Efficient Control for a Multi-Product Quasi-Batch Process via Stochastic Dynamic Programming

Ezgi C. Eren and Natarajan Gautam*
Department of Industrial and Systems Engineering
Texas A&M University
College Station, TX 77843-3131
{ezgi,gautam}@tamu.edu

July 20, 2010

Abstract

We consider a quasi-batch process where items are continuously processed while they move on a conveyor belt. In addition, the products arriving into the processor require variable amounts of processing, which translate into different processor levels. Keeping the processing level constant in such a system results in severe inefficiency in terms of consumption of energy and resources with high production costs and a poor level of environmental performance. To strike a balance between consumption of energy and material, processor performance, and product quality, we formulate a stochastic dynamic programming model. The model minimizes total system-wide cost which is essentially a unified measure across all the objectives. Taking into account high-dimensionality of the state space, we analyze the structural properties of the optimal policy, which is a mapping between the state of the system at any time and the corresponding optimal processor level, and the value functions, which give the best possible value of the objective for the system. Based on some of these results, we develop efficient heuristic methodologies to solve large instances of the problem. We show using several numerical experiments that a significant amount of energy or material resources can be saved and total costs can be reduced considerably compared to the current practices in the process industry. We further provide insights on sensitivity of results with respect to the cost parameters.

Supplementary materials are available for this article. Go to the publisher's online edition of *IIE Trasactions* for additional tables.

Keywords: Efficient process control, energy conservation, environmental impact, process industry, multi-product quasi-batch processing, stochastic dynamic programming, MDP, multi-dimensional state space.

*Corresponding Author

1 Introduction

In this paper, we consider developing control strategies for efficiently operating quasi-batch processes. Although the term *quasi-batch process* has slightly different meanings depending on the context, we restrict our attention to those in production systems. In particular, a quasi-batch process, like standard batch processes, involves processing multiple products simultaneously. However, unlike standard batch processes, in quasi-batch processes individual entities in a batch arrive and depart at different times. Typically products are processed while they move on a conveyor belt from when they arrive at the processor until their departure. Quasi-batch processes can be broadly categorized as: (i) single product where all entities are identical in all respects, and (ii) multi-product where entities can belong to multiple classes depending on their type and processing needs.

We consider a multi-product quasi-batch process which is common in the process industry. Here, the input to the processor is variable and a representative sketch of the described system is provided in Figure 1. Items continuously enter the processor from one end and move on a conveyor belt while being processed. They require different amounts of processing, which would be achieved by different levels of processor setting. There exits a central control knob which is used to change the setting of the processor. Due to the variability of the input into the processor, implementing a static design with a constant processor setting (which is the state of the practice in most industries) in such a system has several undesired outcomes. For example, products requiring lower settings would have been over-processed, whereas the ones requiring higher settings would be under-processed as they exit the processor. This results in an inefficient use and waste of resources and materials in addition to degradation in the quality of output products. Next, we provide a few examples of multi-product quasi-batch process.

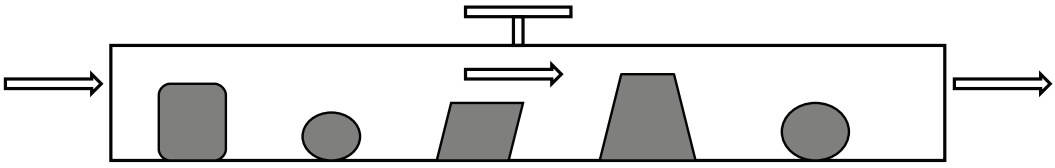

Figure 1: A snapshot of products and conveyor belt in a quasi-batch processor

A typical example of such a multi-product quasi-batch process is tunnel freezing of food items with variable cooling requirements. To illustrate multi-product inputs to tunnel freezers consider the following: meat products chopped in different sizes and shapes; any similar-sized cooked items with varying temperatures; similar-sized items with similar temperatures but with high variability in arrival time. In all these cases the input entities would be considered as "multi-product" because of the variability in their characteristics when they enter the tunnel. The tunnel freezer is typically composed of a tunnel and a conveyor belt, where from one side the items enter and proceed through the tunnel as the conveyor belt moves. In cryogenic tunnel freezers, items are coated with refrigerants such as liquid nitrogen at very low temperatures while in the tunnel, whereas in mechanical freezers the cooling power is generated by continuous supply of energy. Every item coming in has to be served right away due to strict regulations regarding contamination of food items processed in the food industry. Hence, there is no buffering prior to the process. There is a central valve for instantaneous control of the refrigerant flow into the tunnel. The state of the practice is to operate the tunnel freezers continuously at a high fixed level which ensures that the maximum anticipated thermal load will be frozen (Ramakrishnan et al. [17]). This results in high energy or refrigerant consumption with most of the output being overfrozen. Over-freezing is a common problem in the food industry due to high costs of refrigerants as well as increasing energy costs. For cryogenic freezers, refrigerant costs per pound of an item constitute a significant portion of the total production costs per pound. Freezing and

cooling of the products together with the cooling of the production rooms in the food industry account for around 50 % of the total electricity consumption (Mattsson and Sonesson [8]). On the other hand, applying a static design of a low constant freezer level would lead to under-freezing of products with high processing requirements. All these can be avoided by dynamically adjusting the control knob with appropriate monitoring of input into the process, which is the focus of this paper.

Another example is a process for undulating the surface of micro-fibers during their manufacturing process. That requires usage of nonuniform plasma and changing the power input of microwave radiation according to the requirements of different zones of the surface to be produced (Paulauskas et al. [13]). Under-processing of micro-fibers results in failure to obtain the desired surface features, whereas over-processing causes undesired weakening of the fiber and degradation of the product quality. Energy consumption in micro-fiber manufacturing constitutes 20-25 % of overall production costs (Paulauskas et al. [14]). In view of increasing energy costs, those figures are only bound to become more significant. Therefore, the results of this research can be applied to improve efficiency of these systems.

Having described some examples of multi-product quasi-batch processes, next we highlight the key difference between this and other closely-related but well-studied systems. In production systems in general, it is common to have different types of processors involved in a production process. A common example would be packaging of items as discrete products following mixing different ingredients in a batch process to prepare the product and bringing it to an ideal temperature by use of a continuous processor. Such hybrid processes are widely studied in the literature (Moreno-Lizaranzu et al. [10], Dennis and Meredith [2], Rossi et al. [18]). Our focus here is on a single processor which has a hybrid structure itself, which is frequently seen in different settings of the process industry. We consider efficient control of such a processor during the whole process duration.

In summary, there is an opportunity to reduce consumption of energy and other resources via implementation of an efficient process control mechanism which will capture the variability in the system. However, notice that in a quasi-batch process, this variability occurs both in time and space as a batch is composed of non-identical items at any time, which dynamically change as items move forward on the conveyor. The state of the system at any time depends on all the items in the batch and the processor setting at the time. As a result, an intelligent control strategy should rely on all those components with a regular monitoring of the system state and take into account the variability in the process. For that, we model the described system using stochastic dynamic programming (SDP) techniques and investigate the effects of establishing an intelligent control to operate the processor. Our focus in this paper is developing a system-wide model for the described setting, where we establish a control mechanism based on observing the state of the complete system over time in addition to the input into the processor. That way, we are able to analyze overall effects of the proposed control mechanism in addition to the significant energy, material and cost savings obtained.

In terms of broader impacts of this research, in addition to significant cost reduction, process efficiency is also critical to improve environmental performance of industry operations, which has been receiving increasing attention since the environmental regulations have become stricter and the notion of a "sustainable economy" has risen. In this context, Environmentally Conscious Manufacturing (ECM) has been widely addressed in literature to focus on the environmental impacts during the whole product life-time (O'Brien [11], Stuart et al. [19], Gupta and Flapper [4]). Development of intelligent control strategies in certain systems is a fairly recent notion, which is gaining attention with improvements in sensor technologies that enable continuous monitoring of production systems (Gupta and Sinha [3]). Efficient control strategies are promising components of ECM, as they increase environmental performance of operations without introducing any significant costs but rather resulting in a direct decrease in process costs (Carter [1], Mittal [9]).

Next, we briefly describe the objectives of our study. We assume that control of the processor is managed by changing the setting of the processor which we refer to as the processor level. Based on that, we discretize the time, state and action spaces. A description of the problem

including its discrete characteristics and system-wide cost components is presented in Section 2. We formulate an SDP model with a multi-dimensional state space and describe the associated *value functions* (Section 3). We analyze the structure of the optimal policy and value functions and list our findings in Section 4. Based on some of these results, we develop computationally efficient heuristic algorithms for large instances of the problem where value iteration does not work due to "curse of dimensionality" (Section 5). We conduct a numerical study to analyze cost and energy/material savings obtained by the proposed SDP strategy and the performance of heuristic algorithms for a wide range of parameters and different size instances of the problem (Section 6). We finally conclude with summarizing remarks and a discussion of possible future work.

2 Problem Description

In this paper, we consider a single quasi-batch processor into which multiple discrete products with different levels of processing requirements arrive. The products enter from one side of the processor and propagate towards the exit, as the conveyor belt moves. An illustration of the described setting is provided in Figure 1. A typical example would be meat products to be frozen in a tunnel freezer right after they are cut in different sizes and shapes. Another example is quasi-batch processing of micro-fibers to produce undulated surfaces, where successive zones of a fiber are treated as discrete items which require different amounts of processing. In both cases the products are served right away in order to prevent contamination or degradation of quality. Therefore, there is no buffering prior to the process. As described in Section 1, applying a static design with a constant processor setting has undesired results of either under-processing or over-processing of items together with phenomenal amounts of energy and material wastage. The objective of this paper is to develop an optimal strategy to control the processor level over time, based on overall system state. We aim to strike a balance between energy (or material) consumption, product quality and the performance of the processor. For this, we first model the system dynamics and costs, and in the next section we formulate an optimal control framework. Costs are later used as a common denominator to balance various objectives and compare the performance of the proposed method to current practices.

To model this system, we discretize time, state and action spaces. Consistent with the practices in the process industry, we assume that the conveyor belt of the processor runs at a constant velocity, high enough to accommodate the arrival rate. We divide the processor into a fixed number of slots N, according to the arrival rate and the speed of the conveyor belt, where each slot can have at most one item or a single zone of an item at a time. A time period is defined as the constant time it takes for all the items in the processor to proceed one slot further. At the beginning of each time period, as the item in the last slot of the processor exits (if the slot is nonempty), a new item (if any) enters the processor simultaneously occupying the very first slot. The possibility of unoccupied slots in the processor is simply a result of the fact that there may be no arrivals at the beginning of any time period. At the beginning of each time period, an item of class $i = 0, 1, ..., C_{max}$ arrives into the processor with probability p_i. The class of an item is defined in proportion to the processing amount it requires. Class 0 simply represents a case of no-arrival. For example, for the process of tunnel freezing, the classes are defined according to the energy levels that need to be extracted from the item to be frozen. As the energy is a function of mass, specific heat capacity and the difference between the freezing temperature and the initial temperature, class values are suitably discretized to account for weight, material and temperature differences between the items. Throughout the process, the remaining processing amount required by each item decreases as the item propagates in the processor and stays the same once it reaches zero. This decrease can dynamically change from period to period depending on the processor setting. The control of the processor is managed by selecting one of the actions, $a = 0, 1, 2, ..., L_{max}$, at the beginning of each time period, where each a value represents a different processor level. The processing requirements of the items in all slots of the processor decrease by the same amount in a time period (unless some of them have already reached zero), which is defined by the level selected at the beginning of the period.

The remaining amounts of processing required by items in each slot at the beginning of an iteration determine their state values, which may take on values in a range specified according to the possible action and class levels. Note that state 0 means either the slot is occupied by an item which does not need further processing or it is unoccupied. Besides, the state of an item at the very first slot of the processor, is equal to its class value. Together with the current level of the processor, the states of the items in all slots form the state of the overall system. Hence, the state of the system at any time period is an "$N + 1$" dimensional vector.

To give an example, let us consider defining item classes, states and action levels according to the above convention in a tunnel freezer, where the control is power or refrigerant level of the freezer. Let us consider a freezer setting with energy requirements for different items and power levels as follows: The energy requirements are 0, 1, 2, 3, 4, 5 Kilojoules and the power levels are 500 and 1000 Watts. For simplicity assume that the time it takes for each item to propagate one slot, i.e. the time period is just one second. That means that at the beginning of any time period, remaining energy requirements of the items in the tunnel can take on values as 0, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000 Joules, which can be represented by the state values 0, 1, 2, ..., 10 with levels defined as 0, 1 and 2. As a result, arrival classes would be redefined as 0, 2, 4, 6, 8, 10 to preserve the consistency with the state values. Note that an action level 1 means that during the period, the state of each item in the tunnel will decrease by 1 unless it has already reached state 0. The state vector of the tunnel freezer can take on values in the range defined by the possible combinations of state values for all the items in the tunnel and action levels.

The state transitions between successive periods occur based on three components, which are the initial state of the processor, action level chosen, and the class of the arriving item before the next period begins. For example, with $N = 4$ given, if the current state of the system is $\langle 5, 1, 4, 1, \boldsymbol{2}\rangle$ and the new action chosen is 1, the next state will be $\langle i, 4, 0, 3, \boldsymbol{1}\rangle$, where i can be any of the class values defined for the process with probability p_i. However, if the action is 2, the next state will be $\langle i, 3, 0, 2, \boldsymbol{2}\rangle$.

There are costs incurred at each period which are functions of the system state and the action chosen. There is a fixed continuous cost of power or material supply, P per unit time, incurred as long as the processor runs, and additionally p per processor level per unit time. This could also be the cost incurred for any other resources while processing, such as refrigerants used in tunnel freezing processes. We ignore the fixed term, P, as it is a sunk cost. Thus, for any time slot, the incurred power (or material) cost is pa/λ, where a is the action chosen and $1/\lambda$ is equal to the length of a time period. There are also costs associated with the control strategy itself when a change to the processor level occurs. When the chosen action level is higher than the previous one, a one-time cost of switching is incurred associated with the immediate supply of additional resources into the processor. This cost is composed of a fixed term, Q, which represents the one-time fixed cost incurred each time switching to a higher level occurs and a linear term which is a function of the distance to the switched action and a linear cost factor of q. A switching cost equivalent to $Q + q(k - l)$ is incurred for switching from level l to k where $k > l$. This assumption is motivated by the settings of the processes mentioned in Section 1. For example, it is more of a concern to instantly increase the processor level in a tunnel freezer to the desired level by adding refrigerants through the control valves, whereas to decrease the freezer level, just introduction of ambient air is sufficient. Hence, we model this cost associated with an increase in the processor level, although it is worth noting that a symmetric switching cost can as well be modeled without significantly affecting the analysis of the problem. Inclusion of this cost term also serves to set a level of sensitivity for the control strategy to avoid too frequent processor level switching, which typically is a requirement of the setting considered. Finally, we assume that any under-processed items, i.e. items which are at state 1 or greater at the time of departure, result in a one-time penalty, which is a function with a fixed cost term, R and a linear cost term proportional to the state of the departing item. If the item is at state $s > 0$ while exiting the tunnel, the corresponding penalty cost is $R + rs$, where r is the penalty incurred per unit state value. No additional costs for over-processed items are considered as those already correspond to waste of energy or material resources, i.e. excess cost incurred for

redundant power (or material) supply.

3 Stochastic Dynamic Programming Formulation

Based on the assumptions discussed above, we formulate a discrete-time, infinite-horizon SDP model of the problem, in which the objective is to minimize the total discounted costs incurred by the system. The time periods are defined in accordance with the slots of the processor as described in Section 2. We define s_i, $i = 0, 1, ..., N$ to be the state of the item in the i^{th} slot of the processor, which represents its remaining required processing at the beginning of a period. Current level of the processor is denoted by l and can take on any of the values in the set $\{0, 1, ..., L_{max}\}$. As a result, the state space of the system, $\mathcal{S}$, includes $(N + 1)$ dimensional vectors, $\langle \overrightarrow{s}, l\rangle = \langle s_1, s_2, ..., s_N, l\rangle$, where $s_i \in \{0, 1, ..., C_{max}\}$ $\forall i$, $i = \{1, 2, ..., N\}$ and $l \in \{0, 1, ..., L_{max}\}$. Recall that N is the number of the slots in the processor as mentioned in Section 2.

At the beginning of each period, once the class of the arriving item is observed together with the state of the processor, the decision is made regarding staying at the current level l or switching to one of the other levels in the action space, $a \in \mathcal{A} = \{0, 1, ..., L_{max}\}, a \neq l$. Depending on the state and the action chosen, the incurred costs per time period, i.e. the stage costs are

$$c(\overrightarrow{s}, l, a) = \begin{cases} \frac{pa}{\lambda} + R1_{\{(s_N - a) > 0\}} + r(s_N - a)^+ & \text{if} \quad a \leq l, \\ \\ \frac{pa}{\lambda} + R1_{\{(s_N - a) > 0\}} + r(s_N - a)^+ + Q + q(a - l) & \text{if} \quad a > l, \end{cases}$$

where $\overrightarrow{s} = \langle s_1, s_2, ..., s_N\rangle$. Note that $(s - a)^+ = \max\{0, s - a\}$ and for any real s,

$$1_{\{s>0\}} = \begin{cases} 1 & \text{if} \quad s > 0, \\ 0 & \text{otherwise.} \end{cases}$$

The first term in both expressions of $c(\overrightarrow{s}, l, a)$ corresponds to the power (or material) supply costs which are functions of the processor level chosen, and following two terms represent the penalty cost associated with any under-processed items, which is a function of both the state vector and the action level chosen. The last two terms in the second expression constitute the cost of switching, which is incurred only when switching to a higher level occurs. Switching cost is a function of both the action level chosen and the state vector (the element of the vector which represents the current level of the processor) as well.

At the beginning of each time period, t, events occur in the following time order:

1) The new item arrives with a class, i_t, where $i_t \in \{0, 1, ..., C_{max}\}$.

2) The state of the system, $\langle \overrightarrow{s}_t, l\rangle$ is observed.

3) The decision, a_t is taken.

Based on that, we can determine the state at the beginning of the next period. When the state at the beginning of period t is $\langle \overrightarrow{s}_t, l_t\rangle = \langle s_1, s_2, ..., s_N, l\rangle$ and the action $a_t = a$ is chosen, the state at the beginning of the next period $t+1$ becomes $\langle \overrightarrow{s}_{t+1}, l_{t+1}\rangle = \langle i_{t+1}, (s_1 - a)^+, (s_2 - a)^+, ..., (s_{N-1} - a)^+, a\rangle$ with probability $p_{i_{t+1}}$, where i_{t+1} is the class of the item which arrives at the beginning of the next period. Note that the action chosen at period t becomes a part of the system state at the next period, $t + 1$. As a result, the transition probabilities according to the arrival class distribution are given by

$$p(\langle \overrightarrow{r}, a\rangle | \langle \overrightarrow{s}, l\rangle, a) = \begin{cases} p_i & \text{if} \quad \overrightarrow{r} = \langle i, (s_1 - a)^+, (s_2 - a)^+, ..., (s_{N-1} - a)^+, a\rangle, \\ 0 & \text{otherwise,} \end{cases}$$

where $\langle \overrightarrow{s}, l\rangle = \langle s_1, s_2, ..., s_N, l\rangle$ and for $i = 0, 1, ..., C_{max}$.

We define the value function in state $\langle \overrightarrow{s}, l\rangle = \langle s_1, s_2, ..., s_N, l\rangle$ as $V_\beta(\overrightarrow{s}, l) = V(s_1, ..., s_N, l)$ and obtain the following set of optimality equations:

$$V_\beta(\overrightarrow{s}, l) = \min_{a \in \{0,1,...,L_{max}\}} \{c(\overrightarrow{s}, a) + \beta \sum_{j=1}^{C_{max}} p_j V_\beta(j, (s_1 - a)^+, ..., (s_{N-1} - a)^+, a)\}, \qquad (1)$$

where β is a given discount factor (Puterman [15]).

By considering SDP model with the above optimality equation (i.e. Equation (1)), which allows dynamic change of the processor level at the expense of possible costs related to the under-processing of items and the control activity itself (viz. switching costs), we aim at analyzing the trade-off in terms of process efficiency versus product quality.

4 Structural Results

In order to solve Equation (1) to obtain optimal policy and optimal value function, one option is to make use of the value iteration algorithm (Puterman [15]), that would iteratively solve the following optimality equations according to (2) and (3) until convergence occurs:

$$V_{n+1}(\overrightarrow{s},l) = \min_{a\in\{0,1,...,L_{max}\}}\{c(\overrightarrow{s},a)+\beta\sum_{j=1}^{C_{max}} p_j V_n(j,(s_1-a)^+,...,(s_{N-1}-a)^+,a)\}, \quad (2)$$

$$V_0(\overrightarrow{s},l) = 0. \quad (3)$$

Value iteration algorithm works only for cases with small C_{max} and N values, as the high-dimensionality of the state space makes the computation of the optimal solution intractable for large instances of the problem. Hence it is critical to understand the behavior of the optimal policy and value functions, to aid in development of efficient methods to approximate the optimal solution. In this section, we derive structural properties of the original problem and a special case of it taking into account multi-dimensional state space and multiple action choices. Using some of these results, in Section 5, we develop heuristic algorithms that are useful for large instances of the problem.

We begin by showing that the value functions are partially nondecreasing in the state vector described for the quasi-batch system above. For this, we define a partial ordering of the state space. We assume $\langle\overrightarrow{k},a_k\rangle = \langle k_1,k_2,...,k_N,a_k\rangle$ is partially greater than $\langle\overrightarrow{l},a_l\rangle = \langle l_1,l_2,...,l_N,a_l\rangle$, denoted by $\langle\overrightarrow{k},a_k\rangle \succeq \langle\overrightarrow{l},a_l\rangle$, if $k_1\geq l_1, k_2\geq l_2,...,k_N\geq l_N, a_k\leq a_l$ hold. Note that the state vector increases with increasing states of the items in the processor and decreases with increasing processor level. This ordering makes the state space, $\mathcal{S}$ a partially ordered set (poset) as it is reflexive, antisymmetric and transitive (Topkis [20]). In $\mathcal{S}$ there are unordered elements, k and l, which neither satisfy $\langle\overrightarrow{k},a_k\rangle \succeq \langle\overrightarrow{l},a_l\rangle$ nor $\langle\overrightarrow{l},a_l\rangle \succeq \langle\overrightarrow{k},a_k\rangle$, for example $\langle\overrightarrow{k},a_k\rangle = \langle 3,5,2\rangle$, $\langle\overrightarrow{l},a_l\rangle = \langle 4,4,3\rangle$. In a poset, the least upper bound and the greatest lower bound for two different elements are called join and meet of them. With respect to the partial ordering defined above, $\succeq$, the join for $\langle\overrightarrow{k},a_k\rangle = \langle k_1,k_2,...,k_N,a_k\rangle$ and $\langle\overrightarrow{l},a_l\rangle = \langle l_1,l_2,...,l_N,a_l\rangle$ would be given by $\langle\overrightarrow{k},a_k\rangle \vee \langle\overrightarrow{l},a_l\rangle = \langle\max\{k_1,l_1\},...,\max\{k_N,l_N\},\min\{a_k,a_l\}\rangle$, whereas the meet for the same elements would be $\langle\overrightarrow{k},a_k\rangle \wedge \langle\overrightarrow{l},a_l\rangle = \langle\min\{k_1,l_1\},...,\min\{k_N,l_N\},\max\{a_k,a_l\}\rangle$. As $\mathcal{S}$ is a poset which contains the join and the meet of each pair of its elements, it is called a lattice. In fact, it is a complete lattice as every nonempty subset has a supremum and an infimum which are the join and meet of all the elements contained in the set in respective order. Theorem 1 states the monotonicity result for the value functions:

Theorem 1 *$V_\beta(\overrightarrow{s},l)$ is partially nondecreasing in $\langle\overrightarrow{s},l\rangle$ with respect to the partial ordering $\succeq$ defined above, i.e. $V_\beta(\overrightarrow{s},l)\geq V_\beta(\overrightarrow{r},k)$ if $(\overrightarrow{s},l)\succeq(\overrightarrow{r},k)$.*

Proof: We prove the result by induction on the steps of value iteration where $V_n(\overrightarrow{s},l)$ is the value function in the n^{th} step of the algorithm given by Equation (2). Proof proceeds in a similar manner to the proof of Theorem 3.2 in Papadaki and Powell [12]. The monotonicity result holds for

$$\begin{aligned} V_1(\overrightarrow{s},l) &= V_1(s_1,s_2,...,s_N,l) = \min_{a\in\{0,1,...,L_{max}\}} c(\overrightarrow{s},l,a) \\ &= \min_{a\in\{0,1,...,L_{max}\}}\left\{\frac{pa}{\lambda}+R1_{\{(s_N-a)>0\}}+r(s_N-a)^+ + Q1_{\{(a>l\}}+q(a-l)^+\right\}, \end{aligned}$$

as $c(\overrightarrow{s}, l, a)$ is increasing in s_N and decreasing in l and unaffected by any change in the other elements of the state vector for all $a \in \mathcal{A}$.

Assume now that V_n is partially nondecreasing for $n = 1, ..., k$. Our aim is to prove that V_{k+1} is partially nondecreasing based on this induction argument. The value function at $(k+1)^{st}$ step of the algorithm, V_{k+1} is as follows:

$$V_{k+1}(\overrightarrow{s}, l) = \min_{a \in \{0,1,...,L_{max}\}} \{c(\overrightarrow{s}, l, a) + \beta \sum_{j=1}^{C_{max}} p_j V_k(j, (s_1 - a)^+, ..., (s_{N-1} - a)^+, a)\}.$$

Given that the action space is finite, there exists an optimal action $a^*_{k+1} \in \mathcal{A} = \{0, 1, ..., L_{max}\}$ at which the above minimum is attained. Thus the value function can be rewritten as

$$V_{k+1}(\overrightarrow{s}, l) = c(\overrightarrow{s}, l, a^*_{k+1}) + \beta \sum_{j=1}^{C_{max}} p_j V_k(j, (s_1 - a^*_{k+1})^+, ..., (s_{N-1} - a^*_{k+1})^+, a^*_{k+1}).$$

For any $\langle \overrightarrow{s}, l \rangle \succeq \langle \overrightarrow{s}, l \rangle^- := \langle s_1^-, s_2^-, ..., s_N^-, l^+ \rangle := \langle \overrightarrow{s}^-, l^+ \rangle$,

$$\begin{aligned} V_{k+1}(\overrightarrow{s}, l) &\geq c(\overrightarrow{s}^-, l^+, a^*_{k+1}) + \beta \sum_{j=1}^{C_{max}} p_j V_k(j, (s_1^- - a^*_{k+1})^+, ..., (s_{N-1}^- - a^*_{k+1})^+, a^*_{k+1}) \\ &\geq \min_{a \in \{0,1,...,L_{max}\}} \{c(\overrightarrow{s}^-, l^+, a) + \beta \sum_{j=1}^{C_{max}} p_j V_k(j, (s_1^- - a)^+, ..., (s_{N-1}^- - a)^+, a)\} \\ &= V_{k+1}(\overrightarrow{s}^-, l^+). \end{aligned}$$

Note that the first inequality is a result of

$$c(\overrightarrow{s}, l, a^*_{k+1}) \geq c(\overrightarrow{s}^-, l^+, a^*_{k+1}),$$

and

$$V_k(j, (s_1 - a^*_{k+1})^+, ..., (s_{N-1} - a^*_{k+1})^+, a^*_{k+1}) \geq V_k(j, (s_1^- - a^*_{k+1})^+, ..., (s_{N-1}^- - a^*_{k+1})^+, a^*_{k+1}),$$

for all $j = 1, ..., C_{max}$. Hence, $V_n(\overrightarrow{s}, l)$ is nondecreasing $\forall n \in \{0, 1, ...\}$, which implies that $V_\beta = \lim_{n \to \infty} V_n(\overrightarrow{s}, l)$ is nondecreasing. □

Theorem 1 states that as the initial processing requirements of the items in the processor increase and/or the initial level of the processor decreases, the minimum expected costs incurred increases. This result is quite intuitive as greater states for the items in the processor imply higher required amounts of processing and potentially more under-processing, thus larger energy (material) and penalty costs. When the initial processor level is smaller, switching to a level other than the current one corresponds to higher switching costs for all selected levels, which explains the monotonicity with respect to the current processor level.

Having proven the monotonicity of the value functions, we continue to investigate structural properties of the optimal policy. The existence of a switching cost in an SDP generally results in hysteretic optimal policies (Lu and Serfozo [7], Kitaev and Serfozo [6]). Hysteresis may be defined as the lag between the changes in the system state and application of the optimal action level as a response that would be realized immediately if there were no switching costs at all. In the existence of switching costs, depending on the current action, it may be less costly to stay at the current processor level, or switch to another action level. A more formal definition tailored to our problem is provided below:

Definition 1 *The optimal policy* $f(\overrightarrow{s}, l) = \min\{a \in \mathcal{A} = \{0, 1, ..., L_{max}\} : V_\beta(\overrightarrow{s}, l)\}$ *is hysteretic, if* $f(\overrightarrow{s}, l) = a$ *for some* $l \in \mathcal{A}, l \neq a$ *implies* $f(\overrightarrow{s}, a) = a$, $a \in \mathcal{A}$.

Note that the requirement for hysteretic behavior is that if an action is switched from another one for any state, there will be no switching from that action for the same state. Next we present the hysteresis result for the optimal policy and its proof.

Proposition 1 *Optimal policy,* $f(\overrightarrow{s},l) = \min\{a \in \mathcal{A} = \{0,1,...,L_{max}\} : V_\beta(\overrightarrow{s},l)\}$ *is a hysteretic policy.*

Proof: The proof follows by defining

$$V_n(\overrightarrow{s},l) = \min_{a\in\mathcal{A}}\{\widetilde{Q}(l,a) + W_{n-1}(\overrightarrow{s},a)\},$$

where

$$W_{n-1}(\overrightarrow{s},a) = \frac{pa}{\lambda} + R1_{\{(s_N-a)>0\}} + r(s_N-a)^+ + \beta\sum_{j=1}^{C_{max}} p_j V_{n-1}(j,(s_1-a)^+,...,(s_{N-1}-a)^+,a),$$

$$\overrightarrow{s} = \langle s_1,...,s_N\rangle,$$

and

$$\widetilde{Q}(l,a) = Q1_{\{(a-l)>0\}} + q(a-l)^+, \tag{4}$$

similar to the discussion for the proof of Theorem 1 in Lu and Serfozo [7]. The only part left to show is that the switching cost, $\widetilde{Q}(l,a)$ satisfies the condition

$$\widetilde{Q}(l,a) \le \widetilde{Q}(l,c) + \widetilde{Q}(c,a) \quad \text{for all} \quad a,c,l \in \mathcal{A} = \{0,1,...,L_{max}\}, \tag{5}$$

and

$$\widetilde{Q}(a,a) = 0 \quad \text{for all} \quad a \in \mathcal{A}, \tag{6}$$

which are the conditions in Theorem 1 of Hipp and Holzbaur [5]. It is straightforward that Equation (6) holds and Inequality (5) follows by a simple observation. For $a \le d$, Inequality (5) follows directly as $\widetilde{Q}(d,a) = 0$. For $a > d$, $\widetilde{Q}(d,a) = Q + q(a-d)$ and the result follows from observation of all three subcases:
1) $c > a > d$: $Q + q(a-d) \le Q + q(c-d)$,
2) $a > d > c$: $Q + q(a-d) \le Q + q(a-c)$,
3) $a > c > d$: $Q + q(a-d) \le 2Q + q(a-d)$. □

Next, we prove the monotonicity of the optimal policy with respect to s_N, i.e. the state of the item which is closest to the exit of the processor.

Theorem 2 *Optimal policy,* $f(\overrightarrow{s},l) = f(s_1,...,s_N,l)$ *is nondecreasing in* s_N.

Proof: Observe that one-stage cost, $c(\overrightarrow{s},l,a)$, is submodular with respect to s_N and a, as

$$\begin{aligned} & c(s_1,...,s_N+1,l,a+1) + c(s_1,...,s_N,l,a) - c(s_1,...,s_N,l,a+1) - c(s_1,...,s_N+1,l,a) \\ = \;& r[2(s_N-a)^+ - (s_N-a-1)^+ - (s_N+1-a)^+] + R[2(1_{\{s_N>a\}}) - 1_{\{s_N>(a+1)\}}1_{\{(s_N+1)>a\}}] \\ \le \;& 0. \end{aligned}$$

Consider the optimality equation,

$$V_\beta(s_1,s_2,...,s_N,l) = \min_{a\in\{0,1,...,L_{max}\}}\{c(\overrightarrow{s},l,a) + \beta\sum_{j=1}^{C_{max}} p_j V_\beta(j,(s_1-a)^+,...,(s_{N-1}-a)^+,a)\}.$$

Defining

$$g(s_1,s_2,...,s_N,l,a) = \beta\sum_{j=1}^{C_{max}} p_j V_\beta(j,(s_1-a)^+,...,(s_{N-1}-a)^+,a)\},$$

observe that

$$g(s_1,...,s_N+1,l,a+1) = g(s_1,...,s_N,l,a+1),$$

and

$$g(s_1,...,s_N+1,l,a) = g(s_1,...,s_N,l,a),$$

as g is independent of s_N. Hence, we obtain

$$g(s_1,...,s_N+1,l,a+1)+g(s_1,...,s_N,l,a)-g(s_1,...,s_N,l,a+1)-g(s_1,...,s_N+1,l,a)=0,$$

which implies $g(s_1,...,s_N,l,a)$ is submodular with respect to s_N and a as well. As sum of submodular functions is submodular, we conclude that $c(\overrightarrow{s},a)+g(s_1,s_2,...,s_N,l,a)$ is submodular and the result follows from the fact that optimal actions for submodular functions are monotone for the case of minimization (Topkis [20]).□

The monotonicity result is intuitive due to the fact that increasing the state of the last item in the processor implies higher amount of processing requirement for the item and potentially more penalty cost. This motivates selecting a higher level for any current processor level which in turn leads to higher power (or material) and switching costs.

4.1 A special case with no switching cost

Finally, we analyze a special case of the problem, where $Q=0, q=0, R=0$, i.e. no cost is incurred for switching between different processor levels and penalty cost for under-processing is in linear form. We derive the optimal policy in closed form for this case. Furthermore, we utilize this result for the implementation of efficient heuristic policies later in Section 5.

Theorem 3 *For the case with no switching cost, i.e. $q=0$, $Q=0$ and when the penalty cost for under-processing is a linear function with no fixed cost terms, i.e. $R=0$, the optimal action is given by*

$$a^*=\min\{\max\{s_N,s_{N-1}-L_{max},s_{N-2}-2L_{max},...,s_1-(N-1)L_{max}\},L_{max}\} \qquad (7)$$

if $\beta^{(N-1)}r>\frac{p}{\lambda}$ and $C_{max}<NL_{max}$.

Proof:

The proof is based on induction on the steps of value iteration algorithm. Using Equations (2) and (3),

$$\begin{aligned}
V_1(\overrightarrow{s}) &= V_1(s_1,s_2,...,s_N)\\
&= \min_a\left\{\frac{pa}{\lambda}+r(s_N-a)^+ +\beta E^{d_1}V_0((d_1-a)^+,(s_1-a)^+,...,(s_{N-1}-a)^+)\right\}\\
&= \min_a\left\{\frac{pa}{\lambda}+r(s_N-a)^+\right\}:=\min_a g_1(a).
\end{aligned}$$

As

$$\frac{\Delta g_1(a)}{\Delta a}=\frac{p}{\lambda}-r1_{\{(s_N-a)>0\}},$$

it turns out that

$$a^*=\begin{cases}\min\{s_N,\overline{a}\} & \text{if } r>\frac{p}{\lambda},\\ 0 & \text{if } r<\frac{p}{\lambda}.\end{cases}$$

Assuming $r>\frac{p}{\lambda}$ is satisfied, then

$$V_1(s_1,s_2,...,s_N)=\frac{p\min\{s_N,\overline{a}\}}{\lambda}+r(s_N-\bar{a})^+.$$

Continuing with the second iteration,

$$\begin{aligned}
V_2(s_1,s_2,...,s_N) &= \min_a\left\{\frac{pa}{\lambda}+r(s_N-a)^+ +\beta E^{d_1}V_1((d_1-a)^+,(s_1-a)^+,...,(s_{N-1}-a)^+)\right\}\\
&= \min_a\left\{\frac{pa}{\lambda}+r(s_N-a)^+ +\beta\frac{p}{\lambda}\min\{(s_{N-1}-a)^+,\overline{a}\}+\beta r(s_{N-1}-a-\bar{a})^+\right\}\\
&:= \min_a g_2(a).
\end{aligned}$$

As

$$\frac{\Delta g_2(a)}{\Delta a} = \frac{p}{\lambda} - r(1_{\{s_N - a > 0\}} + \beta 1_{\{s_{N-1} - \overline{a} - a > 0\}}) - \beta \frac{p}{\lambda} 1_{\{0 < s_{N-1} - a < \overline{a}\}},$$

by careful observation of two cases $s_N \geq (s_{N-1} - \overline{a})$ and $s_N < (s_{N-1} - \overline{a})$, the optimal action is given by

$$a^* = \min\{\max\{s_N, s_{N-1} - \overline{a}\}, \overline{a}\},$$

under the sufficient condition $\beta r > \frac{p}{\lambda}$ which becomes effective for the case $s_N < (s_{N-1} - \overline{a})$.

Note that the optimal action becomes an interval as $\beta \to 1$, which in the limit is the case with average cost criterion, given by

$$a^* \in [\min\{\max\{s_N, s_{N-1} - \overline{a}\}, \overline{a}\}, \min\{\max\{s_N, s_{N-1}\}, \overline{a}\}].$$

The value function is

$$\begin{aligned} V_2(s_1, s_2, ..., s_N) &= r[(s_N - \bar{a})^+ + \beta(s_{N-1} - 2\bar{a})^+] + \frac{p}{\lambda} \min\{\max\{s_N, s_{N-1} - \overline{a}\}, \overline{a}\} \\ &\quad + \beta \frac{p}{\lambda} \min\{(s_{N-1} - \min\{\max\{s_N, s_{N-1} - \overline{a}\}, \overline{a}\})^+, \bar{a}\}. \end{aligned}$$

Similarly,

$$\begin{aligned} V_3(s_1, s_2, ..., s_N) &= \min_a \Big\{ \frac{pa}{\lambda} + r(s_N - a)^+ \\ &\quad + \beta r(s_{N-1} - a - \bar{a})^+ + \beta^2 r(s_{N-2} - a - 2\bar{a})^+ \\ &\quad + \beta \frac{p}{\lambda} \min\{\max\{(s_{N-1} - a)^+, (s_{N-2} - a)^+ - \overline{a}\}, \overline{a}\} \\ &\quad + \beta^2 \frac{p}{\lambda} \min\{((s_{N-2} - a)^+ - \min\{\max\{(s_{N-1} - a)^+, (s_{N-2} - a - \overline{a})^+\}, \overline{a}\})^+, \overline{a}\} \Big\} \\ &:= \min_a g_3(a), \end{aligned}$$

and

$$\begin{aligned} \frac{\Delta g_3(a)}{\Delta a} &= \frac{p}{\lambda} - r(1_{\{s_N > a\}} + \beta 1_{\{(s_{N-1} - \overline{a}) > a\}} + \beta^2 1_{\{(s_{N-2} - 2\overline{a}) > a\}}) \\ &\quad - \beta \frac{p}{\lambda} 1_{\{(s_{N-1} - \overline{a}) < a\}} 1_{\{(s_{N-2} - 2\overline{a}) < a\}} (1_{\{s_{N-1} > a\}} + 1_{\{(s_{N-2} - \overline{a}) > a\}}), \end{aligned}$$

which results in

$$a_3^* = \min\{\max\{s_N, s_{N-1} - \overline{a}, s_{N-2} - 2\overline{a}\}, \overline{a}\},$$

assuming $\beta^2 r > \frac{p}{\lambda}$ holds.

For the rest of the discussion, we do not present the terms with p in detail but rather represent them as a function of p considering clarity, relevance and space. The logic follows similar to the first few iterations. Observing the expressions of a_1^*, a_2^* and a_3^*, we propose the induction argument

$$a_n^* = \min\{\max\{s_N, s_{N-1} - \overline{a}, ..., s_{N-n+1} - (n-1)\overline{a}\}, \overline{a}\},$$

where $\beta^{n-1} r > \frac{p}{\lambda}$ and

$$\begin{aligned} V_n(s_1, ..., s_N) &= r[(s_N - \bar{a})^+ + \beta(s_{N-1} - 2\bar{a})^+ ... + \beta^{n-1} * (s_{N-n+1} - (n-1)\bar{a})^+] \\ &\quad + \frac{p}{\lambda} a_n^* + B_n(p), \end{aligned}$$

where $B_n(p)$ represents the remaining terms which include the cost parameter p.

As

$$\begin{aligned} V_{n+1}(s_1, s_2, ..., s_N) &= \min_a \Big\{ \frac{pa}{\lambda} + r(s_N - a)^+ \\ &\quad + \beta r(s_{N-1} - a - \bar{a})^+ + ... + \beta^n r(s_{N-n} - a - n\bar{a})^+ + \widetilde{B}_{n+1}(p) \Big\}, \end{aligned}$$

it turns out that

$$a^*_{n+1} = \min\{\max\{s_N, s_{N-1} - \overline{a}, ..., s_{N-n} - n\overline{a}\}, \overline{a}\},$$

and

$$\begin{aligned} V_{n+1}(s_1, ..., s_N) &= r[(s_N - \bar{a})^+ + \beta(s_{N-1} - 2\bar{a})^+ + ... + \beta^n * (s_{N-n} - (n)\bar{a})^+] \\ &\quad + \frac{p}{\lambda} a^*_{n+1} + B_{n+1}(p), \end{aligned}$$

by careful observation of the difference function.

We conclude the proof by explaining how the value iteration algorithm converges. At the N^{th} iteration the optimal policy will be in the form

$$a^*_N = \min\{\max\{s_N, s_{N-1} - \overline{a}, ..., s_1 - (N-1)\overline{a}\}, \overline{a}\}, \tag{8}$$

and the value function will be

$$\begin{aligned} V_N(s_1, ..., s_N) &= r[(s_N - \bar{a})^+ + \beta(s_{N-1} - 2\bar{a})^+ ... + \beta^{N-1} * (s_1 - (N)\bar{a})^+] \\ &\quad + \frac{p}{\lambda} a^*_N + B_N(p), \end{aligned}$$

under the condition $\beta^{N-1} r > \frac{p}{\lambda}$. By the assumption $C_{max} < N\overline{a}$, the penalty cost term with s_1 drops and V_N becomes

$$\begin{aligned} V_N(s_1, ..., s_N) &= r[(s_N - \bar{a})^+ + \beta(s_{N-1} - 2\bar{a})^+ ... + \beta^{N-2} * (s_2 - (N-1)\bar{a})^+] \\ &\quad + \frac{p}{\lambda} a^*_N + B_N(p). \end{aligned} \tag{9}$$

Finally, at the $N + 1^{st}$ iteration, we have

$$V_{N+1}(s_1, ..., s_N) = \min_a \left\{ \frac{pa}{\lambda} + r(s_N - a)^+ + \beta E^{d_1} V_N((d_1 - a)^+, (s_1 - a)^+, ..., (s_{N-1} - a)^+) \right\}$$

which is independent of d_1 due to Equation (9). As a result, V_{N+1} becomes equivalent to V_N resulting in the same optimal policy in Equation (8).□

Note that Equation (7) implies a policy which selects the minimum action that guarantees that no penalty cost will be incurred at any time period. It compares the processor levels required by each item for the period assuming that the maximum action will be applied in the succeeding periods and selects the maximum level among those. This is a conservative approach where no under-processing is allowed. The conditions of $\beta^{(N-1)} r > \frac{p}{\lambda}$ and $C_{max} < NL_{max}$ make sure that the penalty cost is high enough to ensure the avoidance of any under-processing and the maximum processor level is sufficiently large to completely process items of all possible classes.

5 Heuristic Methods

Due to "curse of dimensionality", it becomes computationally intractable to obtain the exact optimal solution using value iteration algorithm for large-scale problems. The number of states increases exponentially with respect to the number of slots in the processor, N and the maximum class value, C_{max}. For this, we develop some heuristic algorithms that are useful for large instances of the problem and propose a decomposition heuristic which efficiently calculates an approximation to the optimal policy. Following this, we analyze performance of algorithms that we propose in Section 6.

We first describe some naive heuristics in Section 5.1, and discuss the development of a decomposition heuristic in Section 5.2.

5.1 Naive Heuristics

Heuristic 1 (responsive): At every departure/ arrival, the processor level is selected according to:

$$a = a^* = \max\left\{\left\lceil\frac{s_1}{N}\right\rceil, \left\lceil\frac{s_2}{N-1}\right\rceil, ..., s_N\right\}, \tag{10}$$

when the state is $\langle s_1, s_2, ..., s_N, l\rangle$.

Each item in the processor implies a certain level that corresponds to the minimal processing required if the controller was set at that value until its departure. This level is a function of the position of the item in the processor and its current state. Heuristic 1 avoids any penalty for under-processing by choosing the maximum of those values for the items in the processor.

Heuristic 2 (smoothing): One expected downside of Heuristic 1 is that it may lead to frequent switching under certain cases. Heuristic 2 is a modification of Heuristic 1 to account for this and avoid too high switching costs while making sure that each item will be processed until it departs. The decision is made according to:

For state $\langle s_1, s_2, ..., s_N, l\rangle$,
if $a^* = \max\left\{\left\lceil\frac{s_1}{N}\right\rceil, \left\lceil\frac{s_2}{N-1}\right\rceil, ..., s_N\right\} < l$
  set $a = a^*$
else if $l < \min\{\max\{s_N, s_{N-1} - L_{max}, s_{N-2} - 2L_{max}, ..., s_1 - (N-1)L_{max}\}, L_{max}\}$
  set $a = a^*$
else
  set $a = l$.

The *else if* condition comes from the policy explained in Equation (7), which consumes energy and material resources conservatively by applying the minimum processor level required to avoid under-processing. If switching is required, i.e. the *if* condition holds, the new level is determined according to the rule in Heuristic 1 which represents a more reasonable value to avoid frequent switching in future periods.

Heuristic 3 (conservative): Heuristic 3 simply applies the conservative policy given by Equation (7) in Theorem 3.

5.2 The Decomposition Heuristic

Next, we present the ***Decomposition Heuristic*** which is based on solving the problem for a version with a reduced number of slots and calculating the implied actions for each state by propagation of the items in the processor step by step. The maximum of the actions implied by the solution of the reduced problem determines the policy for each state of the original problem. The algorithm for a setting with N slots and a reduced number of slots of N' would work as follows:

1. Calculate the optimal actions for the N' setting, $a^*_{N'}(s_1, ..., s_{N'}, l)$, for all states $\langle s_1, s_2, ..., s_{N'}, l\rangle$, where $l \in \{0, 1, ..., L_{max}\}$ and $s_i \in \{0, 1, ..., L_{max} * N'\}$ $\forall i$, $i = \{1, 2, ..., N'\}$, using the value iteration method with the given cost parameters.
2. For all $\langle s_1, s_2, ..., s_N, l\rangle$, where $s_i \in \{0, 1, ..., C_{max}\}$ $\forall i$, $i = \{1, 2, ..., N\}$, and $l \in \{0, 1, ..., L_{max}\}$, calculate the action to be chosen for the N setting, $a^*_N(s_1, ..., s_N, l)$:
   2.1. Calculate $a^r_{N'}$ for $r = 0, ..., N - N'$ using

$$a^r_{N'} = a^*_{N'}((s_{N-N'-r+1} - r * L_{max})^+, ..., (s_{N-r} - r * L_{max})^+, l). \tag{11}$$

   2.2. Select the maximum of the implied actions:

$$a^*_N(s_1, ..., s_N, l) = \max_r a^r_{N'}. \tag{12}$$

Hence, for each state of the original problem, the algorithm finds out the optimal action implied by the last N' slots in the processor closest to the exit and the current control level based on the

optimal policy calculated for the reduced version of the problem. Next, the items are assumed to be slid by one slot throughout the processor according to the *propagation rule* given by Equation (11) and the optimal action implied for the new arrangement of the last N' slots is calculated. This step is repeated until there are no more items in the processor, i.e. $N - N' + 1$ times. Finally, the action for the considered state of the original problem is computed by selecting the maximum of the implied actions using Equation (12).

We in fact develop different versions of the Decomposition Heuristic by changing the propagation rule in Equation (11) which determines the assumed action level for artificially sliding items throughout the processor. Among them, we keep our assumption in Equation (11), as it always outperforms others in terms of proximity to the optimal solution. Note that according to this rule, items are propagated as if the maximum processor level, L_{max} is applied.

6 Computational Results and Benchmarking

We proceed with a numerical study where we investigate the contribution of the proposed control strategies for a wide range of parameters. We analyze results in terms of total cost savings and energy (or material) savings with respect to current practices. We also present our insights on sensitivity of those savings to different cost parameter values.

For benchmarking purposes, we use a policy driven by performance that we call *traditional policy* as it represents the state of the practice in the process industry. We also compare a subset of our results to an alternative control method proposed for tunnel freezers in Ramakrishnan et al. [16] that we call *alternative policy*. Although exemplified from the food processing industry, those are valid strategies for any process industry with similar setting, as they aim at preventing under-processing at all expenses, which is typically much costlier compared to over-processing of the items. The traditional policy represents the case in which the processor is run at the maximum level regardless of the state of the system. The alternative policy makes sure that the processor is run at a level which is sufficient to completely process the item which has the highest class upon arrival among all the products currently present in the processor. It checks the class of the items at the entrance and exit without keeping record of their states throughout the whole processing. Thus, each item is specified by its class upon arrival, which does not change for the duration of the process. Unlike our approach here, the alternative method in [16] does not also consider a cost formulation of energy (or material) consumption, switching and under-processing.

For small instances of the problem, i.e. the cases with small number of slots in the processor, N and small values of maximum possible class of arriving items, C_{max}, it is possible to compute the optimal policy and cost values using the value iteration algorithm in Section 4 in a reasonable amount of time. For large C_{max} and especially large N, value iteration faces "curse of dimensionality", as the size of the state space, $(C_{max}+1)^N L_{max}$, grows exponentially with increasing N and C_{max}. Hence, we divide our numerical analysis into two main sections: First, we briefly analyze the contribution by the SDP approach for relatively small case examples by employing the value iteration method. This enables us to gain insights for savings generated by the proposed control strategy in small processor settings, which are seen in practices where low volume production is essential for a high quality target. More importantly, we use some of these results later to compare the performance of heuristic algorithms to the optimal policy. We also compare the savings obtained by heuristic methods with respect to the traditional policy for larger instances of the problem in Section 6.2.1.

6.1 Numerical Analysis of SDP Approach via Value Iteration

We conduct two initial sets of experiments for two settings with relatively small number of items in the processor, $N = 3$ and $N = 5$. We set the maximum processor level as $L_{max} = 3$ and $L_{max} = 2$ for the two settings in respective order. We calculate the maximum class possible for an item using $C_{max} = L_{max}N$ so that it is feasible to completely process items of all classes

entering the processor until they exit. The values of parameters related to the processor setting are presented in Table 1.

N	L_{max}	C_{max}
3	3	9
5	2	10

Table 1: Setting Parameter Values Used for Numerical Experiments

p	λ	r	R	q	Q
1,2,4	1	2,4	2,4,8	0.5,1,2,4	1,2,4,8

Table 2: Cost Parameter Values Used for Numerical Experiments

The data for the cost parameters are generated using a factorial design with values as listed in Table 2. This design corresponds to 288 experiments with a different parameter set for each N value considered. The factorial design is considered to obtain a large unbiased set of cost data to conduct a complete analysis of system behavior and performance of the proposed method for the case studied. The computations are based on the assumption that the processor starts empty and running at level zero, which corresponds to the state $\langle 0,0,0,0\rangle$ for the case of $N=3$ and $\langle 0,0,0,0,0,0\rangle$ for the case of $N=5$. Thus, the optimal cost is in fact the value attained by the value function at that state, which is $V_\beta(0,0,0,0)$ and $V_\beta(0,0,0,0,0,0)$ in respective order. Finally, for these sets of experiments, the arrival probabilities are assumed to be uniformly distributed over the set of classes. Note that this is not a requirement for our model or analysis.

We begin by comparing the optimal system-wide cost values against the cost values generated by the traditional and alternative policies. The average and maximum values of percentage total cost savings generated by the optimal policy are summarized in Table 3. (Please refer to the supplemental file for a complete list of results). Both $N=3$ and $N=5$ settings show significant reduction in total system-wide costs with respect to the benchmark policies, where results for the latter are slightly less pronounced. This small drop in system-wide costs savings for $N=5$ case is quite intuitive, as the number of possible processor levels for this setting are less than that of $N=3$, which implies less flexibility for switching. Observing results for all sets of cost parameters, one can see that the percentage cost difference with respect to the traditional policy increases as the values of power (or material) cost parameters increase and switching cost parameters decrease. This implies that the SDP approach becomes more profitable as energy and materials become more expensive and changes suggested by the control activity become less costly, which is quite intuitive as the proposed method conserves energy (or material resources) by adjusting the processor level at the expense of switching. The increase in penalty cost parameters results in slightly less significant discrepancies between the two methods. Note that penalty and switching costs are not incurred by the traditional policy at all and act as constraints on the proposed control policy to maintain a certain quality level and avoid too frequent switching activity.

The cost savings with respect to the alternative policy is slightly less than those with respect to the traditional policy. Similar to the trends observed for the cost difference from the traditional policy, the discrepancy between the costs generated by the optimal and alternative policies as well becomes more pronounced as energy and materials become more expensive and switching becomes less costly, with the exception of the case of relatively low power (or material) costs. When the energy and materials are too cheap, percent savings first become less pronounced with an increase in the cost of switching. However, as the switching cost keeps increasing, that trend is reversed such that the savings become more pronounced again. This can be explained by the fact that the optimal policy has the flexibility to cut switching costs and set the processor level to a high fixed value under the given circumstances, whereas the alternative policy continues with frequent switching as usual. The effect of penalty cost is intuitive, as it is

potentially incurred only by the proposed policy.

	w.r.t. Traditional Policy		w.r.t. Alternative Policy	
	Average	Maximum	Average	Maximum
$N = 3$	12.52	32.35	11.25	26.38
$N = 5$	8.77	22.21	8.44	19.34

Table 3: Percentage Total Cost Savings with respect to the Benchmark Policies

It is worth noting that the alternative policy does not outperform the traditional policy in terms of the total system-wide cost generated for all of the parameter sets, whereas our optimal policy outperforms both under all cases. The alternative policy results in around 1% cost savings for the $N = 3$ case and even less for the $N = 5$ case with respect to the traditional policy on average although the maximum savings are as high as 13% (for $N = 3$ case) and 8% (for $N = 5$ case). It becomes undesirable when switching to higher processor levels becomes more costly relative to the energy (or material) and penalty costs, as it results in frequent switching regardless of cost parameters.

Comparison of total system costs gives insights regarding the overall performance of the proposed SDP method over the current practices in the industry and literature, in terms of power (or material) cost which is a measure of the energy and/or material consumption, penalty cost which is a measure of degradation of quality due to under-processing, and the switching cost, which is a measure of the frequency of the changes due to the control activity itself. As penalty and switching costs act as constraining factors on the proposed method, it is not possible to have a clear idea about the energy and material savings obtained by the optimal policy through total cost comparison. For this, we proceed with a brief discussion of the energy/material savings generated by the optimal solution. The average and maximum energy (or material) savings with respect to the traditional and alternative policies are listed in Table 4. As expected, the energy (or material) cost savings are significantly higher than the total cost savings. The penalty and the switching costs are not incurred by the traditional policy at all, thus consideration of those terms favors it over the SDP approach. The average energy/material savings with respect to the benchmark policies show trends similar to the average system-wide cost savings with higher discrepancies on average as mentioned above. Increase in power (or material) costs creates a tendency to save more energy and materials, and decrease in the switching costs increases the flexibility of the control activity ending up with reduced energy (or material) consumption.

	w.r.t. Traditional Policy		w.r.t. Alternative Policy	
	Average	Maximum	Average	Maximum
$N = 3$	24.25	66.53	11.09	60.71
$N = 5$	15.75	36.73	12.08	34.09

Table 4: Percentage Energy Savings with respect to the Benchmark Policies

6.2 Numerical Analysis of Heuristic Methods

Having clearly demonstrated the benefits of our optimal algorithm, we now compare the performance of heuristics described in Section 5 with respect to the exact optimal solution. We conduct an initial set of experiments with $N = 5$, where value iteration still works and benchmark results from Section 6.1. In Section 6.2.1, we will consider larger N values where obtaining the optimal solution via value iteration becomes intractable. The cost values for the heuristics are obtained by the value iteration algorithm after calculation of the policies implied by those methods. The minimum, maximum and the average values of deviations from the optimal are summarized in Table 5. (For the complete list of results, please refer to the supplemental file).

	Heuristic 1	Heuristic 2	Heuristic 3	Decomposition
Average	12.41	3.73	22.78	1.23
Minimum	7.49	0.04	0.12	0.00
Maximum	41.22	23.74	109.44	5.11

Table 5: Percent Deviations of the Total Costs Generated by the Heuristics from the Optimum Total Cost Value

Observing the cost figures in the complete list of results and Table 5, one can see that Heuristic 2 outperforms Heuristic 1 and Heuristic 3 on average and for all of the parameter sets individually. This can be explained by the ability of Heuristic 2 to alleviate the switching activity as well as minimize energy (or material) consumption and avoid under-processing. Heuristic 3 performs quite poorly for most of the cases as long as the switching cost is not set too low, as it tries to minimize the amount of energy (or materials) spent without limiting the switching activity at all. The Decomposition Heuristic results in the closest cost figures to the optimal on average among all the heuristics.

It is worth noting that Heuristic 2 does not always perform worse than the Decomposition Heuristic although it has a larger average percent deviation from the optimal. As switching cost becomes less significant, i.e. the process control becomes more flexible, the relative performance of Heuristic 2 with respect to the Decomposition Heuristic increases. On the other hand, the Decomposition Heuristic performs better in terms of the total system costs with respect to the traditional and alternative policies for all of the parameter sets considered. Heuristic 2 performs on average better than both of the benchmark policies, even though its performance is not consistent for the complete list of parameters.

In terms of computational efficiency, value iteration algorithm is still able to compute the optimal solution in a reasonable time (less than 3 minutes for all cases) for the setting considered. On the other hand, heuristic algorithms compute the solution in less than a second for all the cases considered, although the Decomposition Heuristic works slightly longer in terms of CPU time compared to the naive heuristics. The complete list of the computational times for value iteration, Heuristic 1 and the Decomposition Heuristic for $N = 5$ setting are presented in the supplemental file. (The computational times for the rest of the naive heuristics are excluded as their results are pretty similar to those of Heuristic 1). We consider cases with larger N in Section 6.2.1 where value iteration does not work due to "curse of dimensionality" and our heuristics are still able to approximate the optimal solution in reasonable amount of computational time.

We continue our numerical analysis with the Decomposition Heuristic and Heuristic 2 excluding other naive heuristic algorithms, as they are outperformed by Heuristic 2 in terms of proximity to the optimal solution for all of the parameter sets. First, we compare their deviations from the optimal on average for different power (or material) and switching cost parameter values. As seen in Figure 2, Heuristic 2 results deviate from the optimal the least when the power (or material) cost is at a moderate level and the switching cost is high. This can be explained by the fact that when switching cost parameters are relatively high with respect to the energy (or material) cost, SDP approach cuts the frequency of changes to the processor level, which is similar to the strategy of Heuristic 2 for any parameter set. The performance is the worst for low switching and low power (or material) cost combination, which can be explained by similar reasoning. For the Decomposition Heuristic, the best performance is observed for the low switching and high power (or material) cost combination with an average percent deviation of around 0.03% from the optimal (see Figure 3). Note that for the parameter classes where the deviations from the optimal are relatively high for the Decomposition Heuristic, the deviations corresponding to the same classes are significantly smaller for Heuristic 2 and vice versa. The relative performance of those two heuristics is almost negatively correlated, which implies that running the best algorithm according to the parameters of a specific setting, the percent deviations from the optimal can be reduced even further (to around 0.84% on average for the case considered).

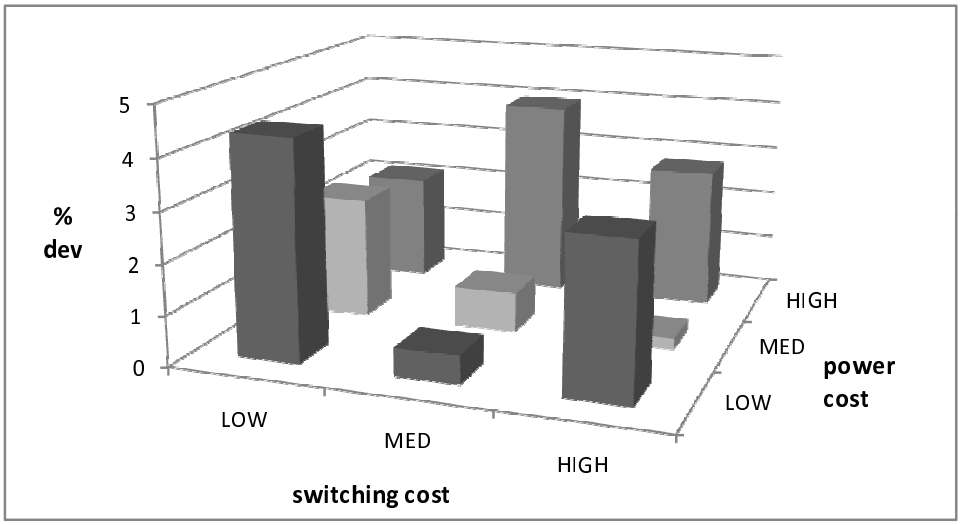

Figure 2: Percent Deviations from the Optimal Cost Value for Heuristic 2

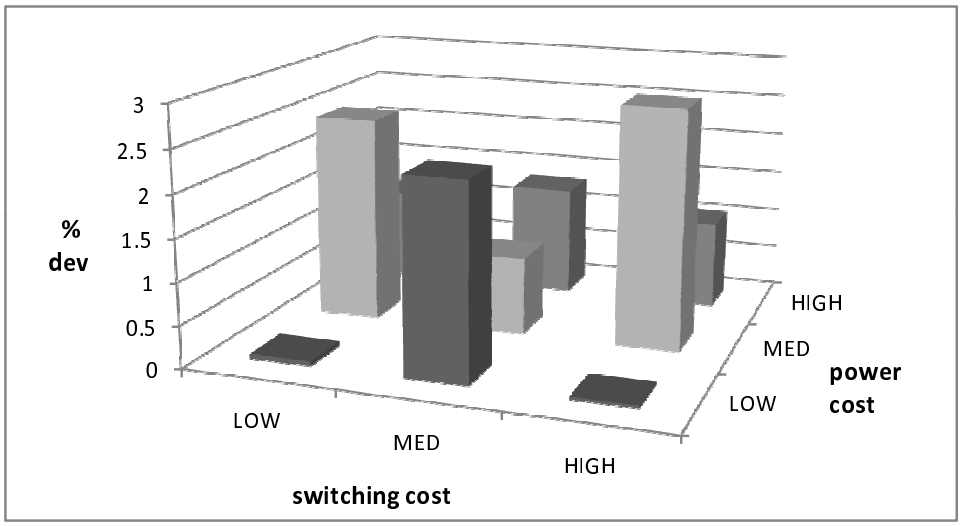

Figure 3: Percent Deviations from the Optimal Cost Value for the Decomposition Heuristic

As far as the percent savings with respect to the traditional policy are considered, Heuristic 2 and the Decomposition Heuristic perform the best for high power (or material) cost and medium switching cost combination (see Figures 4, 5). This is the setting where the motivation to save energy and material resources is the highest and the changes to the processor level are not too costly. Note that Heuristic 2 is outperformed by the traditional policy on average for low power (or material) and low switching cost case, and the savings generated by the Decomposition Heuristic is the lowest for the low power (or material) and high switching cost setting.

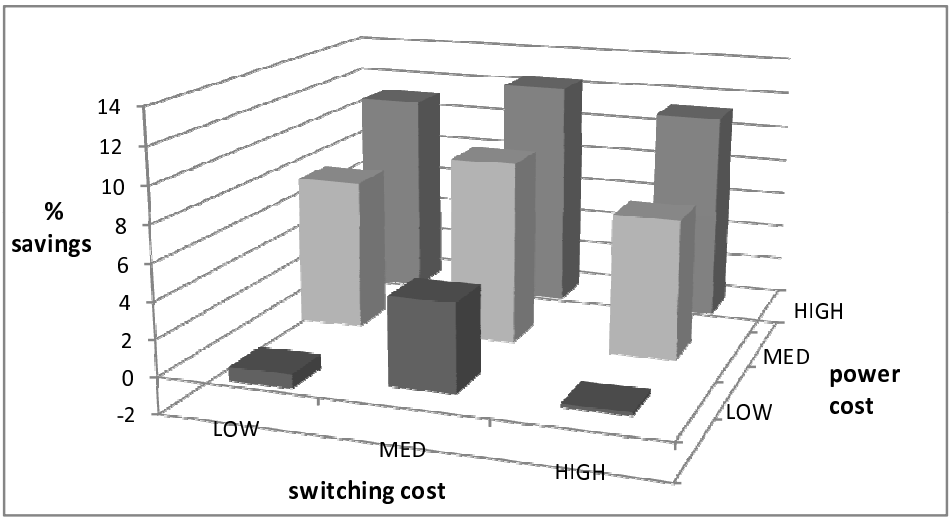

Figure 4: Percentage Cost Savings over the Traditional Policy for Heuristic 2

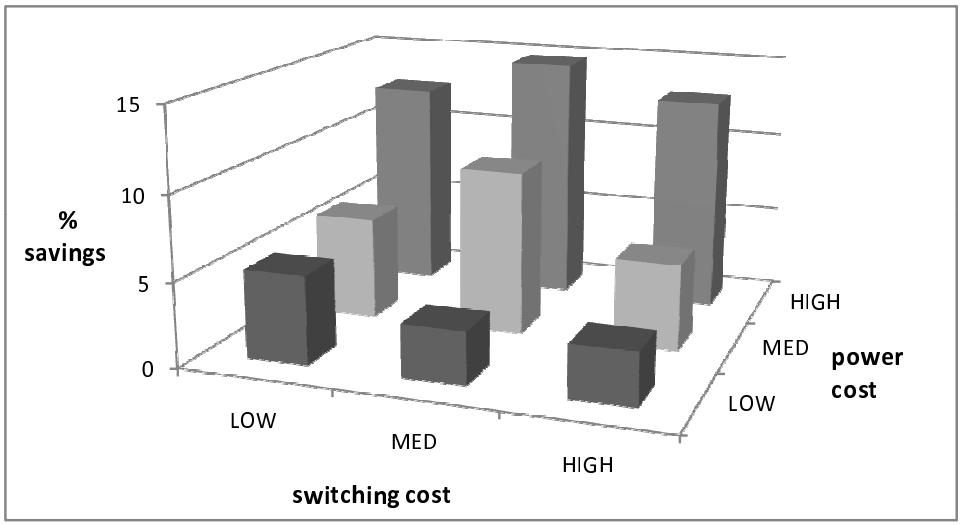

Figure 5: Percentage Cost Savings over the Traditional Policy for the Decomposition Heuristic

In conclusion, the Decomposition Heuristic is capable of approximating the optimal cost solution quite well for almost all the parameter cases. Its performance can be further improved by employing Heuristic 2 for the parameter settings where Heuristic 2 performs relatively better.

6.2.1 Larger Problems

We conclude our numerical study by considering larger processor settings where value iteration becomes computationally intractable. We conduct a set of experiments for the case with $N = 15$, i.e. the number of slots in the processor is 15, which is fairly typical for a quasi-batch processor with a multi-product setting. We also consider a hypothetically larger case with $N = 100$ to test the applicability of the proposed method for even larger instances of the problem. The maximum processor level, L_{max} is 2 for both settings, and the maximum class level for an item is 30 and 200 in respective order. We deviate from the uniform probability assumption considered earlier for the classes of arriving items. In particular, we assume a significant probability of a class zero arrival in a period, p_0, which means on average a certain number of slots in the processor occupy no product at all. This is consistent with the fact that a large-sized processor would typically have highly variable spacing between the items generated by different inter-arrival times of products. The arrival probabilities for the rest of the classes are distributed uniformly from a certain level, C_{min}, to the largest possible class, C_{max}. We also conduct experiments with uniform probability assumption for the sake of completeness and to be able compare results with those in Section 6.2.

As described earlier, due to the size of the state space, it is not computationally tractable to solve these problems via value iteration. We use a combination of the Decomposition Heuristic and Heuristic 2 as suggested in Section 6.2 to approximate the optimal solution and use simulation over multiple runs to generate the cost values for each case. The heuristic algorithm computes solutions in less than 3 minutes for all parameters sets. The average and maximum values for percent total cost and energy (or material) savings with respect to the traditional policy are listed in Table 6. Significant cost savings are generated with more pronounced values for energy (or material) consumption similar to the results for $N = 3$ and $N = 5$ settings. The proposed approach performs better for the case with a significant p_0 value compared to the case with uniform probability assumption. (A complete list of results are provided in the supplemental file for $N = 15$ and $N = 100$ settings with the significant p_0 assumption). The sensitivity of results to the probability distribution of the arrivals is promising in the sense that it is possible to generate further improvement in process efficiency for settings with different input characteristics.

	Cost Savings				Energy Savings			
	Significant p_0		Uniform Probability		Significant p_0		Uniform Probability	
	Average	Maximum	Average	Maximum	Average	Maximum	Average	Maximum
$N = 15$	11.65	19.45	5.96	10.90	18.04	52.24	8.55	24.43
$N = 100$	13.73	16.82	9.68	12.14	15.80	17.65	11.44	13.21

Table 6: Percentage Cost and Energy Savings with respect to the Traditional Policy

7 Conclusions and Future Work

In this paper, we modeled a quasi-batch process with variable input that we call multiple-product by discretizing time, state and action spaces. We formulated and solved a stochastic dynamic program taking into account various cost factors to represent different performance measures of the system in the model developed. We analyzed the model behavior and developed heuristic strategies that are useful to efficiently solve the large instances of the problem. We conducted a numerical study with several experiments for different size instances of the problem to provide insights on the potential cost, energy and material savings and evaluate the performance of proposed heuristics. Our results indicate that by implementing an intelligent control strategy which takes into account system-wide components, it is possible to significantly reduce energy and material consumption compared to a static design. This results in significant cost savings as well as improvements in the environmental performance of the process. Our results also show

that the proposed heuristics approximate the optimal solution extremely well in reasonable computational time.

It is worth noting that dividing the processor into discrete slots would apply to the settings where we have discrete products as well as a continuous product with discrete zones which require different amounts of processing. In other words, by discrete assumption, we imply the change in characteristics of inputs occurring in a discrete manner rather than the products being physically disconnected. In fact the process of micro-fiber surface treatment, which we mentioned in Section 1 constitutes such a case. In addition, we believe as the production processes move to smaller scales with the improvements in micro and nano technologies, capturing the input variability in the processes becomes much more critical.

A future extension for the problem is relaxing the iid assumption for the arrivals. Although addressed in very limited scope in Section 6.2.1, the effect of arrival distributions on the performance of the proposed approach is another matter of interest to be investigated in more detail. Another extension is consideration of an additional line of arrival into the processor by slightly under-processed items.

Acknowledgements

The authors thank the reviewers and editors for their comments and suggestions that led to considerable improvements in the content and presentation of this paper.

References

[1] Carter, C. (2008), "Cool on Energy," *The Engineer 2-15 June 2008*.

[2] Dennis, D. and J. Meredith (2000), "An Emprical Analysis of Process Industry Transformation Systems," *Management Science*, Vol. 46, No. 8, pp. 1085-1099.

[3] Gupta, M. M. and N. K. Sinha (2000), "Soft Computing and Intelligent Systems: Theory and Applications," Academic Press Series in Engineering, San Diego, CA.

[4] Gupta, S. M. and S. D. P. Flapper (1999), "Preface to the Special Issue on the Operational Aspects of Environmentally Conscious Manufacturing," *Computers and Industrial Engineering*, Vol. 36, No. 4, pp. 719-721.

[5] Hipp, S. K. and U. D. Holzbaur (1988), "Decision Processes with Monotone Hysteretic Policies," *Operations Research*, Vol. 36, No. 4, pp. 585-588.

[6] Kitaev, M. Y. and R. F. Serfozo (1999), "M/M/1 Queues with Switching Costs and Hysteretic Optimal Control," *Operations Research*, Vol. 47, No. 2, pp. 310-312.

[7] Lu, F. V. and R. F. Serfozo (1984), "M/M/1 Queueing Decision Processes with Monotone Hysteretic Optimal Policies," *Operations Research*, Vol. 32, No. 5, pp. 1116-1132.

[8] Mattsson, B. and U. Sonesson (2003), "Environmentally-friendly food processing," Woodhead Publishing Limited, Cambridge, UK.

[9] Mittal, G. S. (1997), "Computerized Control Systems in the Food Industry," Marcel Dekker Inc., New York, NY.

[10] Moreno-Lizaranzu M. J., R. A. Wysk, J. Hong and V. V. Prabhu (2001), "A Hybrid Shop-floor Control System for Food Manufacturing,", *IIE Transactions*, Vol. 33, pp. 193-202.

[11] O'Brien, B. (2002), "Global Manufacturing and the Sustainable Economy," *International Journal of Production Research*, Vol. 40, No. 15, pp. 3867-3877.

[12] Papadaki, K. and W. B. Powell (2007), "Monotonicity in Multidimensional Markov Decision Processes for the Batch Dispatch Problem," *Operations Research Letters*, Vol. 35, pp. 267-272.

[13] Paulauskas, F. L., T. L. White, T. S. Bigelow (2003), "Microwave and Plasma-Assisted Modification of Composite Fiber Surface Topography," *United States Patent*, No. US 6,514,449 B1.

[14] Paulauskas, F. L., K. D. Yarborough, T. T. Meek (2002), "Carbon Fiber Manufacturing via Plasma Technology," *United States Patent*, No. US 6,372,192 B1.

[15] Puterman, M. L. (1994), "Markov Decision Processes: Discrete Stochastic Programming," John Wiley & Sons Inc., New York, NY.

[16] Ramakrishnan, S., Gautam, N., and Wysk, R. A. (2002), "Tunnel Freezing Process with Adaptive Control: A Stochastic Modeling Approach," *Proceeding of the Industrial Engineering Research Conference*.

[17] Ramakrishnan, S., R. A. Wysk, V. V. Prabhu (2004), "Prediction of Process Parameters for Intelligent Control of Tunnel Freezers Using Simulation," *Journal of Food Engineering*, Vol. 65, pp. 23-31.

[18] Rossi, T., C. Noe, A. Sianesi (2008), "Modelling Hybrid Production Systems through the ACD Specification: A Case Study in the Fibre-Glass Industry," *International Journal of Production Research*, Vol. 46, No. 8, pp. 2033-2059.

[19] Stuart, J. A., J. C. Ammons, L. J. Turbini (1999), "A Product and Process Selection Model with Multidisciplinary Environmental Considerations," *Operations Research*, Vol. 47, No. 2, pp. 221-234.

[20] Topkis, D. M. (1978), "Minimizing a Submodular Function on a Lattice," *Operations Research*, Vol. 26, No. 2, pp. 305-321.