AN APPROXIMATION METHOD FOR SEQUENCING OF A BATCH MANUFACTURING SYSTEM

by

W. Samuel Weyerman

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

September 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

W. Samuel Weyerman

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Sean C. Warnick, Chair
Date	Jordan Cox
Date	Scott Woodfield

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of W. Samuel Weyerman in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Sean C. Warnick Chair, Graduate Committee
Accepted for the Department	Parris K. Egbert Graduate Coordinator
Accepted for the College	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

AN APPROXIMATION METHOD FOR SEQUENCING OF A BATCH MANUFACTURING SYSTEM

W. Samuel Weyerman

Department of Computer Science

Master of Science

ACKNOWLEDGMENTS

The author would like to thank Dr. Tyler Jarvis and Dr. Jeff Humpherys of the math department. I also thank my wife and daughter for putting up with me coming late while I completed my thesis.

Contents

Chapter 1

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.

We develop a novel approximation 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.

Chapter 2

Literature Review

This chapter needs to be enlarged considerably.

Early work simplified the problem by considering the scheduling of a single machine [3], [7] and [8]. 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 [10] 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 [9] and [6] 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.

Chapter 3

Max-Plus Algebra

We will briefly discuss the max-plus algebra as it is presented in [5]. The max-plus algebra is defined over $\mathbb{R}_{max} = \mathbb{R} \cup -\infty$. We will define three operations for scalars:

$$\begin{aligned} &\forall a, b \in \mathbb{R}_{max} \\ &\qquad a \oplus b &&= \max(a, b) \\ &\qquad a \otimes b &&= a + b \\ &\qquad a \oslash b &&= a - b. \end{aligned}$$

The zero element is defined as $\epsilon = -\infty$, and the unit element is defined as $e = 0$.

Matrix arithmetic is also defined. For matrices, $A, B \in \mathbb{R}_{max}^{nl}$, $C \in \mathbb{R}_{max}^{lm}$ these are defined as:

$$\begin{aligned} [A \oplus B]_{ij} &= a_{ij} \oplus b_{ij} \\ [B \otimes C]_{ik} &= \bigoplus_{j=1}^{l} b_{ij} \otimes c_{jk}. \end{aligned}$$

The zero vector and the unit vector are given by

$$\begin{aligned} \boldsymbol{\epsilon} &= \begin{bmatrix} \epsilon \\ \vdots \\ \epsilon \end{bmatrix} \\ \mathbf{e} &= \begin{bmatrix} e \\ \vdots \\ e \end{bmatrix}. \end{aligned}$$

The identity matrix is

$$I = \begin{bmatrix} e & \epsilon & \dots & \epsilon \\ \epsilon & e & \dots & \epsilon \\ \vdots & & & \\ \epsilon & \epsilon & \dots & e \end{bmatrix}.$$

We can now define a linear state-space system in the max-plus algebra. For $\mathbf{x}(k) \in \mathbb{R}_{max}^n$, and $A \in \mathbb{R}_{max}^{n \times n}$, there is a linear autonomous system,

$$\mathbf{x}(k+1) = A \otimes \mathbf{x}(k).$$

Definition 3.0.1 *We say a max-plus autonomous system is stable if*

$$\forall i \exists v \in \mathbb{R} \ \lim_{k \to \infty} x_i(k) \oslash x_1(k) = v.$$

Throughout this paper we will use the convention that for any $y \in \mathbb{R}_{max}$, the indeterminate form $y \oplus (\epsilon \oslash \epsilon) = y$.

We will also define the 1-norm in the max-plus algebra.

Definition 3.0.2 *The 1-norm of a max-plus vector, $b \in \mathbb{R}_{max}^n$ is*

$$||b||_1 = \bigoplus_{i=1}^{n} b_i = \mathbf{e}^T \otimes b$$

This norm induces a norm on a matrix.

Definition 3.0.3 *The 1-induced norm of an operator $A \in \mathbb{R}_{max}^{n \times m}$ is*

$$||A||_1 = max_x(||A \otimes x||_1 \oslash ||x||_1).$$

Theorem 3.0.1 *Given a matrix $A \in \mathbb{R}_{max}^{n \times m}$, the max-plus 1-induced norm of A is*

$$||A||_1 = \max_{ij} a_{ij}.$$

Proof: Let A be given. Without loss of generality, we will say that $||x||_1 = e$. Note that $||A \otimes x||_1 = \mathbf{e}^T \otimes A \otimes x$. The vector $v^T = \mathbf{e}^T \otimes A$ is the vector containing the max element of each column of A. Therefore, we want to maximize $v^T \otimes x$. Because $||x||_1 = e$, the largest element in x is e. Clearly, to maximize $v^T \otimes x$, we want to make x as large as possible; this means we set $x = \mathbf{e}$ which gives $v^T \otimes x = max_{ij} a_{ij}$. ■

Theorem 3.0.2 *Given a matrix $A \in \mathbb{R}_{max}^{n \times m}$,*

$$\min_x ||A \otimes x||_1 \oslash ||x||_1 = \min_i [\mathbf{e}^T \otimes A]_i.$$

Proof: Let A be given. As in the previous proof we will say that $||x||_1 = e$ and consider $v^T \otimes x$. Now we want to minimize $v^T \otimes x$, so we want x as small as possible. However, having $||x||_1 = e$ requires at least one element of x equal to e. Thus, we need only consider each e_i where $\mathbf{e}_{i,i} = e$ and $\mathbf{e}_{i,j} = \epsilon$ for $j \neq i$. So the minimum $v^T \otimes x$ is $\min_i(v^T \otimes \mathbf{e}_i) = \min_i(v_i) = \min_i[\mathbf{e}^T \otimes A]_i$. ■

Chapter 4

Problem Description

4.1 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.

4.2 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.

4.3 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.

4.4 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 $q = \begin{bmatrix} q_1 & q_2 & \dots & q_m \end{bmatrix}^T$, 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

$$\begin{aligned} z(k+1) &= f(z(k), u(k)) \\ y(k) &= g(z(k), u(k)) \end{aligned} \tag{4.1}$$

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

$$\begin{aligned} \min_{u} \quad & \sum_{k=0}^{K-1} y(k) \\ \text{subject to} \quad & z(k+1) = f(z(k), u(k)) \\ & y(k) = g(z(k), u(k)). \end{aligned} \tag{4.2}$$

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 (4.2).

Chapter 5

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 $R(t) = [r_1(t) \ \dots \ r_m(t)]', r_i(t) \in \mathbb{Z}^*$ 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 $\Psi = [\psi_1 \ \dots \ \psi_m]', \psi_i \in \mathbb{Z}^*$ 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:

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

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:

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

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

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

$$\psi_i(t+1) = \begin{cases} \psi_i(t) + 1, \end{cases}$$

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 [11] and [4]. Four conditions for deadlock are given in [2] and heuristic methods for avoiding and preventing deadlock are given in [12]. 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.

Chapter 6

Approximation Algorithm

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 [7]. The formulation we derive is based on the IP formulation of the TSP given in [13]. 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:

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

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

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

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:

$$\begin{aligned} \min_{X} \quad & \sum_{j=0}^{m}\sum_{i=0}^{m} c_{ij}x_{ij} \\ \text{subject to} \quad & \sum_{i=0}^{m} x_{ij} - \sum_{l=0}^{m} x_{jl} = 0; \\ & j = 0, 1, \ldots, m \\ & \sum_{i=0}^{m} x_{ij} = q_j; \\ & j = 0, 1, \ldots, m \\ & x_{ij} \in \{0, 1, \ldots\}. \end{aligned} \tag{6.1}$$

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}:

$$\sum_{i,j \in \varsigma} x_{ij} < \varsigma_{length} \tag{6.2}$$

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 6.1.

- 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
    - $i = \text{Pop}(Q)$
    - increment j
  - else if X contains more than one outgoing arc from i
    - $\text{Push}(Q, i)$
  - 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 $\text{length}(S_i)! = 1$
    - find S_j such that $\text{front}(S_i) \in S_j$
    - replace occurrence of $\text{front}(S_i)$ in S_j with S_i
  - delete S_i

Table 6.1: Algorithm for determining a schedule based on X.

Chapter 7

A Max-Plus Representation

Because this problem is NP-complete, we wish to analyze this system in order to validate the approximation method. However, due to the recursive nonlinear definition of $f(x, u, k)$ we seek a different representation of this system. To reduce the complexity of the model given in chapter **??** we will consider a further simplification of the factory and allow only a single machine at each workstation. This enables us to model the system using a linear max-plus representation.

To create the model, we will consider the system as a heap model. We will represent the time that workstation i spends processing a batch of job type j as a piece of width one and height $\tau_i(j)$. The piece for job type j and workstation i is given by a matrix $P_i(j)$ which is equal to I except that $[P_i(j)]_{ii} = \tau_i(j)$. We represent the precedence of workstation i over $i+1$ using a piece of width two and height 0. This matrix, R_i, is equal to I except that $[R_i]_{i,i+1} = [R_i]_{i+1,i} = e$.

Using these pieces we can construct the heap created by job type j using the algorithm given in Figure 7.1. This algorithm multiplies several piece matrices together to arrive at a matrix which we refer to as $A(j)$. We will show that this matrix defines the linear max-plus model for (**??**). Thus, the system can now be represented in the max-plus algebra as

$$\begin{aligned} x(k+1) &= A(u_k) \otimes x(k) \\ y(k) &= ||x(k)||_1 \oslash ||x(k-1)||_1. \end{aligned} \tag{7.1}$$

The 1-norm specified here is the max-plus 1-norm.

7.1 Properties

To ease notation we will introduce some new symbols. Given A, we define

$$\begin{aligned} \xi_{ij} &= a_{ij} - a_{i,j+1}, \\ \delta_i &= a_{i+1,1} - a_{i1}. \end{aligned}$$

These representations are meaningful because:

We will give a definition of matrix structure.

Definition 7.1.1 *We will say that a matrix A has the monotone property if:*

1. $a_{ij} \leq a_{i+1,j}$,
2. $a_{ij} \geq a_{i,j+1}$,
3. $\xi_{1i} \geq \xi_{2i} \geq \ldots \geq \xi_{i-1,i} = \ldots = \xi_{ni} \geq 0, \ \forall i \leq n$,
4. $a_{ij} > -\infty, \ \forall i \leq j \leq n$.

Note to self: it seems that property 3 should be modified, I think we only have equality after the diagonal.

Lemma 7.1.1 *A matrix $B \in \mathbb{R}_{max}^{n\times n}$ with the monotone property maintains property 2 of the monotone property after finitely many left multiplications of P_i and R_i for some job type with n stages.*

Proof: Let $B \in \mathbb{R}_{max}^{n\times n}$ with the monotone property, and $i \leq n$ be given. Property 2 of the monotone property is trivially maintained after a left multiply of P_i. Suppose that $i < n$. Consider the product $A = R_i \otimes B$. We want to show that $a_{ij} \geq a_{i,j+1}$ and $a_{i+1,j} \geq a_{i+1,j+1}$ for all $j < n$. We will consider two cases.

Suppose $b_{ij} \geq b_{i+1,j}$. Then $a_{ij} = b_{ij} = a_{i+1,j}$. Because B has property 2 of the monotone property it follows that $b_{ij} \geq b_{i,j+1}$ and $b_{ij} \geq b_{i+1,j} \geq B_{i+1,j+1}$. So $a_{ij} \geq a_{i,j+1}$ and $a_{i+1,j} \geq a_{i+1,j+1}$.

Suppose $b_{i+1,j} \geq b_{ij}$. We achieve the same result as the previous case in the same manner.

Thus, because B maintains property 2 of the monotone property after a left multiply of an arbitrary P_i or R_i, after a finite number of left multiplies, the resulting matrix will satisfy property 2 of the monotone property. ■

Lemma 7.1.2 *A matrix $B \in \mathbb{R}_{max}^{n\times n}$ with the monotone property maintains property 3 of the monotone property after finitely many left multiplications of P_i and R_i for some job type with n stages.*

The proof for this lemma only shows that a weakened version of property 3 is maintained, that where we replace the equalities with inequalities.

Proof: Let $B \in \mathbb{R}_{max}^{n\times n}$ with the monotone property be given. Left multiplying B by P_i adds the same number to every element of row i. As this does not change $b_{ij} - b_{i,j+1}$ for any j, the

resulting matrix still satisfies the desired property. Now suppose that we left multiply B by R_i for some $i < n$. The resulting matrix, A, has rows i and $i+1$ equal. We now have $a_{ij} - a_{i,j+1} = \max(b_{ij}, b_{i+1,j}) - \max(b_{i,j+1}, b_{i+1,j+1}) = a_{i+1,j} - a_{i+1,j+1}$. We will handle each of the four cases separately.

Suppose $b_{ij} \geq b_{i+1,j}$ and $b_{i,j+1} \geq b_{i+1,j+1}$. Then

$$a_{ij} - a_{i,j+1} = b_{ij} - b_{i,j+1}$$

and

$$\begin{aligned} a_{i+1,j} - a_{i+1,j+1} &= b_{ij} - b_{i,j+1} \\ &\geq b_{i+1,j} - b_{i+1,j+1}. \end{aligned}$$

Suppose $b_{ij} \geq b_{i+1,j}$ and $b_{i,j+1} \leq b_{i+1,j+1}$. Then

$$\begin{aligned} a_{ij} - a_{i,j+1} &= b_{ij} - b_{i+1,j+1} \\ &\leq b_{ij} - b_{i,j+1} \end{aligned}$$

and

$$\begin{aligned} a_{i+1,j} - a_{i+1,j+1} &= b_{ij} - b_{i+1,j+1} \\ &\geq b_{i+1,j} - b_{i+1,j+1}. \end{aligned}$$

Suppose $b_{ij} \leq b_{i+1,j}$ and $b_{i,j+1} \geq b_{i+1,j+1}$. Then

$$\begin{aligned} a_{ij} - a_{i,j+1} &= b_{ij} - b_{i,j+1} \\ &\leq b_{i+1,j} - b_{i+1,j+1} \end{aligned}$$

which violates the assumption on B, so this case is not possible.

Finally, suppose $b_{ij} \leq b_{i+1,j}$ and $b_{i,j+1} \leq b_{i+1,j+1}$. Then

$$\begin{aligned} a_{ij} - a_{i,j+1} &= b_{i+1,j} - b_{i+1,j+1} \\ &\leq b_{i,j} - b_{i+1,j} \end{aligned}$$

and

$$a_{i+1,j} - a_{i+1,j+1} = b_{i+1,j} - b_{i+1,j+1}.$$

We have shown that in any case, $a_{ij} - a_{i,j+1} \leq b_{ij} - b_{i,j+1}$ and $a_{i+1,j} - a_{i+1,j+1} \geq b_{i+1,j} - b_{i+1,j+1}$. Clearly for any $k \neq i$ and $k \neq i+1$, $a_{k,j} - a_{k,j+1} = b_{kj} - b_{k,j+1}$ for all j. Thus,

$a_{kj} - a_{k,j+1} \geq a_{k+1,j} - a_{k+1,j+1}$, so property 1 of the monotone property is maintained after a left multiply of R_i.

Because we have shown that the resulting matrix after a left multiply of arbitrary P_i or arbitrary R_i maintains property 3 of the monotone property, it follows that property 3 of the monotone property is maintained after a finite number of left multiplies. ■

Theorem 7.1.1 *Given* $(\mathbf{c}, \boldsymbol{\tau})(\alpha)$, $A(\alpha)$ *has the monotone property.*

Proof: Let $(\mathbf{c}, \boldsymbol{\tau})(\alpha)$ be given. The algorithm for constructing $A(\alpha)$ begins with I. The block of I consisting of i_{11} trivially has the monotone property. Thus we know by the lemmata 7.1.1 and 7.1.2 that this block maintains properties 2 and 3 of the monotone property throughout the algorithm. We will show by induction that by the end of the algorithm the entire matrix has both of these properties. Suppose that at the p^{th} stage of the algorithm, the block from b_{11} to b_{kk} of the current matrix B has properties 2 and 3 of the monotone property and the remainder of the matrix consists of $-\infty$ off the diagonal and 0 on the diagonal. Consider now $C = R_k \otimes B$, we can suppose without loss of generality that this is the next operation the algorithm performs. Row $k+1$ of C is equal to row k of C, so the block from c_{11} to $c_{k+1,k+1}$ has property 3 of the monotone property. The only difference between row k of C and row k of B is that $c_{k,k+1} = 0$ and $b_{k,k+1} = -\infty$. Because we must multiply by P_k before R_k, $c_{kk} > 0 = c_{k,k+1}$, so the block from c_{11} to $c_{k+1,k+1}$ has property 2 of the monotone property. Also, the remainder of C consists of $-\infty$ off the diagonal and 0 on the diagonal. Therefore, by induction, A has these properties.

It remains to be shown that A has properties 1 and 4 of the monotone property. Property 4 comes due to the fact that

$$A = R_{n-1} \otimes B_{n-1} \otimes R_{n-2} \otimes B_{n-2} \otimes \ldots \otimes R_1 \otimes P_1 \otimes I$$

where B_{n-j} is some intermediate matrix sum.

For property 1, consider the last multiplication of P_i for some i. This is necessarily followed by a multiplication of R_i, at this point the resulting matrix, C, has $c_{ij} = c_{i+1,j}$. This matrix is necessarily multiplied by P_{i+1}; this resulting matrix, D, has $d_{ij} \leq d_{i+1,j}$, at this point we know $a_{ij} = d_{ij}$ and $a_{i+1,j} \leq d_{i+1,j}$, so A must have property 1. ■

The fact that A has the monotone property gives two easy results

Corollary 7.1.1 *Given* $(\mathbf{c}, \boldsymbol{\tau})(\alpha)$*, we have the following:*

$$\begin{aligned} ||A(\alpha)||_1 &= a_{n1}(\alpha) \\ \min_x ||A(\alpha) \otimes x||_1 \oslash ||x||_1 &= a_{nn}(\alpha). \end{aligned}$$

Proof: Let A be given. Due to monotonicity of A,

$$\begin{aligned} ||A||_1 &= \max_{ij} a_{ij} \\ &= a_{n1}. \end{aligned}$$

And

$$\begin{aligned} \min_x ||A \otimes x||_1 \oslash ||x||_1 &= \min_i(\mathbf{e}^T \otimes A) \\ &= \min_i([a_{n1}, \ldots, a_{nn}]) \\ &= a_{nn}. \end{aligned}$$

■

7.1.1 stability

We can write this A matrix in terms of a_{11}, the ξ_i's and r, where $r_i = a_{i-1,1} - a_{i,1}$. Then $a_{ij} = a_{11} - \sum_{l=1}^{j-1} \xi_{il} + \sum_{k=1}^{i-1} r_i$.

Given two distinct vertices, v_i and v_j, in A, we will define $P_{v_i v_j}$ as

$$P_{v_i v_j} = \begin{cases} -\sum_{k=v_i}^{v_j - 1} \xi_{v_j,k} & v_i < v_j \\ \sum_{k=v_j}^{v_i - 1} \xi_{v_j,k} & v_i > v_j \end{cases}.$$

Lemma 7.1.3 *Given a circuit,* γ *in* A*,* $\sum_{v_i \in \gamma} P_{v_i v_{i+1}} \leq 0$*.*

Proof: We will first enumerate the vertices in A as $\{1, 2, 3, \ldots, n\}$, similarly we will enumerate the vertices in γ as $\{v_1, v_2, \ldots, v_l\}$, with $|\gamma|_l = l$. We will let k correspond with a vertex in A and show that the sum over each k, $0 < k < n$ is less than or equal to zero, and that the sum over all $0 < k < n$ is the sum in question. Let $0 < k < n$ be given. We will say that if there is some i such that $v_i \leq k < v_{i+1}$ we will add $\xi_{v_{i+1},k}$. If there is some i such that $v_{i+1} \leq k < v_i$ we will add $-\xi_{v_{i+1},k}$. Since γ is a circuit, we will have $v_i \leq k < v_{i+1}$ for some i iff there is some $j \neq i$ such that $v_{j+1} \leq k < v_j$. Let I_k be the set of all $i \leq n$ such that $v_i \leq k < v_{i+1}$ and let J_k

be the set of all $j \leq n$ such that $v_{j+1} \leq k < v_j$. Note that I_k and J_k have the same number of elements. The total sum at k is $\sum_{i \in I_k} \xi_{v_{i+1},k} - \sum_{j \in J_k} \xi_{v_{j+1},k}$. Because $\forall j \in J_k (v_{j+1} \leq k)$ and $\forall i \in I_k (k < v_{i+1})$, it follows that $\forall i \in I_k, j \in J_k, v_{j+1} < v_{i+1}$ and by theorem 7.1.1 $\xi_{v_{j+1},k} \geq \xi_{v_{i+1},k}$. Thus $\sum_{i \in I_k} \xi_{v_{i+1},k} - \sum_{j \in J_k} \xi_{v_{j+1},k} \leq 0$. We will expand the sets I_k and J_k to $I = \{i | v_i < v_{i+1}, v_i, v_{i+1} \in \gamma\}$ and $J = \{j | v_{j+1} < v_j, v_j, v_{j+1} \in \gamma\}$. Then

$$\begin{aligned} \sum_{v_i \in \gamma} P_{v_i v_{i+1}} &= \sum_{i \in I} \sum_{l=v_i}^{v_{i+1}-1} \xi_{v_{i+1},l} - \sum_{j \in J} \sum_{l=v_{j+1}}^{v_j - 1} \xi_{v_{j+1},k} \\ &= \sum_{k=1}^{n-1} (\sum_{i \in I_k} \xi_{v_{i+1},k} - \sum_{j \in J_k} \xi_{v_{j+1},k}) \\ &\leq 0. \end{aligned}$$

■

Lemma 7.1.4 *Let a_{mm} be the maximum diagonal element in A. Then*

$$\sum_{i=1}^{m-1} \xi_{mi} - \sum_{i=1}^{m-1} r_i - \sum_{i=1}^{j-1} \xi_{ji} + \sum_{i=1}^{j-1} r_i \leq 0$$

for any $j \leq n$ with equality iff $a_{mm} = a_{jj}$.

Proof: Let $j \leq n$ be given. $a_{jj} = a_{mm} + \sum_{i=1}^{m-1} \xi_{mi} - \sum_{i=1}^{m-1} r_i - \sum_{i=1}^{j-1} \xi_{ji} + \sum_{i=1}^{j-1} r_i$. Because a_{jj} is on the diagonal, $a_{jj} \leq a_{mm}$, so $\sum_{i=1}^{m-1} \xi_{mi} - \sum_{i=1}^{m-1} r_i - \sum_{i=1}^{j-1} \xi_{ji} + \sum_{i=1}^{j-1} r_i = a_{jj} - a_{mm} \leq 0$ and equality is achieved if and only if $a_{jj} = a_{mm}$. ■

Lemma 7.1.5 *If γ is a circuit in the graph of A and a_{mm} is the maximum diagonal element of A, then $w(\gamma) \leq a_{mm}$ with equality only if $a_{jj} = a_{mm} \forall j \in \gamma$.*

Proof: Let $k = |\gamma|_l$. We will consider $kw(\gamma)$. Then

$$\begin{aligned} kw(\gamma) &= a_{v_2v_1} + a_{v_3v_2} + \ldots + a_{v_kv_1} \\ &= a_{11} - \sum_{i=1}^{v_1-1} \xi_{v_2i} + \sum_{i=1}^{v_1-1} r_i + a_{11} - \sum_{i=1}^{v_2-1} \xi_{v_3i} + \sum_{i=1}^{v_2-1} r_i + \ldots + \\ &\quad a_{11} - \sum_{i=1}^{v_k-1} \xi_{v_1i} + \sum_{i=1}^{v_k-1} r_i \\ &= ka_{mm} + \left(\sum_{i=1}^{m-1} \xi_{mi} - \sum_{i=1}^{m-1} r_i - \sum_{i=1}^{v_k-1} \xi_{v_1i} + \sum_{i=1}^{v_1-1} r_i \right) + \\ &\quad \left(\sum_{i=1}^{m-1} \xi_{mi} - \sum_{i=1}^{m-1} r_i - \sum_{i=1}^{v_1-1} \xi_{v_2i} + \sum_{i=1}^{v_2-1} r_i \right) + \ldots + \\ &\quad \left(\sum_{i=1}^{m-1} \xi_{mi} - \sum_{i=1}^{m-1} r_i - \sum_{i=1}^{v_{k-1}-1} \xi_{v_ki} + \sum_{i=1}^{v_k-1} r_i \right) \end{aligned}$$

We can decompose $\sum_{i=1}^{v_j-1} \xi_{v_{j+1}i}$ as $\sum_{i=1}^{v_{j+1}-1} \xi_{v_{j+1}i} + P_{v_jv_{j+1}}$. Thus by lemma 7.1.4, each

$$\begin{aligned} \sum_{i=1}^{m-1} \xi_{mi} - \sum_{i=1}^{m-1} r_i - \sum_{i=1}^{v_j-1} \xi_{v_{j+1}i} + \sum_{i=1}^{v_1-1} r_i &= \sum_{i=1}^{m-1} \xi_{mi} - \sum_{i=1}^{m-1} r_i - \sum_{i=1}^{v_{j+1}-1} \xi_{v_{j+1}i} + \sum_{i=1}^{v_1-1} r_i + P_{v_jv_{j+1}} \\ &\leq P_{v_jv_{j+1}}. \end{aligned}$$

With equality when $a_{jj} = a_{mm}$.

Thus

$$\begin{aligned} kw(\gamma) &\leq ka_{mm} + \textstyle\sum_{v_j \in \gamma} P_{v_jv_{j+1}} \\ &\leq ka_{mm} \\ w(\gamma) &\leq a_{mm}. \end{aligned} \tag{7.2}$$

We get equality in (7.2) if and only if $a_{jj} = a_{mm}$ $\forall j \in \gamma$. ■

Lemma 7.1.6 *Every vertex in the critical graph of A has a self loop.*

Proof: Let v be a vertex in the critical graph of A. This means that v is in a critical circuit of A which we will call γ. By Lemma 7.1.5, it must be that $w(\gamma) = a_{mm}$ because vertex m has a self loop with average weight a_{mm} which must be in the critical graph. Furthermore, since $v \in \gamma$, $a_{vv} = a_{mm}$, so the self loop on v must also be in the critical circuit of A. ■

Theorem 7.1.2 *Given a recipe (c, τ), the associated max-plus matrix A has cyclicity one.*

Proof: By lemma 7.1.6, every vertex in the critical graph of A has a self loop. In [5] the cyclicity of a matrix is defined to by the cyclicity of the critical graph of that matrix. We will call G the critical graph of A.

Suppose that G is strongly connected. The cyclicity of A is the greatest common divisor of the lengths of all elementary circuits in G. Because every node in G has a self loop, the elementary circuits of those self-loops have length one, thus the cyclicity of G must be one.

Suppose that G is not strongly connected. Then the cyclicity of G (and thus A) is the least common multiple of the cyclicities of all maximal strongly connected subgraphs (m.s.c.s.'s) of G. Again, since each node in each m.s.c.s. of G has a self-loop, the cyclicity of each m.s.c.s. is one and thus the cyclicity of G is one. ■

$Ps = Rs = \text{zeros}(n,1)$
$M = I$
$B = \text{lcm}(c)$ ./ c
while ($\neg$ all($B == Rs$))
  for i = 1:resources
    if ($i == 1$)
      if($Ps_i == Rs_i$)
        $M = P_i \otimes M$
        Ps_i ++
      elif(($Rs_i + 1 == Ps_i$)
                $\wedge \left(Rs_{i+1} == \left\lceil \frac{(Rs_i+1)c_i(u_k)}{c_{i+1}(u_k)} \right\rceil - 1\right)\right)$
        $M = R_i \otimes M$
        Rs_i ++
      end
    elif($i == n$)
      if$\Big(\left(Ps_{i-1} == \left\lceil \frac{(Ps_i+1)c_i(u_k)}{c_{i-1}(u_k)} \right\rceil\right)$
                $\wedge(Ps_i == Rs_i))$
        $M = P_i \otimes M$
        Ps_i ++
      elif($Ps_i == Rs_i + 1$)
        Rs_i ++
      end
    else
      if$\Big(\left(Ps_{i-1} == \left\lceil \frac{(Ps_i+1)c_i(u_k)}{c_{i-1}(u_k)} \right\rceil\right)$
                $\wedge(Ps_i == Rs_i))$
        $M = P_i \otimes M$
        Ps_i ++
      elif(($Rs_i + 1 == Ps_i$)
                $\wedge \left(Rs_{i+1} == \left\lceil \frac{(Rs_i+1)c_i(u_k)}{c_{i+1}(u_k)} \right\rceil - 1\right)\right)$
        $M = R_i \otimes M$
        Rs_i ++
      end
    end
  end
end
$A = M$

Figure 7.1: Algorithm for determining A using the heap model.

Chapter 8

Error Bounds for Approximation Method

We will briefly extend the approximation method given in [14] to the max-plus formulation. For the t-step approximation, given a sequence $U = (u_k, \ldots, u_{k+t})$ we approximate (7.1) as

$$\begin{aligned} \tilde{x}_t(k+1) &= A(u_{k+t}) \otimes \ldots \otimes A(u_k) \otimes x^* \\ \tilde{y}_t(k) &= ||\tilde{x}_t(k)||_1 \oslash ||\tilde{x}_{t-1}(k-1)||_1. \end{aligned}$$

Where we define $x^* = \mathbf{e}$. This approximation eases the solution to (**??**) as it exponentially reduces the number of possible states and therefore number of values to compute at each stage of the dynamic program that solves the problem. This replaces a large NP-complete problem with a much smaller NP-complete problem. Using this approximation, much more complex problems can be solved.

The error of a single step of the t-step approximation is given by

$$\varepsilon_t(k) = |y(k) \oslash \tilde{y}_t(k)|.$$

We can explicitly calculate the error bound for the 0-step approximation for job type j.

Theorem 8.0.3 *The maximum error of the 0-step approximation for job type j is*

$$0 \leq \varepsilon_0(k) \leq \gamma_0(j) = a_{n1}(j) \oslash a_{nn}(j)$$

Proof: Let $A(j)$ be given. The maximum error of the 0-step approximation is given by

$$\gamma_0(j) = \max_x \{(||A \otimes x^*||_1 \oslash ||x * ||_1) \oslash (||A \otimes x||_1 \oslash ||x||_1)\}$$

Using $x^* = \mathbf{e}$, we get

$$||A \otimes x^*||_1 \oslash ||x^*||_1 = a_{n1}.$$

Also by corollary 7.1.1 we get

$$\min_x(||A \otimes x||_1 \oslash ||x||_1) = a_{nn}.$$

Thus,

$$\gamma_0(j) = a_{n1} \oslash a_{nn}$$

■

Note that the 0-step approximation is not very useful as it treats the dynamic system as a static function with a single input. Therefore, every sequence has the same cost in the 0-step approximation.

8.1 1-step Approximation

We will now extend to the error bound of the 1-step approximation and in the next section to the t-step approximation. We will first define some notation.

Definition 8.1.1 *For a job type j, we define*

$$\begin{aligned} Z_i^{(1)}(j) &= [a_{i1} \oslash a_{i-1,1} \cdots a_{in} \oslash a_{i-1,n}] \otimes \mathbf{e} \\ z_i^{(1)}(j) &= a_{i1} \oslash a_{i-1,1}. \end{aligned}$$

where it is understood that each a_{ij} is from $A(j)$.

Lemma 8.1.1 *For job type j, for any $x \in \mathbb{R}^n_{max}$, we have $x(1) = A(j) \otimes x$, and*

$$z_i^{(1)}(j) \leq x(1)_i \oslash x(1)_{i-1} \leq Z_i^{(1)}(j).$$

Proof: Let $x \in \mathbb{R}^n_{max}$ be given. Then $x(1) = A \otimes x$. Suppose that $x_i(1) = a_{ij} \otimes x_j$ and $x_{i-1}(1) = a_{i-1,l} \otimes x_l$. Thus, $a_{ij} \otimes x_j \geq a_{ip} \otimes x_p$ for all $p \leq n$, and $a_{i-1,l} \otimes x_l \geq a_{i-1,p} \otimes x_p$, for all p. Through algebraic manipulation we achieve

$$\begin{aligned} x_j \oslash x_p &\geq a_{ip} \oslash a_{ij} && \forall p \leq n, \\ x_p \oslash x_l &\leq a_{i-1,l} \oslash a_{i-1,p} && \forall p \leq n. \end{aligned}$$

Using these inequalities, we prove the first inequality

$$\begin{aligned} x_i(1) \oslash x_{i-1}(1) &= (a_{ij} \otimes x_j) \oslash (a_{i-1,l} \otimes x_l) \\ &\geq (a_{ij} \otimes a_{il}) \oslash (a_{i-1,l} \otimes a_{ij}) \\ &\geq a_{i1} \oslash a_{i-1,1}. \end{aligned}$$

The second inequality follows similarly

$$\begin{aligned} x_i(1) \oslash x_{i-1}(1) &= (a_{ij} \otimes x_j) \oslash (a_{i-1,l} \otimes x_l) \\ &\leq (a_{ij} \otimes a_{i-1,l}) \oslash (a_{i-1,l} \otimes a_{i-1,j}) \\ &\leq [a_{i1} \oslash a_{i-1,1} \ldots a_{in} \oslash a_{i-1,n}] \otimes \mathbf{e} \end{aligned}$$

■

Now we will examine the error of the 1-step approximation. This is given by

$$\begin{aligned} \gamma_1(u_{k-1}, u_k) \quad = \quad & \max_x \{(||A(u_k)A(u_{k-1})x^*||_1 \\ & \oslash ||A(u_{k-1})x^*||_1) \\ & \oslash (||A(u_k)A(u_{k-1})x||_1 \\ & \oslash ||A(u_{k-1})x||_1)\} \end{aligned} \tag{8.1}$$

Note that since $x^* = \mathbf{e}$, the numerator of this fraction is a constant calculated from

$$\begin{aligned} ||A(u_k) \otimes A(u_{k-1}) \otimes x^*||_1 \quad = \quad & [a_{n1} \ldots a_{nn}](u_k) \\ & \otimes [a_{11} \ldots a_{n1}]^T (u_{k-1}) \end{aligned}$$

and

$$||A(u_{k-1}) \otimes x^*||_1 = a_{n1}(u_{k-1}).$$

Which give us

$$[a_{n1}(u_k) \otimes a_{11}(u_{k-1}) \oslash a_{n1}(u_{k-1}), \cdots, a_{nn}(u_k)] \otimes \mathbf{e}$$

Because we ultimately want to know the difference between the 0-step approximation and the 1-step approximation, we are interested in the quantity

$$(||A(u_k) \otimes x^*||_1 \oslash ||x^*||_1)$$
$$\oslash(||A(u_k)A(u_{k-1})x^*||_1 \oslash ||A(u_{k-1})x^*||_1).$$

Which by our previous calculations we can calculate and we define

$$\begin{aligned}\delta_1 =&[a_{n1}(u_{k-1}) \oslash a_{11}(u_{k-1}),\\ &(a_{n1}(u_k) \otimes a_{n1}(u_{k-1})) \oslash (a_{n2}(u_k) \otimes a_{21}(u_{k-1})),\\ &\cdots, a_{n1}(u_k) \oslash a_{nn}(u_k)] \otimes \mathbf{e}\\ >&e\end{aligned}$$

The action of maximizing (8.1) is done by minimizing the denominator. So we wish to solve

$$\min_x(||A(u_k)A(u_{k-1})x||_1 \oslash ||A(u_{k-1})x||_1).$$

This problem is similar to the zero step approximation, except that we have $A(u_{k-1})x$ everywhere we used to have just x. So now we have a similar problem, but with an extra constraint.

As when calculating

$$\min_x(||A \otimes x||_1 \oslash ||x||_1)$$

we wanted $x_n = e$ with $x_i = \epsilon$ for all other i, we want $[A(u_{k-1}) \otimes x]_n = e$ and $[A(u_{k-1}) \otimes x]_i$ as small as possible. By lemma 8.1.1, we know that once we fix $[A(u_{k-1}) \otimes x]_n = e$ the smallest $[A(u_{k-1}) \otimes x]_i$ can be is $e \oslash \bigotimes_{j=i+1}^{n} Z_j^{(1)}(u_{k-1})$. We will pick x so that

$$A(u_{k-1}) \otimes x = [e \oslash \bigotimes_{j=2}^{n} Z_j^{(1)}(u_{k-1}), \cdots, e]^T.$$

This value achieves

$$\begin{aligned}&\min_x(||A(u_k) \otimes A(u_{k-1}) \otimes x)||_1 \oslash ||A(u_{k-1}) \otimes x||_1)\\ =&[a_{n1}(u_k) \oslash (\bigotimes_{i=2}^{n} Z_i^{(1)}(u_{k-1})), \cdots, a_{nn}(u_k)] \otimes \mathbf{e}\end{aligned}$$

Again, we are interested in the difference between this value and that of the 0-step approximation, we will define

$$\begin{aligned}\delta_2 =& \min_x(||A(u_k)\otimes A(u_{k-1})\otimes x||_1 \oslash ||A(u_{k-1})\otimes x||_1)\\ & \oslash \min_x(||A(u_k)\otimes x||_1 \oslash ||x||_1)\\ =& [a_{n1}(u_k)\oslash a_{nn}(u_k)\oslash (\bigotimes_{i=2}^{n} Z_i^{(1)}(u_{k-1})),\ldots,e]\otimes \mathbf{e}\\ \geq& e.\end{aligned}$$

This proves the following theorem.

Theorem 8.1.1 *The bound of the error of the 1-step approximation for the two job type sequence* (u_1, u_2) *is*

$$0 \leq \gamma_1(u_1,u_2) \quad = \quad \gamma_0(u_2)\oslash\delta_1\oslash\delta_2.$$

8.2 t-step Approximation

We will now extend this result to the t-step approximation. The t-step approximation error bound for sequence $U = (u_1,\ldots,u_t)$ is given by

$$\begin{aligned}\gamma_t(U) \quad = \quad & \max_x(||A(u_t)\otimes\ldots\otimes A(u_1)\otimes x^*||_1\\ & \oslash||A(u_{t-1})\otimes\ldots\otimes A(u_1)\otimes x^*||_1)\\ & \oslash(||A(u_t)\otimes\ldots\otimes A(u_1)\otimes x||_1\\ & \oslash||A(u_{t-1})\otimes\ldots\otimes A(u_1)\otimes x||_1)\end{aligned} \tag{8.2}$$

We will define a recursive definition which we will use to bound $x(k)$

Definition 8.2.1 *Given a sequence of length t of job types, $U = (u_1, u_2, \ldots, u_t)$,*

$$\begin{aligned}
Z_l^{(t)}(U) &= \bigoplus_{\substack{d_1 \in D_Z(l-1) \\ d_2 \in D_Z(l) \\ d_1 \le d_2}} \left(a_{ld_2}(u_t) \oslash a_{l-1,d_1}(u_t) \otimes \bigotimes_{j=d_1+1}^{d_2} Z_j^{(t-1)}(U^{t-1}) \right) \\
z_l^{(t)}(U) &= \min_{\substack{d_1 \in D_z(l-1) \\ d_2 \in D_z(l) \\ d_1 \le d_2}} \left(a_{ld_2}(u_t) \oslash a_{l-1,d_1}(u_t) \otimes \bigotimes_{j=d_1+1}^{d_2} z_j^{(t-1)}(U^{t-1}) \right).
\end{aligned}$$

With

$$\begin{aligned}
D_Z(i) &= \{d \mid \forall k < d \left(a_{ik} \oslash a_{id} \le \bigotimes_{j=k+1}^{d} Z_j^{t-1}(U^{t-1}) \right) \} \\
D_z(i) &= \{d \mid \forall k > d \left(a_{id} \oslash a_{ik} \ge \bigotimes_{j=d+1}^{k} z_j^{t-1}(U^{t-1}) \right) \}
\end{aligned}$$

Here we use the notation U^{t-1} to mean $(u_1, \ldots, u_{t-1})$. Here, we still have $Z_l^{(1)}(u_1)$ and $z_l^{(1)}(u_1)$ as in definition 8.1.1.

Note that due to the monotonicity of A, Z_i is achieved with the maximum $d_1 \in D_Z(i-1)$ and $d_2 \in D_Z(i)$, and z_i is achieved with the minimum $d_1 \in D_z(i-1)$ and $d_2 \in D_z(i)$.

We will develop a lemma similar to that in the previous section

Lemma 8.2.1 *Given a sequence of length t, $U = (u_1, \ldots, u_t)$, for any $x \in \mathbb{R}_{max}^n$, we have $x(t+1) = A(u_t) \otimes \ldots \otimes A(u_1) \otimes x$ with*

$$z_i^{(t)}(u) \le x_i(t+1) \oslash x_{i-1}(t+1) \le Z_i^{(t)}(u).$$

Proof: We will prove this lemma by induction. We already have the base case in lemma 8.1.1. Define $x(k) = A(u_{k-1}) \otimes \ldots \otimes A(u_1) \otimes x$ and suppose that

$$z_i^{(k-1)}(U^{k-1}) \leq x_i(k) \oslash x_{i-1}(k) \leq Z_i^{(k-1)}(U^{k-1}). \tag{8.3}$$

Now, considering $x(k+1) = A(u_k) \otimes x(k)$, we wish to determine bounds on $x_i(k) \oslash x_{i-1}(k)$. Suppose that $x_n(k) = 0$ and $x_i(k+1) = a_{ij}(u_k) \otimes x_j(k)$ and $x_{i-1}(k+1) = a_{i-1,p}(u_k) \otimes x_p(k)$. These conditions require

$$\begin{aligned} a_{ij}(u_k) \otimes x_j(k) &\geq a_{il}(u_k) \otimes x_l(k) \qquad \forall l \leq, \\ a_{i-1,p}(u_k) \otimes x_p(k) &\geq a_{i-1,l}(u_k) \otimes x_l(k) \quad \forall l \leq n. \end{aligned}$$

This is

$$a_{il}(u_k) \oslash a_{ij}(u_k) \leq x_j(k) \oslash x_l(k) \qquad \forall l \leq n, \tag{8.4}$$

$$a_{i-1,l}(u_k) \oslash a_{i-1,p}(u_k) \leq x_p(k) \oslash x_l(k) \qquad \forall l \leq n. \tag{8.5}$$

Which leads us to

$$\begin{aligned} a_{ip} \otimes x_p(k) &\leq a_{ij} \otimes x_j(k) \\ \Rightarrow \quad a_{ip} \oslash a_{ij} &\leq x_j(k) \oslash x_p(k). \end{aligned}$$

Similarly we get $a_{i-1,p} \oslash a_{i-1,j} \geq x_j(k) \oslash x_p$. This gives us

$$a_{i-1,p} \oslash a_{i-1,j} \geq a_{ip} \oslash a_{ij}.$$

By the monotonicity of A, this implies $p \leq j$.

Combining (8.4) and (8.5) with (8.3), we get

$$a_{il}(u_k) \oslash a_{ij}(u_k) \leq \bigotimes_{r=l+1}^{j} Z_r^{k-1}(u^{k-1}) \qquad \forall l < j, \tag{8.6}$$

$$a_{ij}(u_k) \oslash a_{il}(u_k) \geq \bigotimes_{r=j+1}^{l} z_r^{k-1}(u^{k-1}) \qquad \forall l > j, \tag{8.7}$$

and

$$a_{i-1,l}(u_k) \oslash a_{i-1,p}(u_k) \leq \bigotimes_{r=l+1}^{p} Z_r^{k-1}(u^{k-1}) \qquad \forall l < p, \tag{8.8}$$

$$a_{i-1,p}(u_k) \oslash a_{i-1,l}(u_k) \geq \bigotimes_{r=p+1}^{l} z_r^{k-1}(u^{k-1}) \qquad \forall l > p. \tag{8.9}$$

We are interested in $x_i(k+1) \oslash x_{i-1}(k+1)$, which equals

$$a_{ij} \otimes x_j(k) \oslash (a_{i-1,p} \otimes x_p(k)).$$

Because $p \leq j$, we get

$$\bigotimes_{r=p+1}^{j} z_r^{(k-1)}(U^{k-1}) \leq x_j(k) \oslash x_p(k) \leq \bigotimes_{r=p+1}^{j} Z_r^{(k-1)}(U^{k-1})$$

This gives us

$$a_{ij}(u_k) \oslash a_{i-1,p}(u_k) \otimes \bigotimes_{r=p+1}^{j} z_r^{(k-1)}(u^{k-1}) \tag{8.10}$$

$$\leq x_i(k+1) \oslash x_{i-1}(k+1) \tag{8.11}$$

$$\leq a_{ij}(u_k) \oslash a_{i-1,p}(u_k) \otimes \bigotimes_{r=p+1}^{j} Z_r^{(k-1)}(u^{k-1}). \tag{8.12}$$

To generalize this result for any p, j, we want to minimize (8.10) and maximize (8.12) subject to the constraints we have shown for p and j. In the case of minimizing (8.10), we want to pick p and j as small as possible.

Considering our constraints, if (8.6) is violated by some j and l, then we have

$$a_{il}(u_k) \oslash a_{ij}(u_k) > \bigotimes_{r=l+1}^{j} Z_r^{k-1}(u^{k-1}).$$

But this means there is some $j' < j$, namely l, such that

$$a_{ij'}(u_k) \oslash a_{ij}(u_k) \geq \bigotimes_{r=j'+1}^{j} z_r^{k-1}(u^{k-1}).$$

Clearly, if j' satisfies the other constraints it will be a better choice than j, so we ignore constraint (8.6) and similarly (8.8) when minimizing (8.10). Using a similar argument, we ignore constraints

(8.7) and (8.9) when maximizing (8.12). Therefore, (8.10 - 8.12) become

$$z_i^{(k)}(U^k) \leq x_i(k+1) \oslash x_{i-1}(k+1) \leq Z_i^{(k)}(U^k).$$

By induction we conclude with the lemma. ■

Using these bounds, we will prove the following lemma.

Lemma 8.2.2 *Given a sequence $U = (u_1, \ldots, u_t)$,*

$$A(u_t) \otimes \ldots \otimes A(u_1) \otimes x^* = \begin{bmatrix} x_n \oslash \left(\bigotimes_{i=2}^{n} z_i^{(t)}(U) \right) \\ \vdots \\ x_n \oslash z_n^{(t)}(U) \\ x_n \end{bmatrix}$$

Proof: We will prove this by induction. The base case is trivially true. We will suppose that

$$\begin{aligned} x(k) &= A(u_{k-1}) \otimes \ldots \otimes A(u_1) \otimes x^* \\ &= \begin{bmatrix} x_n \oslash \left(\bigotimes_{r=2}^{n} z_r^{(k-1)}(U^{k-1}) \right) \\ \vdots \\ x_n \oslash z_n^{(k-1)}(U^{k-1}) \\ x_n \end{bmatrix} \end{aligned}$$

We will calculate $x(k+1) = A(u_k) \otimes x(k)$. Consider $x_i(k+1)$. Suppose that $x_i(k+1) = a_{ij} \otimes x_1(k) \otimes \bigotimes_{r=2}^{j} z_r^{(k-1)}(U^{k-1})$. This implies that

$$a_{ij} \otimes \bigotimes_{r=2}^{j} z_r^{(k-1)}(U^{k-1}) \geq a_{il} \otimes \bigotimes_{r=2}^{l} z_r^{(k-1)}(U^{k-1}) \quad \forall l \leq n$$

Therefore, j is the smallest index for which (8.7) holds. Similarly, $x_{i-1}(k+1) = a_{i-1,p} \otimes x_p(k)$ has the smallest index p for which (8.9) holds. It also follows as before that $p \leq j$. Therefore, we get

$$x_i(k+1) \oslash x_{i-1}(k+1) = z_i^k(U^k).$$

And we conclude with the lemma. ■

Theorem 8.2.1 *Given a sequence of job types, $U = (u_1, \ldots, u_t)$,*

$$\gamma_t(U) \leq \gamma_{t-1}(u_2, \ldots, u_t).$$

Proof: The equation for $\gamma_t(U)$ is given in (8.2). To calculate $\gamma_t(U) \oslash \gamma_{t-1}(u_2, \ldots, u_t)$, which we will hereafter refer to simply as γ_t and γ_{t-1}, we will consider two parts. First we will consider

$$\begin{aligned}
&(||A(u_t) \otimes \ldots \otimes A(u_1) \otimes x^*||_1 \\
&\oslash ||A(u_{t-1}) \otimes \ldots \otimes A(u_1) \otimes x^*||_1) \\
&\oslash (||A(u_t) \otimes \ldots \otimes A(u_2) \otimes x^*||_1 \\
&\oslash ||A(u_{t-1}) \otimes \ldots \otimes A(u_2) \otimes x^*||_1) .
\end{aligned}$$

Using lemma 8.2.2 we write this as

$$\begin{aligned}
&\left[a_{11}(u_t) \oslash \left(\bigotimes_{j=2}^{n} z_j^{(t-1)}(u^{t-1}) \right) \right. \\
&\left. \cdots a_{n-1,1}(u_t) \oslash z_n^{(t-1)}(u^{t-1}), a_{nn}(u_t) \right] \otimes \mathbf{e} \\
&\oslash \left[a_{11}(u_t) \oslash \left(\bigotimes_{j=2}^{n} z_j^{(t-2)}(u_2, \ldots, u_{t-1}) \right) \right. \\
&\left. \cdots a_{n-1,1}(u_t) \oslash z_n^{(t-2)}(u_2, \ldots, u_{t-1}), a_{nn}(u_t) \right] \otimes \mathbf{e}. \\
\leq \quad & e.
\end{aligned}$$

Where the last line is a result of the fact that $z_i^{(t-1)}(U^{t-1})$ is more constrained than $z_i^{(t-2)}(u_2, \ldots, u_{t-1})$, which implies $z_i^{(t-1)}(U^{t-1}) \geq z_i^{(t-2)}(u_2, \ldots, u_{t-1})$

The other part that we wish to consider is

$$\begin{aligned}
&\min_x (||A(u_t) \otimes \ldots \otimes A(u_1) \otimes x||_1 \\
&\oslash ||A(u_{t-1}) \otimes \ldots \otimes A(u_1) \otimes x||_1) \\
&\oslash \min_x (||A(u_t) \otimes \ldots \otimes A(u_2) \otimes x||_1 \\
&\oslash ||A(u_{t-1}) \otimes \ldots \otimes A(u_2) \otimes x||_1) \\
\geq \quad & e.
\end{aligned}$$

Where we get the final line because the first minimization is more constrained, and therefore greater than or equal to the latter minimization.

These two results give us $\gamma_t \oslash \gamma_{t-1} \leq 0$. ■

Chapter 9

Conclusions and Future Work

Bibliography

[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] B. Heidergott, G. J. Olsder, and J. van der Woude. *Max Plus at Work: Modeling and Analysis of Synchronized Systems: A Course on Max-Plus Algebra and its Applications*. Princeton Series in Applied Mathematics. Princeton University Press, 2006.

[6] 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.

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

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

[9] 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.

[10] 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.

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

[12] W. Stallings. *Operating Systems*. Prentice Hall, 4$^{\text{th}}$ edition, 2001.

[13] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, 2$^{\text{nd}}$ edition, 2001.

[14] W. Weyerman, D. West, and S. Warnick. A decision-friendly approximation technique for scheduling multipurpose batch manufacturing systems. *Proc. IEEE American Control Conference*, 2006.