in the context of manufacturing process typical in discrete manufacturing. Here, we focus on processes that involve physical or virtual inventories of products, parts and materials that are used to anticipate uncertainties on supply or throughputs. These inventories are called work-in-progress inventories. Over time, the state of the inventories and processes changes until completion. We use the term Buffered Temporal Flow Processes (BTFP) to describe them. The terms BTF processes and BTFP are used interchangeably in this paper. According to the product process matrix [14], a process structure may be classified into jumbled flow processes, disconnected line flow or batch processes, connected line flow or assembly processes, and continuous flow processes. The BTFP formalism presented in this paper can be used to model connected line flow or assembly processes that involve inventories whose state varies at discrete time intervals. Such processes can be found in many different areas of manufacturing and supply chain. A particularly important BTF process is in the area of discrete manufacturing such as in automotive assembly floor, furniture assembly line, smartphone manufacturing, and aircraft assembly line.

For such a process, a process engineer may want to perform a variety of what-if analysis and optimization tasks. As an example of a what-if analysis question, a process engineer may ask: given a particular planned process throughput, and load-distribution among the processes, if one of the processes has to stop working or the speed of a highly energy efficient process has to be increased, what would be production output, 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 processes should be on and off, and how to set-up controls of every operational process, and distribute load among the processes, 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? Given the diversity of manufacturing processes, it is highly desirable to have a system that allows the flexible specification of manufacturing processes, and is able to answer declarative analysis and optimization queries to end users.

To support what-if analysis and optimization of BTF processes, there is a need to accurately model systems and processes. These models need to capture (a) metrics of processes (such as cost, energy consumption, and emission) as a function of control variables, (b) process routing that describes the flow of materials thorough the manufacturing floor, and (c) work-in-progress for inventories. In BTF processes, this needs to be modeled over a temporal sequence and include stochasticity of throughput and supply.

In order to enable a process engineer to model BTF processes and perform what-if analysis and optimization on them, in this paper, we propose a mathematical model that encapsulates the atomic process or the manufacturing floor as a whole. We call such models base BTF process and composite BTF process, respectively. We argue that these BTF processes can be used for modular composition of BTFP-like manufacturing floors to enable efficient formulation and solution of what-if analysis and optimization problems. More specifically, the contributions of this paper are:

- A formal BTFP framework, including syntax and semantics of the base and hierarchically composed processes, metrics, feasibility and optimality. The composed BTFP processes would naturally allow a process engineer to store and reuse process definition models in a process model repository.
- A graphical notation for hierarchical composition of BTF processes, which is easy to use by production engineers and can be automatically translated to the formal BTFP representation and show it on a manufacturing process example.
- Implementation of BTFP optimization by developing an MILP model using OPL.

Please note that this paper only provides the formulation for deterministic BTFP optimization. For the stochastic variant, we direct the reader to [15].

The rest of this paper is organized as follows. Section II discusses related work and III gives a motivating example for the BTF processes along with its graphical notation. Section IV formally defines base and composite BTF process. Section V provides the syntax and semantics for the BTFP optimization and shows the formulation and solution of BTFP optimization on a simple example. Section VI shows the implementation of BTFP optimization in OPL and, finally, section VII concludes and discusses future work.

## II. Related Work

The related approaches for building model-based DSS for what-if analysis and optimization is examined in greater detail in this section. As outlined in the previous section, this related work can be broadly classified into three categories: (1) customized domain-specific solutions for optimization of manufacturing processes, (2) simulation-based systems, and (3) optimization solvers and modeling languages based on MP/CP. Customized domain-specific solutions for BTF processes are designed for specific, limited setting of a manufacturing process, and would typically provide a graphical user interface that is easy to use by end users. Examples include [4], [5], [6]. 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 (1) typically not extensible to additional aspects of processes and metrics, and (2) perform a "silo" optimization, which would not achieve optimality if an extended underlying system needs to be optimized.

Simulation-based systems allow to accurately model a system and its inner workings. They are usually object-oriented, modular, extensible, and reusable. Furthermore, many simulation tools provide an easy-to-use graphical user interface. Tools like SIMULINK [7] and Modelica-based ones [8] allow users to model complex systems in mechanical, hydraulic, thermal, control, and electrical power. However, simulation-based optimization is significantly inferior to optimization solutions based on MP/CP in terms of optimality of results and computational complexity. This is because simulation-based optimization amounts to a heuristically-guided trial and error search, which does not utilize the mathematical structure of the underlying problem the way MP/CP methods do.

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 AMPL [9], OPL [10], or GAMS [16]. However, MP and CP modeling 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.

Significant research as also been conducted on manufacturing process optimization. One such approach describes how to build models for real-world scenarios and solve optimization problems on these models based on MILP [17]. In [18], the authors propose an optimization approach based on the ordinal optimization technique and particle swarm optimization to find the operational variables while minimizing cost. The research in [19] studies simultaneous optimization of workload and buffer allocations. The work in [12] and [13] discusses optimization for water bottle and mold manufacturing processes respectively. Additionally, there has been extensive research on analysis and optimization of BTFP-like processes (e.g., see [20], [19], [21] for overview). Although process optimization has been widely researched, unlike our approach, these techniques do not generate solutions for general, composable BTFP. Also, these techniques are either hardwired to a specific problem or they provide models with a low level of abstraction that cannot be reused or modified easily by the process engineer. To further motivate the need for a general, composable BTFP framework, an example BTF process is given in the next section.

## III. Motivating example

This section introduces an example BTFP problem to describe the challenges motivating modeling and optimization

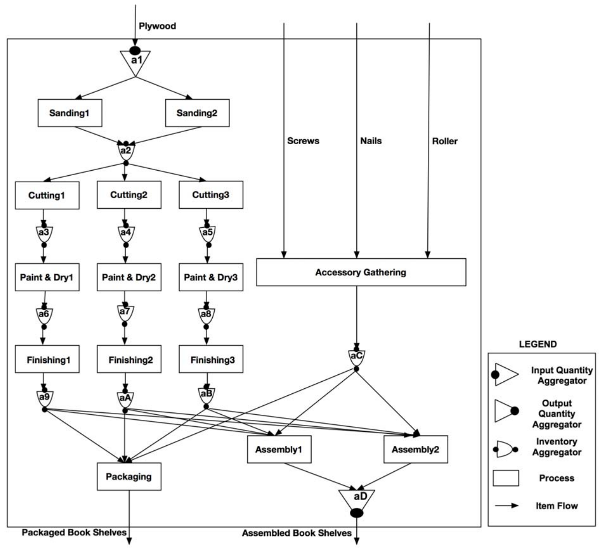

Figure 1. A graphical notation for the bookshelf manufacturing floor

of such manufacturing processes. The example consists of a simplified bookshelf packaging and assembly system. A graphical notation of its process interaction is shown in Figure 1. The processes, shown as rectangular boxes, are interleaved with buffers used to store and/or distribute the items produced by one process to another. Raw materials for the bookshelf manufacturing floor consist of plywood logs and accessories such as nails, screws, and rollers. The plywood goes first through two sanding processes where the plywood logs entering the floor are sanded to smoothen the wood. Next, the left plank, right plank and the shelves are cut. This stage involves the wood first undergoing the rough cutting of appropriate size from the log of plywood, precise cutting of the edges, and sanding of the edges. After this, the left plank, right plank and shelves are painted and dried. After the drying is complete, a finishing touch is given to the left plank, right plank, and the shelves. This involves drilling holes and cleaning the respective pieces of wood so that they are ready for use. In parallel, the accessories are gathered. Then, the finished left plank, right plank, shelves and the accessories are either packaged in a box to be sent to a warehouse or they are assembled into a bookshelf to be put on display.

A system could include controllable and non-controllable parameters from all the processes. For instance, a controllable parameter could be the speed of the sanding process. A processes outputs parts to be used by a succeeding process, which may not be able to immediately work on these parts. Hence, all processes are connected by physical *buffers* that can hold parts until the succeeding process(es) are ready to take these parts as input. These buffers are constrained by their capacity. Also, there may be multiple processes working at different speeds and efficiencies, at the same stage of production. Finally, the flow of items between the processes and buffers is bounded by the number of items that the process can handle and/or by the number of items that can be stored in the buffers.

The book-shelf manufacturing floor may require optimization to be performed in order to maximize the number of bookshelves packaged or minimize the energy consumed. In addition, some types of what-if analysis queries can also be performed in order to know what will be the increase in the energy consumption of the Sanding process if the speed of the process is increased by 15%? These kinds of queries require a composable, easy-to-use, and closed mathematical model that can easily be used in an optimization problem. The next two sections give details of such a mathematical model and explain how one could pose an optimization problem on top of such a model.

## IV. BTFP Formalism

In BTFP, the state of the inventories and the process changes until process completion. BTF processes can be used to model either atomic process (see $Sanding1$ process in Figure 1) or to model the entire manufacturing floor (like the whole bookshelf process shown in Figure 1). To do this, BTF processes need to capture the variables, constraints and metrics that capture the workings of the atomic processes and the manufacturing floor on a temporal sequence over the time horizon. 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 this section, we will provide the mathematical instance (syntax) of these BTF processes, i.e., we explain the variables, metrics and constraints contained in these processes.

The BTF process that captures the working of an atomic process is called base BTF process (see $Sanding1$ process in Figure 1). More formally, a base BTF process is defined as:

*Definition 1:* A base BTF process is a tuple of *bp = (I, out, capacity, inputPerOutput, throughputControl, accumAmt, leftOver, $\mathcal{M}$)* where:

1) $I$ is the set of input flows to the base BTF process. E.g., $I$ for $Sanding1$ is the flow (arrow) from $a1$ to $Sanding1$.
2) $out$ is the output flow from the base BTF process. E.g., $out$ for $Sanding1$ is the flow (arrow) from $Sanding1$ to $a2$.
3) $capacity \in \mathbb{R}^+$, which is the maximum throughput at all time points. E.g., $capacity = 23.12$ for $Sanding1$.
4) $inputPerOutput : I \rightarrow \mathbb{Z}^+$, i.e., inputPerOutput(i) gives the quantity of input i required per output produced at each time point $t \in \mathbb{T}$, where $\mathbb{T} = \{1, 2, \ldots, n\}$. E.g., $inputPerOutput(from\_aA) = 4$, i.e., $Packaging$ process (in Figure 1) requires 4 input shelves (from input flow $from\_aA$) for each packaged book shelf output
5) $throughputControl : \mathbb{T} \rightarrow \mathbb{R}^+$, i.e., throughputControl(t) gives throughput of the process at time point t. E.g., $throughputControl(2) = 2.85$ for $Sanding1$.
6) $accumAmt : \mathbb{T} \rightarrow \mathbb{R}^+$, i.e., accumAmt(t) gives the amount of items accumulated in the process at time point t. E.g., $accumAmt(2) = 3.12$ for $Sanding1$.
7) $leftOver : \mathbb{T} \rightarrow \mathbb{R}^+$, i.e., leftOver(t) gives the quantity of items left over in the process at time point t. E.g., $leftOver(2) = 0.5$ for $Sanding1$.
8) $\mathcal{M} : \mathbb{T} \times \mathbb{R}^+ \rightarrow D_1 \times \cdots \times D_k$, i.e., $\mathcal{M}$ gives the metrics $M_1(t, X)$ through $M_k(t, X)$ from their domains $D_1$ through $D_k$, where X is the $throughputControl$ variable at time point $t \in \mathbb{T}$. E.g., $D_1 = [0, \infty)$ for cost, $D_2 = [0, \infty)$ for power, and $\mathcal{M}(2, 2.85) = (110.0, 2.3)$ meaning that for $t = 2$ and $throughputControl = 2.85$, the cost is \$110.0 and the power required is 2.3 kW.

We say that a base BTF process is feasible if it satisfies the following constraints:

1) $capacityConstraint :$
$$\underset{t\in\mathbb{T}}{\forall}\ throughputControl[t] \leq capacity$$
2) $accumAmtConstraint : \underset{t\in\mathbb{T}}{\forall}\ accumAmt[t] =$
$$leftOver[t-1] + throughputControl[t]$$
3) $outputQtyConstraint :$
$$\underset{t\in\mathbb{T}}{\forall}\ periodQty_{out}[t] = \lfloor accumAmt[t] \rfloor$$
4) $leftOverConstraint :$
$$\underset{t\in\mathbb{T}}{\forall}\ leftOver[t] = accumAmt[t] - periodQty_{out}[t]$$
5) $inputQtyConstraint : \underset{t\in\mathbb{T}}{\forall}\ \underset{in\in I}{\forall}\ periodQty_{in}[t] =$
$$periodQty_{out}[t] \times inputPerOutput[in]$$

Each base BTF process contains a set of input flows ($I$) that bring in the input items and one output flow ($out$) that takes out the produced items. The $throughputControl$ signifies the speed of the base process and is a controllable variable. The $capacity$ variable is the maximum speed possible for the base process or may be constrained by some sustainability metric. The other variables from Definition 1 are self-explanatory. Each base BTF process contains a set of metrics ($\mathcal{M}$) that could include the cost, energy consumption, water usage, waste disposal etc. The metrics could be used as an indicator by the process operator to check how good or bad the process is performing. These metrics might be used as a monitoring, analysis or planning indicators. Although these metrics serve as a good local indicator of performance, they may also be added to the other metrics of the composite process to provide a global performance indicator of the manufacturing floor.

Each base BTF process contains a number of constraints that capture the actual constraints of the process as shown in Definition 1. The speed of the base process is constrained by the maximum speed that this process can be operated at as shown in the first constraint. The number of items accumulated in the process at each time period is the sum of any left over items from the previous time periods and the number of items produced during the current time period ($throughputControl$) as shown in the second constraint. The number of items produced by the process for a time period is the minimum amount of "finished" items that the output flow can carry as shown in the third constraint. The number of items left over at a time period is the difference between the number of items accumulated in the process so far and the number of items that left the process via the output flow as shown in the fourth constraint. The number of items the process requires at its input is the number of output items that need to be produced times the number of particular input required per output produced ($inputPerOutput$) as shown in the fifth constraint.

The BTF process that captures the working of the entire manufacturing floor or a part of it is called composite BTF process. The composite BTF process can contain other composite processes and/or base processes (collectively called as processes). Additionally, the composite BTF process may also contain aggregators and flows. The aggregators here may be the input quantity aggregator ($IQA$) that distributes the inputs of the composite BTF process among sub processes (see $a1$ in Figure 1) or inventory aggregator ($IA$) that stores and distributes the items from one set of sub processes to other (see $a2$ in Figure 1) or input quantity aggregator ($OQA$) that collect the items from multiple sub processes and dispenses it out of the composite BTF process (see $aD$ in Figure 1). The flows is called item flows ($IF$) that act as an interface between the sub processes and aggregators of the composite BTF process. More formally, the $IQA$, $IA$, $OQA$, $IF$, and composite BTF process can be defined as in Definitions 2 - 6 respectively.

*Definition 2:* The aggregator, $IQA$ is a tuple of *iqa* = ($\overrightarrow{I1}$, $\overrightarrow{O1}$, $\overrightarrow{outAllocRatio1}$, $\overrightarrow{totalTPAlloc1}$) where:

1) $\overrightarrow{I1} = \{I_i \subseteq$ *IF: set of input flows of* $IQA$ $i | i \in IQA\}$. E.g., $I_{a1}$ is the flow (arrow) to $a1$.
2) $\overrightarrow{O1} = \{O_i \subseteq$ *IF: set of output flows of* $IQA$ $i | i \in IQA\}$. E.g., $O_{a1}$ is the flow (arrow) from $a1$ to $Sanding1$ and $Sanding2$.
3) $\overrightarrow{outAllocRatio1} = \{outAllocRatio_i : O_i \to \mathbb{R}^+ | i \in IQA\}$, i.e., for IQA i, $outAllocRatio_i(out)$ gives the portion of $totalTPAlloc_i$ to be allocated to output $out$. E.g., $outAllocRatio_{a1}(toSanding1) = 0.4$ and $outAllocRatio_{a1}(toSanding2) = 0.6$.
4) $\overrightarrow{totalTPAlloc1} = \{totalTPAlloc_i : \mathbb{T} \to \mathbb{Z}^+ | i \in IQA\}$, i.e., $totalTPAlloc_i(t)$ is the sum of the allocation of inputs of $IQA$ i at time point t $\in \mathbb{T}$, where $\mathbb{T} = \{1, 2, \ldots, n\}$. E.g., $totalTPAlloc_{a1}(2) = 15$.

We say that an $IQA$ is feasible if it satisfies the following constraints:

1) $totalAllocConstraint$ :
$$\underset{i \in IQA}{\forall} \underset{t \in \mathbb{T}}{\forall} totalTPAlloc_i[t] = \sum_{in \in I_i} tpAlloc_{in}[t]$$
2) $outputAllocConstraint$ :
$$\underset{i \in IQA}{\forall} \underset{t \in \mathbb{T}}{\forall} \underset{out \in O_i}{\forall} tpAlloc_{out}[t] = outAllocRatio_i[out] \times totalTPAlloc_i[t]$$
3) $inputQtyConstraint$ : $\underset{i \in IQA}{\forall} \underset{t \in \mathbb{T}}{\forall}$
$$\left( \sum_{in \in I_i} periodQty_{in}[t] = \sum_{out \in O_i} periodQty_{out}[t] \right)$$

The mathematical representation of an $IQA$ is described in Definition 2. The $IQA$ takes all the items entering via its input flow ($\overrightarrow{I1}$) and distributes them among all its output flows ($\overrightarrow{O1}$). At each time point, the total allocation of the inputs of the $IQA$ is the sum of allocations on all its inputs as shown in the first constraint. At each time point, the output allocation on each output of the $IQA$ is the product of the ratio allocated to that output and the total allocation of the inputs as shown in the second constraint. Because the $IQA$ does not have storage, all incoming items into the $IQA$ need to be distributed among processes. Hence, at each time period, the number of outputs coming out of the $IQA$ should be the same as the number of inputs coming into the $IQA$ as shown in the third constraint.

*Definition 3:* The aggregator, $IA$ is a tuple of *ia* = ($\overrightarrow{I2}$, $\overrightarrow{O2}$, $\overrightarrow{capacity}$, $\overrightarrow{initInv}$, $\overrightarrow{inAllocRatio2}$, $\overrightarrow{outAllocRatio2}$, $\overrightarrow{invQty}$, $\overrightarrow{totalQty}$) where:

1) $\overrightarrow{I2} = \{I_i \subseteq$ *IF: set of input flows of* IA $i | i \in IA\}$. E.g., $I_{a2}$ is the flow (arrow) from $Sanding1$ and $Sanding2$ to $a2$.
2) $\overrightarrow{O2} = \{O_i \subseteq$ *IF: set of output flows of* IA $i | i \in IA\}$. E.g., $O_{a2}$ is the flow (arrow) from $a2$ to $Cutting1$ and $Cutting2$.
3) $\overrightarrow{capacity} = \{capacity_i : \in \mathbb{Z}^+ | i \in IA\}$, which is the maximum number of items allowed in $IA$ i at all time points. E.g., $capacity_{a2}$ = 30.
4) $\overrightarrow{initInv} = \{initInv_i \in \mathbb{Z}^+ | i \in IA\}$, which is the initial number of items present in $IA$ i. E.g., $initInv_{a2}$ = 1.
5) $\overrightarrow{inAllocRatio2} = \{inAllocRatio_i : I_i \to \mathbb{R}^+ | i \in IA\}$, i.e., $inAllocRatio_i(in)$ gives the portion of remaining space of $IA$ i to be allocated to input $in$. E.g., $inAllocRatio_{a2}(fromSanding1) = 0.45$ and $inAllocRatio_{a2}(fromSanding2) = 0.55$.
6) $\overrightarrow{outAllocRatio2} = \{outAllocRatio_i : O_i \to \mathbb{R}^+ | i \in IA\}$, i.e., $outAllocRatio_i(out)$ gives the portion of items of $IA$ i to be allocated to output $out$. E.g., $outAllocRatio_{a2}(toCutting1) = 0.3$, $outAllocRatio_{a2}(toCutting2) = 0.3$, and $outAllocRatio_{a2}(toCutting3) = 0.4$.
7) $\overrightarrow{invQty} = \{invQty_i : \mathbb{T} \to \mathbb{Z}^+ | i \in IA\}$, i.e., $invQty_i(t)$ is the number of items in $IA$ i at time point t $\in \mathbb{T}$, where $\mathbb{T} = \{1, 2, \ldots, n\}$. E.g., $invQty_{a2}(2) = 12$.
8) $\overrightarrow{totalQty} = \{totalQty_i \in \mathbb{Z}^+ | i \in IA\}$, which is the

actual maximum number items stored in $IA$ i across all time points. E.g., $totalQty_{a2}$ = 20.

We say that an $IA$ is feasible if it satisfies the following constraints:

1) $initInvConstraint: \underset{i \in IA}{\forall} invQty_i[0] = initInv_i$
2) $capacityConstraint: \underset{i \in IA}{\forall} totalQty_i \leq capacity_i$
3) $invQtyConstraint:$
$$\underset{i \in IA}{\forall} \underset{t \in \mathbb{T}}{\forall} invQty_i[t] = invQty_i[t-1] + \sum_{in \in I_i} periodQty_{in}[t] - \sum_{out \in O_i} periodQty_{out}[t]$$
4) $inputAllocConstraint:$
$$\underset{i \in IA}{\forall} \underset{t \in \mathbb{T}}{\forall} \underset{in \in I_i}{\forall} tpAlloc_{in}[t] = inAllocRatio_i[in] \times (totalQty_i - invQty_i[t])$$
5) $outputAllocConstraint:$
$$\underset{i \in IA}{\forall} \underset{t \in \mathbb{T}}{\forall} \underset{out \in O_i}{\forall} tpAlloc_{out}[t] = outAllocRatio_i[out] \times invQty_i[t]$$

The mathematical representation of an $IA$ is described in Definition 3. The $IA$ stores and distributes items coming in from processes via its input flows ($\overrightarrow{I2}$) and going out to processes via its output flows ($\overrightarrow{O2}$). The $IA$ may not require to use its space entirely because either the inputs are flowing slowly, there is a high demand on the output flows, or the demand is low. Hence the total space used during the time the $IA$ is active is captured by the $\overrightarrow{totalQty}$ variable. Initially, the number of items in the $IA$ is set via the $initInv$ variable as shown in the first constraint. As the $IA$ performs storage and distribution, the total number of items is upper bounded by the physical capacity of the $IA$ as shown in the second constraint. At each time point, the number of items in the $IA$ is the sum of the items remaining from the previous time intervals and the items that came in via the input flows but not the items that were dispensed via its output flows as shown in the third constraint. At each time point, the number of items that the $IA$ accepts from an input flow is the product of the ratio allocated to that input and the space remaining in the $IA$ as shown in the fourth constraint. At each time point, the number of items that the $IA$ dispenses via an output flow is the product of the ratio allocated to that output and the number of items in the $IA$ as shown in the fifth constraint.

*Definition 4:* The aggregator, $OQA$ is a tuple of *oqa* = *($\overrightarrow{I3}$, $\overrightarrow{O3}$, $\overrightarrow{inAllocRatio3}$, $\overrightarrow{totalTPAlloc3}$)* where:

1) $\overrightarrow{I3} = \{I_i \subseteq$ *IF: set of input flows of OQA* $|i \in OQA\}$. E.g., $I_{aD}$ is the flow (arrow) from $Assembly1$ and $Assembly2$ to $aD$.
2) $\overrightarrow{O3} = \{O_i \subseteq$ *IF: set of output flows of OQA* $|i \in OQA\}$. E.g., $O_{aD}$ is the flow (arrow) from $aD$.
3) $\overrightarrow{inAllocRatio3} = \{inAllocRatio_i : I_i \rightarrow \mathbb{R}^+ | i \in OQA\}$, i.e., for OQA i, $inAllocRatio_i(in)$ gives the portion of $totalTPAlloc_i$ to be allocated to input $in$. E.g., $inAllocRatio_{aD}(fromAssembly1) = 0.5$ and $inAllocRatio_{aD}(fromAssembly2) = 0.5$.
4) $\overrightarrow{totalTPAlloc3} = \{totalTPAlloc_i : \mathbb{T} \rightarrow \mathbb{Z}^+ | i \in OQA\}$, i.e., $totalTPAlloc_i(t)$ is the sum of the allocation of outputs of $OQA$ i at time point t $\in \mathbb{T}$, where $\mathbb{T} = \{1, 2, \ldots, n\}$. E.g., $totalTPAlloc_{aD}(2) = 1$.

We say that an $OQA$ is feasible if it satisfies the following constraints:

1) $totalAllocConstraint:$
$$\underset{i \in OQA}{\forall} \underset{t \in \mathbb{T}}{\forall} totalTPAlloc_i[t] = \sum_{out \in O_l} tpAlloc_{out}[t]$$
2) $inputAllocConstraint:$
$$\underset{i \in OQA}{\forall} \underset{t \in \mathbb{T}}{\forall} \underset{in \in I_i}{\forall} tpAlloc_{in}[t] = inAllocRatio_i[in] \times totalTPAlloc_i[t]$$
3) $outputQtyConstraint: \underset{i \in OQA}{\forall} \underset{t \in \mathbb{T}}{\forall}$
$$\left( \sum_{out \in O_i} periodQty_{out}[t] = \sum_{in \in I_i} periodQty_{in}[t] \right)$$

The mathematical representation of an $OQA$ is described in Definition 4. The $OQA$ collects items entering via its input flow ($\overrightarrow{I3}$) and dispenses them via output flows ($\overrightarrow{O3}$). It is obvious that the $OQA$ is symmetrical to the $IQA$. Hence, the three constraints discussed in Definition 4 are also symmetrical to the constraints of the $IQA$ discussed in Definition 2.

*Definition 5:* The flows, $IF$ is a tuple of *if* = *($\overrightarrow{tpAlloc}$, $\overrightarrow{periodQty}$)* where:

1) $\overrightarrow{tpAlloc} = \{tpAlloc_i : \mathbb{T} \rightarrow \mathbb{Z}^+ | i \in IF\}$, i.e., $tpAlloc_i(t)$ is the maximum quantity of items allowed in item flow i at time point t $\in \mathbb{T}$, where $\mathbb{T} = \{1, 2, \ldots, n\}$. E.g., $tpAlloc_{fromSanding1}(2) = 4$.
2) $\overrightarrow{periodQty} = \{periodQty_i : \mathbb{T} \rightarrow \mathbb{Z}^+ | i \in IF\}$, i.e., $periodQty_i(t)$ is the quantity of items in item flow i at time point t $\in \mathbb{T}$. E.g., $periodQty_{fromSanding1}(2) = 3$.

We say that an $IF$ is feasible if it satisfies the following constraint:

1) $qtyConstraint:$
$$\underset{i \in IF}{\forall} \underset{t \in \mathbb{T}}{\forall} periodQty_i[t] \leq tpAlloc_i[t-1]$$

The mathematical representation of an $IF$ is described in Definition 5. The $IF$ acts as an interface between the sub processes and aggregators of the composite BTF process. Each $IF$ contains one constraint, which is the number of items that the flow can carry at each time period ($periodQty$) is the upper bounded by the amount of items that the aggregators can accept at each time period ($tpAlloc$).

Using the definitions for the base BTF process, aggregators, and flows, it is easy to model a composite BTF process. A composite BTF process may contain one or more base processes (e.g., $Sanding1$, $Sanding2$ etc. in Figure 1), aggregators (e.g., $a1$, $a2$ etc. in Figure 1), and flows (e.g., flows (arrows) in Figure 1). The metrics of the composite BTF process can serve as a global performance indicator of a manufacturing floor. One of these metrics can be used as the objective in the optimization problem. Examples of such objectives include minimization of cost, maximization of some utility etc. A composite process may also contain other composite processes. More generally, a BTF process is defined formally as shown below.

*Definition 6:* Recursively, a BTF process p is defined as either:

1) A base BTF process (see Definition 1), or
2) A composite BTF process, defined as a tuple of cp = *(I, O, SP, IQA, $IA$, $OQA$, $IF$, $\mathcal{M}$)* where:
   a) $I \subseteq IF$: set of input flows to the composite BTF process
   b) $O \subseteq IF$: set of output flows from the composite BTF process
   c) $SP$: set of sub processes, each is recursively, a BTF process
   d) $IQA$: set of all IQAs (see Definition 2)
   e) $IA$: set of all IAs (see Definition 3)
   f) $OQA$: set of all OQAs (see Definition 4)
   g) $IF$: set of all IFs (see Definition 5)
   h) $\mathcal{M} : \mathbb{T} \times \mathbb{R}^{+} \rightarrow D_1 \times \cdots \times D_k$, i.e, $\mathcal{M}$ gives the metrics $M_1(t, X)$ through $M_k(t, X)$ from their domains $D_1$ through $D_k$, where X is the sum of the corresponding metrics among all sub processes at time point t $\in \mathbb{T}$, where $\mathbb{T} = \{1, 2, \ldots, n\}$

We say that process p is feasible:

1) If p is a base BTF process (case 1), it is a feasible base BTF process
2) If p is a composite BTF process (case 2), p is feasible if:
   a) All sub processes in SP are feasible, and
   b) All IQAs in $IQA$ are feasible, and
   c) All IAs in $IA$ are feasible, and
   d) All OQAs in $OQA$ are feasible, and
   e) All IFs in $IF$ are feasible.

## V. BTFP Optimization

It is now possible to use the BTF process defined in the previous section in an optimization problem. A process engineer may want to pose optimization or what-if analysis questions against the BTF process to get actionable recommendations from the composed model. In this sections, we provide the syntax and semantics of such an optimization problem on a BTF process. Additionally, we also use a smaller version of the motivating example to show how an optimization problem can be posed against a BTF process.

More formally, the syntax of an optimization problem on the BTFP is given as:

*Definition 7:* A BTFP optimization problem is a tuple of *(min/max, variableBTFP, objective, additionalConstraints)* where:

- *min/max* is a flag to indicate whether minimum or maximum value of the $objective$ is requested.
- $variableBTFP$ is a BTF process where:
  – parameter variables are recursively instantiated with constants, i.e, variables 1 through 4 in all base BTF processes (Definition 1), variables 1 through 3 in all IQAs (Definition 2), variables 1 through 6 in all IAs (Definition 3), variables 1 through 3 in all OQAs (Definition 4), and variables a, b in all composite BTF processes (Definition 6) are initialized with constants.
  – some controllable variables are annotated as decision variables using the character "?". E.g., $throughputControl$ (variable 5 in Definition 1) is annotated with ?.
  – other variables that are computed using the constraints are recursively annotated as expressions using the character "=", i.e, variables 6, 7 in all base BTF processes (Definition 1), variable 4 in all IQAs (Definition 2), variables 7, 8 in all IAs (Definition 3), variable 4 in all OQAs (Definition 4), variables 1, 2 in all IFs (Definition 5), and variables a, b in all composite BTF processes (Definition 6) are annotated with =.
- $objective$ is a metric or a combination of metrics from $\mathcal{M}$ that needs to be minimized or maximized
- $additionalConstraints$ are optional additional set of constraints imposed on the metrics from $\mathcal{M}$

The result (semantics) of the BTFP optimization is a BTF process p such that:

- if there does not exist a feasible BTFP instance of $variableBTFP$, then $variableBTFP$ is copied into p without any changes and we say that the optimization is infeasible
- if there exists feasible BTFP instance(s) of $variableBTFP$, then the decision variables and expressions in p are instantiated to values that gave the $min/max$ objective value among the instance(s) and we say that the optimization is optimal.

We now consider an example BTF process to show how an optimization problem can be posed against it. Due to lack of space, a smaller version of the motivating example (discussed in Section III) called sandCut is used here. A superior example of BTF process was discussed in [22]. The parameter variables of a composite sandCut BTF process will be initialized and the optimization problem will try to find feasible speeds of the individual processes ($throughputControl$) so as to minimize the totalCost metric such that all the constraints of the sandCut process are satisfied.

Consider the example in Figure 2, which is a smaller version of the bookshelf example described in Figure 1. In this example, a part of the plywood goes to the *sand1*

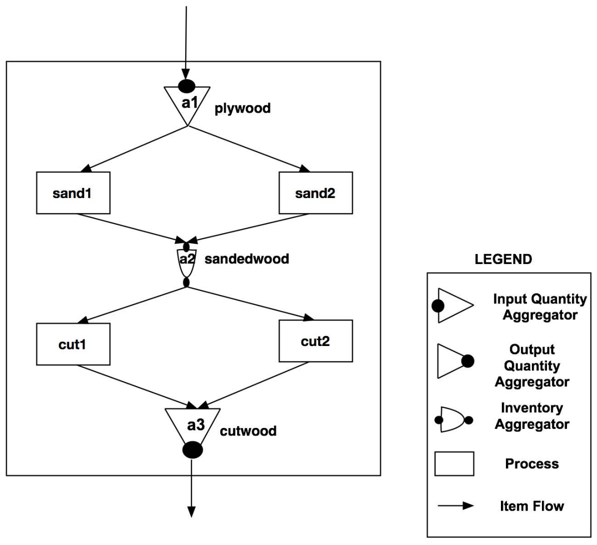

Figure 2. A graphical notation for the sandCut BTF process

Table I
OVERVIEW OF INPUT VARIABLE INITIALIZATION FOR BASE BTF PROCESS

| VariableName | Sand1Value | Sand2Value | Cut1Value | Cut2Value |
|---|---|---|---|---|
| I | {toSand1} | {toSand2} | {toCut1} | {toCut2} |
| O | {fromSand1} | {fromSand2} | {fromCut1} | {fromCut2} |
| throughputControl | ? | ? | ? | ? |
| capacity | 50 | 40 | 40 | 50 |
| inputPerOutput | toSand1:1 | toSand2:1 | toCut1:1 | toCut2:1 |

process and the remaining goes to the *sand2* process. The sanded plywood is then buffered and redistributed among the *cut1* and *cut2* processes. The sanded plywood is cut in these processes and finally, the cut plywood is collected and provided as output from the sandCut manufacturing floor. In this example we will assume that there are 5 time points in the time horizon, i.e., $\mathbb{T} = \{1, \dots, 5\}$

The initialization of the inputs for each base BTF process (see Definition 1) are shown in Table I. The input of the

Table II
OVERVIEW OF INPUT VARIABLE INITIALIZATION FOR IA A2

| VariableName | a2 value |
|---|---|
| I | {fromSand1, fromSand2} |
| O | {toCut1, toCut2} |
| initInv | 0 |
| capacity | 80 |
| inAllocRatio | fromSand1: 0.5, fromSand2: 0.5 |
| outAllocRatio | toCut1: 0.4, toCut2: 0.6 |

Table III
OVERVIEW OF INPUT VARIABLE INITIALIZATION FOR IQA A1

| VariableName | a1 value |
|---|---|
| I | {plywood_in} |
| O | {toSand1, toSand2} |
| outAllocRatio | toSand1:0.5, toSand2:0.5 |

Table IV
OVERVIEW OF INPUT VARIABLE INITIALIZATION FOR OQA A3

| VariableName | a3 value |
|---|---|
| I | {fromCut1, fromCut2} |
| O | {cutwood_out} |
| inAllocRatio | fromCut1:0.4, fromCut2:0.6 |

$sand1$ process is the IF coming from $a1$ and the output is the the IF going to $a2$. The $throughputControl$ or the speed of the $sand1$ process is a controllable process settings that is the decision variables in the optimization problem and hence it is annotated with "?". The $capacity$ for $sand1$ process is set to 50 and finally $inputPerOutput$ is set so that for each sanded plywood produced by $sand1$ process, one plywood would be required. Other prcoess variables are initialized similarly.

Table II gives the variable initialization of the inputs for the IA $a2$. The input and output is a set of IF ids of the sand and the cut processes respectively. There are 0 items initially in the inventory and $a2$ is set to have a capacity of 80. Also, the allocation ratios for the input and output are set to percentage values so that the the proportion of those amounts can be allocated to the respective input and output flows. Table III and Table IV gives the variable initialization of the inputs for the IQA a1 and OQA a3 respectively. These tables show the input and output flow ids along with the respective allocation ratios that are also percentage values. The other variables of the whole sandCut BTF process that are not mentioned in these tables are the expressions and they are annotated with "=".

The metric used in the sanding and cutting processes is that of real-valued cost metric and is computed as a piece-wise linear function of the $throughputControl$ decision variable over the time horizon. We omit the formulation of the piecewise linear function here. Using the individual cost metric of each sanding and cutting process, the objective metric for the whole sandCut BTF process can be given as:

$$sandCut.cost = \sum_{t \in \mathbb{T}} (sand1.cost[t] + sand2.cost[t] + cut1.cost[t] + cut2.cost[t]).$$

Using the initialized sandCut BTF process and the cost objective, we can now formulate the optimization problem. This optimization problem formulation follows from Definition 7. The optimization task is to find the process speed control variables ($sand1.throughputControl$, $sand2.throughputControl$, $cut1.throughputControl$, $cut2.throughputControl$) so as to minimize the cost of the SandCut BTF process ($sandCut.cost$) subject to the the constraints in the SandCut BTF process (described in Definition 6) being feasible. Thus, this task can be summarized using the tuple $opt\_in = (min, sandCut, sandCut.cost)$. The optimal output of the problem is given as $sandCutOut$ where $sandCutOut$ has the exact same structure as the $sandCut$

BTF process but with the $throughputControl$ variable and expressions initialized to values where $sandCutOut.cost$ is minimum if the result of the optimization is feasible.

## VI. Implementation of BTFP Optimization in OPL

This section describes the implementation of the optimization model for a BTF process to Optimization Programming Language (OPL) by IBM [10]. OPL is an optimization software package that solves integer programming and very large linear programming problems. The optimization results are then provided as decision guidance to decision makers. BTF processes are modeled in OPL as a generic module. In this paper, these generic modules are described with the help of code-snippets from OPL. Using these generic modules, it is possible to generate specific models that are linear to the size of the original manufacturing floor data. This can be done by using the data files in OPL. We describe the method of manually initializing such a data file for a specific manufacturing floor setting so that a process operator can run an optimization query using the generic module files and the data file for any BTFP optimization problem formulation

First the time horizon is set. Given the number of periods ($noPeriods$), the last time point ($lastTP$) is the same as the number of periods. Except that the time points range from 0 to $lastTP$ and the periods range from 1 to $noPeriods$ as shown by the following OPL code snippet.

```
int noPeriods = ...;
int lastTP = noPeriods;
range timeRange = 0..lastTP;
range periodRange = 1..noPeriods;
```

In order to generalize the composite BTF process, base BTF process, IF, IA, IQA and OQA, are stored as an array of their respective ids. The variables of each module are indexed on their respective ids. If the variables is also indexed on other units such as time points, then the variable is stored as a 2D array, first indexed on the respective ids and then indexed on the units. Further, the variables of the components whose values are provided as input have the values of ... in the implementation. This is to indicate that OPL will search for the values of these variables in the data file. The variables of the components whose values are defined as decision variables are defined as $dvar$ variables in the implementation. These variables are decision variables in the optimization problem and will be determined by OPL while satisfying the constraints and/or optimizing some numeric value. An example of this strategy is shown via the code snippet of the IF variable definition in OPL.

```
{string} IFIds = ...;
dvar int+  IF_tpAlloc
               [IFIds][timeRange];
dvar int+  IF_periodQty
             [IFIds][periodRange];
```

The constraints that were described in the BTF process definition are also encoded in OPL. For instance the encoding of the IF constraint is shown here. For all indexes in the IF component ids and for all periods, this constraint expression bounds the $periodQty$ value by the respective $tpAlloc$ value.

```
forall(id in IFIds){
    forall(p in 1..noPeriods){
        IF_periodQty[id][p] <=
         IF_tpAlloc[id][p-1];
    }
}
```

The other constraints are expressed similarly in OPL.

In order to manually translate a system layout to an OPL data file that can be run with the OPL model, the following steps are taken.

1) Convert the system layout to the graphical language by generating a composite process graph for the manufacturing floor like the one shown in Figure 2 for the sandCut BTF process example.
2) Divide the composite process graph into one or more composite process modules
3) For each composite process module, identify the aggregators (IA, IQA, OQA), the base processes (base BTF process) and then the flows that connect them.
4) Add each IF identified as a unique IF id to the data file.
5) Provide the inputs and outputs for each of the other components (IA, IQA, OQA and base BTF process) as the same ids added as IF ids thus connecting them manually with each other. Add these components to the data file.
6) Provide the other interface constants for IA, IQA,OQA and baseProcess.
7) Perform steps 3-6 for all the composite process modules identified in #2.
8) Connect the different composite process modules by using the same IF id as inputs/outputs to these modules
9) Run min/max query on the composed model in OPL.

## VII. Conclusion and future work

This paper considers BTFP problems and maps the entities of such manufacturing floors using modular components called BTF processes. A graphical notation is shown to describe these BTF processes. The paper also presented a mathematical formulation of the base BTF process and composite BTF process. The base BTF process contains variables, constraints and metrics that map it directly onto a atomic process on the manufacturing floor. The composite BTF process contains the sub processes, IA, IQA, OQA, and IF that enable it to capture the variables, constraints and metrics that map the process to an entire or a part of the manufacturing floor.

This paper formally defines the syntax and semantics of an optimization problem on these BTF processes. We show the optimization scenario by using an example sandCut BTFP. We also described the implementation of BTFP optimization problem in OPL and show how a BTF process can be formulated as a deterministic MILP problem by manually converting the processes into a deterministic OPL model and the associated data file.

We are currently investigating: (a) the incorporation of real-world constraints like sequence-dependent setup times, nested time lags etc., (b) developing graphical user interfaces for optimizing BTF processes with online examples, instances, and code, with a step-by-step tutorial, (c) a library of the components discussed here for different manufacturing scenarios, (d) modeling BTFP as a job-scheduling problem and to use a feedback loop, and (e) developing specialized algorithms that can utilize preprocessing of stored (and therefore, static) components to speed up optimization, generalizing the results in [23].

## Acknowledgement

The authors are partly supported by the National Institute of Standards and Technology Cooperative Agreement 70NANB12H277. The authors would also like to thank the reviewers for their valuable comments, which helped improve the manuscript.

## References

[1] J. Davis, T. Edgar, Y. Dimitratos, J. Gipson, I. Grossmann, P. Hewitt, R. Jackson, K. Seavey, P. Porter, R. Reklaitis, and B. Strupp, "Smart Process Manufacturing: An Operations and Technology Roadmap," Smart process manufacturing engineering virtual organization steering committee, Los Angeles, CA, Tech. Rep., 2009.

[2] S. Weiner, "Manufacturing and the data conundrum: Too much? Too little? Or just right?" The Economist Intelligence Unit, eds. David Line, Washington DC, Tech. Rep., 2014.

[3] A. Fink, N. Kliewer, D. Mattfeld, L. Mönch, F. Rothlauf, G. Schryen, L. Suhl, and S. Voß, "Model-based decision support in manufacturing and service networks," *Business & Information Systems Engg.*, vol. 6, no. 1, pp. 17–24, 2014.

[4] K. Tierney, S. Voß, and R. Stahlbock, "A mathematical model of inter-terminal transportation," *European Journal of Operational Research*, vol. 235, no. 2, pp. 448 – 460, 2014.

[5] T. Grünert and H. Sebastian, "Planning models for long-haul operations of postal and express shipment companies," *European Journal of Operational Research*, vol. 122, no. 2, pp. 289 – 309, 2000.

[6] S. Melouk, N. Freeman, M. Miller, and M. Dunning, "Simulation optimization-based decision support tool for steel manufacturing," *International Journal of Production Economics*, vol. 141, no. 1, pp. 269 – 276, 2013.

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

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

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

[10] P. Van Hentenryck, *The OPL optimization programming language*. Cambridge, MA, USA: MIT Press, 1999.

[11] R. A. Sarker and C. S. Newton, *Optimization Modelling: A Practical Approach*. Boca Raton, FL: CRC Press, 2007.

[12] V. Jovanovic, B. Stevanov, D. Seslija, S. Dudic, and Z. Tesic, "Energy efficiency optimization of air supply system in a water bottle manufacturing system," *Journal of Cleaner Production*, vol. 85, pp. 306 – 317, 2014.

[13] L. Grandguillaume, S. Lavernhe, Y. Quinsat, and C. Tournier, "Mold manufacturing optimization: A global approach of milling and polishing processes," *Procedia* {*CIRP*}, vol. 31, pp. 13 – 18, 2015.

[14] W. J. Hopp and M. L. Spearman, *Factory Physics*. Long Grove, IL: Waveland Pr Inc; 3rd edition, 2011.

[15] 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*, 2015, pp. 30–46.

[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. Kallrath and T. I. Maindl, *Real Optimization with SAP APO*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[18] R. Zhang, W. Chiang, and C. Wu, "Investigating the impact of operational variables on manufacturing cost by simulation optimization," *International Journal of Production Economics*, vol. 147, Part C, pp. 634 – 646, 2014.

[19] M. S. Hillier and F. S. Hillier, "Simultaneous optimization of work and buffer space in unpaced production lines with random processing times," *IIE Transactions*, vol. 38, no. 1, pp. 39–51, 2006.

[20] F. Altiparmak, B. Dengiz, and A. Bulgak, "Optimization of buffer sizes in assembly systems using intelligent techniques," in *Simulation Conference, 2002. Proceedings of the Winter*, vol. 2, Dec 2002, pp. 1157–1162.

[21] G. Gurkan, "Simulation optimization of buffer allocations in production lines with unreliable machines," *Annals of Operations Research*, vol. 93, no. 1-4, pp. 177–216, 2000.

[22] D. A. Menasce, M. Krishnamoorthy, and A. Brodsky, "Autonomic smart manufacturing," *Journal of Decision Systems*, vol. 24, no. 2, pp. 206–224, 2015.

[23] 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*, Jan 2013, pp. 1124–1133.