Proceedings of the 2006 American Control Conference
Minneapolis, Minnesota, USA, June 14-16, 2006

WeA16.1

A Decision-Friendly Approximation Technique for Scheduling Multipurpose Batch Manufacturing Systems

W. Weyerman, D. West, S. Warnick

Information and Decision Algorithms Laboratories
Department of Computer Science, Brigham Young University, Provo, UT 84602
http://idealabs.byu.edu

Abstract— **This paper presents a method for computing sub-optimal schedules for certain complex manufacturing systems. These systems are used to produce various products using a fixed factory infrastructure. Such systems can be relatively easy to model and simulate, but computing optimal schedules for them is often intractable. Here, we sample a simulation of the complex system to develop an integer program approximating the complex relationship between admissible schedules and their overall performance, measured in terms of system throughput. Solving this integer program yields a sub-optimal schedule that seems close to optimal on examples.**

I. Introduction

Multipurpose batch manufacturing systems (BMS) are used by various industries to produce different products using the same factory infrastructure. These systems are typically modeled as a sequence of processes necessary to manufacture the desired products. Optimally scheduling factory resources to deliver a specified suite of products turns out to be a hard problem. Nevertheless, understanding this problem is necessary to quantify the capacity, and hence the profitability, of the manufacturing system.

Early work simplified the problem by considering the scheduling of a single machine [3], [6] and [7]. This problem naturally generalizes to the flexible job shop which processes several different jobs with different routes and allows for multiple machines at any workstation. The BMS considered in this paper differs from the job shop in that in a job shop, a job needs processing only on a single machine in a workstation: workstations are not allowed to have different capacities. Nevertheless, this is a property of batch manufacturing. A graphical solution to the general multipurpose batch plant is given in [9] which works well for simple examples. However, when there are machines of drastically different sizes or complex recipes, the problem grows to an intractable size. Others such as [8] and [5] use a graphical representation of a multipurpose batch plant to derive a mixed integer linear program (MILP) formulation to determine the exact answer to solve several objectives. However, since these methods are exact, they are also computationally intractable for a complex manufacturing system. Two heuristic methods of solution to the minimum makespan problem are given in [1], the better of the two reduces decision variables in the MILP by a linear factor. Although this does allow for the solution of much more complex problems, it is still difficult to compute the makespan for a manufacturing system containing a workstation with a much larger capacity or longer processing time than another workstation.

This work is supported in part by grants from ATK Thiokol, Sandia National Laboratories, and the Office of Research and Creative Activities at BYU. Please direct comments and questions to W. Sam Weyerman at `wsweyerman@gmail.com` or Sean Warnick at `sean@cs.byu.edu`.

We develop a novel decomposition method that simplifies the BMS to a single machine with sequence dependent setup costs. This problem is known to be similar to the Traveling Salesperson Problem (TSP). We propose an integer programming formulation to then solve this simpler problem. The reduction approximates a large optimization problem with a significantly smaller problem. This allows for a sub-optimal solution to the actual problem by offering a tractable computational approach.

II. Problem Description

A. Notation and Definitions

The fundamental processing unit in our manufacturing system is the machine. A *machine* is a resource that performs a specific job in processing various products. Common examples include mixers, extrusion presses, air dryers, heaters, etc. Note that the resource represented by a machine has different characteristics depending on the product it is being used to produce. For example, when manufacturing one product, a boiler may only process 5 kgs of material for 1 hour. Nevertheless, while manufacturing another product, the same boiler may process 10 kgs for 24 hours. Thus, the availability of the resource represented by a single boiler may be very different depending on the product to which it is assigned.

A manufacturing system, or *factory*, then, is a finite set of machines $\mathcal{F} = \{M_1, M_2, ..., M_n\}$. Because a factory often has multiple machines that perform the same job, we partition this set into a set of *workstations*, $\{W_1, W_2, ..., W_p\}$, where $p \leq n$. The set of workstations characterizing a given manufacturing system define the factory's functional capabilities, while the machines assigned to each workstation indicate the amount of that particular resource available at the factory.

Any particular manufacturing system represents a collection of resources that are capable of producing a set of products $\mathcal{P} = \{P_1, P_2, ..., P_m\}$. Each product is characterized by a *recipe* that defines a sequence of resources that are required to manufacture the product. Specifically, the recipe for product i is a triple $\mathcal{R}_i = \{S_i, B_i, T_i\}$, where S_i is a sequence of o_i workstations, indicating the sequence of processing steps needed to create the product i; B_i is a sequence of o_i vectors, where the j^{th} entry of the k^{th} vector is a rational number, b, that indicates the batch size of product i on machine j of workstation $S_i(k)$ during production step k of product i; and T_i is a sequence of o_i vectors, where the j^{th} entry of the k^{th} vector is a real number, τ, indicating the amount of time machine j of workstation $S_i(k)$ would be occupied during production step k of product i. Thus, the manufacture of product i may require a sequence of processing steps that revisits the same workstation multiple times, leading to a total number of processing steps o_i that is larger than the total number of workstations at the factory. Moreover, this sequence of processing steps may skip other workstations altogether. Likewise, note that not only may the batch sizes and processing times of the same machine be different when processing different products, but they may also change from one processing step to another in the manufacture of a single product if the recipe recirculates to the machine multiple times.

B. Operational Constraints

There are three constraints that characterize the particular class of manufacturing systems that we consider here. These constraints are hard constraints, imposed by either the production process itself, or by management to satisfy some competing objectives such as safety requirements, regulatory standards, etc. These are:

1) A machine must be loaded to capacity before it is allowed to run. As a result, the recipes of the products are determined such that a machine's capacity is a rational multiple of the previous machine's capacity. There can be many reasons for this kind of constraint, including that the partially processed product will not develop correctly except as a full batch, that polluting emissions become unacceptable if partial batches are processed, etc.
2) We only consider the no intermediate storage (NIS) queuing policy. Thus any machine must wait for the next machine to be available before it can unload any processed product. This constraint complicates the analysis of the factory since buffers between stages of production are removed, thus tightly connecting the behavior from one machine to the next. Nevertheless, this constraint is also fundamental to various manufacturing systems. For example, a "hot ingot" system requires processing to occur immediately, and a waiting period or inter-processing queue would drastically affect the production process. Likewise, safety regulations may prevent any kind of stockpiling certain combustible materials between processing steps, thus forcing machines to hold their product until the next workstation in the production sequence becomes available.
3) No preemption is allowed, meaning that once a product has begun processing, it must run to completion. This is consistent with the NIS queuing policy, as any preempted material would have no storage and would have to be discarded.

C. Simplifying Assumptions

In addition to operational constraints that narrow the scope of the manufacturing systems we consider, we also impose some simplifying assumptions that facilitate our analysis and make exposition more clear. Unlike the operational constraints, however, these assumptions are not essential to problem definition nor or they critical to our results. In particular, our assumptions are:

1) All recipes end their production sequence at the same workstation. This assumption is made without a loss of generality since we can always augment any recipe to include a "final" workstation, such as storage of the final product.
2) All machines in the same workstation are identical. This assumption simplifies notation and the data structures used to store recipe information, but it is not essential. Note in particular that some machines in the workstation may process one product while the others work on an entirely different product.
3) The only schedules considered are permutation schedules. Thus, once a decision has been made to manufacture a sequence of products, resources are assigned to the processing of those products in the same order as they were scheduled. Note that this assumption is, in many cases, restrictive since parallel machines at workstations could allow one product to be held dormant while another "passes" it in the production sequence. Nevertheless, we impose this assumption in this work and leave its relaxation for future research.
4) The factory works as designed. That is to say, machines are reliable and do not break down, nor does their performance in processing products deteriorate over time. The recipes, then, will produce exactly the desired products, and no quality control is needed to tune recipes or adapt to machine failure, etc. Clearly our results can be extended to consider stochastic models of failure rates or time variation of production processes, but this is left for future work.

From these assumptions, we can then develop some specialized definitions that are meaningful for discussing this particular class of production environments. In particular, given no intermediate queues and our restriction to permutation schedules, a decision to process product i begins with loading relevant materials into the machines at the first workstation of product i's recipe. From there, these materials will pass from workstation to workstation as defined by the recipe until product i is processed on the final workstation. Nevertheless, since each workstation operates on a possibly different batch size of materials, and since the workstations operate only when full, it may be the case that many batches of product i must be processed in tandem before some machine in the production sequence is filled and can begin processing. Thus, the concept of a load for each product becomes meaningful. A *load* of a product is the minimum amount of material required to complete processing of that product in the factory. A load will always be an integer multiple of the largest batch size of any machine used in the production sequence. Note that the permutation schedule assumption and the no-preemption constraint imply that loads can not be interrupted; all operational decisions of the factory boil down to scheduling a sequence of loads of various products, which we write as $u = \{u(0), u(1), ...\}$, where $u(k) \in \mathcal{P} = \{P_1, P_2, ..., P_m\}$ and means that the k^{th} load processed by the factory will be a load of product $u(k-1)$.

Another useful concept is the notion of product dominance of the factory; a product is said to be *dominating* the factory while the final workstation is processing its load. A product's *runtime* at load or step k, $r(k)$, is the amount of time that that load dominates the factory when processed at step k. Finally, the *idle time of transition* k, $\Delta(k)$, is the idle time of the final workstation once the $k-1$ load of any product leaves the machine until the k^{th} load enters the machine and begins dominating the factory.

Moreover, under the assumption that all the machines in the same workstation are identical, we can consolidate the recipe information for the entire production set $\mathcal{P}$ in three $m \times o_{max}$ matrices, where o_{max} is the length of the longest production sequence over all products. Note that this assumption implies that the vector sequences $B_i(k)$ and $T_i(k)$, where k is the processing step, can be replaced by sequences of numbers $b_i(k)$ and $\tau_i(k)$, since a distinction between different machines at the same workstation is no longer meaningful. We define the i^{th} row of the first matrix, O_w, to be the workstation sequence S_i, appended with zeros to match the length of the longest sequence. Likewise, let the i^{th} row of the matrices O_b and O_τ be the batch sequence b_i and the time sequence τ_i, each appended with zeros as necessary. Thus, the collection of recipes for all products supported by the specific manufacturing system define the factory's *admissible operations* $O = \{O_w, O_b, O_\tau\}$. Equipped with these definitions, we can now formulate the problem and develop its solution.

D. Problem Formulation

This general framework sets the stage for our analysis of manufacturing systems. The focus of the analysis is on the interaction between the factory resources (machines grouped into workstations) and products (characterized by recipes). In particular, we are interested in the production capacity of the factory, and the optimal schedule realizing this maximal throughput.

The flexibility of the manufacturing system, however, to produce multiple products suggests that throughput alone is not sufficient to meaningfully answer the capacity question. Clearly throughput may change drastically depending on which product is being produced, so the capacity question is meaningful only with respect to a particular quota. Let $q = \begin{bmatrix} q_1 & q_2 & \ldots & q_m \end{bmatrix}^T$, where q_i is an integer number of loads of product $i \in \mathcal{P}$ be the desired quota of each of the m products manufactured by the factory. The schedule to maximize throughput then becomes the same schedule to minimize the amount of time required to produce q. Note that the total number of manufacturing loads is $K = |q|_1$, which also defines the total length of the scheduling sequence $u(k)$, $0 \leq k \leq K-1$. We say u is an *admissible schedule* if the total number of loads scheduled for product i equals q_i.

Given the constraints and assumptions imposed on the manufacturing system, it is clear that a given finite production sequence $u(k)$ will generate an output sequence in the same order. What is not clear is the completion time of this final output sequence. Let $y(k) = \Delta(k) + r(k)$ be the work time of the final workstation associated with load k. The total completion time is then simply $\sum_{i=0}^{K-1} y(k)$.

We observe that the factory can be thought of as a complex dynamic system mapping $u(k)$ to $y(k)$. The state of the system at stage k is a vector $z \in \mathbb{R}^n$, where $z_i(k)$ is the amount of time from when load k enters the first workstation in its recipe until machine i completes all processing associated with load k. The initial state of the factory, $z(0)$, thus indicates the times when factory resources (machines) become available for processing the first load $u(0)$. The action of a production schedule $u(k)$ on the factory then becomes

$$\begin{array}{rcl} z(k+1) & = & f(z(k), u(k)) \\ y(k) & = & g(z(k), u(k)) \end{array} \tag{1}$$

for some complex functions f and g. We thus have the problem: given a factory, a production set $\mathcal{P}$ with their associated recipes $\mathcal{R}$, and a quota q, find an admissible schedule u such that

$$\begin{array}{rl} \min\limits_{u} & \sum\limits_{k=0}^{K-1} y(k) \\ \text{subject to} & z(k+1) = f(z(k), u(k)) \\ & y(k) = g(z(k), u(k)). \end{array} \tag{2}$$

Note that the complexity of f and g make it very difficult to determine an optimal schedule u, even when it may not be terribly difficult to simulate (f, g) given a particular candidate schedule. We capitalize on this fact to develop a discrete event realization of (f, g) in the next section. This is then followed by an approximation method that samples the simulation to characterize certain transition costs, yielding an integer program to compute a suboptimal schedule, approximating a solution to (2).

III. Simulation of the Factory

In order to simulate the factory, we model it as a discrete event system. Initially each machine can be viewed as a finite-state automata with a set of states Θ containing three states: loading (θ_l), running (θ_r), and unloading (θ_u). These states can be defined in terms of the portion of a full batch the machine contains, $\ell \in [0, 1]$, and the time the machine has been running, $T \in \mathbb{Z}^*$. The current state, θ is defined as follows:

$$\theta = \begin{cases} \theta_l & when \quad \ell \leq 1 \quad T = 0 \\ \theta_r & when \quad \ell = 1 \quad T < \tau \\ \theta_u & when \quad T \geq \tau. \end{cases}$$

Assume a factory with n machines that can manufacture m different products. In order to model the factory as a dynamic system, let the state of a machine be defined by $[\ell \ T \ \pi \ o]'$, where ℓ and T are defined as before, $\pi \in \mathbb{N}$ is the current product in the machine, and $o \in \mathbb{N}$ is the operation step of the product in the machine. Let $R(t) = [r_1(t) \ \ldots \ r_m(t)]', r_i(t) \in \mathbb{Z}^*$ be the remaining supply, or raw materials, for each product given in number of batches of the first workstation for this product's route for each time $t \in 0, 1, 2, \cdots ....$ Also, let $\Psi = [\psi_1 \ \ldots \ \psi_m]', \psi_i \in \mathbb{Z}^*$ be the amount of each product completed, given in batches of the final machine in the product's route. Allowing for workstations requires a trivial mapping from machine number to workstation number. The schedule, u, creates a sequence, R_K, where K is the number of elements in the schedule, by setting R_{i,u_i} to the number of batches of the first workstation in a load of the product at step i, $u(i)$ and all other values of R_i set to 0. Define $d \in \mathbb{N}$ as the index into R, where $d(0) = 1$.

We will first handle the update rules of general machines, we will handle special cases next. The amount possibly loaded into a machine that is not in the first workstation of the current product's route is given by:

$$\ell_i^+(t+1) = \ell_i(t) + \min\left\{1 - \ell_i(t), \frac{\ell_{prev}(t)}{O_{b,\pi,prev}}\right\}.$$

The amount possibly unloaded from a machine that is not the last machine of the current product's route is given by:

$$\ell_i^-(t+1) = \ell_i(t) - \min\left\{\ell_i(t), (1 - \ell_{next}(t))(O_{b,\pi,next})\right\}.$$

If the machine is in the last workstation of the current product's route ($o_i = o_{final}$), it unloads all of its product. So the load of machine i changes according to:

$$\ell_i(t+1), = \begin{cases} \ell_i^+(t+1), & \theta_i = \theta_l \ and \ \theta_{prev} = \theta_u \\ \ell_i^-(t+1), & \theta_i = \theta_u \ and \ \theta_{next} = \theta_l \\ 0, & o_i = o_{final} \\ \ell(t) & otherwise. \end{cases}$$

When machine i is running, T is incremented at each time step. If the machine is empty, T is reset to 0. The update function for T is given by:

$$T_i(t+1) = \begin{cases} T_i(t) + 1 & \theta_i = \theta_r \\ 0 & \ell_i = 0 \\ T_i(t) & otherwise. \end{cases}$$

The values for π and o are set when loading a product from machine i to machine j (this implies that machine j is not in the first workstation of the current product's route) the update functions for π and o are:

$$\pi_j(t+1) = \begin{cases} \pi_i(t) & \theta_i = \theta_u \ and \ \ell_j = 0 \\ \pi_j(t) & otherwise \end{cases}$$

$$o_j(t+1) = \begin{cases} o_i(t) + 1 & \theta_i = \theta_u \ and \ \ell_j = 0 \\ o_j(t) & otherwise. \end{cases}$$

The storage, Ψ is incremented whenever a machine in the last workstation of the current route is unloaded, so

$$\psi_i(t+1) = \{ \ \psi_i(t) + 1,$$

A machine in the first workstation of a product's route loads from R. Say the non-zero location in $R(t)$ (at most there will be one) is a. Let the machine in the first workstation be i, then machine i is updated as follows:

$$\ell_i(t+1) = \begin{cases} 1 & \theta_i = \theta_l \\ \ell_i^-(t+1), & \theta_i = \theta_u \ and \ \theta_{next} = \theta_l \\ \ell(t), & otherwise \end{cases}$$

$$\pi_i(t+1) = \begin{cases} a, & \theta_i = \theta_l \\ pi_i(t), & otherwise \end{cases}$$

$$o_j(t+1) = 1$$

R is updated as follows:

$$r_i(t+1) = \begin{cases} U_{d+1,i}, & R(t) = 0 \\ r_i(t) - 1, & \theta_j = \theta_l \quad d(t) = d + 1. \\ r_i(t), & otherwise. \end{cases}$$

Using this simulation we can then calculate each transition's idle time, $\Delta(k)$. $\Delta(k)$ is the idle time of the final workstation once the entire load of the product scheduled at step $k-1$, $u(k-1)$, has left the machine until the product scheduled at step k, $u(k)$, begins using the machine, or dominates the factory. Note that although the final workstation may be idle at various times during the production of a given load for a specified product, this idle time does not contribute to $\Delta(k)$, which really captures the inefficiency related to load transitions. $r(k)$ follows simply as the total amount of time that the product scheduled at step k, $u(k)$, dominates the factory.

Finally, an important issue for such general manufacturing systems is deadlock. Deadlock is discussed in [10] and [4]. Four conditions for deadlock are given in [2] and heuristic methods for avoiding and preventing deadlock are given in [11]. We implemented a method of deadlock prevention by preventing one of the conditions necessary for deadlock. To do this, we define a linear ordering of the workstations. We then only allow products to occupy workstations in an increasing order. If the recipe dictates that a product use a machine in a decreasing order, we require that all workstations of lower order be reserved, meaning that there must always be one free machine in that workstation, until the product has completed processing of the earlier stages. Although this is a conservative policy, it will prevent deadlock from occurring.

IV. Method of Solution

In order to determine a schedule we reduce the factory to a model of a single-machine manufacturing system with sequence dependent setup times. We define a *transition cost* to be the one-step approximation of the idle time of the transition plus the run time of product transitioned to. That is, to calculate the transition cost from product i to product j, we construct the schedule $u = (i, j)$ and compute $\Delta(1) + r(1)$ for this schedule, this is the transition cost.

The single-machine problem with sequence dependent setup times is known to be a combinatorial problem solved as the TSP as discussed in [6]. The formulation we derive is based on the IP formulation of the TSP given in [12]. We will view scheduling as traversing a graph. If a load of product i is followed by a load of product j, it will be represented by an arc from node i to node j. The cost of transitioning from node i to node j will be given by the transition cost from product i to product j. We wish to minimize total transition cost while still producing a set amount of loads of each product. The derivation of the IP formulation is given.

Let the matrix C be the cost matrix for a directed graph representing the transition costs of every product that we desire to make where c_{ij} is the transition cost from product i to product j. If a non-cyclical schedule is desired, this graph must contain a dummy node (0) which represents the empty factory. Because there is no product to transition to when emptying the factory, we define the transition cost from any product to the empty factory to be 0. In the case of a non-cyclical schedule, C is a $[m+1 \times m+1]$ matrix. The matrix X specifies which arcs to traverse to reach quota and is the same size as C. If we let x_{ij} represent the number of times the arc from node i to node j is traversed and c_{ij} the cost of that traversal, the objective function for the IP problem is thus:

$$\min_X \sum_{j=0}^{m} \sum_{i=0}^{m} c_{ij} x_{ij}.$$

Because we want a tour, each node must have the same number of incoming arcs as outgoing arcs. This adds the constraint:

$$\sum_{i=0}^{m} x_{ij} - \sum_{l=0}^{m} x_{jl} = 0; \quad j = 0, 1, \ldots, m.$$

Let q be the quota vector. We will set the quota of node 0 to be 1. The following constraint is added:

$$\sum_{i=0}^{m} x_{ij} = q_j; \quad j = 0, 1, \ldots, m.$$

The values of X are limited to be integer values because they are the number of times each arc is to be traversed. The full IP problem is:

$$\begin{aligned} \min_X \quad & \sum_{j=0}^{m} \sum_{i=0}^{m} c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{i=0}^{m} x_{ij} - \sum_{l=0}^{m} x_{jl} = 0; \\ & j = 0, 1, \ldots, m \\ & \sum_{i=0}^{m} x_{ij} = q_j; \\ & j = 0, 1, \ldots, m \\ & x_{ij} \in \{0, 1, \ldots\}. \end{aligned} \tag{3}$$

However, this formulation allows for disjoint sets. Let ς be the smallest subtour or disjoint set, ς is a set containing each arc in the subtour. Since we want to find a single tour, this problem may be iteratively solved adding a constraint each iteration to break the smallest subtour, ς, of length ς_{length}:

$$\sum_{i,j \in \varsigma} x_{ij} < \varsigma_{length} \tag{4}$$

Once there is a single tour, the matrix X will represent a directed graph and will specify the number of times each arc is to be traversed. An algorithm to determine a schedule based on X is given in Table I.

```
• Initialize $i := 0$, $j := 1$
• For every arc in $X$ do:
    – while $x_{ii}! = 0$
        ∗ Add($S_j$, $i$) and decrement $x_{ii}$
    – if $X$ contains no more outgoing arcs from $i$
        ∗ $i$ = Pop($Q$)
        ∗ increment $j$
    – else if $X$ contains more than one outgoing arc from $i$
        ∗ Push($Q, i$)
    – Add($S_j$, $i$)
    – Arbitrarily traverse an available arc and remove that arc from the graph
• For all $S_k$, $k \in \{2, \ldots, j\}$ do:
    – if length($S_i$)! = 1
        ∗ find $S_j$ such that front($S_i$) $\in S_j$
        ∗ replace occurrence of front($S_i$) in $S_j$ with $S_i$
    – delete $S_i$
```

TABLE I

ALGORITHM FOR DETERMINING A SCHEDULE BASED ON X.

V. EXAMPLE

Let a factory with 11 different workstations and nine products be defined as follows (all times are in minutes):

$$F : \begin{pmatrix} 2 & 2 & 2 & 2 & 6 & 3 & 5 & 2 & 1 & 1 & 1 \end{pmatrix}$$

$$O_b : \begin{pmatrix} 1 & 0.0833 & 2 & 1 & 138 & 5 & 0.2 & 0.1 & 75 & 0.0133 & 0 & 0 \\ 1 & 0.0833 & 2 & 138 & 5 & 0.2 & 0.0909 & 1 & 20 & 0.05 & 0 & 0 \\ 1 & 0.0833 & 2 & 630 & 0.2 & 0.1 & 1 & 20 & 0.05 & 0 & 0 & 0 \\ 1 & 0.0833 & 2 & 120 & 5 & 0.2 & 0.1 & 1 & 20 & 0.05 & 0 & 0 \\ 1 & 0.0833 & 2 & 1 & 120 & 5 & 0.2 & 0.0909 & 55 & 0.02 & 20 & 0.05 \\ 1 & 0.0833 & 2 & 1 & 16 & 20 & 0.05 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0.0833 & 2 & 1 & 75 & 5 & 0.2 & 10 & 0.0133 & 0 & 0 & 0 \\ 1 & 0.0833 & 2 & 1 & 75 & 5 & 0.2 & 10 & 0.0133 & 0 & 0 & 0 \\ 1 & 0.0833 & 2 & 120 & 0.1 & 20 & 0.05 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$O_c : \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 9 & 10 & 11 & 0 & 0 \\ 1 & 2 & 3 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 0 & 0 \\ 1 & 2 & 3 & 6 & 7 & 8 & 9 & 10 & 11 & 0 & 0 & 0 \\ 1 & 2 & 3 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 0 & 0 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 6 & 9 & 10 & 11 \\ 1 & 2 & 3 & 4 & 7 & 10 & 11 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 10 & 11 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 10 & 11 & 0 & 0 & 0 \\ 1 & 2 & 3 & 5 & 9 & 10 & 11 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$O_\tau : \begin{pmatrix} 40 & 3 & 6 & 4 & 7080 & 11520 & 1560 & 60 & 786 & 20 & 0 & 0 \\ 40 & 3 & 13 & 2160 & 8640 & 480 & 15 & 60 & 786 & 20 & 0 & 0 \\ 50 & 3 & 13 & 2880 & 1920 & 60 & 60 & 786 & 20 & 0 & 0 & 0 \\ 45 & 3 & 13 & 2640 & 17280 & 360 & 60 & 60 & 786 & 20 & 0 & 0 \\ 45 & 3 & 11 & 20 & 2640 & 8640 & 420 & 60 & 2880 & 60 & 786 & 20 \\ 45 & 3 & 11 & 20 & 2880 & 786 & 20 & 0 & 0 & 0 & 0 & 0 \\ 30 & 3 & 12 & 18 & 1620 & 1440 & 240 & 786 & 20 & 0 & 0 & 0 \\ 30 & 3 & 12 & 5 & 2160 & 5280 & 540 & 786 & 20 & 0 & 0 & 0 \\ 40 & 3 & 13 & 7200 & 60 & 786 & 20 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$Q : \begin{pmatrix} 1 & 3 & 2 & 2 & 2 & 2 & 1 & 1 & 3 & 2 \end{pmatrix}$$

The transition cost matrix, C, for these products was calculated to be:

$$\begin{pmatrix} 0 & 43687 & 42499 & 21754 & 36791 & 35453 & 38350 & 12905 & 15425 & 10899 \\ 0 & 21107 & 25795 & 11725 & 13779 & 13196 & 35221 & 2654 & 2987 & 2355 \\ 0 & 21598 & 25795 & 11725 & 14639 & 16066 & 36210 & 2654 & 2987 & 2345 \\ 0 & 31646 & 30318 & 11725 & 24658 & 23750 & 36210 & 2654 & 3673 & 2345 \\ 0 & 16107 & 25795 & 11725 & 11725 & 16066 & 36210 & 2654 & 2987 & 2345 \\ 0 & 26199 & 25795 & 11862 & 20107 & 12725 & 35221 & 2654 & 2987 & 2358 \\ 0 & 39523 & 38321 & 17540 & 32610 & 31306 & 36396 & 8765 & 11276 & 6747 \\ 0 & 37903 & 36621 & 17387 & 30879 & 29689 & 35871 & 7125 & 9625 & 5137 \\ 0 & 33169 & 31983 & 13314 & 26278 & 25012 & 35871 & 2654 & 4938 & 2347 \\ 0 & 35287 & 33422 & 12643 & 27649 & 26391 & 35221 & 3857 & 7691 & 2349 \end{pmatrix}$$

The IP was able to find two optimal tours, or schedules (u_1, u_2) which respectively are:

$$6, 9, 3, 8, 9, 3, 2, 1, 4, 1, 4, 1, 5, 5, 2, 8, 8, 7$$
$$6, 9, 3, 8, 9, 3, 2, 2, 1, 4, 1, 4, 1, 5, 5, 8, 8, 7$$

Four more schedules were created for testing purposes: a block schedule based on the solution to the associated TSP $(0, 6, 9, 3, 2, 4, 1, 5, 8, 7)$, u_3; a random block schedule, u_4; a completely random schedule, u_5; and the worst schedule found by the IP (by maximizing over X rather than minimizing), u_6. These strategies are:

$$u_3 : 6, 9, 9, 3, 3, 2, 2, 4, 4, 1, 1, 1, 5, 5, 8, 8, 8, 7$$
$$u_4 : 8, 8, 8, 2, 2, 7, 4, 4, 3, 3, 6, 9, 9, 5, 5, 1, 1, 1$$
$$u_5 : 8, 6, 8, 3, 9, 9, 7, 3, 5, 1, 1, 2, 1, 8, 5, 4, 2, 4$$
$$u_6 : 2, 3, 5, 3, 5, 9, 8, 4, 8, 4, 8, 2, 6, 1, 7, 1, 9, 1$$

Schedule	IP Cost	Simulation Time
u_2	250386	243747
u_3	254050	248674
u_1	250386	249308
u_4	306925	294243
u_5	308813	295860
u_6	368080	309906

TABLE II

COMPARISON OF TOTAL PRODUCTION TIMES BETWEEN THE SIMULATION AND ITS IP APPROXIMATION OVER VARIOUS SCHEDULES.

The results of running each of these strategies are found in Table II. Note that the production times are similar for a wide variety of schedules, and that good schedules can lead to 15-20% improvement over poor ones, which can be a significant savings to the enterprise. Using the approximation technique and solving the IP can find such good schedules, while computing them from the simulation would be intractable.

VI. CONCLUSION

This paper presents a technique to approximate the complex dynamics of certain manufacturing processes. This approximation can be used to compute scheduling policies that appear to be close to optimal, even when the original system would make computing such decisions intractable.

REFERENCES

[1] F. Blömer and H. Günther. Scheduling of a mutli-product batch process in the chemical industry. *Computers in Industry*, 36:245–259, 1998.

[2] E. G. Coffman, Jr., M. J. Elphick, and A. Shoshani. System deadlocks. *Computing Surveys*, 3(2):67–78, June 1971.

[3] S. French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Mathematics and its Applications. Ellis Horwood Limited, 1982.

[4] A. Gürel, S. Bogdan, and F. L. Lewis. Matrix approach to deadlock-free dispatching in multi-class finite buffer flowlines. *IEEE Transactions on Automatic Control*, 45(11):2086–2090, November 2000.

[5] E. Kondili, C. C. Pantelides, and R. W. H. Sargent. A general algorithm for short-term scheduling of batch operations - I. MILP formulation. *Computers and Chemical Engineering*, 17(2):211–227, 1993.

[6] R. G. Parker. *Deterministic Scheduling Theory*. Chapman and Hall, 1995.

[7] M. L. Pinedo. *Planning and Scheduling in Manufacturing and Services*. Springer Series in Operations Research. Springer Science+Business Media, Inc., 2005.

[8] S. H. Rich and G. J. Prokopakis. Scheduling and sequencing of batch operations in a multipurpose plant. *Ind. Eng. Chem. Process Des. Dev.*, 25:979–988, 1986.

[9] E. Sanmartí, L. Puigjaner, T. Holczinger, and F. Friedler. Combinatorial framework for effective scheduling of multipurpose batch plants. *AIChE Journal*, 48(11):2557–2570, Nov 2002.

[10] T. I. Seidman. 'First come, first served' can be unstable! *IEEE Transactions on Automatic Control*, 39(10):2166–2171, October 1994.

[11] W. Stallings. *Operating Systems*. Prentice Hall, 4th edition, 2001.

[12] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, 2nd edition, 2001.