The Asynchronous t-Step Approximation for Scheduling Batch Flow Systems

David R. Grimsman

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Sean Warnick, Chair
John Hedengren
Bryan Morse

Department of Computer Science

Brigham Young University

June 2016

ABSTRACT

The Asynchronous t-Step Approximation for
Scheduling Batch Flow Systems

David R. Grimsman
Department of Computer Science, BYU
Master of Science

Heap models in the max-plus algebra are interesting dynamical systems that can be used to model a variety of tetris-like systems, such as batch flow shops for manufacturing models. Each heap in the model can be identified with a single product to manufacture. The objective is to manufacture a group of products in such an order so as to minimize the total manufacturing time. Because this scheduling problem reduces to a variation of the Traveling Salesman Problem (known to be NP-complete), the optimal solution is computationally infeasible for many real-world systems. Thus, a feasible approximation method is needed.

This work builds on and expands the existing heap model in order to more effectively solve the scheduling problems. Specifically, this work:

1. Further characterizes the admissible products to these systems.
2. Further characterizes sets of admissible products.
3. Presents a novel algorithm, the asynchronous t-step approximation, to approximate these systems.
4. Proves error bounds for the system approximation, and show why these error bounds are better than the existing approximation.

Keywords: Batch flow systems, Scheduling, Max-plus algebra, Dynamic programming, Heap models

ACKNOWLEDGMENTS

This work reflects the support that I have received as graduate student, first and foremost from my wife, Kalisha. She has made extraordinary efforts and sacrifices so that I can have this opportunity to pursue an education. This work is also a tribute to my adviser and friend, Prof. Sean Warnick, who has been my biggest advocate in the academic world to help me accomplish my goals. Finally, thank you to my commitee and my colleagues in the IDeA Labs who have offered criticism and insight to this work.

Table of Contents

1 Introduction **1**

1.1 Problem Formulation 2

2 Related Works **6**

3 Background **9**

3.1 Max-Plus Algebra 9

3.2 Non-Rigid Heap Model 11

3.3 Communication Graphs 13

3.4 Dynamic Programming and Bellman's Equation 15

3.5 The Symmetric t-Step Approximation 17

4 Subsets of $\mathcal{M}^n$ **18**

4.1 Rigid Matrices and Eigenvalues 18

4.2 System Equilibria 20

5 Approximation Method **25**

5.1 Systems that Always Depend on x_0 25

5.2 Symmetric Systems 26

5.3 Asymmetric Systems 28

5.4 Approximation Method 28

5.5 Example 31

6 Error Bounds **35**

6.1 Example 38

7 Future Work **40**

References **42**

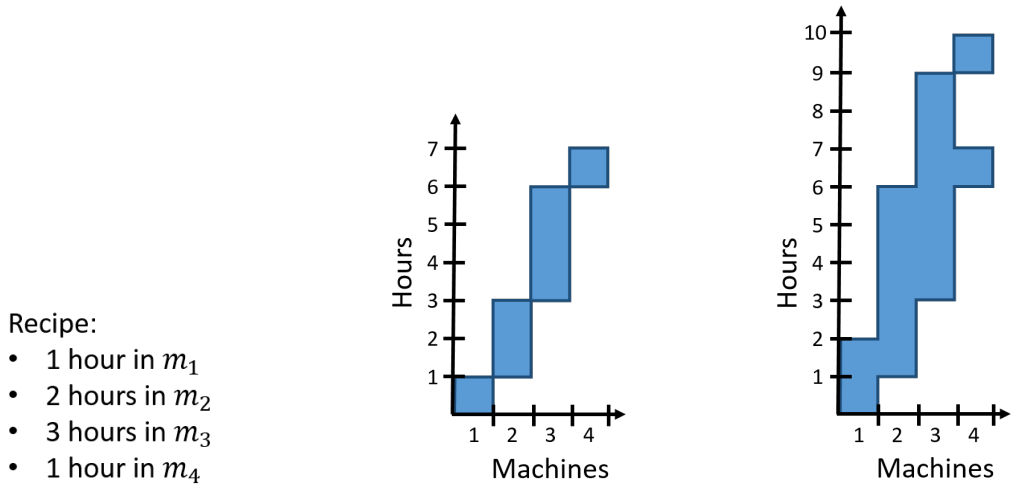
Chapter 1

Introduction

As an introduction to the topic of this work, consider as a prototype application a chemical manufacturer that makes N chemical mixtures using M machines. Denote the chemical mixtures as $c_1, c_2, ..., c_N$ and the machines as $m_1, m_2, ..., m_M$. This work assumes that each machine will only work at full capacity - an important constraint for many machines of this nature. In general, each machine will have a different capacity, denoted as $v_1, v_2, ..., v_M$.

One problem that the manufacturer faces with each machine is how to determine the *batch size* based on the recipes for the chemical mixtures. Because each machine must run at capacity, the batch size is fixed by each recipe. For instance, if $v_1 = 1$ and $v_2 = 2$, then each chemical must run 2 units through m_1 before running through m_2. The first unit will be stored in m_2 to wait for the second. Alternatively, if $v_1 = 2$ and $v_2 = 1$, half the unit in m_1 will have to remain there while the other half is processed in m_2. Therefore, each batch recipe will require an exact amount of c_i be created at a time - this amount is the batch size (see Figure 1.1 for an additional example). While determining the batch size is important, this work assumes that this problem has been solved.

Instead this work focuses on the associated sequencing problem: once the batch recipes have been created and the batch sizes set, given a quota of batch recipes, determine the shortest sequence that meets the quota. For instance, consider a quota of 1 batch of c_1 and 1 batch of c_2. There are 2 possible sequences, and in general, each sequence takes a different amount of time to complete. The solution to the sequencing problem is finding the sequence that minimizes the total time.



(a) (b) (c)

Figure 1.1: Two batch recipes using the same recipe, but with different machine capacities. The recipe is shown in (a). If $v_1 = v_2 = v_3 = v_4$, the block representation for the batch recipe is shown in (b). If $v_1 = v_3 = v_4$ and $v_2 = 2v_1$, the block representation for the batch recipe is shown in (c). Note that the batch size for (c) will be double that of (b), since more of c_i will need to be processed at one time. The focus of this work will not be determining batch size, rather, how to sequence the blocks once they've been created.

One way to model this situation is by using blocks, shown in Figure 1.1. The horizontal component of each block represents each machine in the chemical manufacturing plant. The vertical component is time. Using this model, a sequence of batch recipes is equivalent to stacking these blocks, as shown in Figure 1.2. These blocks are *non-rigid*, since they may "stretch" when waiting for the next machine to complete its processing. This can be considered idle time for the waiting machine.

1.1 Problem Formulation

Generalizing the chemical manufacturing example, a *batch flow shop* has M workstations and N distinct batch recipes. Batch flow shops can be applied to numerous real-world systems, including construction, computer processing, baking, and assembly lines. This work is narrowed to batch flow shops with the following restrictions:

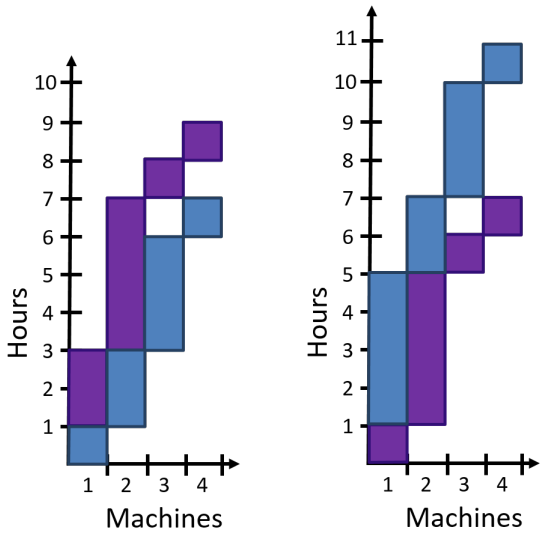


Figure 1.2: This figure shows an example of how the sequence affects the total time. On the left, there is one sequence of two blocks, and on the right, the reverse sequence. Notice how the blocks stretch in order to accommodate times where the mixtures are waiting for the next machine to be available. This is the consequence of the no-intermediate storage and full-capacity constraints in the system.

- A workstation is a single machine, in contrast to environments where a workstation can be multiple machines working in parallel. Initially solving the single machine problem will allow future work to relax this requirement to solve the more general problem.
- There is no intermediate storage, although machines themselves can act as storage. There are situations where intermediate storage is impossible, so this simplification directly maps to some real-world problems. Additionally, a scheduling problem where intermediate storage is allowed would naturally break into multiple scheduling problems where it is not allowed. Thus this constraint forces us to solve the more difficult problem where the machine schedules are coupled.
- Each machine must run at full capacity - also a requirement of many machines, since running at partial capacity can be harmful to the machine.
- Each batch recipe must access the workstations in the same order. This is by definition of a flow shop.

The purpose of this work is to solve the following problem:

Problem 1. *Using the following definitions:*

- ***Items**: let $n = \{n_1, ..., n_N\}$ be a set of batch recipes.*
- ***Sequencing rule**: a sequence s of length K is $n_{i_1} n_{i_2} \dots n_{i_K}$, where $n_{i_j} \in n$. A sequence s is created by running batch recipe n_{i_1}, followed by n_{i_2}, etc.*
- ***Sequence measurement**: the measurement $H(s, x_0)$ is the time it takes for the sequence to complete if x_0 is the state of the machines when s begins.*

Let a quota $q = [q_1\ q_2 \dots q_N]$, where each q_i represents the number of n_i required in the quota. Also, let the set $S_q = \{s \mid s$ is a sequence of length $||q||_1$ and contains q_i of item $n_i\}$. The the problem can be formulated to solve for s^*:

$$s^* = arg \min_{s \in S_q} H(s, x_0) \tag{1.1}$$

Notice that when using the block (or heap) model, the following can be redefined to find an equivalent s^*:

- **Items**: n is a set of blocks
- **Sequencing rule**: a sequence s is a heap of blocks created by stacking block n_{i_1}, followed by n_{i_2}, etc.
- **Sequence measurement**: the measurement $H(s, x_0)$ measures the height of heap s with initial condition x_0.

It is also shown in Chapter 3 that these items can be redefined again in the max-plus algebra when considering a different model for the problem.

For an arbitrarily large quota, Problem 1 can be associated with a variation of the Traveling Salesman Problem, where each city must be visited q_i times, and the distance between cities depends on the previous path. Thus it is NP-Complete and not computationally

tractable. Therefore, the method of this work to solve Problem 1 is to approximate the system and then solve the lower-order approximation.

Chapter 2

Related Works

Theory and algorithms to understand and solve general scheduling and sequencing problems are abundant in the literature (see [1], [26], [22] for example). Because this problem is known to NP-Complete, tractable algorithms can only guarantee an estimate of the optimal schedule or sequence [12]. Thus, modeling the specifics of a problem instance can lead to better estimates. To this end, some researchers have created problem formulations specific to batch scheduling and sequencing. For instance, [8] considers multi-purpose batch schedules, where alternative routing is allowed. The authors formulate the problem as a mixed integer non linear optimization. In another case, [20] considers batches that can be merged or split as required, where the problem is formulated as a mixed integer linear programming problem.

This work only considers batch flow shops, where all products follow the same sequence of production. In this space, much work has been done to solve the scheduling problem, however, the solution depends on the objective. In [27], various objectives are presented, including minimizing makespan, maximizing throughput, minimizing weighted tardiness, and minimizing lead time. For instance, consider the objective to minimize weighted tardiness. With this objective each job has a due date, and the goal is to make sure that every job gets completed before its due date. Therefore, each job has a different priority and the solution to the scheduling problem is a schedule that minimizes the tardiness (in a weighted sense) of each job. An algorithm for computing an estimate for this type of schedule is found in [29].

As shown in Problem 1, the objective for this work is to minimize the makespan, or total manufacturing time. Traditionally, this has been the objective for batch flow shop

scheduling problems, but still the models and methods vary. For instance, [19] considers an algorithm for a simple two-machine job, later enhanced in [18]. Another simplified model is the single-machine with setup costs, examined in [31] and [4]. Other research, such as [25], schedules the jobs according to a heuristic called "slope index", showing optimality bounds on this technique. Later work has focused on using variations of branch-and-bound techniques, each with its own heuristic. For a survey on such heuristics, see [6]. Using the properties developed in max-plus algebra, this work seeks to create a new heuristic for the algorithm for the batch flow shop scheduling problem.

One complexity of batch systems is determining the optimal capacity, also called the lot sizing problem. The literature shows that others have developed methods to estimate the optimal lot sizing (see [7], [10], [15]). This work assumes that the lot sizing problem has already been solved. Additionally, this work assumes that the batch sizes have already been determined using the process in [33]. Therefore, the recipe for each batch recipe is given and the scheduling problem is reduced to a sequencing problem.

Using the heap model to simulate the dynamics of the batch flow shop was first developed by Gaubert and Mairesse [13]. This model equates a batch recipe with a rigid tetris-like block, and thus the objective to minimize the makespan is equated to the objective to minimize the height of the heap of blocks. Following in this tradition, a model for non-rigid heaps, as described in Chapter 1, was created in [33]. This work is meant to extend the work done in [33] using the non-rigid heap model.

In [13], Gaubert and Mairesse also used the max-plus algebra as a tool for problem formulation and analysis. The max-plus algebra is precisely defined in Chapter 3, but suffice it to say here that it is a semiring over the real numbers and $-\infty$. The authors show that each batch recipe is equivalent to a matrix representation, where the multiply operator simulates stacking. This definition is also used in [33], but the work is limited to a closed set of matrices representing the non-rigid heaps - the set $\mathcal{M}^n$. Thus, this work can leverage much of the max-plus theory created in the literature. For instance, this work relies heavily on the

spectral theory presented in [17] to help analyze the properties of these matrices. Specifically, $\mathcal{M}^n$ is a subset of matrices called *irreducible* matrices. The work in [17] also contains proofs using connection graphs, which are also utilized here.

Irreducible matrices have also been well-studied. In [5], the author presents properties of powers of irreducible matrices. Also in [24] an analysis of sequences of matrices that exhibit a memory-loss property is presented. This property is fundamental in this work for an approximation algorithm.

In Chapter 5 a method is presented to approximate the systems described in Chapter 1. Once this approximation is completed, a dynamic programming solver can be used to solve the simpler system. Dynamic programming is a well-studied topic, with many applications, including control theory [3], artificial intelligence [30], and operations research [28]. Methods include Q-learning [32], SARSA, approximate value functions, and step size rules. Most of these methods, however, assume an infinite horizon, whereas this work is concerned with finite horizons, represented by the quota. For this subset of dynamic programming problems, algorithms such as Dykstra's algorithm [9] or A* [16] can provide an exact solution. If the approximation remains intractable for one of these algorithms, other algorithms have been developed for further simplification, including IDA* [21] and SMA*[11].

Chapter 3

Background

3.1 Max-Plus Algebra

This project leans heavily on analysis in the max-plus algebra, which has the following conventions:

- $a \oplus b = \max\{a, b\}$
- $a \otimes b = a + b$
- $a \oslash b = a - b$
- $\varepsilon = -\infty$

The max-plus semiring $\mathbb{R}_{max}$ is the set $\mathbb{R} \cup \{-\infty, \infty\}$ equipped with the above operations. Matrix operations are also defined for max-plus, analogous to normal matrix algebra. For $A, B \in \mathbb{R}_{max}^{n \times l}$, $C \in \mathbb{R}_{max}^{l \times m}$:

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

Likewise, if x, $y \in \mathbb{R}_{max}^{n}$, then $y = A \otimes x$ with the following methodology:

$$y_i = \bigoplus_{j=1}^{l} a_{ij} \otimes x_j \tag{3.1}$$

Here the notation for subscripts on a vector is abused: y_i is the i^{th} component of y. The rest of this work uses x_k to indicate the value of vector x at time k unless otherwise noted. Using

these definitions, an autonomous linear system for max-plus algebra can be defined. For $x(k) \in \mathbb{R}^n_{max}$, $k \in \mathbb{N}$, where $\mathbb{N}$ denotes the non-negative integers, and $A \in \mathbb{R}^{n\times n}_{max}$, the following recursion-generated sequence

$$x_{k+1} = A \otimes x_k, \tag{3.2}$$

is well-defined and characterized by

$$x_k = A^{\otimes k} \otimes x_0,$$

where $x_0 \in \mathbb{R}^n_{max}$ is the initial condition, and $A^{\otimes k}$ denotes the matrix A raised to the k^{th} power, in the max-plus sense of matrix multiplication.

Definition 1. *Let $x \in \mathbb{R}^n_{\max}$. The vector $\mathbb{C}(x) \in \mathbb{R}^n_{\max}$ is called the contour vector of x and is given by $\mathbb{C}(x) = x \oslash \min x$, where $\min x$ is the minimum element of x.*

One important property to note for the systems considered is that the first element in x is always the minimum value, hence the first element of $\mathbb{C}(x)$ is always 0.

Definition 2. *Let $x,\ y \in \mathbb{R}^n_{\max}$. Then x and y are said to have the same contour if $\mathbb{C}(x) = \mathbb{C}(y)$.*

In [24] and [14] a memory-loss property (MLP) is described for arbitrary sequences of max-plus matrices. The property is restated here as it pertains to this work:

Definition 3. *Consider a sequence of p matrices $A_1, A_2, ...A_p \in \mathbb{R}^{n\times n}_{\max}$. The sequence is defined as having the memory-loss property (MLP) if $\mathbb{C}(A_1 \otimes A_2 \otimes ... \otimes A_p \otimes x_0)$ is independent of x_0.*

The asymptotic behavior of sequences of matrices has been studied in several places, including [23]. Note that when $A_1 = A_2 = ... = A_n$, the sequence characterizes the system described in (3.2). In this case it is said that the matrix A has the MLP when $\mathbb{C}(x(k))$ is independent of x_0 for sufficiently large p.

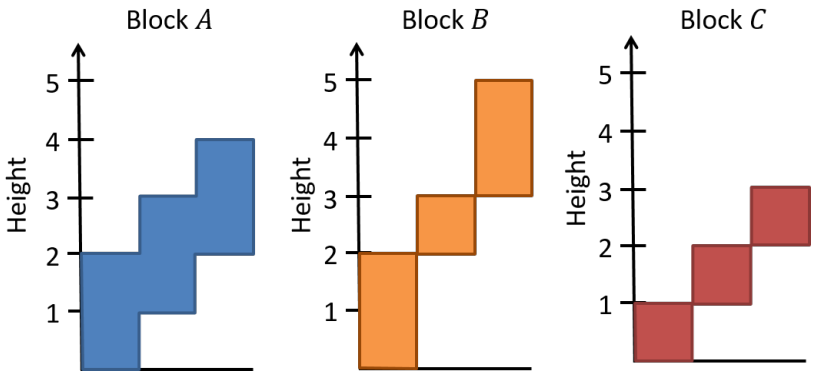


Figure 3.1: Blocks A, B, and C.

Block	Upper Contour	Lower Contour	Matrix
A	$u = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$	$l = \begin{bmatrix} e \\ 1 \\ 2 \end{bmatrix}$	$M(A) = \begin{bmatrix} 2 & 1 & e \\ 3 & 2 & 1 \\ 4 & 3 & 2 \end{bmatrix}$
B	$u = \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix}$	$l = \begin{bmatrix} e \\ 2 \\ 3 \end{bmatrix}$	$M(B) = \begin{bmatrix} 2 & e & \varepsilon \\ 3 & 1 & e \\ 5 & 3 & 2 \end{bmatrix}$
C	$u = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	$l = \begin{bmatrix} e \\ 1 \\ 2 \end{bmatrix}$	$M(C) = \begin{bmatrix} 1 & e & \varepsilon \\ 2 & 1 & e \\ 3 & 2 & 1 \end{bmatrix}$

Table 3.1: Mathematical modeling of a 3-block system. Note that these blocks correspond to the shapes in Figure 3.1, and the upper contour of stacking shapes, as in Figure 3.2, corresponds to the result of matrix multiplication of the corresponding matrices operating on the appropriate initial condition.

3.2 Non-Rigid Heap Model

The stacks of "stretchy" blocks described in Chapter 1 are referred to as *non-rigid heaps*. Each block P has a corresponding matrix M that can be created using u and l, the upper and lower contour vectors of P. The process is found in [33]:

$$[M(P)]_{ij} = \begin{cases} u_i(P) - l_j(P) & \text{if } u_i(P) - l_j(P) \geq 0 \\ \varepsilon & \text{otherwise.} \end{cases} \tag{3.3}$$

Here again the vector subscripts in u and l represent elements in the vectors. For the blocks in Figure 3.1, the corresponding matrices are found in Table 3.1. The key to

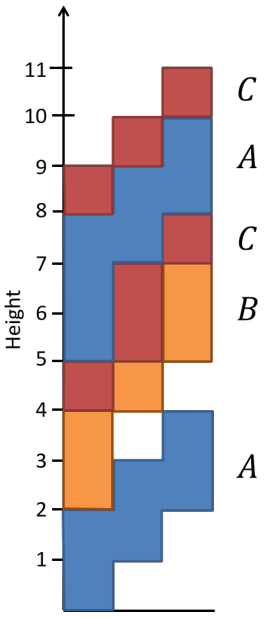


Figure 3.2: Heap resulting from processing products A, B, C, A, and C from the example in Figure 3.1. This shows that blocks can take different shapes depending on the state of the system, or shape of the contour below the block.

using max-plus algebra in this context is that stacking blocks is equivalent to multiplying the corresponding matrices. Thus, the theory from the previous section can be leveraged to gain insight into these types of systems. As an example, consider an initial condition $x_0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ followed by the sequence A, B, C, A, C. The resulting non-rigid heap is shown in Figure 3.2. Using max-plus multiplication,

$$\begin{aligned} x_{final} &= C \otimes A \otimes C \otimes B \otimes A \otimes x_0 \\ &= \begin{bmatrix} 9 & 10 & 11 \end{bmatrix}^T \end{aligned} \tag{3.4}$$

Each entry in x_f corresponds to the height of the resulting heap at each workstation as illustrated in Figure 3.2. A heap resulting in stacking the same block A repeatedly would result in the linear system described in (3.2). Also note that the overall height of the heap is given by the max entry in x_f, or $||x_f||_\infty$.

Definition 4. [33] *A matrix A is in $\mathcal{M}^n \subset \mathbb{R}_{max}^{n\times n}$ if:*

$$a_{i+1,j} \geq a_{ij}, \;\; i \leq n-1, \; j \leq n, \tag{3.5}$$

$$a_{ij} \geq a_{i,j+1}, \;\; i \leq n, \; j \leq n-1, \tag{3.6}$$

$$\xi_{1j}(A) \geq \ldots \geq \xi_{nj}(A), \;\; j \leq n, \tag{3.7}$$

$$a_{ij} > -\infty, \;\; j \leq i+1, \; i \leq n, \; j \leq n. \tag{3.8}$$

The set $\mathcal{M}^n$ is a subspace and every matrix created using (3.3) is in $\mathcal{M}^n$. Additionally, $\mathcal{M}^n$ is closed under the max-plus multiply operation. Thus the focus of this work is limited to those matrices in $\mathcal{M}^n$, and Problem 1 can be equivalently redefined as follows:

- **Items**: n is a set of matrices in $\mathcal{M}^n$
- **Sequencing rule**: a sequence s is a chain of max-plus matrix multiplications beginning with x_0 and then multiplying by n_{i_1}, followed by n_{i_2}, etc.
- **Sequencing measurement**: the measurement $H(s, x_0)$ is $||x_K||_\infty$, where x_K is the resulting vector from the sequence s.

3.3 Communication Graphs

Studying the properties of matrices in $\mathcal{M}^n$ often requires insight from a corresponding model: the graph model. Viewing a matrix $A \in \mathcal{M}^n$ as an adjacency matrix for a communication graph, $G(A)$, additional properties can be inferred. $G(A)$ is created from A with the following convention: the arc from node i to node j in $G(A)$ has weight a_{ij}. If $a_{ij} = \varepsilon$, then no arc exists in $G(A)$ from i to j (see Figure 3.3).

Multiplying two matrices A and B together is equivalent to connecting $G(A)$ and $G(B)$ together and solving for a maximum path. In order to better understand this process, this project leverages the following graph properties for matrices in $\mathcal{M}^n$:

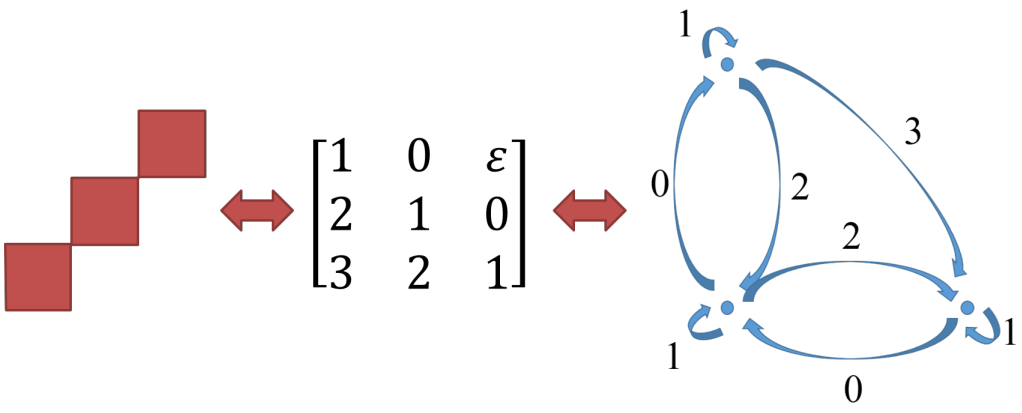


Figure 3.3: A simple block with its corresponding matrix and graph

1. A *circuit* in $G(A)$ is a path that begins and ends with the same node. A circuit is *elementary* if the nodes in the circuit have only one incoming and one outgoing arc [17]. A *critical circuit* is one that has maximum average weight per arc for $G(A)$.

2. The *critical graph* of A, $G_c(A)$ is the subgraph of $G(A)$ comprised only of nodes and arcs from critical circuits.

3. The *cyclicity* of $G(A)$ is the greatest common divisor of all the lengths of the elementary circuits in $G(A)$.

4. $G(A)$ is *strongly connected* if every node in $G(A)$ communicates with every other node.

5. A is *irreducible* if there exists a path in $G(A)$ from every node to every other node.

6. A *maximal strongly connected subgraph (MSCS)* of $G(A)$ is a strongly-connected subgraph of $G(A)$ that is not strongly connected to any other subgraph. An MSCS must have at least 1 arc.

7. An irreducible matrix is *primitive* if for every MSCS of $G_c(A)$, the lengths of all cycles are coprime.

3.4 Dynamic Programming and Bellman's Equation

Dynamic programming is a well-studied method that was first introduced by Richard Bellman in the 1950s in order to solve multistage decision problems [2]. These problems assume the following elements:

- A state space X that contains every possible state of the system x_k for all k
- An input space U that contains every possible input u_k for all k
- A random vector of information w_k whose value is not known until after u_k has been applied to x_k.
- A transition function $f(x_k, u_k, w_k)$ that governs how x evolves
- A cost function $c(x_k, u_k)$ that gives the cost of taking action u_k while in state x_k, but before w_{k+1} is applied

The goal is to choose u_k so as to minimize some objective function. However, one does not know in advance which u_k is best until time period k - in other words, it depends on the value w_{k-1}, which was randomly given by the system. Thus, it is necessary to use feedback of the current state in order to guide the choice for u_k. The solution to a dynamic programming problem then comes in the form of an optimal policy π^*, which can be considered a function or look-up table for how u_k should be chosen based on x_k.

In his original work, Bellman introduced a function $V(x)$, which represents the best possible total value achieved when starting at state x. Then $V(x_0)$ could be recursively solved for using the following:

$$V(x) = \min_{u \in U}(c(x, u) + \mathbb{E}\{V(f(x, u, w))\}) \tag{3.9}$$

In other words, $V(x_k)$ can be obtained by finding the best u_k to minimize the current cost function plus $V(x_{k+1})$. But, in order to find $V(x_{k+1})$, one needs to know $V(x_{k+2})$ and so on until the end of the horizon at time K. Once $V(x_K)$ is known, $V(x_0)$ can be found by

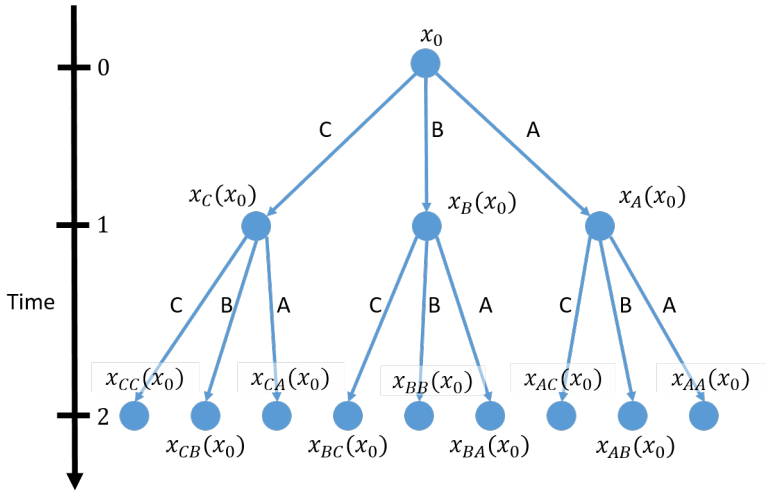


Figure 3.4: A decision tree for the blocks in Figure 3.1 for $k = 0, 1, 2$. The weight on each edge is the incremental height to add the corresponding block to the top of the heap. For the heap shown in Figure 3.2, notice adding B on top of A increments the height of the heap by 3 for $x_0 = 0$. Thus the weight of the branch between x_A and x_{AB} is 3.

working backwards. This also produces policy π^*, because u_k is simply chosen as the one that gives the best current value plus expected value for the next time step.

Problem 1 can be solved using dynamic programming. In this system, x_k is the value of the upper contour of the heap after k blocks have been stacked, u_k is the block stacked at time k (this also takes into account the quota), $f(x, u)$ is the sequencing operation, $c(x_k, u_k)$ is the amount that the height of the stack increments when the block chosen by u is placed on a stack with upper contour x, and K is the size of the quota. This system is simpler than the general multistage decision process described above, in that there is no random information w_k, since it is assumed that the job types are fixed. With this assumption, π^* becomes a sequence $u_0, ..., u_{K-1}$ that can be completely determined in advance.

The biggest drawback of this approach is what Bellman termed *the curse of dimensionality*, in other words, for large problems, this solution technique is not tractable. In Problem 1, this curse is found in the number of possible states at each time k. For instance, consider the blocks in Figure 3.1. For $k = 2$, all possible states are shown in the decision tree in Figure 3.4. Note that the number of states is 3^k, and each state could potentially depend

on x_0. Therefore, without mitigation, it would quickly be intractable to consider all possible paths in the tree.

3.5 The Symmetric t-Step Approximation

The reason for this explosion in the state space is because each x_k could depend on x_0. Therefore, it is unclear whether whether all possible x_k are the same as x_{k+1} and whether any of the edge weights repeat. The symmetric t-step approximation (STA), as presented in [33], is the current approach for mitigating this complexity. Rather than computing the weights for all edges back to $k = 0$, a value t is chosen which represents how far to look back to estimate what states are possible at k.

For example, consider again the decision tree in Figure 3.4 and assume $t = 2$ and $x_0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$. The value of the state $x_{BBB} = \mathbb{C}(B^{\otimes 3} \otimes x_0) = \begin{bmatrix} 0 & 1 & 3 \end{bmatrix}^T$. However, if $x_0 = \begin{bmatrix} 0 & 0 & 4 \end{bmatrix}^T$, then $x_{BBB} = \begin{bmatrix} 0 & 2 & 4 \end{bmatrix}^T$. Under the STA, $x_{BBB} = x_{BB} = C(B^{\otimes 2} \otimes \begin{bmatrix} \varepsilon & \varepsilon & 0 \end{bmatrix}^T) = \begin{bmatrix} 0 & 2 & 3 \end{bmatrix}^T$, and the actual value of x_0 is not considered. A more detailed discussion of the STA and how it affects decision trees is found in Chapter 5.

Chapter 4

Subsets of $\mathcal{M}^n$

In order to create a better approximation than that provided by STA for the state space of Problem 1, it is helpful to understand some partitions of matrices within the set $\mathcal{M}^n$. This chapter focuses on properties of individual matrices and sequences of a single matrix repeated. Chapter 5 then focuses on heterogeneous sequences of matrices.

4.1 Rigid Matrices and Eigenvalues

Definition 5. *A matrix $A \in \mathcal{M}^n$ is described as rigid if $\mathbb{C}(A \otimes x)$ does not depend on x, where $x \in \mathbb{R}^n_{\max}$.*

In terms of the block model, a rigid block is one that has a fixed upper contour regardless of the state of the heap it is stacked on. A rigid matrix by definition has the MLP, but some non-rigid matrices may also have the MLP. Thus only some of the matrices in $\mathcal{M}^n$ are rigid. The next theorem completely characterizes these matrices.

Theorem 6. *$A \in \mathcal{M}^n$ is rigid if and only if A is rank 1 in the max-plus sense.*

Proof. Suppose A is rank 1. Then $\dim(\mathcal{R}(A)) = 1$ and $A \otimes x = c \otimes x_A$ for some x_A and scalar c, and for all x. Since $\mathbb{C}(c \otimes x_A) = \mathbb{C}(x_A)$, then A is rigid from Definition 5. Conversely, if A is rigid, then $A \otimes x$ is independent of x. Call the vector $A \otimes x$, x_A. Then $A \otimes x = c \otimes x_A$ for all x. Thus, A is rank 1. □

Theorem 6 yields a quick way to find whether a matrix is rigid: simply test whether it is rank 1. An example of such a matrix is A in Table 3.1: $\mathbb{C}(A \otimes x) = \begin{bmatrix} 0 & 1 & 2 \end{bmatrix}^T$ for all x.

Now $\mathcal{M}^n$ is divided into 2 subsets of matrices: rigid and non-rigid. To take a closer look at the properties of both requires graph theory.

Lemma 7. *Every $A \in \mathcal{M}^n$ has cyclicity 1.*

Proof. By definition, the cyclicity is the greatest common divisor of the lengths of the elementary circuits in $G(A)$. Since $a_{ii} \neq \varepsilon$ for $i = 1, 2, ..., n$, $G(A)$ has self-loops at every node. This implies that $G(A)$ has elementary circuits with length one. No matter what other critical circuits exist in $G(A)$, the greatest common divisor for the lengths of all the circuits must be 1. Therefore, the cyclicity is 1. □

Lemma 8. *Every matrix $A \in \mathcal{M}^n$ is irreducible.*

Proof. From our set $\mathcal{M}^n$, since ε's can only exist in the upper right corner strictly above the first superdiagonal, it follows that every node in $G(A)$ has a path to every other node. Thus, every matrix $A \in \mathcal{M}^n$ is also irreducible. □

This result allows us to take advantage of other results concerning irreducible matrices.

Lemma 9. [17] *Let A be an irreducible matrix. Then there exists a positive integer c (the cyclicity of A), $\lambda \in \mathbb{R}_{max}$ (unique eigenvalue of A), and a positive integer K such that*

$$\forall k \geq K,\ A^{\otimes k+c} = \lambda c \otimes A^{\otimes k} \tag{4.1}$$

Because every matrix in $\mathcal{M}^n$ has cyclicity 1, (4.1) can be modified to the following:

$$\forall k \geq K,\ A^{\otimes k+1} = \lambda \otimes A^{\otimes n} \tag{4.2}$$

This implies that for the linear system described in (3.2), $\mathbb{C}(x_k)$ converges to a single vector. In block model terms, a block stacked on itself over and over converges to an upper contour. More precisely stated, there exists K such that $x_{k+1} = \lambda \otimes x_k$ for $k \geq K$. Note also that x_k is an eigenvector for A, as is $c \otimes x_k\ \forall c \in \mathbb{R}$.

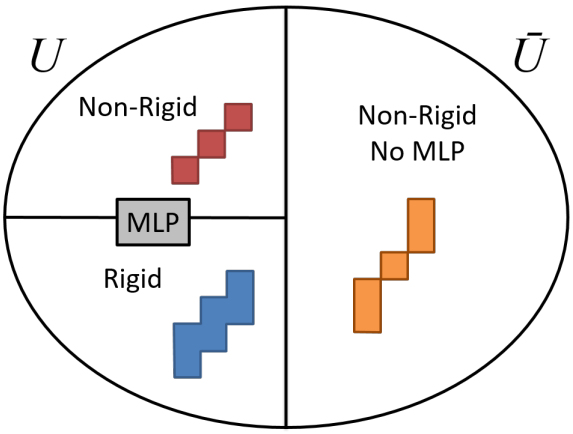


Figure 4.1: This is the set $\mathcal{M}^n$, partitioned to reflect fundamentally different kinds of blocks. The right partition is $\bar{U}$, comprised of non-rigid matrices without the MLP. The left is U, with both rigid and non-rigid matrices that have the MLP. An example of each type of block from Table 3.1 is shown.

Theorem 10. *Let $A \in \mathcal{M}^n$. Then A has a unique eigenvalue λ equal to its maximum diagonal entry.*

Proof. According to [17], knowing that A is irreducible implies that λ is unique and that its value is the maximum average circuit weight for $G(A)$. It is also known from [33] that the average circuit weight for every circuit in $G(A)$ is less than or equal to the maximum diagonal entry. Therefore, the maximum diagonal entry is the eigenvalue. □

4.2 System Equilibria

This section attempts to further classify the matrices in $\mathcal{M}^n$ by looking at the eigenvectors and revisiting the MLP.

Definition 11. *For matrix A, $dim_\lambda(A)$ is the dimension of the eigenspace of A in the max-plus sense. Note that $dim_\lambda(A) = 1$ if and only if A has the MLP. Also, $dim_\lambda(A) = 1$ is equivalent to the statement that the system in* (3.2) *has a unique equilibrium.*

Because rigid matrices have the MLP, it follows that $dim_\lambda(A) = 1$. However, this need not be the case for all matrices in $\mathcal{M}^n$ - see Figure 4.1. In order to find an easy way to distinguish matrices that have the MLP, some results from the literature are restated.

Lemma 12. [17] *Let $A \in \mathcal{M}^n$ Then every vertex in the critical graph of A has a self loop.*

This lemma is intuitive given the results and proof of Theorem 10. This also implies that the vertices in the critical graph correspond to the maximum diagonal entries in A.

For the following lemma, note that in [5], a *strongly irreducible* matrix A is one where $A^{\otimes k}$ is also irreducible for every value of k. Since $A \in \mathcal{M}^n$ implies $A^{\otimes k} \in \mathcal{M}^n$, every matrix in $\mathcal{M}^n$ is strongly irreducible by Lemma 8.

Lemma 13. [5] *For a strongly irreducible A, $dim_\lambda(A^{\otimes k}) = \sum_i GCD(r_i, k)$, where*

- *$GCD(x, y)$ is the greatest common denominator between x and y*
- *r_i is the GCD of the lengths of all critical cycles of A in the i^{th} connected component of $G_c(A)$.*

If $A \in \mathcal{M}^n$, then $r_i = 1$ (Lemma 12), $GCD(r_i, k) = 1$, and therefore $dim_\lambda(A^{\otimes k}) = \sum_i GCD(r_i, k) =$ # maximal strongly connected subgraphs (MSCS's) in $G_c(A)$.

Theorem 14. *Assume a matrix $A \in \mathcal{M}^n$. The following are equivalent statements:*

1. *$dim_\lambda(A) = 1$*
2. *$G_c(A)$ has a single MSCS*
3. *A has the MLP.*

Proof. The equivalence of 1) and 2) follow from the above discussion. Because the $dim_\lambda(A) = 1$, $\mathbb{C}(x(k))$ is unique and independent of x_0 for sufficiently large k. Therefore, 3) is equivalent. □

Thus, in $\mathcal{M}^n$, a simple test for the MLP is whether $G_c(A)$ has multiple MSCS's. This also implies that matrices in $\mathcal{M}^n$ with the MLP are dense in the space. With small perturbations to the diagonal entries, a matrix without the MLP can easily be converted to a matrix with the MLP. In terms of blocks, this corresponds to making small modifications to the upper and lower vectors.

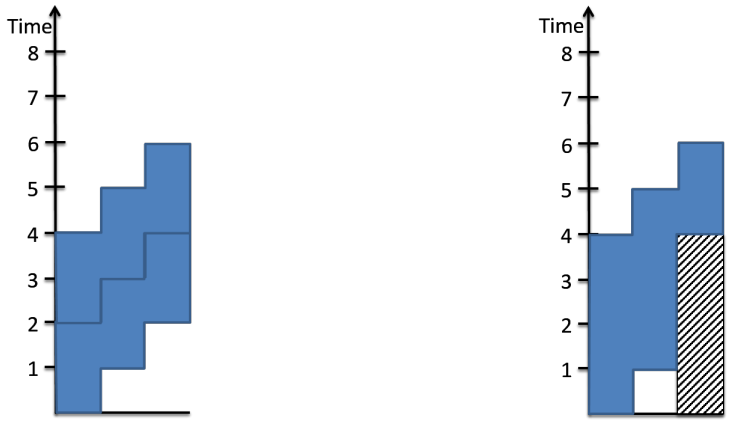


Figure 4.2: Block A is a rigid element of $\mathcal{M}^n$, which means that the resulting upper contour is invariant to initial conditions ($x_0 = [0\ 0\ 0]^T$, left, and $x_0 = [0\ 0\ 4]^T$, right, respectively). Note that as an element of $\mathcal{M}^n$, block A still stretches without affecting its upper contour (right figure).

While limiting our system to one block may seem overly restrictive, note that a single block can represent any number of blocks already sequenced together. Thus, the analysis of a single matrix also leads to conclusions about periodic sequences.

This section concludes by showing an example matrix for each subset. Again the blocks in Figure 1 are employed. Block A is a rigid matrix in the partition U (see Figure 4.1). As expected, one can see in Figure 4.2 that the heap immediately converges to an upper contour, and that upper contour is the same, regardless of the initial condition. Block C corresponds to a non-rigid matrix in the partition U. In Figure 4.3, the upper contour after stacking the block once is different depending on the initial condition. However, notice that the upper contour at convergence is the same either way. Block B corresponds to a matrix in the partition $\bar{U}$. Notice in Figure 4.4 regardless of the initial condition, the upper contour converges after stacking it once. This is not indicative of matrices in $\bar{U}$ in general, but it is convenient for an example. Notice also that the upper contour is different on the left than on the right, so the upper contour at convergence is dependent on the initial condition.

The point of this discussion is to show that there exist sequences of matrices of size 1 whose equilibrium does not depend on x_0, yet there also exist large finite sequences whose

Figure 4.3: Block C is non-rigid and has a unique equilibrium. After stacking it once, the resulting upper contour is different depending on what the initial condition is. However, after stacking the block again, the upper contour is the same because it converged to the same equilibrium value regardless of the initial condition.

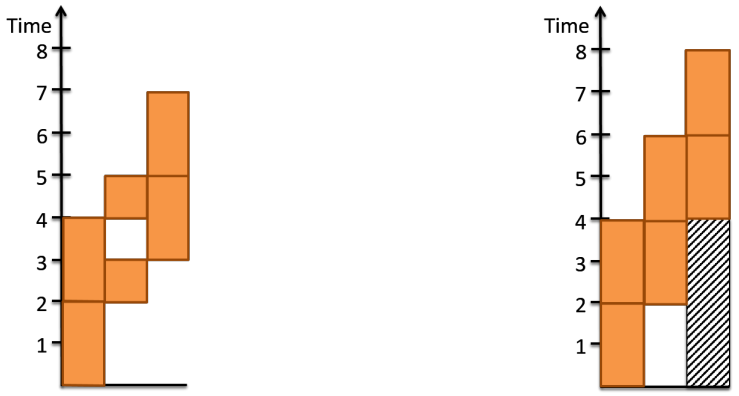


Figure 4.4: Block B is non-rigid and has multiple equilibria. Like block C, this block also converges after being stacked twice. Nevertheless, the resulting upper contour is different depending on the initial condition.

equilibrium does. This chapter characterizes these two groups using spectral and graph analysis in the max-plus algebra.

Chapter 5

Approximation Method

This chapter seeks to characterize sequences of matrices, and then use that characterization to develop a method for approximating the system described in Problem 1.

5.1 Systems that Always Depend on x_0

As described in the previous chapter, there are some groups of matrices that, when multiplied together in some order, never exhibit the MLP. This means that regardless of how long the system runs, the possible states at any time k are dependent on the initial condition.

As an example, consider a non-rigid heap system, as in Equation (3.2), with only two possible block types. The first block is given by block B from Table 3.1, and the second block is described by the following matrix:

$$D = \begin{bmatrix} 3 & e & \varepsilon \\ 4 & 1 & e \\ 7 & 4 & 3 \end{bmatrix}. \tag{5.1}$$

Now suppose that the objective is to sequence two B blocks and four D blocks in a heap with minimal total height.

When considering this problem, there are 15 possible sequences to choose from. Out of these, 10 result in a heap with the property that the upper contour of the heap depends on the initial condition. For instance, the sequence

$$D \otimes B \otimes D \otimes B \otimes B \otimes B = \begin{bmatrix} 16 & 13 & 11 \\ 18 & 15 & 14 \\ 20 & 17 & 16 \end{bmatrix} \tag{5.2}$$

is non-rigid, since the max plus rank is 2. The initial condition $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ yields a resulting contour $\begin{bmatrix} 0 & 2 & 4 \end{bmatrix}^T$ while the initial condition $\begin{bmatrix} 0 & 0 & 5 \end{bmatrix}^T$ yields a resulting contour $\begin{bmatrix} 0 & 3 & 5 \end{bmatrix}^T$. Thus this path in the system is dependent on x_0.

5.2 Symmetric Systems

Some non-rigid heap systems are characterized by a set of blocks that become rigid after a fixed number of steps no matter how they are combined. These systems exhibit a degree of deadbeat-like behavior symmetrically. For example, consider a system with the following three block matrices:

$$E = \begin{bmatrix} 0 & 0 & \varepsilon \\ 3 & 3 & 1 \\ 5 & 5 & 3 \end{bmatrix}, F = \begin{bmatrix} 0 & 0 & \varepsilon \\ 2 & 2 & e \\ 6 & 6 & 4 \end{bmatrix}, G = \begin{bmatrix} 4 & 1 & \varepsilon \\ 7 & 4 & 2 \\ 7 & 4 & 2 \end{bmatrix} \tag{5.3}$$

Any multiplication of two of these matrices together yields a rigid matrix. Thus, every sequence in the system has the MLP and all initial conditions are lost after 2 time steps. This allows an enormous reduction in the possible state values the system can exhibit. If each choice of matrix led to a unique state, then after k time steps, there would be 3^k possible states. However, knowing that the system can be approximated symmetrically with $k = 2$ means that there are really only 9 possible states (1 for every potential sequence of 2) at time

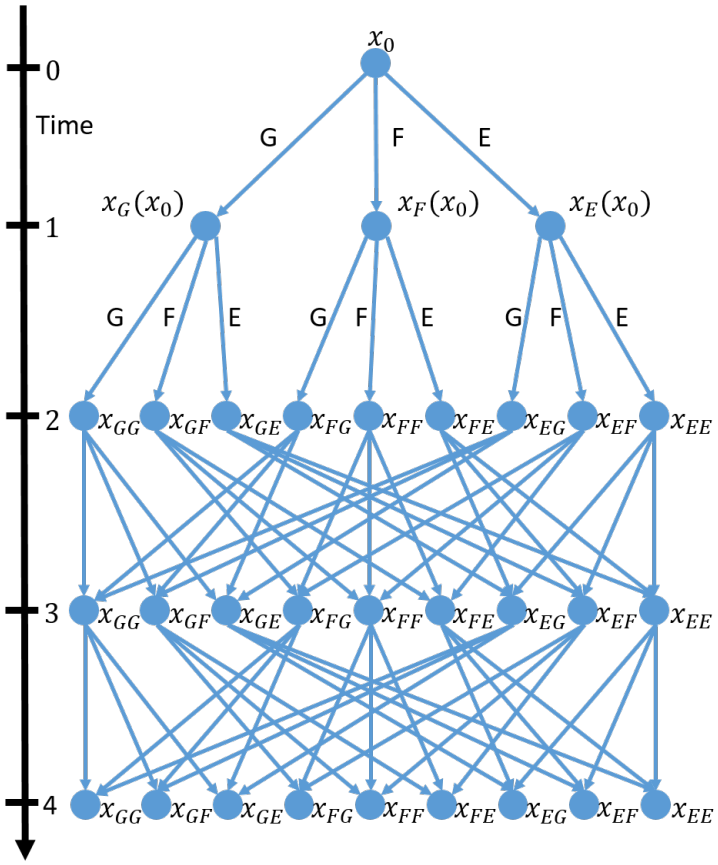


Figure 5.1: The right edge coming out of each node corresponds to choosing E, the middle F, and the left G. Note that the possible states of the system at time 2 is the same as those at time 3 and at time 4. Additionally, the edge weights between all states at time 2 and time 3 are the same as those between 3 and 4

step 2. Additionally, *these 9 states are the only ones that are possible for all remaining time steps.*

The decision tree for this system is shown in Figure 5.1. Notice that all states at time 1 are functions of the initial condition, but all states at time 2 are not. Notice also that at time step 2, the path EE leads to the far right state. Likewise, starting from any other state, such as the F state in time 1, will end up in the far right state after two subsequent steps choosing EE. Thus once the topology of the graph is understood for the transition between time steps 2 and 3, the rest of the graph is known until the quota is complete. Note that rigid heap systems, or systems with blocks that are all rigid, are just special cases of systems with symmetric approximation, where the symmetric approximation occurs in only one time step.

5.3 Asymmetric Systems

Finally, the most generic non-rigid heap system is one characterized by a set of block matrices such that some combinations become rigid while others do not. As a result, some paths in the decision tree take longer to exhibit rigid behavior than others. In this context, consider again the matrices A and C in Table 3.1, combined with the following matrix:

$$L = \begin{bmatrix} 8 & e & \varepsilon \\ 9 & 2 & e \\ 12 & 6 & 4 \end{bmatrix} \tag{5.4}$$

Recall that A is rigid, while C and L are not. Some length-2 sequences in the system are non-rigid, including CC, and some length-3 sequences are non-rigid, including LLL. All length-4 sequences are rigid. Therefore, a similar shrinking of the possible state space can be seen, illustrated in Figure 5.2. Note the number of arrows connecting to the far right state, which is the state after choosing A, and only one arrow to the far left state, corresponding to the sequence $LLLC$. Figure 5.2 also shows that all possible states are known at by $k = 4$.

5.4 Approximation Method

The goal of this section is to present a novel approximation method for a given system that shrinks the state space while staying as close as possible to the original dynamics. In [33], recall that the STA is developed and error bounds are shown. Essentially, this method approximates a system with a symmetric approximation: all paths of length t are assumed rigid. As t increases, the closer the approximation becomes to the actual system.

Understanding the dynamics of these systems, however, suggests that one can do better given complexity constraints. An asymmetric t-step approximation (ATA) is proposed in this section. This approximation leverages the fact that once a sequence is rigid, there is no need to traverse the decision tree beyond that point when searching for the state value.

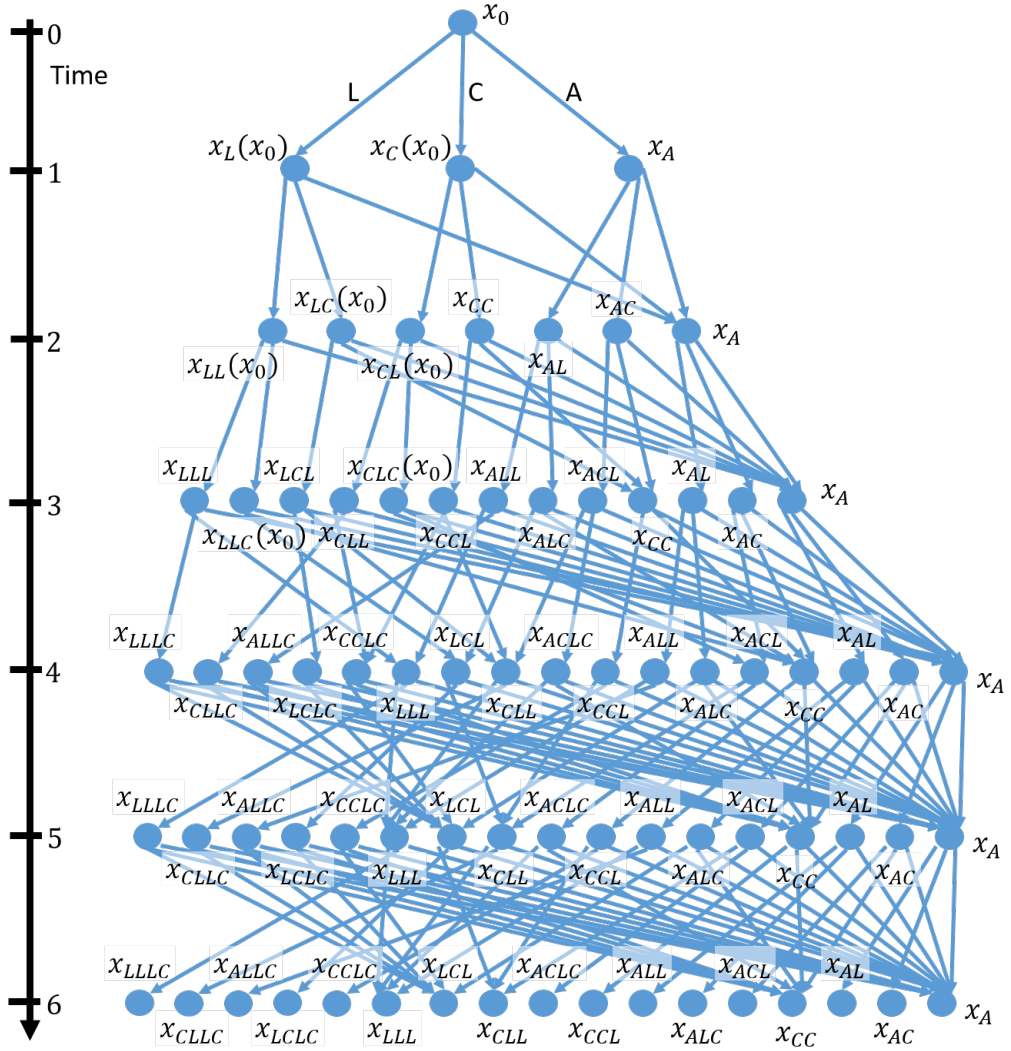


Figure 5.2: A decision tree showing an asymmetric approximation. The right edge coming out of each node corresponds to choosing A, the middle C, and the left L. Note that all the states and edges repeat after $k = 4$.

In Figure 5.2, if there were an A anywhere in a path, one need not look back any farther to know the current state or weight on an edge.

For non-rigid matrices, the natural next step is to consider how rigid a matrix is. Sequences of matrices tend toward being rigid, since $rank(A \otimes B) \leq min(rank(A), rank(B))$. Therefore, using rank as a metric for rigidness, an algorithm for approximating a system asymmetrically can be developed. As mentioned, the STA reduces the number of nodes in the decision tree in a uniform way, allowing a reduction in complexity. The ATA leverages the rank to better determine which nodes are redundant.

Here the term *approximation* is used because it could be the case that the system is fundamentally too complex to solve the sequencing problems given the available resources. In Section 5.2, the system was able to condense its decision tree while still maintaining the exact same dynamics. For instance, in Figure 5.1 if there is only space to store 5 nodes instead of 6, an approximation to the system is needed. The method uses a table to store all possible states in the system that are independent of x_0. Once each entry is independent, the system dynamics are preserved. Otherwise, the set of states is an approximation.

The ATA Algorithm

1. Initialize $\mathbf{P}$ to be the set of matrices n from Problem 1. Initialize $j = 0$ and create 2 tables:
   - A state table, initialized with only the state x_0
   - A weights table, initialized with $w(0, p)$ for each $p \in \mathbf{P}$
2. For each new $p \in \mathbf{P}$:
   - Add the state $x_p = p \otimes x_0$ to the states table.
   - Add the weight $w(p, n_i) = n_i \otimes x_p$ to the weights table for all $n_i \in n$.
3. Remove each rigid p from $\mathbf{P}$

4. Find p_{max}, the element of $\mathbf{P}$ that represents the longest sequence. In case of a tie, choose one with the highest rank.

5. Remove p_{max} from $\mathbf{P}$ and add $n_i \otimes p$ to $\mathbf{P}$ for all $n_i \in n$.

6. If there is space remaining in the tables and $\mathbf{P}$ is not empty, increment j and go to Step 2.

7. If $\mathbf{P}$ is not empty, for every $p \in \mathbf{P}$:

   - Update $x_p = p \otimes \begin{bmatrix} \varepsilon & \cdots & \varepsilon & 0 \end{bmatrix}^T$.
   - Update $w(p, n_i)$ as in Step 2 with the new value of x_p.

5.5 Example

As an example, consider again the system in Section 5.3 and assume that storage capacity is 19 states (including the corresponding 57 edge weights). Initially, $\mathbf{P} = \{A, C, L\}$, the state table contains x_0, and the weights table contains $w(0, A)$, $w(0, C)$, and $w(0, L)$. In Step 2 x_A, x_C, x_L, and their associated weights are stored in the tables. The matrix A is removed from $\mathbf{P}$ because it is rigid. Since both C and L are rank 2, choose $p_{max} = C$ and remove C from $\mathbf{P}$. The matrices AC, CC, and LC are then added to $\mathbf{P}$ and the loop repeats. The set $\mathbf{P}$ after one iteration is $\{L, AC, CC, LC\}$.

The algorithm continues until there are 19 states in the state table and 57 weights in the weight table (see Figure 5.3). At this point, $\mathbf{P} = \{CLC, LLC\}$, which are both still rank 2. Therefore, by Step 7, x_{CLC} and associated weights are updated. Recall that this is simply an approximation to the system from Section 5.3, since there are actually 25 unique states. In Figure 5.2, notice that at time 3 there are only 13 possible states. The ATA described above would assume that these 13 states are the only possible states at each time $k \geq 3$.

Notice that in the above example, ATA stores the same amount of information about the system as STA for $t = 3$, but with less space. STA here would have required storage of 40 states. For STA with $t = 4$, requiring 121 states, ATA can make an equivalent system

Init		$j = 0$		$j = 1$	
$\boldsymbol{P} = \{A, C, L\}$		$\boldsymbol{P} = \{A, C, L\}$		$\boldsymbol{P} = \{L, AC, CC, LC\}$	
States	**Weights**	**States**	**Weights**	**States**	**Weights**
x_0	$w(0, A)$	x_A	$w(A, A)$	x_{AC}	$w(AC, A)$
	$w(0, C)$	x_C	$w(A, C)$	x_{CC}	$w(AC, C)$
	$w(0, L)$	x_L	$w(A, L)$	x_{LC}	$w(AC, L)$
			$w(C, A)$		$w(CC, A)$
			$w(C, C)$		$w(CC, C)$
			$w(C, L)$		$w(CC, L)$
			$w(L, A)$		$w(LC, A)$
			$w(L, C)$		$w(LC, C)$
			$w(L, L)$		$w(LC, L)$

$j = 2$		$j = 3$		$j = 4$	
$\boldsymbol{P} = \{LC, AL, CL, LL\}$		$\boldsymbol{P} = \{LC, LL, ACL, CCL, LCL\}$		$\boldsymbol{P} = \{LC, AL, L\ CLL, LLL\}$	
States	**Weights**	**States**	**Weights**	**States**	**Weights**
x_{AL}	$w(AL, A)$	x_{ACL}	$w(ACL, A)$	x_{ALL}	$w(ALL, A)$
x_{CL}	$w(AL, C)$	x_{CCL}	$w(ACL, C)$	x_{CLL}	$w(ALL, C)$
x_{LL}	$w(AL, L)$	x_{LCL}	$w(ACL, L)$	x_{LLL}	$w(ALL, L)$
	$w(CL, A)$		$w(CCL, A)$		$w(CLL, A)$
	$w(CL, C)$		$w(CCL, C)$		$w(CLL, C)$
	$w(CL, L)$		$w(CCL, L)$		$w(CLL, L)$
	$w(LL, A)$		$w(LCL, A)$		$w(LLL, A)$
	$w(LL, C)$		$w(LCL, C)$		$w(LLL, C)$
	$w(LL, L)$		$w(LCL, L)$		$w(LLL, L)$

$j = 5$		Finish
$\boldsymbol{P} = \{ALC, CLC, LLC\}$		$\boldsymbol{P} = \{CLC, LLC\}$
States	**Weights**	Memory limit reached: update x_{CLC}, x_{LLC} and the associated weights
x_{ALC}	$w(ALC, A)$	
x_{CLC}	$w(ALC, C)$	
x_{LLC}	$w(ALC, L)$	
	$w(CLC, A)$	
	$w(CLC, C)$	
	$w(CLC, L)$	
	$w(LLC\ A)$	
	$w(LLC, C)$	
	$w(LLC, L)$	

Figure 5.3: The ATA algorithm performed on the system from Section 5.3 and assuming that capacity is to store 19 states. The table shows all states and weights that are added to the table at each iteration of the loop.

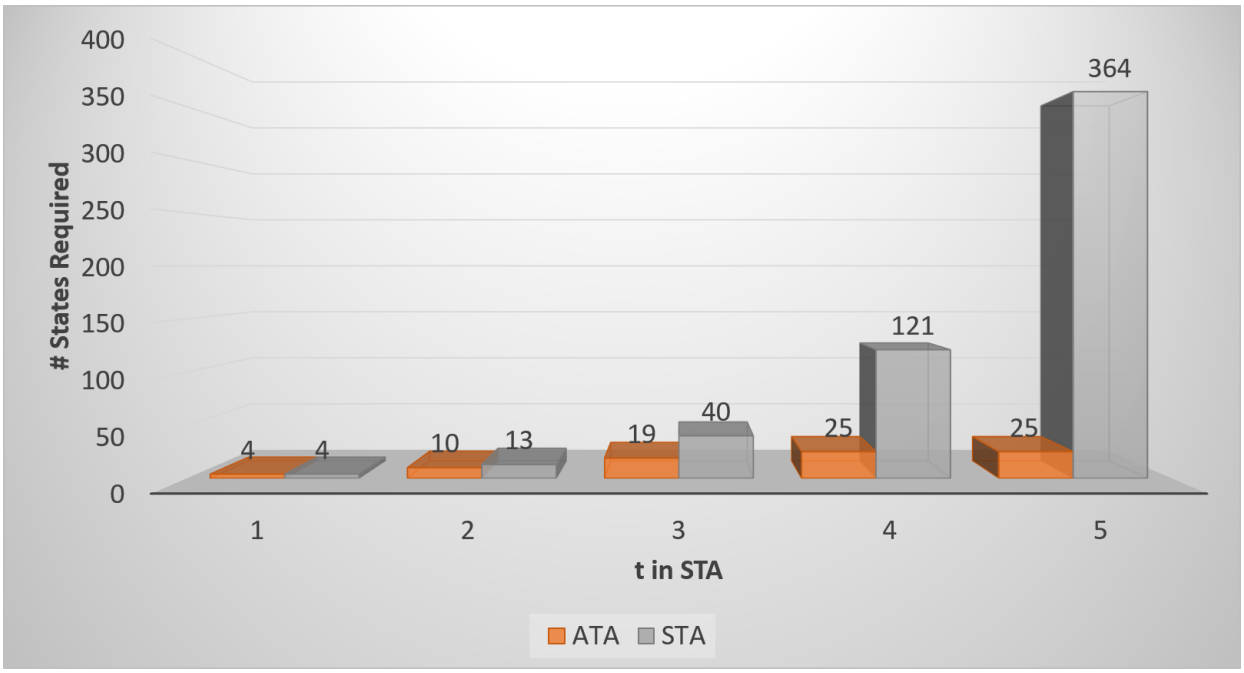


Figure 5.4: A comparison of the STA and ATA for the system in Section 5.3. The x-axis represents the value of t for the STA, and the chart shows how many states need to be stored for each value. This is compared to how many states need to be stored for the ATA to yield an equivalent approximation.

approximation in this example with only 25 states (see Figure 5.4). This illustrates the fact that ATA does no worse than STA for any system, given the same resources. This is shown more formally in Chapter 6.

Recall that Problem 1 is

$$s^* = arg\min_{s \in S} H(s, x_0) \tag{5.5}$$

Since the ATA provides an approximation to the system, the metric $H(s, x_0)$ is replaced by $\hat{H}(s, x_0)$, which approximates the weight of each branch in the decision tree as described above. Using this method a new problem can be formulated:

Problem 2. *Using the same symbolic definitions as in Problem 1 and $\hat{H}$ as described above, find*

$$\hat{s}^* = arg\min_{s \in S} \hat{H}(s, x_0) \tag{5.6}$$

Problem 2 can either be solved or approximated using an approximate dynamic programming approach or a shortest path tool, some of which are described in Chapter 2. Regardless of the chosen solver, Problem 2 is be less complex than Problem 1. The chosen solver will need to take into account which paths are invalid based on the quota, and manage which subset of states is possible at time k.

Chapter 6

Error Bounds

This chapter describes the error bounds of the ATA. The claim of this work is that the ATA is a closer system approximation than the STA and that the error bounds of the ATA are the same or tighter than those of the STA for given computational constraints. As mentioned, error bounds for the STA are given in [33]. Since the ATA is similar in nature, we leverage some of these results and show why the ATA is an improvement.

First begin with a few definitions:

- Let $x_{min} = \begin{bmatrix} \varepsilon & \cdots & \varepsilon & 0 \end{bmatrix}^T$
- Let $\hat{t}_{STA}$ be the value of t used in the STA
- Let $\hat{t}_{ATA}$ be the maximum sequence length found in the ATA state table
- Let t_{max} be the maximum sequence length in S that is not rigid

Lemma 15. [33] *Suppose $A, B \in \mathcal{M}^n$, and s_1 is the sequence (A, B) and s_2 is the sequence (B). Then*

$$x_{min} = arg \min_{x \neq \varepsilon} (H(s_1, x) - H(s_2, x)) \tag{6.1}$$

Since B could be the combination of any sequence of matrices, Lemma 15 essentially states that starting with the vector x_{min} guarantees the minimum change in H for any matrix A.

Lemma 16. *Let s be a sequence $(n_1, ..., n_N)$, and define:*

- $\boldsymbol{W}(s,i) = H((n_1,...,n_i),x_0) - H((n_1,...,n_{i-1}),x_0)$ *is the cost of choosing* n_i
- $\hat{\boldsymbol{W}}^t(s,i) = H((n_{i-t-1},...,n_i),x_{min}) - H((n_{i-t-1},...,n_i),x_{min})$
- $\hat{\boldsymbol{W}}^t(s,i) = \boldsymbol{W}(s,i)$ *when* $i \leq t+1$

Then $\boldsymbol{W}(s,i) \geq \hat{\boldsymbol{W}}^{t+1}(s,i) \geq \hat{\boldsymbol{W}}^t(s,i)$.

Proof.

$$\begin{aligned}
\hat{\mathbf{W}}^t(s,i) &= H((n_{i-t-1},...,n_i),x_{min}) - H((n_{i-t-1},...,n_{i-1}),x_{min}) \\
&\leq H((n_{i-t-1},...,n_i),x) - H((n_{i-t-1},...,n_{i-1}),x)\ \forall x, \text{ by Lemma 15} \\
&\text{Let } x = n_{i-t-2} \otimes x_{min} \\
&= H((n_{i-t-2},...,n_i),x_{min}) - H((n_{i-t-2},...,n_{i-1}),x_{min}) \\
&= \hat{\mathbf{W}}^{t+1}(s,i)
\end{aligned}$$

The left side of the inequality is true by extension, since $N \geq t+1$. □

Theorem 17. *Let* $\hat{H}_{STA}$, $\hat{H}_{ATA}$ *be the STA, ATA respectively for* H *in Problem 1. Then*

$$H(s,x_0) \geq \hat{H}_{ATA}(s,x_0) \geq \hat{H}_{STA}(s,x_0)$$

Proof. Note that from the discussion in Chapter 5, $t_{max} \geq \hat{t}_{ATA} \geq \hat{t}_{STA}$. By Lemma 16, $\mathbf{W}(s,i) \geq \hat{\mathbf{W}}^{t_{ATA}}(s,i) \geq \hat{\mathbf{W}}^{t_{STA}}(s,i)$. Since $H(s,x_0) = \sum_i \mathbf{W}(s,i)$, $\hat{H}_{ATA}(s,x_0) = \sum_i \hat{\mathbf{W}}^{t_{ATA}}(s,i)$, and $\hat{H}_{ATA}(s,x_0) = \sum_i \hat{\mathbf{W}}^{t_{ATA}}(s,i)$ the theorem statement holds. □

The above theorem states that the ATA yields an $\hat{H}$ that is the same distance or closer to the value of H than that of STA for all admissible sequences. Therefore, the ATA is considered a closer approximation to the real system.

The next results shows that the error bounds to Problem 2 are smaller when using the ATA versus the STA. Here error is defined $e = H(\hat{s}^*,x_0) - H(s^*,x_0)$. In order to show this result, the following are also defined:

- Let $s(t,k)$ be the subsequence of s of t items starting at time k
- Let $\gamma^t(s) = \max_x\{(H(s(t,0),x) - H(s(t-1,0),x) - (H(s(t,0),x_{min}) - H(s(t-1,0),x_{min}))\}$
- Let $\Gamma^t = \max_{s\in S(t)} \gamma^t(s)$, where $S(t)$ is the set of all admissible sequences of length t.

Intuitively, $\gamma^t(s)$ shows the worst-case error for a given sequence of length t and Γ^t shows the worst-case error for all sequences of length t. If $t < \hat{t}_{max}$, then $\gamma^t(s) = 0$ for all s, and thus $\Gamma^t = 0$. This corresponds to the case when the ATA perfectly approximates the system. Finding Γ^t is itself a combinatorial optimization problem, but one that is no more complex than Problem 2.

Lemma 18. [33] *For the STA,*

1. $e_{STA} \leq \Gamma^{\hat{t}_{STA}}[||q||_1 - (\hat{t}_{STA}+1)] = e^{max}_{STA}$
2. $\Gamma^t \geq \Gamma^{t+1}$ *for any* t

Essentially, Lemma 18 states that a worst-case error is $\Gamma^{\hat{t}_{STA}}$ repeated for each $k > \hat{t}_{STA}$. To analyze the ATA, a similar statement can be made, with a few subtle differences.

Theorem 19. *Using the ATA as a system approximation yields the following:*

1. $e_{ATA} \leq \Gamma^{\hat{t}_{ATA}}(||q||_1 - \hat{t}_{ATA}) = e^{max}_{ATA}$
2. $e^{max}_{ATA} \leq e^{max}_{STA}$

Proof. The key to this theorem is to again note that for given computational constraints, $\hat{t}_{STA} \leq \hat{t}_{ATA}$. In other words, with ATA, one is able to approximate more states and edge weights, a point clearly shown in the previous chapter. Also note that unlike STA, $\hat{t}_{ATA}$ does not ensure that *all* sequences of length $\hat{t}_{ATA}$ are correctly estimated, just at least one. Thus for a worst-case analysis, $\Gamma^{\hat{t}_{ATA}}$ is repeated for every $k > \hat{t}_{ATA} - 1$. Statement 1 therefore follows directly from Lemma 18.

In order to prove Statement 2, consider 2 cases:

1. $\hat{t}_{STA} < \hat{t}_{ATA}$ In this case, by Lemma 18 Statement 2 it follows that $\Gamma^{t_{STA}} \geq \Gamma^{t_{max}}$, and therefore $e^{max}_{ATA} \leq e^{max}_{STA}$.

2. $\hat{t}_{STA} = \hat{t}_{ATA}$. In this case, for the ATA, since there are equivalent computational constraints, all sequences of length $\hat{t}_{ATA}$ have an entry in the state table. Therefore, we have a tighter bound: $e^{max}_{ATA} = \Gamma^{\hat{t}_{ATA}}[||q||_1 - (\hat{t}_{ATA} + 1)]$. Using this tighter bound we see that $e^{max}_{ATA} = e^{max}_{STA}$.

□

The above theorem shows tighter error bounds for the ATA, but note that it could be the case that solving Problem 2 using the STA could yield a better sequence in some cases. This point is illustrated in the example below.

6.1 Example

In order to understand the meaning of the above error bounds and to further illustrate the difference between STA and ATA, this section presents an example system. Consider four 10×10 matrices: O, Q, R, W and quota $q = \begin{bmatrix} q_O & q_Q & q_R & q_W \end{bmatrix}^T$, where $q_O = 2$, $q_Q = 4$, $q_R = 4$, and $q_W = 2$. The initial condition is $x_0 = \begin{bmatrix} 0 & \cdots & 0 & 11 \end{bmatrix}^T$ In this small system there are $207,900$ different possible sequences, with $377 \geq H(s, x_0) \geq 329$ for any admissible sequence s. There are 14 sequences that all yield the minimum score. Figure 6.1 shows each s, ordered in descending order by $H(s, x_0)$.

We first approximate this system using the STA, with $1 \leq t \leq 5$. Then, for each value of t, the $arg\max_s \hat{H}_{STA}(s)$ is found and shown in Figure 6.1. For instance, for $t = 1$, the best sequence found in the STA is $WQRQRQRQRWOO$, where $H(s, x_0) = 358$. The STA for $t = 5$, which requires 1123 states, is equivalent to the original system, therefore a search of the STA yields $QWOQQWOQRRRR$, which has the minimal score.

In comparison, the ATA approximated the system more closely with a fewer number of states. For instance, the minimum sequence found on the STA with $t = 3$ has a score of

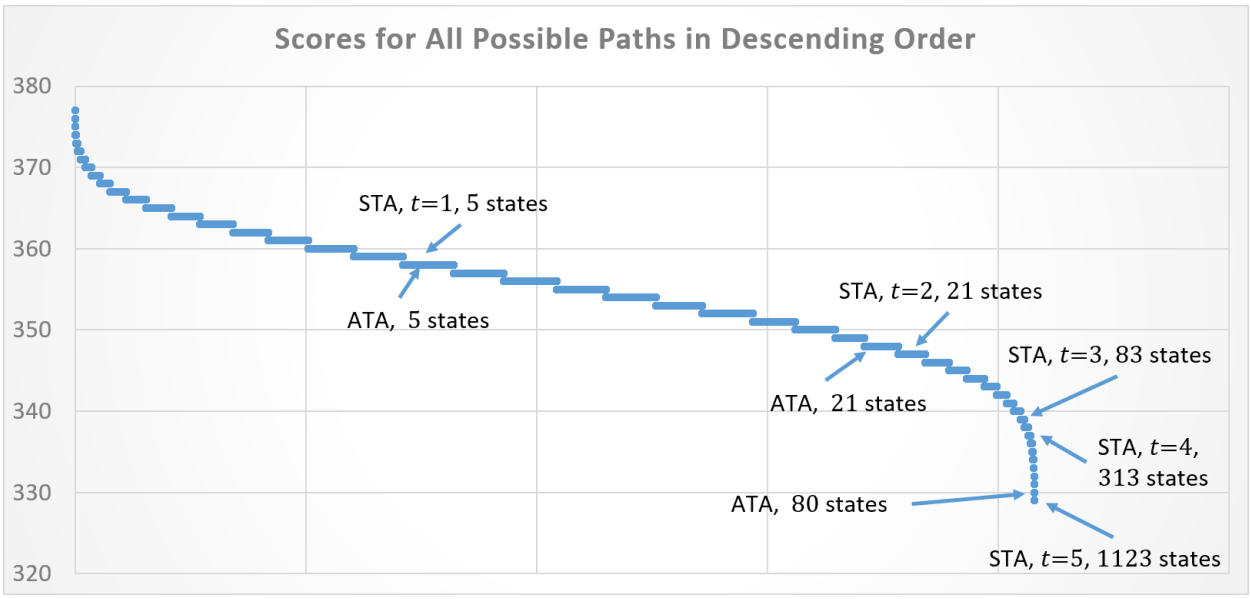


Figure 6.1: The example system

339. The minimum sequence found on the ATA with an equivalent number of states (80) is a minimum sequence for the original system. In fact, for this example, the ATA equivalent to the original system only requires 150 states. Notice that STA, $t = 1$ is the same as an ATA with 5 states, since both only include x_0 and a state for each matrix. Therefore, both approximations yield the same sequence.

This example also illustrates the subtle point at the end of the previous section: sometimes a worse system approximation can yield a better result. Consider in Figure 6.1 the STA, $t = 2$, and the ATA with 21 states. The ATA more closely approximates the system, since the state table contains weights for sequences of length 3, whereas the STA is only considering sequences of length 2. However, since both are approximations, it turns out that the the minimum sequence found by traversing the STA scores better on the actual system than that found by traversing the ATA. This statement does not conflict with Theorem 19, since only the error bounds are guaranteed. The metric $\hat{H}(s, x_0)$ found in the ATA is closer to $H(s, x_0)$ than the metric in the STA for all s. The anomaly arises when s fares better comparatively against other sequences in the STA than s does in the ATA, and s also happens to have the minimum metric score.

Chapter 7

Future Work

Future work could either generalize the ATA to other types of problems or drill down into the specifics of Problem 1. In a generalization of the ATA, the concept of being *rigid* would need to be explored in other application realms. As explained, many dynamic programming approximations exist to overcome the curse of dimensionality, including those that create equivalence classes among the states. The ATA could be explored and defined outside of max-plus and in relation to these methods.

Drilling down into the specifics of Problem 1, open questions can be addressed. For example, the ATA algorithm uses the rank operator to determine which sequence to expand next. However, there could be a better monotonic metric available that determines how rigid a sequence is. Additionally, if a matrix A is non-rigid, this work assumes that the output x_k could be any vector in the range of A. It could be the case, though, that the set of possible initial conditions for A will only yield values for x_k that is a subset of the range of A. Theoretically, this could mean that a matrix could be rigid for a given initial condition, quota, and other matrices of the system.

This work has not considered the specifics of algorithms or tools that will compute a solution to Problem 2. In showcasing the example from Section 6.1, an exhaustive search was used in order to focus on the ATA versus the STA. In practice, it will be important to use an efficient tool or algorithm to conduct such a search, especially for large problems. An open question is whether there is a tool or algorithm that searches the ATA better than others. Perhaps a custom method could be created that also tracks the quota.

In summary, this work has leveraged max-plus algebra in order to analyze certain properties of batch flow shops and develop a system approximation for closer solutions to Problem 1. Specifically, the set $\mathcal{M}^n$ has been divided into three subsets: rigid, non-rigid with the MLP, and non-rigid without the MLP. Rigid matrices are those that have rank 1 - a test which can be performed quickly. Matrices that have the MLP are those which converge to a rigid matrix. Those that do not have the MLP never converge to a rigid matrix, thus a homogeneous sequence of such a matrix will never forget its initial condition. Systems of heterogeneous sequences of matrices were also explored, showing that most systems exhibit an asymmetric behavior - one that is not addressed by the STA. Thus, a competing approximation method, the ATA, has been presented. The ATA has been shown to be an equivalent or better approximation to the STA in all cases.

References

[1] Kenneth R Baker and Dan Trietsch. *Principles of sequencing and scheduling*. John Wiley & Sons, 2013.

[2] Richard Bellman. The theory of dynamic programming. Technical report, DTIC Document, 1954.

[3] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[4] M. Boccadoro and P. Valigi. A modelling approach for the dynamic scheduling problem of manufacturing systems with non negligible setup times and finite buffers. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 5, pages 5472–5477 Vol.5, Dec 2003. doi: 10.1109/CDC.2003.1272508.

[5] P. Butkovi and R.A. Cuninghame-Green. On matrix powers in max-algebra. *Linear Algebra and its Applications*, 421(23):370 – 381, 2007. ISSN 0024-3795. doi: http://dx.doi.org/10.1016/j.laa.2006.09.027. URL `http://www.sciencedirect.com/science/article/pii/S0024379506004514`. Special Issue in honor of Miroslav Fiedler.

[6] David G Dannenbring. An evaluation of flow shop sequencing heuristics. *Management science*, 23(11):1174–1182, 1977.

[7] Xiaotie Deng, Haodi Feng, Guojun Li, and Guizhen Liu. A ptas for minimizing total completion time of bounded batch scheduling. *International Journal of Foundations of Computer Science*, 13(06):817–827, 2002.

[8] YM Dessouky, CA Roberts, MM Dessouky, and G Wilsons. Scheduling multi-purpose batch plants with junction constraints. *International journal of production research*, 34 (2):525–541, 1996.

[9] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[10] Lionel Dupont and Clarisse Dhaenens-Flipo. Minimizing the makespan on a batch machine with non-identical job sizes: an exact procedure. *Computers & Operations Research*, 29(7):807–819, 2002.

[11] Mikhail Simin-Arjang Fahim and Marco Voltorta. Efficient memory-bounded search methods.

[12] Michael R Garey, David S Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129, 1976.

[13] S. Gaubert and J. Mairesse. Modeling and analysis of timed petri nets using heaps of pieces. *Automatic Control, IEEE Transactions on*, 44(4):683–697, Apr 1999. ISSN 0018-9286. doi: 10.1109/9.754807.

[14] Stéphane Gaubert and Dohy Hong. Series Expansions of Lyapunov Exponents and Forgetful Monoids. Technical Report RR-3971, INRIA, July 2000. URL `http://hal.inria.fr/inria-00072677`. Projet MCR.

[15] Celia A Glass, Chris N Potts, and Vitaly A Strusevich. Scheduling batches with sequential job processing for two-machine flow and open shops. *INFORMS Journal on Computing*, 13(2):120–137, 2001.

[16] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[17] Bernd Heidergott and Jacob W. van der Woude. *Max Plus at work: modeling and analysis of synchronized systems: a course on Max-Plus algebra and its applications*, volume 13. Princeton University Press, 2006.

[18] James R Jackson. An extension of johnson's results on job idt scheduling. *Naval Research Logistics Quarterly*, 3(3):201–203, 1956.

[19] Selmer Martin Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1):61–68, 1954.

[20] E Kondili, CC Pantelides, and RWH Sargent. A general algorithm for short-term scheduling of batch operationsi. milp formulation. *Computers & Chemical Engineering*, 17(2):211–227, 1993.

[21] Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[22] Eugene L Lawler, Jan Karel Lenstra, Alexander HG Rinnooy Kan, and David B Shmoys. Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, 4:445–522, 1993.

[23] Jean Mairesse and Laurent Vuillon. Asymptotic behavior in a heap model with two pieces. *Theoretical Computer Science*, 270(1):525–560, 2002.

[24] Glenn Merlet. Memory loss property for products of random matrices in the max-plus algebra. *Math. Oper. Res.*, 35(1):160–172, February 2010. ISSN 0364-765X. doi: 10.1287/moor.1090.0434. URL `http://dx.doi.org/10.1287/moor.1090.0434`.

[25] D.S. Palmer. Sequencing jobs through a multi-stage process in the minimum total time–a quick method of obtaining a near optimum. *OR*, pages 101–107, 1965.

[26] R Gary Parker. *Deterministic scheduling theory*. CRC Press, 1996.

[27] Michael Pinedo. *Planning and scheduling in manufacturing and services*, volume 24. Springer, 2005.

[28] Warren B. Powell. *Approximate Dynamic Programming*. Wiley and Sons, 2011.

[29] Marcos Singer and Michael Pinedo. A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE transactions*, 30 (2):109–118, 1998.

[30] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. MIT Press, 1998.

[31] J.AW. van Eekelen, E. Lefeber, and J.E. Rooda. Feedback control of 2-product server with setups and bounded buffers. In *American Control Conference, 2006*, pages 6 pp.–, June 2006. doi: 10.1109/ACC.2006.1655413.

[32] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.

[33] W. Weyerman, A. Rai, and S. Warnick. Model approximation for batch flow shop scheduling with fixed batch sizes. *Discrete Event Dynamic Systems*, April 2014.