Scheduling Shutdowns for Manufacturing Systems with an Application to Automotive Production Lines: Optimization Models and Computation

by

Blake E. Nicholson

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Industrial and Operations Engineering)
in The University of Michigan
2008

Doctoral Committee:

Professor Robert L. Smith, Co-Chair
Associate Professor Marina A. Epelman, Co-Chair
Professor S. Jack Hu
Daniel J. Reaume, General Motors Corporation

ACKNOWLEDGEMENTS

I would like to thank my advisors, Robert Smith and Marina Epelman, for their guidance and assistance throughout my time in graduate school. I appreciate Professor Smith's emphasis on intuition and understanding the bigger picture, while Professor Epelman's detail orientation ensured that my analyses were valid and correct.

I am grateful to Jack Hu and Dan Reaume for agreeing to sit on my dissertation committee. I know Professor Hu has a busy schedule and I valued his input as one of my committee members. Dan not only served on my committee, but was my mentor at General Motors as well. His advice and critiques proved invaluable, particularly with the development of the computational experiments. Thank you to both Jack and Dan for all your work.

I also want to thank the IOE department's faculty, staff and students. I learned a lot during my time at Michigan both in and out of the classroom. I enjoyed my time in Ann Arbor and am sad to leave.

Thank you to General Motors and the National Science Foundation for the funding that supported this research. This work was partially supported by the National Science Foundation under Grants DMI-0422752 and DMI-0114368, and also by the General Motors Collaborative Research Laboratory in Advanced Vehicle Manufacturing at the University of Michigan.

I thank my in-laws for their help during busy times. They took care of our dogs

on many occasions and helped us out in whatever way they could.

I appreciate my family for their love and support through the years. My Dad and Mom instilled values and a work ethic that have served me well in life. I also thank my brother and sister for keeping in touch, even as we are spread throughout the country. I always enjoyed taking a break from my work to catch up with family and friends.

Last, but certainly not least, I want to thank my wife, Seraphin. Graduate school has been a challenging time, but we made it through. I obviously learned a lot academically from my graduate school experience, but being married has taught me more about myself and interpersonal relationships than I ever imagined it would.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF APPENDICES

Appendix

CHAPTER I

Introduction

Manufacturing systems are complicated networks of many potentially unreliable parts. The analytical study of such complex systems can prove difficult, if not impossible, and **computational models** are frequently used to analyze such systems. Through construction of both analytical and computational models, we analyze and optimize the shutdown process of an automotive production line.

To assist the reader that may not be familiar with an automotive manufacturing environment, we first provide an overview of an automotive production line. Then we introduce the End-State problem, the focus of this dissertation. This chapter concludes with a brief summary of each of the remaining chapters and the specific contribution of each.

1.1 Description of an Automotive Assembly Plant

We now describe an automotive assembly plant to provide the reader with context to the problem we address. An automotive assembly plant is comprised of three main areas: body shop, paint shop, and general assembly. We describe each of these three areas in turn. This discussion is based upon a truck assembly plant due to the author's familiarity with the inner workings of such a facility. However, the concepts behind this discussion are generally applicable across all automotive manufacturing

plants. Throughout this discussion, a **job** refers to a vehicle that is being built.

1.1.1 Body Shop

The body shop, highlighted in Figure 1.1, is the area in which the skeleton of the car is assembled. Pieces of metal of all shapes and sizes are attached via various means: welds, glue, rivets, etc. This area is highly automated, with armies of robots handling the difficult and dangerous tasks required.

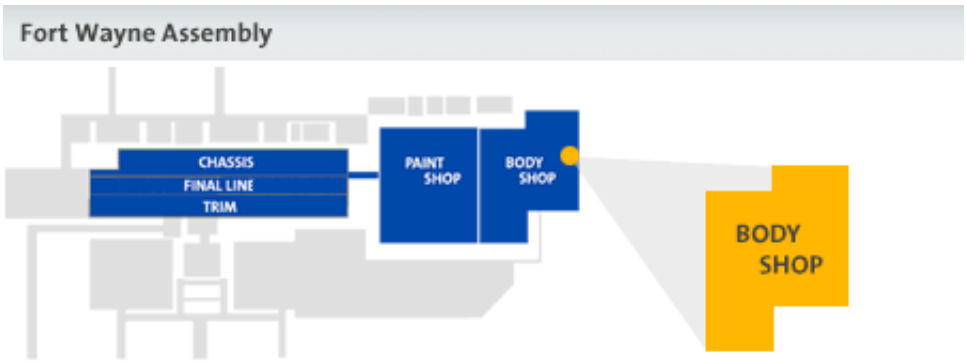

Figure 1.1: Body Shop (Courtesy: General Motors Corporation, 2008a)

The general structure of the body shop is a serial main line with sub-assemblies that are fed in at key points. In a truck plant, the cab and bed are assembled separately and are not attached to one another until late in the production process. Figure 1.2 shows a partially assembled truck cab.

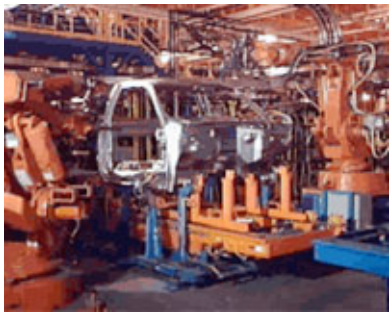

Figure 1.2: Assembly of truck cab (Courtesy: General Motors Corporation, 2008b)

Once the truck cab, bed, doors, and front-end sheet metal (FESM) are assembled, the job proceeds to the paint shop to be painted.

1.1.2 Paint Shop

Far more occurs in the paint shop than simply applying a coat of paint to some metal. In fact, the paint shop consists of four main steps that last of which is the application of the finished coat.

The process starts with cleaning. The entire collection of metal parts for a job are first dipped into a phosphate bath. This cleans and chemically etches the metal. Next the parts are put through an electrodeposition (ELPO) bath filled with electrically charged primer paint to apply the first coat. Finally, the job is baked in an oven to complete the first coat.

The following two steps are the application of sealant and a second primer coat. Robots apply sealant to the seams where two metal pieces join together in order to keep water out. The application of the second primer coat, shown in Figure 1.3, is carried out by robots in a paint booth. The job again goes through an oven after this second primer coat.

Figure 1.3: Primer coat application (Courtesy: General Motors Corporation, 2008c)

The final phase in the paint shop is the application of paint of the appropriate color. This coat, known as the top coat, is again sprayed on by robots in a paint booth. A final coat, known as the clear coat, is applied to give the vehicle a nice, shiny appearance. The job is sent through an oven one final time to cure the paint job.

Jobs are resequenced in the paint shop for a variety of reasons. Throughout the paint shop, there are inspection stations at which the quality of work is inspected and any jobs not up to par are cycled back through the process. Some jobs need additional coats of paint based upon the color they are to receive. Jobs may also be resequenced to batch together jobs to be painted the same color. Such changes to the ordering of jobs creates challenges for the algorithms to be introduced, and the focus of the work described herein is in the body shop and general assembly areas.

1.1.3 General Assembly

The last area of the assembly process is general assembly, which is further broken down into chassis, trim, and final line. Coming out of paint, jobs first go through the trim area, where a variety of subsystems are installed, including seats, the instrument panel, and so on. The parts of a job (cab, bed, doors, and front-end sheet metal) are separated and sent on separate paths for trim processing. Later, these various components merge back together before continuing along the line. This type of split/merge motivates the work in Chapter III.

In chassis, shown in Figure 1.4, the underside of the truck is assembled. This includes the installation of the axles, fuel lines, gas tank, tires, and engine.

Figure 1.4: Chassis Assembly (Courtesy: General Motors Corporation, 2008d)

The chassis and body come together at the chassis-body marriage point and pro-

ceed along final line. During this last phase of the assembly process, the finishing touches are put on the truck. The various fluid reservoirs are filled, emblems are attached, and a final inspection of the vehicle is carried out. Once vehicles have successfully passed all inspections, they are driven out of the plant and into a storage lot outside.

Having given the reader a brief background on the automotive manufacturing process, we now move on to describe the End-State problem.

1.2 End-State Problem

Maximizing equipment utilization is essential to the profitability of capital-intensive production processes. Although much research addresses how to optimally schedule planned system downtime and execute tasks during the downtime, little has been written about how to most effectively coordinate production leading up to the scheduled downtime to enable task completion. This is the focus of the End-State problem.

The End-State problem seeks to find a shutdown policy that optimizes a trade-off among various potentially conflicting goals to achieve as well as the shutdown time of the line. The goals to achieve are driven by tasks that need to be accomplished during the planned downtime as well as by the desire to have a successful start-up of the line when it is brought back on-line. The shutdown time of the line is important in that running beyond the end of the shift incurs overtime costs, while stopping early leads to lost production. Chapter II provides greater detail about how the goal values and time penalties are calculated as well as a comparison of the advantages and disadvantages of a time-based versus a job-based shutdown policy. A time-based shutdown policy shuts down each line element at a particular time, while a job-based

shutdown policy shuts down line elements based upon the flow of jobs through the line.

Planned downtime is useful for a variety of critical tasks, including preventive maintenance, calibrations, installations, and upgrades, that can be performed only when a station is down. What makes scheduling such tasks challenging is that the state of the production system when it shuts down may constrain their performance. For example, in a production line consisting of stations separated by buffers, consider the task of upgrading a particular station. Safety or accessibility needs might dictate that this station be empty of jobs when the upgrade is performed. Moreover, validating the upgrade requires a supply of jobs of appropriate types immediately upstream of the station, together with sufficient empty space downstream to accept these jobs after they are processed. The challenge of achieving as many such requirements (called end-state goals in the rest of the dissertation) as possible while trading off potential lost production or overtime costs is often an exceedingly difficult problem.

Leaving the line in a state that facilitates the initiation of work to be conducted during the planned downtime is critical. Not doing so leads to fewer tasks being accomplished and has a negative effect on the long-term production output of the line. Missing production line maintenance schedules has the potential to have an adverse impact on the quality of product produced, while missing new product launch testing can delay the critical timing of a new product introduction.

We next put the End-State problem within the broader context of maintenance optimization and downtime planning.

1.3 Literature Review

As described above, the End-State problem considers how to operate a production line in anticipation of completing a set of tasks during scheduled downtime. Two related areas of research are maintenance optimization and downtime planning.

Many articles have been written under the umbrella of maintenance optimization, an area of research primarily concerned with developing a maintenance schedule that balances the costs and benefits associated with performing maintenance (see McCall, 1965; Pierskalla and Voelker, 1976; Sherif and Smith, 1981; Valdez-Flores and Feldman, 1989; Cho and Parlar, 1991, and references therein). The criteria that may be used when developing a maintenance schedule include minimizing the total downtime or minimizing the operating cost per unit time. In addition, there also exists a large body of research about optimizing the use of resources during a period of planned downtime using methodologies such as the Critical Path Method (CPM) and Material Requirements Planning (MRP) (Duffuaa et al., 1999; Samaranayake et al., 2002).

To our knowledge, the problem of how to optimally control a production line in the time leading up to a scheduled downtime has been largely unaddressed in the literature. Note that although we specifically discuss an automotive assembly application, this methodology is applicable to a variety of systems involving work-in-process (WIP) inventory. Examples include oil refineries, chemical processing plants, semiconductor manufacturing, transactional back-office operations, and new product development and introduction. Cheung et al. (2004) describe one such example faced by chemical production facilities. Nelson (2007) and Kimber et al. (2006) highlight the importance of downtime planning for a clinical information system and

telecommunications networks, respectively.

1.4 Contribution

Chapter II starts with a simple serial line and a dynamic programming approach to the End-State problem. The key to this approach is to make use of the capacity and ordering constraints of the line to dramatically simplify the feasible decisions of the dynamic program. In addition to developing an efficient dynamic programming formulation of the End-State problem, this chapter also applies the mathematical model to data from an actual production line and presents sample results from simulation experiments showing that the shutdown policy recommended by the algorithm is superior to one obtained by a rule of thumb approach. While the dynamic programming algorithm works well for pure serial lines, it cannot be applied to splits and merges seen in general assembly.

To handle splits and merges, we introduce a hierarchical decomposition of the production line in Chapter III. This decomposition simplifies a line with splits and merges into a series of dependent serial lines that use a similar approach to that of Chapter II. Another benefit of this approach is that a broader range of goals can be handled. Due to a change in the way time penalties are assessed, the time required to execute the algorithm when applied to a serial line drops dramatically.

The previous two chapters did not consider stochasticity when developing a shutdown policy. In Chapter IV, we return to the analysis of a serial line, but now incorporate stochasticity into the development of a shutdown policy. While an optimal solution proves to be intractable, we evaluate two different heuristic techniques for incorporating stochasticity into a shutdown policy.

CHAPTER II

A Dynamic Programming Approach to Achieving an Optimal End-State Along a Serial Production Line

2.1 Introduction

This chapter focuses on the End-State problem for a serial line topology. This topology arises frequently in a manufacturing environment, particularly in the body shop of an automotive manufacturing plant. Both a mathematical model and a dynamic programming formulation are developed for the serial line case.

Among its contributions, this chapter:

- Develops an efficient dynamic programming formulation of the problem that leverages the constraints imposed by the ordering and capacities of the line elements to limit the size of the space of feasible solutions, which enables the use of the algorithm in real time.
- Applies the mathematical model to data from an actual production line at one of General Motors' plants, demonstrating that the model and the algorithm have real world utility.
- Presents sample results from simulation experiments showing that the shutdown policy recommended by the algorithm is superior to one obtained by a rule of thumb approach.

This chapter also charts new territory in looking at how to operate a production line in anticipation of completing a set of tasks during scheduled downtime.

This chapter is organized as follows. Section 2.2 introduces an abstract production line model, and formally states the problem of finding an optimal shutdown policy when considering both end-state and production goals. Section 2.3 presents an efficient dynamic programming formulation for finding an optimal shutdown policy. Section 2.4 summarizes the computational experiments. Finally, Section 2.5 summarizes lessons learned from this case study.

2.2 A Model of the End-State Planning Problem

In this section we describe a network representation of a serial production line. We then formulate the optimization problem of balancing the value of satisfying end-state goals against the costs of overtime and lost production time.

2.2.1 A Network Represention of a Production Line

A typical production line consists of two types of line elements, stations and buffers, connected together. Stations perform manufacturing tasks (welding, hemming, etc.) and can store WIP, while buffers just store WIP.

By modeling stations and buffers as nodes, and their connecting conveyors as arcs, we can describe a general class of production systems as directed graphs. To simplify the problem, we focus on the most common topology, a serial line, for the rest of this chapter. Figure 2.1 illustrates a serial line topology. Note that shutdown decisions are only made at the nodes in this network.

Each job entering the production line belongs to one of several types characterized by one or more distinguishing characteristics. For example, jobs processed by a truck body assembly line of an auto manufacturer can be distinguished by having

an extended cab and/or a sun roof, thus resulting in four job types that may require different processing at some of the stations. An ordered list of jobs (and their types) to be processed on the line is typically specified ahead of time (it is referred to as the **build schedule**), and is known to the managers controlling the production line.

We label the line elements sequentially in ascending order from the tail of the line to the head of the line. Jobs, numbered sequentially from 1 to J, therefore enter the line at line element N, proceed through the line in sequence, and exit at line element 1. The rationale for this numbering will become clear later on.

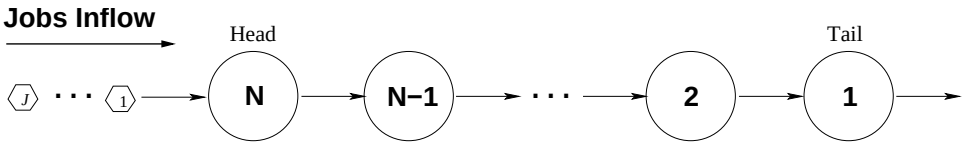

Figure 2.1: A serial production line. Jobs enter at line element N, and exit at line element 1.

An end-state goal is, broadly speaking, a description of the desired contents of a line element (for example, the desired quantity of jobs) when the production line shuts down, which would enable performance of a particular task during downtime. In general, satisfying all end-state goals specified for all line elements may not be feasible or desirable because: 1) multiple goals specified for an individual station or buffer may conflict with one another; 2) the build schedule causes conflicts between stations and/or buffers; 3) the line would have to be run beyond the desired shutdown time, requiring excessive overtime; 4) the line would have to be shut down prohibitively early, leading to lost production. These potential conflicts are what makes the end-state planning problem challenging.

We now formally introduce notation for the end-state planning problem:

- Let $\mathbf{N} = \{1, \ldots, N\}$ denote the index set of the line elements. As already noted, line elements are labeled sequentially in ascending order going from the tail of

the line to the head of the line.

- Let $\mathbf{J} = \{1, \ldots, J\}$ denote the index set of the jobs flowing into the production line. Jobs are labeled in ascending order starting with the first job to enter the line, and for each job, the build schedule specifies its type.
- Let m_n be the capacity, measured in jobs, of line element n, $n \in \mathbf{N}$.
- Let $r_n = (i_n^1, \ldots, i_n^m),\ m \leq m_n,\ n \in \mathbf{N},\ i_n^1, \ldots,\ i_n^m \in \mathbf{J}$, denote the tuple of WIP, described as an ordered list of jobs contained in line element n at the time of line shutdown. Since the jobs traverse the production line in order, $i_n^m = i_n^1 + (m-1)$. If $m = 0$, the line element is empty.
- Let R_n be the set of all tuples of WIP, any of which would satisfy the end-state goal for line element n.
- Let v_n be the value awarded if, at shutdown, $r_n \in R_n$, and let p_n be the penalty assessed if, at shutdown, $r_n \notin R_n$.
- Let T_d be the desired line shutdown time, e.g., the end of the normal shift. An overtime or lost production time penalty is assessed if the shutdown policy induces a shutdown time other than T_d; in particular:
  - Let p_o denote overtime cost per unit time.
  - Let p_l denote lost production time penalty per unit time.
- Let $T_{\max}$ represent a "hard" upper bound on the time by which all line elements must be shut down, e.g., the start time of the next shift.

2.2.2 The Formal Definition of the End-State Planning Problem

The goal of the end-state planning problem, as described earlier, is to optimize the trade-off between meeting end-state goals versus meeting production targets. In

particular, we would like to maximize the net value of meeting end-state goals minus the penalty due to overtime or lost production time.

The shutdown schedule, or shutdown policy, can be specified either in terms of an absolute shutdown time of each line element, or in terms of locations of jobs in the production line at the time of shutdown. For our purposes, we will specify the shutdown policy in the latter form by specifying, for each $n \in \mathbf{N}$, the index of the last job $j_n \in \mathbf{J}$ to exit this line element. (To handle line element N, which presents a special case for this definition, we introduce a dummy line element $N+1$ with capacity large enough to hold all jobs in the build schedule, and specify j_{N+1}.) One of the reasons for this choice is that it is easier to ascertain feasibility of a shutdown policy specified in this form, ensuring that no line element is shut down in mid-cycle and verifying the capacity constraints at each line element. In addition, if some components of the production system are subject to uncertainty (e.g., line elements may break down at random and need to be repaired), job-based specification remains easy to work with in such a stochastic setting.

With a (feasible) shutdown policy specified in the above form, it is also easy to identify which end-state goals are met by calculating, for each line element $n \in \mathbf{N}$, the tuple of WIP r_n contained in that line element using the build schedule (see Section 2.3 for details). If $r_n \in R_n$, i.e., the corresponding end-state goal is met, a reward of value v_n is awarded; otherwise, a penalty of value p_n is assessed. It should be noted that we express goal satisfaction in the above form of set containment for notational convenience. Although occasionally an end-state goal for a line element is so specific that the corresponding set R_n consists of only a small number of WIP tuples, often the goal is fairly general, e.g., "5 jobs regardless of their types," or "at least one job of type 1." In such cases the contents of the set R_n will be described

using predicates $\leq$, $=$, and $\geq$ on the number of jobs, or jobs of particular types, and containment $r_n \in R_n$ will be checked simply by verifying that the tuple r_n satisfies the resulting constraints.

In contrast to end-state goal satisfaction, with the shutdown policy specified in the above form, the time of shutdown of each line element and the overall shutdown time is not so easy to compute. Even in a perfectly predictable, i.e., **deterministic** line with known job cycle times on each station and no unpredictable breakdowns, interactions between elements of a capacitated serial production line, such as starving and blocking, result in lack of an analytical expression for the time at which line element n releases job j_n, and thus shuts down. To overcome this difficulty, a computer simulation can be built to approximate the actual system. In a deterministic environment, the simulation can handle the recursive calculations required to compute these exit/release times (see Section 2.3). In a stochastic environment, the simulation can also generate random numbers, such as failure times of stations and durations of their repairs, given data on mean cycles between failure (MCBF) and mean time to repair (MTTR) observed for the production line in question.

The simulation can be viewed as a function, $F((j_1, \ldots, j_N, j_{N+1}),\ T_{\max})$, that takes the decisions at the line elements, $(j_1, \ldots, j_N, j_{N+1})$, and the upper bound on the production line running time, $T_{\max}$, as inputs. The outputs of the simulation are $((r_1, \ldots, r_N),\ T_s)$, where r_n's are as described above, and T_s represents the time at which the production line comes to a full stop as a result of the decisions.

While line elements in the production line may stop at different times, here we define the shutdown time of the line as the latest shutdown time over all line elements. This definition is particularly well-suited for highly automated production lines, which tend not to have direct labor operators assigned to each line element,

but rather floating personnel that are on the clock as long as some portion of the line is running. The body shop section of an automotive assembly plant, where various pieces of metal are attached together to form the body of the vehicle, is an example of such a highly automated area. Given the shutdown time of the line, the associated time penalty is easily computed. Recall that T_d denotes the desired stopping time. When $T_s > T_d$, overtime cost is incurred at the rate of p_o per unit time. Otherwise, when $T_s < T_d$, a penalty associated with lost production time is charged at the rate of p_l per unit time. (The model and the forthcoming DP formulation can be easily modified to consider other possible definitions of overtime/lost production time penalty.)

If we define the set $\tilde{\mathbf{J}} \subset \mathbf{J}^{N+1}$ as the set of all decision vectors that satisfy line capacity and ordering constraints, the End-State problem can be formally defined as:

$$
\begin{aligned}
(2.1) \quad \max_{j_1,\ldots,j_N,j_{N+1}} \quad & \sum_{n\in\mathbf{N}} [v_n \mathbf{1}_{r_n\in R_n} - p_n \mathbf{1}_{r_n\notin R_n}] - p_o(T_s - T_d)^+ - p_l(T_d - T_s)^+ \\
\text{s.t.} \quad & \\
& ((r_1,\ldots,r_N),\ T_s) = F((j_1,\ldots,j_N,j_{N+1}),\ T_{\max}) \\
& \mathbf{1}_{r_n\in R_n} = \begin{cases} 1, & r_n \in R_n \\ 0, & \text{o/w} \end{cases},\ \forall\, n \in \mathbf{N} \\
& (j_1,\ldots,j_N,j_{N+1}) \in \tilde{\mathbf{J}} \subset \mathbf{J}^{N+1}.
\end{aligned}
$$

The above formulation implicitly assumes that there is at most one end-state goal associated with each line element, but an obvious extension to the above formulation can incorporate multiple (separate) goals as well (see Section 2.4). For the sake of notational simplicity, we refer to formulation (2.1) in Section 2.3.

2.3 Deterministic Dynamic Programming Formulation

In this section we develop an efficient dynamic programming formulation of the end-state problem (2.1) in a deterministic setting. That is, in the implementation of the simulation function $F(\cdot)$ that approximates the system, we assume that the equipment is reliable and there are no unpredictable breakdowns. As the goal in this chapter is to introduce the End-State problem and to provide an optimization approach that produces solutions superior to the ones currently used, we leave the development of a model and algorithm that incorporate stochasticity to Chapter IV. However, as discussed in the previous section, any shutdown policy specified in our chosen format can still be implemented in a stochastic environment. In Section 2.4 we investigate how an optimal solution of (2.1) found with a deterministic implementation of $F(\cdot)$ performs in a stochastic setting.

Toward providing an efficient algorithm for solving (2.1), observe that in a serial production line, shutdown decisions $(j_1, \ldots, j_N, j_{N+1})$ can be made sequentially along the production line. Moreover, once a decision has been made at one line element, feasible decisions at neighboring line elements are significantly restricted. In addition, we do not need to know all the decisions made at the line elements considered previously; instead, condensed information that summarizes the effect of past decisions is sufficient. Such summary information is called the **state** of the system. Once the potential states of the system — its **state space** — are defined, we can solve the optimization problem by using **dynamic programming** (DP). With a properly defined state space, dynamic programming can efficiently solve sequential decision problems (for more detail, refer to Denardo, 1982).

In subsections 2.3.1 and 2.3.2, we describe analytical derivations of both the end-

state and the shutdown time; i.e., we describe the specifics of the implementation of the function $F(\cdot)$. Subsection 2.3.3 presents the dynamic programming model and algorithm for the end-state planning problem.

2.3.1 Deriving the End-State from the Shutdown Policy

A shutdown policy $(j_1, j_2, \ldots, j_N, j_{N+1})$ must be jointly feasible in the sense that it does not violate the ordering of the jobs given by the build schedule or the capacities of the line elements. For two consecutive line elements n and $n-1$, j_n must be at least j_{n-1}. In addition, since the difference of j_n and j_{n-1} indicates the number of jobs left in line element $n-1$, we require that $j_n - j_{n-1} \leq m_{n-1}$. Summarizing the above two observations, the values for j_n are constrained as follows:

$$j_n \in \begin{cases} \mathbf{J} & , n = 1, \\ \{j_{n-1}, j_{n-1}+1, \ldots, j_{n-1}+m_{n-1}\} & , n > 1. \end{cases} \tag{2.2}$$

With all j_n feasible, the end-state of line element n is then:

$$r_n = (j_n + 1, \ldots, j_{n+1}), \quad n \leq N, \tag{2.3}$$

where line element n is empty if $j_n = j_{n+1}$. Note that the knowledge of both j_n and j_{n+1} is required to determine line element n's content.

2.3.2 Computing the Shutdown Time from the Shutdown Policy

Let $e_{j,n}$ denote the time when job j exits line element n. The matrix $\{e_{j,n}, j \in \mathbf{J}, n \in \mathbf{N}\}$ is referred to as the **flow matrix** as it contains information about the flow of the jobs through the line.

Let $t_{j,n}$ be the processing, or cycle, time of job j at line element n (the dummy element $N+1$ is assumed to have zero processing time), and assume that the processing time also includes the transfer time of the job between line elements $n+1$

and n. When job j completes processing at line element n, it can move on to line element $n-1$ if there is spare capacity available. Therefore, the time $e_{j,n}$ at which job j can exit line element n has to satisfy three conditions:

1. Job j must exit line element $n+1$, which occurs at time $e_{j,n+1}$, and complete processing at line element n, which takes $t_{j,n}$ units of time. Therefore, $e_{j,n} \geq e_{j,n+1} + t_{j,n}$.

2. Job $j-1$ must exit line element n, which occurs at time $e_{j-1,n}$, and job j must subsequently be processed at line element n, requiring $t_{j,n}$ units of time. This yields $e_{j,n} \geq e_{j-1,n} + t_{j,n}$.

3. Line element $n-1$ must have available capacity to accept job j. Since the capacity of line element $n-1$ is m_{n-1}, there will be room for job j in line element $n-1$ once job $j-m_{n-1}$ exits. As this event occurs at time $e_{j-m_{n-1},n-1}$, we have $e_{j,n} \geq e_{j-m_{n-1},n-1}$.

Since we assume that the line operates without interruptions, $e_{j,n}$ can be computed by taking the maximum over these three lower bounds, yielding the recursive equation:

for $j = 1, \ldots, J$ **do**

for $n = N, \ldots, 1$ **do**

$$e_{j,n} = \max\{e_{j,n+1} + t_{j,n}\ ,\ e_{j-1,n} + t_{j,n}\ ,\ e_{j-m_{n-1},n-1}\} \tag{2.4}$$

end for

end for

where we set $e_{j,n} = 0$ if either $j \leq 0$ or $n \leq 0$ or $n > N$.

We can compute the production line shutdown time from the flow matrix $\{e_{j,n}\}$

and the collection of decisions $\{j_n\}$ as:

$$T_s = \max_{n \in \mathbf{N}} \{e_{j_n,n}\}. \tag{2.5}$$

Alternatively, if we let T_n be the maximum shutdown time of line elements 1 to n, the production line shutdown time could be computed recursively as

$$T_0 = 0,\ T_n = \max\{e_{j_n,n},\ T_{n-1}\},\ n = 1, \ldots, N, \tag{2.6}$$

and $T_s = T_N$.

2.3.3 Dynamic Programming Model

From the above assumptions and derivations, the problem can be cast as a sequential decision process, where a decision is made at each line element, starting from line element 1. From Equation (2.2), we see that the set of feasible decisions at line element n is constrained by j_{n-1}. The time when each job leaves each line element, assuming the line element has not yet been shut down, can be computed a priori as shown in Equation (2.4), and the resulting flow matrix is considered to be input data for the problem. Then, given T_{n-1}, we can compute T_n as soon as j_n is chosen using Equation (2.6).

From the above description, the minimal amount of information required to make a decision at each line element includes: n, the current line element ID, j_{n-1}, the decision from the downstream line element, and T_{n-1}, the maximum shutdown time up to and including line element $n-1$. When decision j_n at line element n is chosen, T_n is calculated based on the corresponding element of the flow matrix and T_{n-1}. The reward/penalty for satisfying the goal specified for line element $n-1$ is obtained by first computing r_{n-1} according to Equation (2.3), then checking to see if $r_{n-1} \in R_{n-1}$. If so, the decision garners a reward of v_{n-1}, otherwise it incurs a penalty of p_{n-1}.

Note that we can calculate the reward/penalty at line element n only after we have made a decision for line element $n+1$. This is because the contents of line element n are not known until the decision at line element $n+1$ is made (see Equation (2.3)). Recall that we define a dummy line element, $N+1$, to control the contents of line element N.

When we reach line element $N+1$, the beginning of the line, we set $T_s = T_N$, and the overtime/lost production cost can be computed accordingly.

Formally, the DP formulation is as follows:

- State (n, j, T) of the DP:
  - n is the stage of the DP, representing the ID of the current line element,
  - j is the decision at line element $n-1$; it serves as the lower bound on j_n,
  - T is the maximum of shutdown times of line elements from 1 to $n-1$.

  For the initial stage $n = 1$, there is only one state, and that is $(n, T) = (1, 0)$.

- Feasible decisions at state (n, j, T):

$$j_n \in \begin{cases} \mathbf{J}, & n = 1 \\ \{j, j+1, \ldots, j+m_{n-1}\}, & n > 1. \end{cases} \tag{2.7}$$

- State transition functions are as described above.

- Reward function at state (n, j, T) with decision j_n:

  (2.8)

$$V(1,0) = 0, \text{ and } V(n, j; j_n) = \left(\mathbf{1}_{r_{n-1} \in R_{n-1}} \cdot v_{n-1} - \mathbf{1}_{r_{n-1} \notin R_{n-1}} \cdot p_{n-1}\right),\ 2 \le n \le N+1,$$

  where

$$r_{n-1} = (j+1, \ldots, j_n) \text{ and } \mathbf{1}_{r_{n-1} \in R_{n-1}} = \begin{cases} 1, & r_{n-1} \in R_{n-1} \\ 0, & \text{o/w.} \end{cases}$$

Note that a reward/penalty is assessed at stage n for meeting the end-state goal at line element $n-1$, as discussed above. As described in Equation (2.3) and duplicated in the equation for r_{n-1} above, one can compute the contents of line element $n-1$ once the decision at line element n is made. Thus the reward/penalty associated with making decision j_n at line element n is based upon the content and goal satisfaction at line element $n-1$.

- Terminal cost:

$$L(T) = p_o(T - T_d)^+ + p_l(T_d - T)^+, \tag{2.9}$$

where T is the shutdown time of the line. The terminal cost represents the overtime or lost production time penalty incurred as a result of a shutdown time that is different from the desired shutdown time.

- Functional equation at state (n, j, T): let $f(n, j, T)$ be the maximum value one can attain by acting optimally from line element n to $N+1$, if the current state is (n, j, T).

For $n = 1$:

$$f(1, 0) = \max_{j_1 \in \mathbf{J}} \{f(2, j_1, e_{j_1,1})\} \tag{2.10}$$

For $2 \leq n \leq N$:

$$f(n, j, T) = \max_{j_n \in \{j, j+1, \ldots, j+m_{n-1}\}} \{V(n, j; j_n) + f(n+1, j_n, \max\{T, e_{j_n,n}\})\} \tag{2.11}$$

For $n = N + 1$:

$$f(N+1, j, T) = \max_{j_{N+1} \in \{j, j+1, \ldots, j+m_N\}} \{V(N+1, j; j_{N+1}) - L(T)\} \tag{2.12}$$

- The optimal value of the End-State problem is given by $f(1, 0)$.

In the model so far we have essentially ignored the upper bound $T_{\max}$ on the shutdown time (or, in other words, implicitly assumed that the shutdown times of all line elements do not exceed $T_{\max}$). In the following subsection we discuss how to prune the states of the developed DP to incorporate this upper bound, as well as to allow for presence of jobs in the production line at initialization.

Pruning DP states: Starting the Production Line with Initial Content and Shutting Down No Later Than $T_{\max}$

In the DP model described in the previous section, we implicitly assumed that the production line is started empty, with job 1 about to enter the line. However, this is not always the case — for instance, jobs can be positioned at line elements at the beginning of the production run, and/or, as is often the case, the manager may begin planning shutdown activities only, say, an hour prior to the desired shutdown time. We can incorporate this variation by pruning appropriate states from the DP.

Suppose the system is initialized at time 0, which can represent either the beginning of the production run or the time during production at which the End-State problem is being considered, and we are given an initial content of the line, indicating the position of each job. Let $n(k)$ be the ID of the line element where job k is located at initialization (if job k has not yet entered the system, let $n(k) = N + 1$), and assume that the flow matrix is computed to reflect any remaining cycle times of jobs which are being processed at time 0. Since job k starts at line element $n(k)$, all line elements upstream — with ID greater than $n(k)$ — cannot use job k as their shutdown decision. As a result, all states (n, j, T) with $n > n(k)$ and $j \leq k$ are pruned from the DP.

We can use a similar approach to ensure that the shutdown time resulting from the decision $(j_1, \ldots, j_N, j_{N+1})$ generated by the DP does not exceed $T_{\max}$ by pruning

all states (n, j, T) with T exceeding $T_{\max}$ from the DP.

Computational Complexity of DP Formulation

Here we compute an upper bound on the computational effort required to solve the above dynamic program by the standard "backward" dynamic programming algorithm. We assume that the values of $t_{j,n}$'s and $T_{\max}$ are integers (i.e., they are measured in whole seconds), and thus the values of T that need to be considered as part of state descriptions are also integer.

Let t_v be an upper bound on the computational time required to calculate the value of an argument of the maximum in expressions (2.11) – (2.12) for any of the states considered in the DP formulation, expressed as the number of flops. Then, referring to Equations (2.10) – (2.12), we see that the computational effort required to compute the value $f(n, j, T)$ for each state (n, j, T) with $2 \leq n \leq N + 1$ is approximately $(m_{n-1} + 1)t_v$, plus the effort required to perform m_{n-1} comparisons. Since t_v typically exceeds the effort required to perform a comparison, this effort can be further bounded by $(2m_{n-1} + 1)t_v$. The computational effort required to compute $f(1, 0)$ is the effort needed to perform J comparisons, and thus is bounded by $J\, t_v$.

Given that n, the line element ID, ranges from 1 to $N+1$, j, the job ID, ranges from 1 to J, and T, the shutdown time, ranges from 0 to $T_{\max}$, the total computational effort required to solve the DP can be bounded from above by:

$$\sum_{n=2}^{N+1} J\,(T_{\max} + 1)\,(2m_{n-1} + 1)t_v + Jt_v = N\,J\,(T_{\max} + 1)\,(2\bar{m} + 1)t_v + Jt_v$$

$$= J\,t_v\,((2\bar{m} + 1)N\,(T_{\max} + 1) + 1)\,,$$

where $\bar{m}$ is the average capacity of the line elements. Using values from the computational example discussed in Section 2.4, which is based on a GM assembly plant, we have $N = 66$ and $\bar{m} = 1.167$. (In most cases, the capacity of line elements

representing stations is 1, while the capacity of buffers varies greatly depending on the part of the plant. In this specific example, the average buffer capacity is 1.44.). Setting $J = 200$ and $T_{\max} = 4800$ (seconds), as an example, we obtain a numerical upper bound for the complexity:

$$J\, t_v \left((2\bar{m} + 1)N\left(T_{\max} + 1\right) + 1\right) \approx 2.11\mathrm{e} + 10^8 \cdot t_v.$$

Modern CPUs, with operating frequency measured in GHz, can provide computational performance in the range of several GFLOPS (10^9 floating point operations per second). Suppose we are equipped with a machine with one GFLOPS capability. If the value of t_v roughly equals the time required to perform 100 flops, the upper bound above suggests that the problem can be solved in under 21 seconds. Even with $t_v = 1000$, the upper bound is still under 3.5 minutes.

2.4 Computational Experiments

To demonstrate the benefit of using DP for the end-state planning problem, we used a hypothetical yet realistic end-state situation from a General Motors' production line. The scenario was based upon discussions with plant personnel about typical situations that they experience. The line topology and parameters such as cycletimes and capacities in this scenario are from an actual line at a GM plant. The build schedule is randomly generated according to the proportion of job types produced in the plant.

The values of parameters v_n, p_n, p_o and p_l used in the scenario are unitless for the purposes of preserving data confidentiality, and have no direct economic meaning. However, the values were chosen in an effort to preserve the relative proportions among corresponding parameters.

In discussions with GM staff, the cost of overtime was fairly easy to assess; estimates of the cost of lost production time were also reasonably easy to ascertain. On the other hand, production line managers have not had experience explicitly considering values of satisfying goals. Thus, although there is usually an understanding of which shutdown goals have higher priority than others, it is difficult to associate specific numeric values and penalties with the goals at hand. To estimate the value and penalty associated with a particular goal, one may consider 1) the cost of labor and materials required to perform the maintenance task at hand, 2) the labor and material cost of "manually" attaining the desired end-state to perform the task (e.g., manually off-loading jobs from a line element that is supposed to be empty), 3) the likelihood and cost of correcting quality problems or breakdowns resulting from a task left undone due to an unmet end-state goal, etc.

At the present time, most of these estimates are difficult to obtain since, in practice, managers use rules of thumb that aim to satisfy production targets (i.e., stopping on time), and decisions on which tasks to perform and thus which end-state goals to meet are made in an ad hoc manner. In the scenario described in this section, we assigned values and penalties to end-state goals that are fairly low relative to the cost of overtime and lost production. Despite this, these computational examples demonstrate that significant improvement in goal achievement can be attained with minimal sacrifice of production time. It is our hope that having access to a decision support tool such as this model will encourage a more detailed assessment of benefits and costs associated with meeting end-state goals and performing maintenance tasks.

What follows is a description of the scenario investigated, and then a comparison of the performance of an optimal policy obtained by the dynamic programming model

against a "rule-of-thumb" (ROT) policy.

2.4.1 Background and Description of a Typical Scenario

Since automotive manufacturing is extremely capital intensive and plants typically produce several different models of vehicles, new models are typically launched concurrently with existing production. This requires a complex and choreographed installation of new equipment, re-calibration of new and old equipment, and confirmation that changes do not impair existing production. The following scenario description is representative of a realistic scenario occurring in practice.

In this scenario, a plant is just starting to produce a small number of prototype builds of a new model. We refer to the current models as job types #1, #2, and #3, and to the new model as job type #4. Currently, these four job types constitute 30%, 35%, 20%, and 15%, respectively, of the plant production, and the build schedule during a typical shift would consist of a sequence of jobs of the four types in proportions roughly equal to the above percentages, in no particular order.

When launching a new vehicle, the most significant changes occur in the body shop area of the plant, so this scenario focuses on two zones of a body shop — engine compartment (EC) and underbody (UB) — depicted schematically in Figures 2.2 and 2.3. In these figures, the larger squares labeled with identification numbers represent stations (each with capacity 1), while the smaller rounded squares labeled with capacities represent buffers. As stated previously, the line segment used for these experiments is based upon an actual production line, using the topology, cycle times, and capacities of that line segment. Together these two zones will be treated as a serial production line.

We define eight high-level objectives we would like to achieve during a downtime period and the end-state goals based on each of these objectives. The goals are desig-

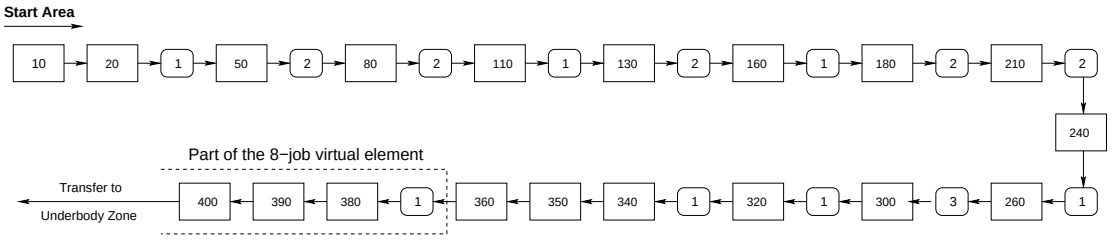

Figure 2.2: Schematic graph for the engine compartment zone.

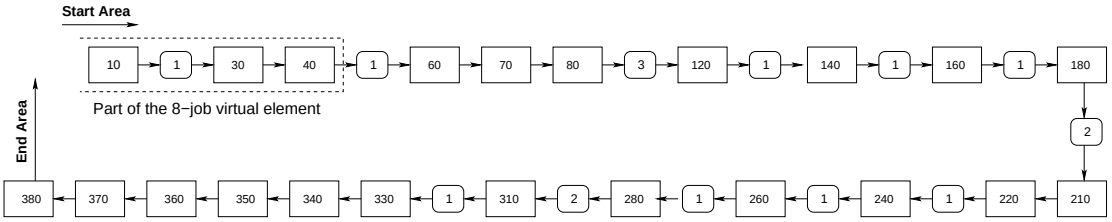

Figure 2.3: Schematic graph for the underbody zone.

nated to be of high, medium, or low priority, based on the emphasis the management wants to put on the objectives they accomplish.

1. The EC and UB zones are experiencing downtime as a result of changes to support the new model and are bottlenecking production. Therefore it is desirable to keep these areas operating as much as possible by filling every station and buffer position with a vehicle of some sort. Since each extra job present impacts throughput only slightly, the goals thus defined are of low priority.

2. EC stations 20, 50, 80, 130, 180, and 260 should be empty to allow verification that material can be loaded into them from newly modified conveyor systems. Although a problem left undetected could be costly, the tests can be delayed; alternatively, jobs could be manually offloaded during the downtime using forklifts, clearing the stations for verification. Therefore, each of these goals is of medium priority (note that these goals directly conflict with those defined by the first objective above.)

3. EC station 160 should have a job of type 4 in it to allow for training of the

welding robot to follow a new weld path. This is a high priority goal since this test is critical to launch timing. The buffers immediately before and after this job should be empty to allow engineers leeway to stop the line to better examine issues as this validation build progresses through the system. These latter goals are of low priority, since the only impact of not achieving them is lower throughput.

4. EC station 300 and 320 were re-calibrated yesterday to better process the new model. Unfortunately, there is worry that this may have caused problems for model type 2. These two stations and their immediately preceding buffers should contain models of type 2 to allow for testing. These goals are of high priority since it is unacceptable to produce low quality current vehicles and it is very difficult to test the calibration in any other way. The next four line elements after these stations should be empty to allow for jobs to be moved through stations 300 and 320. These latter goals are of medium priority, since jobs could be manually offloaded.

5. New equipment is being installed for UB station 350. To ensure adequate working space, the area from station 330 to 370 inclusive must be emptied. These are medium priority goals since jobs could be manually offloaded.

6. The eight-job area in the connecting part of the EC and UB zones from the buffer prior to EC station 380 up to UB station 40 should contain jobs of various types for testing of the new equipment. Sequences where four distinct job types follow four distinct job types are highly preferred, since they would provide the best opportunity to evaluate how the equipment adjusts from producing one type of vehicle to another. Slightly less preferable are sequences where each job type

still appears twice among the eight jobs (e.g., 1-2-1-2-3-4-3-4). Less preferable still are sequences where each job type appears at least once among the eight jobs (e.g., 1-2-3-4-3-3-3-3). To capture these considerations, we associate three goals with this area: a high priority goal, which would be met by the most desirable job sequences only, a medium-priority goal, which would also be met by the less desirable sequences, and a low priority goal, which would be met by any 8-job sequence containing at least one job of each of the four types.

7. UB stations 140, 180, 210, 220, 240, and 260 are slated for re-calibration this evening for jobs of type 1 or 4. Having a job of either type 1 or type 4 in each such station is a medium priority goal.

8. To enable precise measurements, UB stations 80 and 120 should be emptied. These are medium priority goals. Verification of the resulting quality requires that the job immediately preceding each of these stations be of type 4. These are medium priority goals.

We assume an early shutdown costs 10 units per minute due to lost production, while a late shutdown costs 5 units per minute due to overtime expenses. These values can be computed based upon the lost revenue due to an early shutdown or the extra expense of running overtime. We classify goals into three categories, with high, medium, and low value goals, respectively, earning 20, 5, and 1 units if achieved, and costing 7, 3, and 1 units if not achieved. One can view high value goals as those that will have the greatest impact on throughput and quality while low value goals have far less of an effect. As stated previously, these values are unitless, but represent the relative proportions of the actual values.

We assume that the planning horizon for shutdown policy optimization is one

hour before the end of the shift, so the line is initially filled with jobs, and the desired shutdown time is $T_d = 3{,}600$ seconds. This approximately coincides with the planning horizon of plant personnel. At most 10 minutes of overtime are allowed, so $T_{\max} = 4{,}200$ seconds.

Note that in item 6 we described goals associated with a set of line elements rather than a single line element. To represent this type of goal without modifying the DP model, we define a virtual line element that aggregates the area dealt with in item 6, beginning with the buffer prior to EC station 380 up to UB station 40. This aggregate line element has capacity 8, the sum of the capacities of the line elements it contains, and processing time equal to the sum of the processing times of its contained line elements. Physical line elements included in this virtual line element also have goals associated with each one of them. To reflect these goals, we need to modify the reward function associated with the line element immediately preceding the virtual line element. To be more specific, suppose the virtual line element has ID n, and recall that the decision made at line element $n+1$ determines the content of the virtual line element. For each feasible decision j_{n+1}, besides evaluating $V(n+1, j; j_{n+1})$ (as defined in Equation (2.8)), which looks at the goals defined on the virtual line element, we must also consider values and penalties resulting from satisfaction of goals associated with line elements within the virtual line element. This leads to an optimization sub-problem that can be solved by a DP formulation similar to the overall DP.

With this information, we are ready to generate representative problem instances and solve for their optimal shutdown policies.

2.4.2 Experimental Results

Comparison of the Optimal Policy and a Rule-of-Thumb Policy

To generate a problem instance, we created a build schedule by sampling jobs at random in accordance with the given percentages of the four job types. The resulting problem was solved by the DP algorithm described in Section 2.3 in approximately 6 seconds on a 3.4 GHz Pentium 4 running Red Hat Linux. The optimal policy found stops the production line at $T_s = 3,705$ seconds (105 seconds later than the desired time), and achieves 68 of the 92 goals defined. The objective value of the optimal policy, i.e., the sum of rewards for goals achieved minus the sum of the penalties for missed goals and minus any penalty due to a shutdown time that deviates from the desired shutdown time, was 197.25.

As mentioned in Section 2.4.1, for some of the line elements, multiple goals associated with each of them are in conflict with one another; thus, no feasible shutdown policy would capture the rewards for all of these goals. To get a sense of how much of the potential value was in fact achieved by the optimal policy, we estimated the maximum achievable goal value for each line element by calculating the value associated with each non-conflicting combination of goals. For example, if a line element has a low priority goal specifying that the line element be empty at shutdown and a medium priority goal specifying that the line element contain a job of type 3, the potential achievable value is estimated as 4: a value of 5 if a job of type 3 is in the line element, minus a penalty of 1 for not satisfying the low priority goal. Note that these potential values only provide an estimate (in fact, an upper bound) on the value attainable by satisfying the goals, since they do not take into account possible conflicts among goals associated with different line elements, nor do they consider the specifics of the build schedule. Figure 2.4 shows these estimates of potential achiev-

able values for each line element (gray bars), along with the actual values achieved by the optimal policy (black bars). If only a black bar is shown, the net value associated with that line element was in fact the maximum achievable. The presence of a gray bar indicates that the optimal policy did not garner all of the estimated potential value, with the difference between the two bars indicating the difference between the estimated potential and actual value.

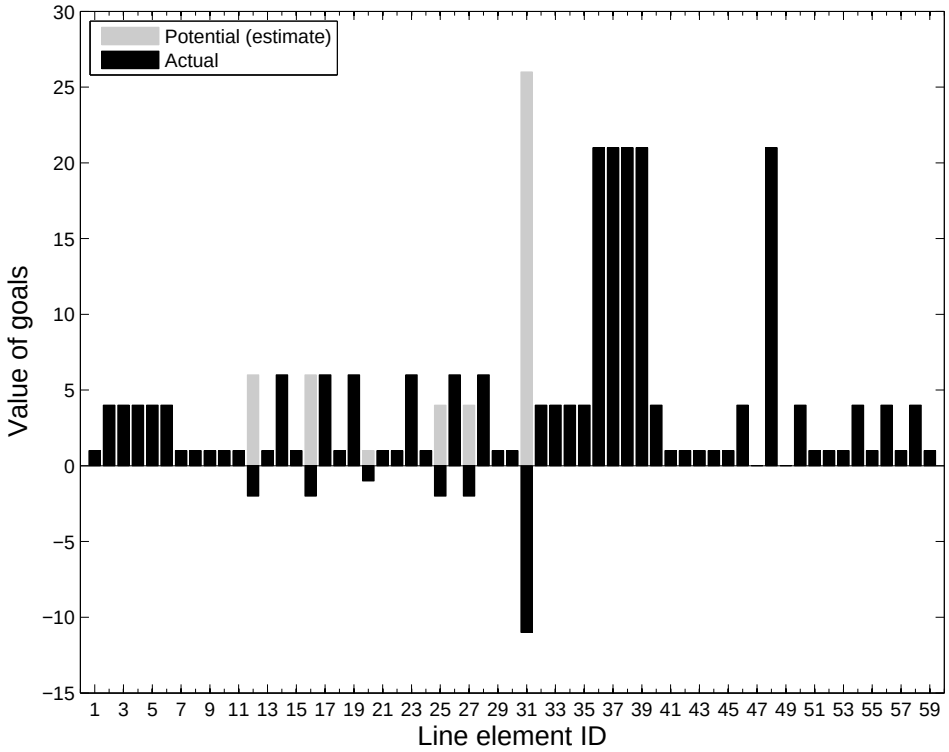

Figure 2.4: Estimate of potential achievable value and value achieved by optimal policy.

Figure 2.5 shows the shutdown time of each line element. Recall that the line's shutdown time, T_s, is the maximum shutdown time over all the line elements. The figure demonstrates that, overall, the line elements at the head of the line tend to shut down earlier than those at the tail, with the shutdown time of 3,705 seconds dictated by line elements 1 and 2.

To assess the benefits of the optimal shutdown policy, we set out to compare it

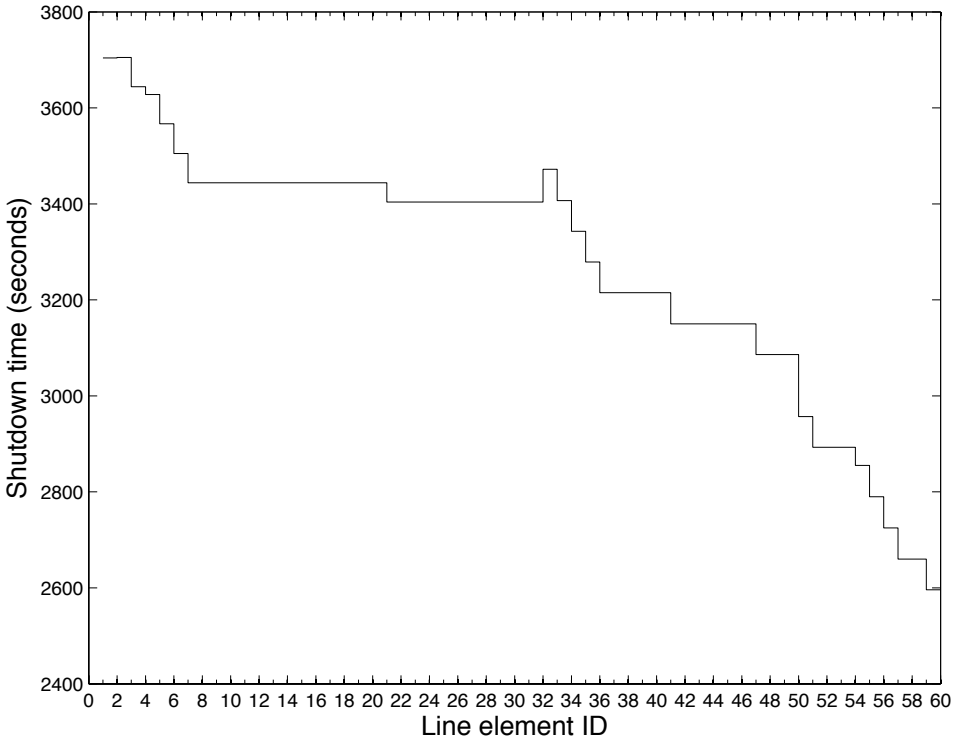

Figure 2.5: Shutdown time for each line element.

to a realistic policy currently in use. Typically, shutdown policies are determined by managers of the production line, several minutes to an hour prior to the desired shutdown time. Since human planners without access to an optimization model are unlikely to be able to quickly evaluate the 92 goals specified in the scenario, as well as the current state of the production line and the build schedule, to come up with an optimal shutdown plan, they typically use rule-of-thumb (ROT) guidelines to come up with a shutdown policy.

For example, GM plant managers report that a common guideline (although not used exclusively) can be roughly described as "shut the line down as close as possible to the desired shutdown time, T_d, while meeting as many goals as possible." While this policy may sound simple (if vague), it is difficult, if not impossible, to mimic the decisions of an experienced manager attempting to meet "as many goals as possible"

without the aid of a formal algorithm. In practice, it is fairly easy for a manager to ensure that the line is shut down at the desired time, but choices made with respect to goal satisfaction are difficult to formalize. Therefore, as a basis for comparison with the optimal policy, we define a **formal ROT policy** which finds an optimal shutdown plan, subject to the constraint that shutdown time T_s is as close to T_d as is feasible for the given flow matrix. This formal ROT policy will have the same shutdown time as the real policy deployed by a manager, but perform better in terms of goal satisfaction. Therefore, the objective value of the formal ROT policy will provide an upper bound on the objective value of any real policy based on the above guideline.

The formal ROT policy can be computed using the existing DP solver by setting overtime and lost production costs to extremely high values. The resultant shutdown plan would meet goals optimally subject to stopping the line as close as possible to the desired shutdown time, and its objective value can be computed by using the goal values and penalties and the true penalties for shutdown time deviations.

When the formal ROT policy is applied to the sample scenario, we find that the line is shut down at exactly 3,600 seconds. The number of goals satisfied is 64 (compared to 68 satisfied by the optimal policy), and the associated objective value is 118 (which is roughly 40% lower than the optimal value of 197.25). Recall that the formal ROT policy is likely to perform much better than any real policy based on this guideline would, and thus the added value of the optimal strategy is likely to be even higher compared to the state of practice.

Testing a Variety of Build Schedules

In the previous subsection we demonstrated the benefit of using an optimal policy compared to an ROT policy on a particular problem instance. However, the struc-

ture and performance of any shutdown policy depend on the build schedule defining the problem instance. To explore the impact of build schedules, we extended the experiments of the previous subsection as follows: we took a sample of 100 build schedules, each generated by sampling jobs at random in accordance with the given percentages of the four job types. For each build schedule, we found the optimal shutdown policy and the formal ROT policy. In all instances the shutdown time of ROT policies was exactly equal to the desired stopping time, while the average of shutdown times dictated by optimal policies exceeded the desired time by 22 seconds.

Over the 100 build schedules, the average value of the optimal policies was 182.88, while the average value of the formal ROT policies was 123.26. For each build schedule we computed the percentage of value lost by using the formal ROT policy, as compared to the optimal policy. The average of these percentages over the 100 build schedules was 32%. Once again, since the formal ROT policy performs better than a real policy based on the ROT guidelines, the value lost in practice by not using the optimal policies is likely to be even higher. These results suggest that the ROT guideline used in practice, in effect, gives too much weight to on-time shutdown compared to goal satisfaction, which significantly lowers the overall value attained.

Testing Policies in a Stochastic Setting

In the experiments up to this point we assumed that the line elements were completely reliable, and thus the flow matrix for a production run could be computed a priori based on the build schedule, the line topology and the cycle times of line elements. A real production line, however, experiences breakdowns of line elements which then need to be repaired in order to resume production. For each line element, the number of cycles between breakdowns is modeled as an exponential random variable; once a breakdown at a line element has occurred, the time until it is repaired is

also modeled as an exponential random variable. These modeling choices, as well as the rates of all the exponentials involved, were based on the data collected at the GM plant which served as a basis for the scenario discussed in Section 2.4.1. It should be noted that, although overall reliability of this particular line is quite high, even infrequent and short downtimes of a few line elements can have a significant impact on the flow matrix, and thus on the value of a shutdown policy.

The experiment was conducted as follows. A build schedule was generated, and a "deterministic" flow matrix was constructed based on the cycle times of the line elements, ignoring the possibility of breakdowns during production (i.e., exactly as in the previous experiments). An optimal shutdown policy for the resulting flow matrix was then found by solving the DP (we refer to the resulting policy as the **basic** policy). To estimate the expected performance of the basic policy within a stochastic environment, we then generated a sample of 500 flow matrices by sampling the times of line element breakdowns and repair times, and calculated the average of values attained by the basic policy on these flow matrices. (In most cases, the set of goals satisfied by the basic policy will remain the same for different flow matrices, while the shutdown time will change accordingly. However, in certain cases, due to changes in the flow matrix, line elements will have to be shut down at $T_{\max}$ rather than upon releasing the job specified by the policy.) The average value attained by the basic policy was 143.36, with a standard deviation of 38.65. As a point of comparison, we solved the DP for each of the sampled 500 flow matrices; the average value of such **optimal policies with hindsight** was 180.31. The latter provides an upper bound on the expected value that can be attained by **any policy** within the stochastic environment for the given build schedule, since it assumes full hindsight, i.e., complete knowledge of the timing of breakdowns and repair times of

line elements during the production process. As another point of comparison, we defined the **formal ROT policy with hindsight** as found by applying the formal ROT policy to the sampled flow matrix; the average value of the formal ROT policy with hindsight over the 500 flow matrices was 148.15.

The above experiment was repeated for 10 different build schedules. Average values attained by the basic policies ranged from 112.07 to 166.26, with a mean of 144.86, while the average values of optimal policies with hindsight ranged from 150.18 to 208.09, with a mean of 180.67 (the average values of the formal ROT policies with hindsight ranged from 118.30 to 166.24 with a mean of 148.22). For the ten build schedules examined, the basic policy came within 14.6% to 26.2% (with a mean of 19.9%) of the average value attained by the optimal policy with hindsight. The formal ROT policy with hindsight came, on average, within 21.2% of the upper bound.

The results show that the basic policy performs almost as well as the ROT policy with hindsight in spite of the advantage (perfect hindsight) we have granted the ROT policy. The basic policy performed fairly well as compared to the optimal policy with hindsight, which provides an upper bound on the expected value of the (unknown) optimal policy within the stochastic framework. The observed difference reflects the upper bound on the (potential) benefit in developing an optimization model that includes stochasticity explicitly. Such a model will be the focus of Chapter IV, with the potential to improve shutdown planning, especially for less reliable production lines.

2.5 Conclusions

We have shown that a dynamic programming model can be used in making complex decisions of when to shut down each element of a production line, considering end-state goal fulfillment, lost production costs, and overtime costs. In addition, a numerical case study was presented in which the superiority of our optimization over a rule-of-thumb method was demonstrated. A software implementation based in part on the work described in this chapter was developed and is in the process of being deployed as part of a pilot installation at a General Motors' assembly plant.

This work has laid the foundation for the next two chapters. Among the two most important areas of subsequent work are non-serial production lines and explicit inclusion of stochasticity into the optimization model. A model that considers non-serial production will be able to account for line topologies such as merging of sub-assemblies into a main line and parallel production, and is the subject of the next chapter. In this chapter, we tested the performance of the optimal solution of the deterministic model within a stochastic environment, and the results suggest potential opportunity in explicitly modeling the stochastic effects of machine breakdowns within a stochastic optimization algorithm. This is the topic of Chapter IV.

Yet another opportunity exists in a joint optimization of the shutdown policy and the build schedule. Section 2.4 illustrated the fact that the build schedule can have an appreciable effect on the value of a shutdown policy. This result suggests that we may be able to modify the ordering of jobs in concert with the development of the shutdown policy to improve the objective function value.

CHAPTER III

A Hierarchical Network Approach to the Non-serial End-State Problem

3.1 Introduction

In Chapter II, we introduced the End-State problem formulation and presented a dynamic programming approach for determining an optimal shutdown policy for a serial production line. In this chapter, we extend this formulation beyond basic serial lines to handle the case where a sub-assembly merges into the main line, either from an independent sub-assembly line or by a split of the sub-assemblies of a job into separate lines that eventually merge back together. The work presented in this chapter extends the mathematical model to the general assembly area of the plant.

Chapter II demonstrated that the policy yielded by the dynamic program exhibited superior performance to an optimistic rule of thumb resembling policies used in practice. This chapter makes two modifications to the earlier formulation that enable the extension of this model to certain types of non-serial lines:

- We assess time penalties on an individual line element basis. This enables the removal of the maximum shutdown time from the state space, reduces the computational requirements of the serial line case, and also leads to solutions that push more work in process (WIP) downstream than the algorithm from

Chapter II.

- We extend the network model to incorporate a hierarchical view of the stations and buffers of a production line. As a result, new line topologies and types of goals can be handled.

This chapter is structured as follows: Section 3.2 introduces the new concepts for the hierarchical modeling approach. Section 3.3 creates a new dynamic programming formulation based upon the new model. Section 3.4 details computational experiments conducted to evaluate the performance of the new formulation. Section 3.5 summarizes the contribution of this work and the results of the analysis and experiments.

3.2 Network Model

As in Chapter II, the fundamental step in developing an algorithm to solve the End-State problem is to model the production line as a network. In this section, we first introduce the concepts underlying the hierarchical decomposition of the production line. Then we describe the hierarchical decomposition of a production line that expands the applicability of the model to certain non-serial lines. Subsequently, we summarize the notation related to the hierarchical decomposition. Finally, we present a revised mathematical programming model for the End-State problem.

3.2.1 Hierarchical Network Concepts

The original model could only handle serial lines. In this chapter, a hierarchical decomposition of the production line enables the modeling of certain types of non-serial lines commonly seen within the general assembly area of an automotive manufacturing environment. While it is common for certain stations and buffers to

be logically grouped around producing a particular sub-assembly, this decomposition is at an even finer level of granularity. The remainder of this section describes the decomposition of the line, and refers to elements of Figure 3.1.

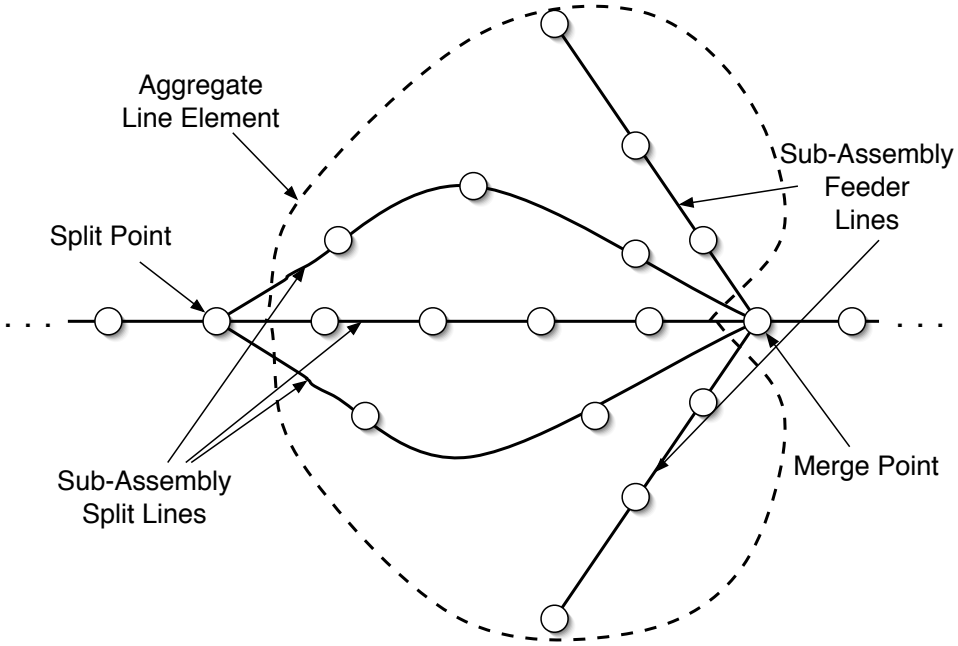

Figure 3.1: Partial Network Representation of a Production Line

We consider a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where the nodes of the graph are the stations and buffers of the production line, collectively referred to as **line elements** as in Chapter II. The arcs of the directed graph represent the various mechanisms used to transfer jobs from one line element to another. The arc (n, n') exists in $\mathcal{A}$ if and only if a job can be transferred from line element n to line element n' without going through any other line elements. Line element n is referred to as a **predecessor** of n', while n' is a **successor** of n. Denote by $\mathbf{P}_n$ the set of predecessors of line element n and by $\mathbf{S}_n$ the set of successors of line element n.

The purpose of the hierarchical decomposition is to extend the existing serial line model to handle two common topologies seen in actual production lines. The model

presented in this chapter can handle sub-assemblies merging into the main production line. In addition, this model can also handle the case where a production line splits, sending separate parts of a job to independent areas for processing, and then later merges back together. Such a split is referred to as an **assembly split**, while the corresponding merge is referred to as an **assembly merge**. In contrast, **production split** and **production merge** refer to the case where multiple, identical lines exist to perform work on different jobs in parallel. Production splits and merges are not applicable in our setting and are not covered by the model in this chapter.

Several ideas must be introduced to facilitate the new modeling approach. The most important is that of an aggregate line element. An **aggregate line element** is used to group serial line segments that must coordinate their decisions to satisfy the line capacity and ordering constraints. In Figure 3.1, the dotted line illustrates the aggregate line element for this particular example.

In addition to the idea of an aggregate line element, we must also introduce split and merge points. A **split point** is a line element from which different sub-assemblies of the same job are sent on different paths. A **merge point** is the line element at which sub-assemblies that were previously separated at a split point merge back together, and/or where sub-assemblies are fed into the line. By definition, a split point has at least two arcs coming out of it, while a merge point has at least two arcs going into it. We can formally describe the sets of split points and merge points, denoted $\mathcal{S}_{\mathbf{N}}$ and $\mathcal{M}_{\mathbf{N}}$, respectively:

$$\mathcal{S}_{\mathbf{N}} = \{n \in \mathbf{N} : |\mathbf{S}_n| > 1\}, \tag{3.1}$$

$$\mathcal{M}_{\mathbf{N}} = \{n \in \mathbf{N} : |\mathbf{P}_n| > 1\}, \tag{3.2}$$

where $|\mathbf{X}|$ denotes the cardinality of set $\mathbf{X}$.

Section 1.1.3 described a topology seen in general assembly of a truck plant where this type of split and merge occurs. The cab and bed of a truck are separated coming out of the paint shop and sent on separate paths for processing before coming back together prior to the marriage of the cab and bed with the chassis.

All serial lines within an aggregate line element are either sub-assembly split lines or sub-assembly feeder lines. **Sub-assembly split lines** are those serial line segments that are restricted by both a split point and a merge point in the sense that the sequence of jobs entering the merge point must match the sequence of jobs exiting the corresponding split point. **Sub-assembly feeder lines**, on the other hand, are only restricted by a merge point.

The final two concepts needed for this new approach are those of a child and parent line element. A (possibly aggregate) line element is a **child** of the smallest aggregate line element within which the line element is contained. Similarly, the **parent** of a line element is the smallest aggregate line element that contains that line element. Denote by $\mathbf{C}_n$ the set of children of line element n. Note that any number of the children of an aggregate line element could also be aggregate line elements (see, e.g., Figure 3.2).

3.2.2 Decomposition of a Production Line

Having defined the new concepts above, we can now move on to describe how to decompose a given production line. To summarize the decomposition approach, an aggregate line element is created for each set of sub-assembly split lines that share the same split point and merge point, as well as for each set of sub-assembly feeder lines that feed into a merge point that does not have any sub-assembly split lines going into it. For example, in Figure 3.1, the three sub-assembly split lines all split at the same split point and merge back together, along with two sub-assembly feeder

lines, at the same merge point. In this case, only one aggregate line element need be created to contain the three sub-assembly split lines and two sub-assembly feeder lines.

We start with a given network representation of a production line, a directed, acyclic graph composed of nodes and arcs. Recall that the nodes of the network represent line elements (the generic term for stations and buffers), while the **arcs** of the network represent the connections between line elements. A connection exists between line element n and line element n' if and only if the directed arc $(n, n') \in \mathcal{A}$. By chaining several arcs (and their corresponding line elements) together, we can construct a **path**. A path from line element n to line element n', denoted $\pi(n, n')$, exists if there exist arcs $(n, n_1), (n_1, n_2), \ldots, (n_m, n') \in \mathcal{A}$. The set of all paths is denoted by Π, while the set of all paths from line element n to n' is denoted $\Pi(n, n') \subset \Pi$. The set of line elements on a path π is denoted by $\mathcal{L}(\pi)$.

Next we proceed with identification of split-merge pairs from which we identify the aggregate line elements that must be formed. A split-merge pair is a pair of line elements, one a split point, the other a merge point, such that at least two distinct paths exist from the split point to the merge point with no intermediate line elements in common. The set of all split-merge pairs is characterized by:

$$\begin{aligned} \mathcal{P} = \{(n_s, n_m) |\ |\Pi(n_s, n_m)| \geq 2, \mathcal{L}(\pi^i) \bigcap \mathcal{L}(\pi^j) = \{n_s, n_m\}, \\ \forall i \neq j, \pi^i, \pi^j \in \Pi(n_s, n_m), n_s \in \mathcal{S}_{\mathbf{N}}, n_m \in \mathcal{M}_{\mathbf{N}}\}. \end{aligned} \tag{3.3}$$

An aggregate line element is created for each split-merge pair, as well as for each merge point with no corresponding split point (meaning only sub-assembly feeder lines connect to the merge point). In the case of an aggregate line element for a split-merge pair, the aggregate line element will contain all line elements on sub-assembly split lines connecting this pair as well as the line elements on sub-assembly

feeder lines that feed into the merge point.

The line elements that are contained within the aggregate line element associated with split-merge pair (n_s, n_m), called the descendants of the aggregate line element, can be characterized by the following set:

$$\mathcal{D}_{n_s,n_m} = \{n \in \mathbf{N} | \exists \pi(n, n_m) \in \Pi, \nexists \pi(n, n_s) \in \Pi\}. \tag{3.4}$$

Note that some of these line elements will be children of the aggregate line element while others will exist at a deeper level within the hierarchy under the aggregate line element. An example that clarifies the details of this hierarchy is given in Section 3.3.

We do not allow split-merge pairs to be interleaved with one another. In other words, for any two split-merge pairs, $(n_{s_1}, n_{m_1}), (n_{s_2}, n_{m_2})$, corresponding to aggregate line elements n_1 and n_2, respectively, either the aggregate line elements do not share any descendants or one of the aggregate line elements is wholly contained within the other.

Figure 3.1 shows an example of a network representation of a production line and identifies the various components of the network using the terminology introduced here. The key feature is that we decompose the line in a way that simplifies the problem to the serial case. The distinguishing difference between a sub-assembly feeder line and a sub-assembly split line is that the upstream end of a sub-assembly split line is restricted by the necessary coordination of all split lines emanating from its split point, whereas a sub-assembly feeder line has no such restriction. Further details about the constraints on feasible solutions are given in Section 3.3. For a full background on analytical models of manufacturing systems, the reader is referred to Gershwin (1994).

3.2.3 Notation for the Hierarchical End-State Problem Formulation

The decision variables to be determined are the last job to enter each line element, denoted j_n for line element n, as well as the last job to exit the most downstream line element with no parent. The vector of decisions is denoted by $\mathbf{j} \in \tilde{\mathbf{J}} \subset \mathbf{J}^{N+1}$, where the set $\tilde{\mathbf{J}}$ represents all decision vectors that satisfy the capacity and ordering constraints of the line (see Section 3.3 for a detailed discussion). The first N members of the decision vector are the decisions for each of the N line elements. The last component of the decision vector is the last job to exit the furthest downstream line element with no parent. Due to the special nature of this particular line element, we give it a special name and refer to it as the **terminal line element**, denoted by n_t. Recall that in the original formulation, the decision variables represented the last job to exit each line element. By modifying the definition, we obtain the benefit that decision j_n controls the content of line element n, rather than that of its successor line element, which was the case with the original formulation. For more details on the trade-offs when deciding between a job-based decision variable and a time-based decision variable, please refer to Section 2.2.2.

The content of line element n at any point in time is represented by an ordered list $r_n = (i_n^1, \ldots, i_n^m), m \leq m_n, n \in \mathbf{N}, i_n^1, \ldots, i_n^m \in \mathbf{J}$. Line element n has a set of goals, denoted $\mathbf{G}_n$, that we wish to satisfy. Note that it is entirely possible, in fact likely, that the goals associated with a given line element are mutually exclusive. For example, one goal could specify that a buffer needs to contain at least three of a particular type of job, while another goal could require that the same buffer be empty. The set $\mathbf{R}_n^g$ specifies content tuples that satisfy goal g of line element n. For each goal g in $\mathbf{G}_n$, a value v_g is awarded at shutdown if $r_n \in \mathbf{R}_n^g$, while a penalty p_g is assessed otherwise.

In addition to the rewards and penalties associated with goal satisfaction, a penalty is applied for each line element for which the shutdown time deviates from a pre-specified desired shutdown time, T_d. To formally define the shutdown time of a line element, we consider an example. Consider a line element $n \in \mathbf{N}$ with successor $n' \in \mathbf{N}$. Assume the decisions made at these two line elements are $j_n \in \mathbf{J}$ and $j_{n'} \in \mathbf{J}$, respectively. Define $e_{j,n}$ as the time at which job j enters line element n. The matrix $\{e_{j,n}, j \in \mathbf{J}, n \in \mathbf{N}\}$ is called the flow matrix. Notice the change in the definition of $e_{j,n}$ compared to Chapter II. The shutdown time of line element n, denoted T_n, can be written:

$$T_n = \max(e_{j_n,n}, e_{j_{n'},n'}), \tag{3.5}$$

where $e_{j_n,n}$ represents the time job j_n enters line element n. As this is the last job allowed to enter line element n, no more jobs enter line element n after this time. Similarly, $e_{j_{n'},n'}$ is the time job $j_{n'}$ enters line element n'. No more jobs exit line element n (not n') after this time. Thus, after time T_n, no jobs enter or exit line element n, and it is said to be shutdown at that time.

Time penalties are computed according to the difference between T_n and T_d for each line element. A penalty of p_o is applied for each unit of overtime required, while a penalty of p_l is applied for each unit of lost production time. In the original serial line formulation of this problem, shutdown time penalties were applied to the line as a whole, while in this chapter these penalties are applied on an individual line element basis. As alluded to earlier, and to be discussed in more detail in Section 3.3.3, this modeling change yields several significant benefits. For a given feasible shutdown policy, certain line elements may incur lost production penalties for stopping early, while other line elements may incur overtime penalties for stopping late. This model better reflects the reality of the line in general assembly where the assembly process

is far less automated than in the body shop. Most stations in general assembly are staffed by at least one worker (in contrast to the floating teams in the body shop).

3.2.4 Mathematical Programming Model

With the notational definitions established, the mathematical programming model of the non-serial End-State problem can be formulated. As mentioned previously, the components of the objective function are the rewards and penalties associated with satisfying or not satisfying goals, respectively, as well as penalties for either lost production or overtime.

$$
\begin{aligned}
(3.6)\ \max_{\mathbf{j}} \quad & \sum_{n\in\mathbf{N}}\left[\sum_{g\in\mathbf{G}_n}(v_g\mathbf{1}_{r_n\in\mathbf{R}_n^g}-p_g\mathbf{1}_{r_n\notin\mathbf{R}_n^g})-p_o(T_n-T_d)^+-p_l(T_n-T_d)^-\right] \\
\text{s.t.} \quad & (\{r_n, n\in\mathbf{N}\},\{T_n, n\in\mathbf{N}\}) = F(\mathbf{j}\in\tilde{\mathbf{J}}, T_{\max}) \\
& \mathbf{1}_{r_n\in R_n^g} = \begin{cases} 1, & r_n\in R_n^g \\ 0, & \text{o/w} \end{cases}, \ \forall\, n\in\mathbf{N} \\
& \mathbf{j}\in\tilde{\mathbf{J}}\subset\mathbf{J}^{N+1}.
\end{aligned}
$$

As in Chapter II, the function F represents a computational model of the production line, the inputs of which are the decisions, $\mathbf{j}\in\tilde{\mathbf{J}}$, and the maximum horizon length, $T_{\max}$. Internally, the computational model performs the recursive calculations necessary to construct the flow matrix, $\{e_{j,n}, j\in\mathbf{J}, n\in\mathbf{N}\}$. This flow matrix is used to determine the content of the line elements at shutdown as well as the actual shutdown time of the line elements, $\{T_n, n\in\mathbf{N}\}$. $T_{\max}$ is a hard upper bound on the shutdown time of the line.

We next introduce the dynamic programming formulation for the hierarchical End-State problem that can be used to find an optimal shutdown policy.

3.3 Dynamic Programming Model

In this section, we formulate the problem (3.6) of finding an optimal shutdown policy as a dynamic program. The general flow of this algorithm is to start by determining the last job to exit the terminal line element n_t, and then recursively work one's way up the line ("descending" into any aggregate line elements encountered) until the last job to enter each line element has been selected. To provide an intuitive understanding of the decomposition process, we consider an example before presenting the formal statement of the dynamic program.

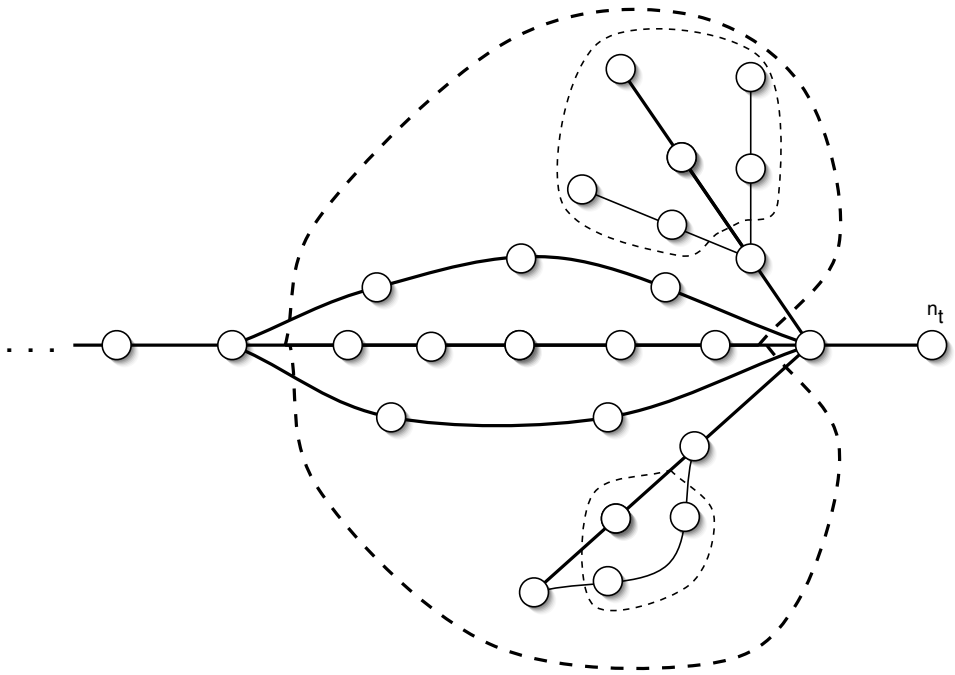

Figure 3.2: Example Production Line

Figure 3.2 illustrates an example of a complex segment of a production line. Notice that there is one top-level aggregate line element, while there are two aggregate line elements nested within this top-level aggregate line element. As mentioned in Section 3.2.2, the goal of decomposition is to simplify the problem to the serial line case for which we can easily find a solution with a DP algorithm similar to the one described

in Chapter II. The algorithm starts with the top-level line elements that have no parent, shown in Figure 3.3. Notice how the decomposition process simplified the problem to a serial line. The solid black circle represents the "collapsed" aggregate line element. The first decision made is the last job to exit the terminal line element. Starting with the terminal line element and proceeding upstream along the line, the last job to enter each element is selected. The restrictions on the decision at each line element will be described shortly. However, decisions made at this level of decomposition are insufficient to compute the objective function value for this line segment. Decisions must be made for the line elements within the aggregate line element, and the associated value computed.

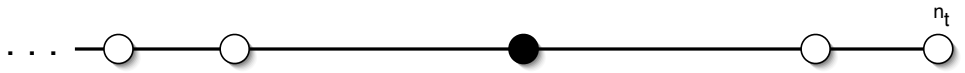

Figure 3.3: Example Production Line (Collapsed)

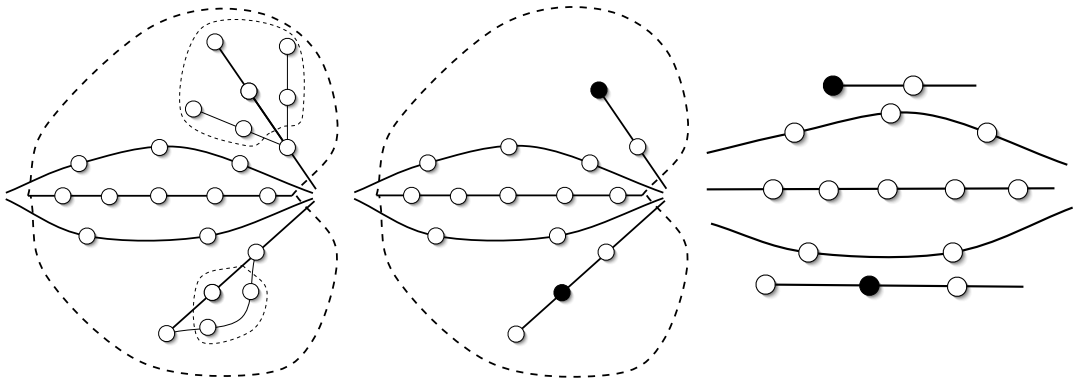

Figure 3.4: Decomposition of Top-level Aggregate Line Element

This brings the focus to the aggregate line element. Figure 3.4 shows the decomposition process for the top-level aggregate line element. Application of the decomposition procedure within the aggregate line element results in two aggregate

line elements **within** the top-level aggregate line element. The child aggregate line elements are denoted by the solid black circles. Notice how the children of the top-level aggregate line element form five serial lines: two sub-assembly feeder lines and three sub-assembly split lines. In addition, note that when the decisions for these child line elements are made, decisions have already been made at both the parent of these child line elements (the top-level aggregate line element) as well as the merge point that follows this aggregate line element. Another important observation is the fact that this process can continue to any finite depth: there is no theoretical restriction on the depth of the hierarchy.

To be able to make decisions at the child line elements, the restrictions on the feasible decisions must be established. The next two sections introduce the feasibility constraints for a sub-assembly feeder line and a sub-assembly split line. The complete statement of the dynamic programming formulation for the hierarchical end-state model follows the feasibility discussion.

3.3.1 Feasibility for Sub-assembly Feeder Lines

Figure 3.5 shows a segment of a line corresponding to a sub-assembly feeder line. For the purposes of the analysis here, refer to the first line element in the line segment as the head, while the last line element is called the tail. The set $\mathbf{N}_S$ contains the indices for the line elements on line segment S. For an arbitrary line element $n \in \mathbf{N}_S$, denote its successor by $n' \in \mathbf{N}$. The successor of the tail of the line segment is the merge point of the associated aggregate line element.

Since the line segment is upstream from the merge point, the decisions for the sub-assembly feeder line are determined after the merge point's decision has been made. Thus, we can view a sub-assembly feeder line as a serial line in which the last job to leave the line segment is fixed at the decision made by the merge point,

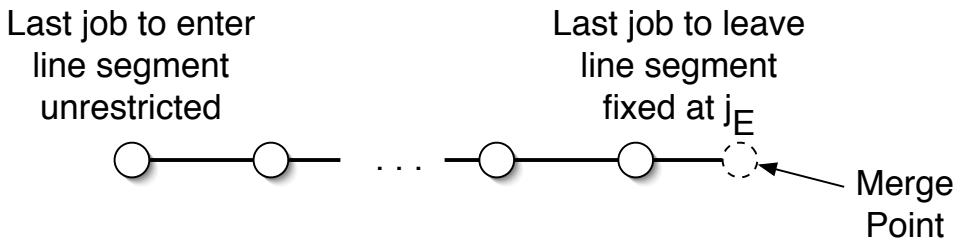

Figure 3.5: Sub-Assembly Feeder Line Segment

denoted j_E (the E stands for 'end'), and the last job to enter the line segment is only restricted by the capacity and ordering constraints of the line segment.

Similarly to the model in Section 2.3.1, when the decision at line element n's successor is given by $j_{n'}$, the feasible decisions at line element n are given by:

$$j_n \in \{j_{n'}, j_{n'} + 1, \ldots, j_{n'} + m_n\}, \tag{3.7}$$

where the merge point's decision is j_E. Note that the feasible set above differs slightly from that in Section 2.3.1 due to the fact that the decision variable represents the last job to enter a line element, whereas Chapter II had a decision variable of the last job to exit a line element.

Based upon the above, we can recursively compute the optimal shutdown plan for the sub-assembly feeder line segment, given the decision at the merge point, j_E, using the techniques of Chapter II.

3.3.2 Feasibility for Sub-Assembly Split Lines

The set of feasible decisions gets a bit more complicated when we consider sub-assembly split lines. As shown in Figure 3.6, now both the last job to exit the line segment and the last job to enter the line segment are fixed, with the latter denoted j_B (B is for 'beginning'). Let the set of line elements on line segment S that are upstream from line element n be denoted by $U_S(n)$. We now have additional capacity

constraints that we must satisfy to remain feasible:

$$j_B - j_n \leq \sum_{k \in U_S(n)} m_k \qquad n \in \mathbf{N}_S, \tag{3.8}$$

$$j_n - j_E \leq \sum_{k \notin U_S(n)} m_k \qquad n \in \mathbf{N}_S. \tag{3.9}$$

Intuitively, one can view the above as ensuring that no decision is made at the current line element n that would force an infeasible decision later as the algorithm progresses up the production line.

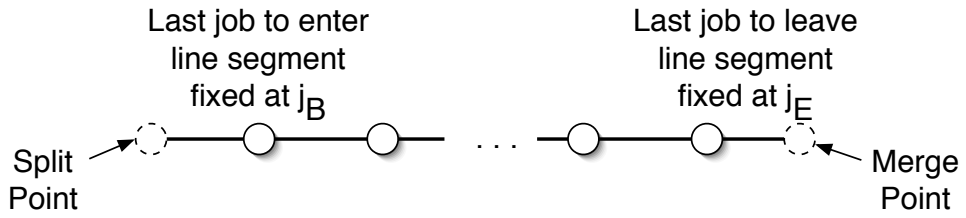

Figure 3.6: Sub-Assembly Split Line Segment

These constraints can be combined with the constraints from the sub-assembly line segment to yield:

$$\max(j_{n'}, j_B - \sum_{k \in U_S(n)} m_k) \leq j_n \leq \min(j_E + \sum_{k \notin U_S(n)} m_k, j_{n'} + m_n, j_B), \tag{3.10}$$

where, by definition, the decision at the head of line segment S must be j_B.

With the feasibility requirements for both sub-assembly feeder lines and sub-assembly split lines established, we can now consider the dynamic programming formulation to optimize the decisions throughout the production line.

3.3.3 Dynamic Programming Formulation

The above feasible decision analysis illuminates the information needed to determine the set of feasible decisions. Using the nomenclature of dynamic programming, this is referred to as the state of the dynamic programming model. As mentioned

earlier, the assessment of time penalties on an individual line element basis enables the removal of the maximum shutdown time from the state described in Section 2.3.3. The associated reduction of the state space due to the removal of the maximum shutdown time helps offset an increase in the state space due to the additional information added to support the hierarchical decomposition.

The state is given by the tuple (n, j, j_B, j_E). The state's first parameter, n, is the index of the line element for which we are making a decision. The next parameter, j, is the decision made at line element n's successor, and represents the last job to enter line element n's successor. Next is j_B, which is the last job to enter line element n's parent. And finally we have j_E, which represents the last job to exit line element n's parent, and is also the decision made at the merge point associated with the aggregate line element. The j_B and j_E terms in the state only come into play for sub-assembly split lines. In the case of sub-assembly feeder lines, $j = j_E$ for the state associated with the most downstream line element of the line segment and thus j_E does not need to be specified separately. Top-level line elements with no parent obviously can not have j_B or j_E specified. The top-level line elements form a serial line restricted on the end by the decision of the last job to exit the terminal line element. As with sub-assembly feeder lines, this restriction shows up in the state via the j parameter.

Given the above state, we must decide upon a value for j_n, the last job to enter line element n, for each line element n in $\mathbf{N}$, as well as the last job to exit the terminal line element, n_t. These decisions dictate the contents of line element $n \in \mathbf{N}$. Given the decision at line element n's successor, $j_{n'}$, and line element n, j_n, the content of line element n is specified by the following equation (where line element n is empty

if $j_n = j_{n'}$).

$$r_n = (j_{n'} + 1, \ldots, j_n). \tag{3.11}$$

Knowing the content of the line enables the determination of the rewards and penalties due to goals. For each of line element n's goals in G_n, we grant a reward of v_g if the content vector r_n is in R_n^g, and assess a penalty of p_g otherwise. Similarly, we assess time penalties based upon the shutdown time of line element n, T_n. If T_n exceeds the desired shutdown time, T_d, we assess an overtime penalty of $p_o(T_n - T_d)$. Otherwise, we assess a lost production penalty of $p_l(T_d - T_n)$.

The functional equation, which defines the value of making a specific decision at a given state and proceeding optimally thereafter is defined as follows:

(3.12)

$$f(n, j, j_B, j_E) = \begin{cases} \max_{j_n \text{ feasible}} & \sum_{g \in \mathbf{G}_n} v_g \mathbf{1}_{r_n \in \mathbf{R}_n^g} + p_g \mathbf{1}_{r_n \notin \mathbf{R}_n^g} & \text{goals} \\ & + \sum_{p \in \mathbf{P}_n} f(p, j_n, j_B, j_E) & \text{predecessors} \\ & + \sum_{c \in \widehat{\mathbf{C}}_n} f(c, j, j_n, j) & \text{terminal children} \\ & + p_o(T_n - T_d)^+ + p_l(T_n - T_d)^- & \text{time penalty,} \end{cases}$$

where $\mathbf{G}_n$, $\widehat{\mathbf{C}}_n$, and $\mathbf{P}_n$ represent the set of goals associated with line element n, the set of terminal children of line element n, and the set of predecessors of line element n, respectively (Terminal children are the children of an aggregate line element that are predecessors of the merge point of the aggregate line element).

The optimal solution is found by determining:

$$\max_{j \in \mathbf{J}} f(n_t, j, -, -), \tag{3.13}$$

where j represents the last job that will be allowed to exit the terminal line element. The dashes shown for the third and fourth parameter reflect the fact that these two parameters are not used by the top-level line elements with no parent.

A few observations are in order. First, this formulation lends itself very well to parallel computation. The optimal policy and value for each child line segment can be computed in parallel. Second, assigning a goal to a group of line elements is trivial. One could create an aggregate line element for the purpose of assigning a goal that spans several line elements. A final observation is that one could easily tune an implementation of this algorithm to prune unnecessary computations. If no children of an aggregate line element have any goals specified, for example, then the invocation of the functional equation for the children can be skipped, if one accounted for the time penalties within the aggregate line element.

3.3.4 Serial Line Special Case

This new formulation not only handles more complicated topologies, but in fact is faster when applied to a serial line than the formulation from Chapter II. Note that when we are dealing with a simple serial line, each line element has only a single predecessor and no children. In order to make a fair comparison between the two formulations, assume no line element has more than one goal associated with it. The earlier complexity calculation implicitly made this assumption. Thus, Equation (3.12) is the same as Equation (2.11), except that the former has an additional expression to compute the time penalty for the line element. The simplified version of the expression within the maximization in Equation (3.12) can be written (p denotes line element n's predecessor):

$$v_n \mathbf{1}_{r_n \in \mathbf{R}_n} + p_n \mathbf{1}_{r_n \notin \mathbf{R}_n} + f(p, j_n, -, -) + p_o(T_n - T_d)^+ + p_l(T_n - T_d)^-,$$

where we have used v_n and p_n instead of v_g and p_g since there is only one goal per line element. Similarly, we simplify $\mathbf{R}_n^g$ to $\mathbf{R}_n$. Due to the focus on a serial line, there are no aggregate line elements and the j_B and j_E terms can be ignored.

Computational Complexity of Serial Line Special Case

We now compare the computational requirements of this formulation when applied to a serial line to the original formulation from Section 2.3.3. The complexity of the expression inside the maximization in Equation (3.12) will be approximately the same as t_v, as defined in Section 2.3.3. The effort required to find the optimal decision for a state is bounded by $(2m_n+1)t_v$, following the same line of logic as in Section 2.3.3. The effort required to do the comparisons of the $f(n_t, \cdot, -, -)$ values is bounded by Jt_v. The values for n range from 1 to N, while the values for j range from 1 to J, leading to a bound on complexity given by:

$$\sum_{n=1}^{N} J(2m_n + 1)t_v + Jt_v \quad = \quad Jt_v(N(2\bar{m} + 1) + 1),$$

where, as with Section 2.3.3, $\bar{m}$ is the average capcity of the line elements.

With $N = 66$, $\bar{m} = 1.167$, and $J = 200$, we get:

$$Jt_v(N(2\bar{m} + 1) + 1) \approx 4.42e + 10^4 \cdot t_v.$$

Assume again that we have a machine with one GFLOPS capability. If t_v is 100 flops, the problem can be solved in less than one one-hundredth of a second. Even with a value for t_v of 1000, the time to solve is still less than one-tenth of a second. This algorithm performs dramatically better than the algorithm of Chapter II, where the upper bound on the computational time with $t_v = 1000$ was 3.5 minutes.

3.4 Computational Experiments

Having defined the hierarchical model for solving the End-State problem, we now evaluate the performance of the algorithm when applied to a realistic scenario. The scenario is based upon a real production line, and uses the capacities and cycletimes of actual line elements, but has been simplified to a pair of sub-assembly split lines that

are enclosed within a single aggregate line element. As in Chapter II, the values of the parameters are unitless, but were selected to preserve relative proportions among the parameters. Similarly, the discussion about the determination of parameter values from Section 2.4 applies here as well.

We first describe the scenario studied in detail, and subsequently present the experiments run along with their results.

3.4.1 Scenario Description

While Chapter II presented computational experiments related to the launch of a new vehicle, here we present a scenario based upon the shutdown of the production line at the end of the week. There are three main considerations that go into the end of week shutdown: (1) producing as much as possible prior to shutdown, (2) positioning the line to facilitate the work of weekend maintenance crews, and (3) ending in a state that ensures a smooth and fast start-up of the line at the beginning of the following week.

Producing as much output as possible prior to shutdown may seem like an obvious objective, but reducing the production rate of the line is a common strategy employed to ensure a smooth and successful shutdown. As an example, if a line typically produces sixty jobs per hour, a production line manager might reduce the production rate to fifty jobs per hour for the last two hours of production. In a truck plant, where each vehicle contributes approximately $10,000 profit, the 20 lost jobs each week for 45 weeks leads to a lost profit of nearly $10 million per year.

Positioning the line in a state that facilitates weekend maintenance work reduces the amount of overtime necessary to move jobs around so work can be initiated. In general, when a maintenance crew conducts work on a line element, at minimum the line element must be empty. If a crew arrives at a work site and finds a station

occupied by a job, the crew must radio for a forklift operator to remove the job. This process generally takes at least an hour, if not more. Further, this process can take several hours for cases where an area after the line element must be clear and the area before a line element must be full with a variety of job styles. Under the assumptions that a fully loaded maintenance worker costs $150 per hour, there are four maintenance workers per maintenance crew, ten different work sites will experience an average of 2.5 hours of delay any given week, this leads to over $500,000 in potential savings over a 45-week work year.

The third objective is to ensure that the start of production at the beginning of the following week can begin immediately and runs smoothly. Among the contributing factors to this objective is an even distribution of jobs throughout the line and no jobs in line elements that tend to be problematic. An example of this latter preference is the fact that we do not want to install a sunroof immediately upon the start of production. This operation is more error-prone than others and has the potential to disrupt production for an entire shift. If a slow start to the week causes a loss of 5 jobs per week on average, this leads to a lost profit of $2.25 million per year.

Having established the objectives we wish to achieve with the end of week shutdown, we now describe the specific scenario used for the computational experiments. The plant under consideration produces four different styles of vehicle based on the four combinations of regular cab versus extended cab, and regular truck bed versus extended truck bed. We assume each of the four combinations are equally likely when randomly generating a build schedule. The area under consideration, shown in Figure 3.7, is a line segment where the cab and truck bed are together coming out of the paint shop at the split point, and then are sent on two different paths for processing, before coming back together at the merge point. The merge point also

serves as the terminal line element.

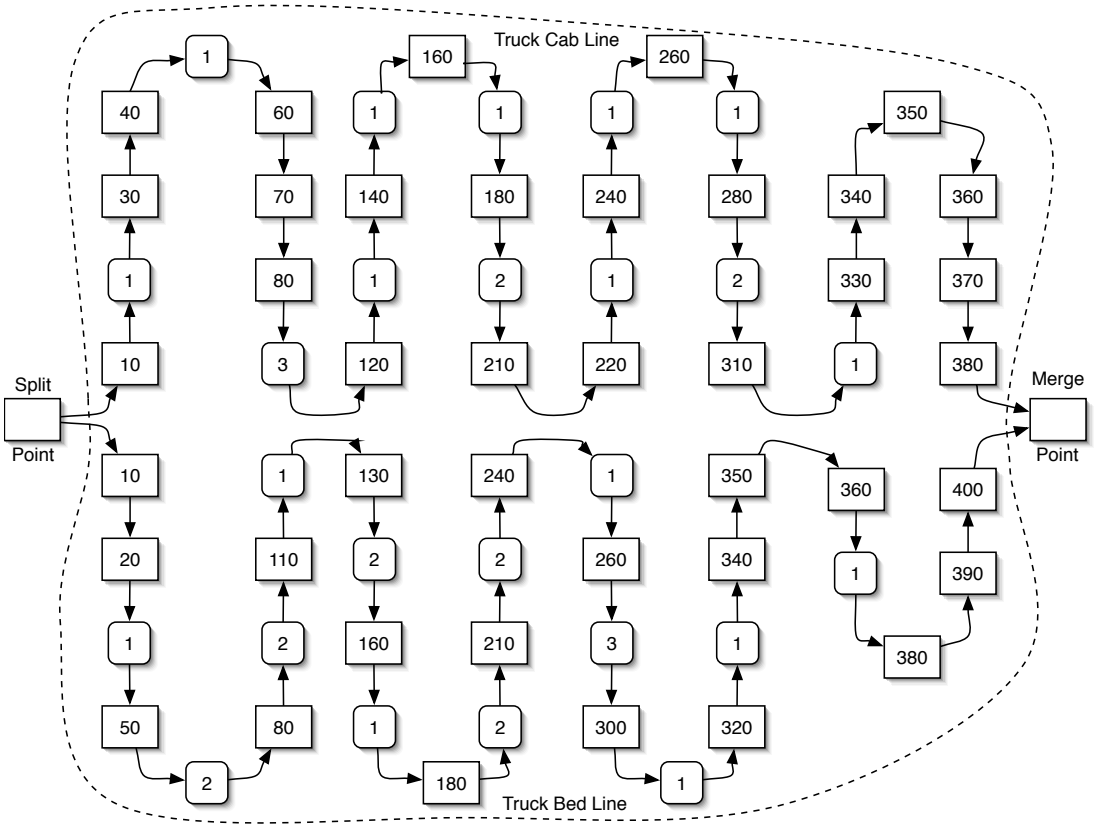

Figure 3.7: Production line topology for computational experiments scenario.

In Figure 3.7, rectangles with identification numbers represent stations (of capacity one) at which work is performed, while the rounded squares are buffers with the number inside indicating their capacities. As the time penalty is now computed on an individual line element basis, we divide the overtime and lost production penalties used in Chapter II by the number of line elements in the scenario used there, sixty-six. This yields a lost production penalty of 0.152 per minute and an overtime penalty of 0.0758 per minute. We again have three different categories of goals: high, medium, and low with rewards of 20, 5, and 1, and penalties of 7, 3, and 1, respectively.

The scenario under consideration consists of seven separate objectives, each of

which consists of multiple, potentially conflicting, goals:

1. To ensure that all line elements have work available at the start of production next week, we would like to fill every station and buffer with a job. This is a low priority goal.

2. Truck bed line stations 20, 50, 80, 130, 180, and 260 should be empty to allow the installation of new supply shelves at each of these stations. Each of these goals can be delayed since they do not directly impact production, but the completion of these jobs is expected to reduce errors as these new shelves have scanners that ensure the proper delivery of part supply bins. These are medium priority goals.

3. Truck bed line station 160 should have an extended bed for an ergonomic evaluation to be conducted by one of the plant's industrial engineers. The employees at this station have reported awkward shoulder movements and the ergonomist would like to explore alternative arrangements of the material in the station to prevent potential injuries. This is a high priority goal as employee health and safety is an overriding priority. If this goal is not accomplished via the execution of the shutdown, there is no way to manually achieve this goal as access to the station with a forklift is not practically possible. The buffers immediately before and immediately after this station should be empty to allow the ergonomist plenty of room to move around while conducting testing. These goals are of low priority because the work can still be completed even if these goals are not satisfied.

4. Truck bed line stations 300 and 320 need new tools installed to assist with the installation of wiring harnesses in the truck bed. The tools must then be

tested against both the regular truck bed and extended truck bed to ensure that everything works properly prior to the start of production on Monday. Each of these stations should contain a regular truck bed and their immediately preceding buffers should contain an extended truck bed for testing. These goals are high priority because they have the potential to impact the quality of product produced when the production line is started up. The next four line elements after these stations should be empty to allow for jobs to move through stations 300 and 320 after completing testing. These goals are medium priority as they could be completed manually if needed.

5. A new hydraulic lift is being installed for cab line station 350. This lift is used to hoist the instrument panel and carefully install it into the truck cab. The installation of this lift requires a forklift and the area from station 330 to 370 must be cleared to avoid damage to work-in-process inventory by the forklift. These goals are of medium priority because the jobs could be moved manually, if needed.

6. Cab line stations 140, 180, 210, 220, 240, and 260 need to have testing performed on extended cab jobs based on a recent modification to the body style. Having an extended cab job in each station is a medium priority goal.

7. Cab line stations 80 and 120 need to be emptied to enable access by a maintenance crew to the conveyor chain, a medium priority goal. To verify their work, the crew needs a job immediately prior to each of these two stations. These are medium priority goals.

To strike a balance between the time required to execute the optimization and the flexibility given to the optimization, we assume the optimization is performed

one hour prior to the desired shutdown. If the time at which the optimization is run is defined to be time zero, then T_d is 3600. We allow up to 30 minutes of overtime, implying that $T_{\max}$ is 5400. We allow for more overtime than in Section 2.4 due to the fact that this shutdown occurs at the end of the week. When the optimization is run at time zero, the line is assumed to be full. This initial state is similar to the state in which we would expect to find the line when executing the optimization in the midst of a shift as we are.

Having established the scenario, we can now describe the experiments run and their associated results.

3.4.2 Experimental Results

To test the performance of the optimization, we evaluated it against the same rule of thumb policy as in Chapter II. This rule of thumb seeks to stop the production line at the desired stopping time while obtaining as much value from goal satisfaction as possible within this constraint. In addition, we sought to determine what fraction of an upper bound on the objective function value the optimization was able to obtain for the given flow matrix. This value was determined by executing the optimizer with no time penalties.

The approach we took was to randomly generate 100 build schedules and apply each of the three techniques (rule of thumb, optimization, optimization with no time penalties) to each build schedule. We could then compare the results of the optimization against the other two strategies to see how it performs.

First we describe the method used to determine the value from the rule of thumb policy using the software framework. In this case, we set the time penalties sufficiently high to force the selection of a policy that shuts down every line element as close to the desired shutdown time as possible. The value of the rule of thumb

policy was then the accumulated value due to goal rewards and penalties, with the large time penalties being ignored as they were used for the purpose of forcing a shutdown at (or very close to) the desired shutdown time. For the purposes of the equations below, we denote the objective function value of the rule of thumb policy when applied to the i^{th} build schedule by r_i. Across 100 build schedules, the mean value of the rule of thumb policy was 226.5 (with a minimum of 199.0, a maximum of 255.0, and a standard deviation of 12.1).

The second approach used was to simply apply the end-state optimization algorithm to a given build schedule. In this case, we used the time penalties and goal rewards and penalties described earlier in this section. This is the algorithm we are evaluating and we would like to see it perform appreciably better than the rule of thumb, and as close as possible to the optimization with no time penalties approach. In the equations below, the objective function value of applying the end-state optimization algorithm to the i^{th} build schedule is denoted s_i. Using the same 100 build schedules as above, the mean value was 268.1 (with a minimum of 243.1, a maximum of 279.4, and a standard deviation of 6.3). Not only did the end-state optimization algorithm perform better than the rule of thumb, but it had a standard deviation that was nearly half that of the rule of thumb policy.

The final approach used, alluded to earlier, is the optimization with no time penalties. This approach enables the determination of the maximum value due to goals that could possibly be attained for a given build schedule. Since all the other components of the objective function reduce the associated value, this maximum value due to goals provides an upper bound on the maximum objective function value that could possibly be achieved for the given build schedule and any flow matrix. This objective function value is denoted by t_i below. For the given build schedule and

any flow matrix realization, this objective function value could be reduced by time penalties or by a reduction in goal value. The reduction in goal value could be due to using a policy that does not attain the maximum goal value or that gets adjusted due to running against $T_{\max}$. This approach was also applied to the 100 generated build schedules, yielding a mean value of 292.7 (with a minimum of 287.0, a maximum of 293.0, and a standard deviation of 0.9). The upper bound on the objective function value falls within a fairly tight range across the 100 build schedules.

The question we seek to answer with the computational experiments is how do the s_i values compare with the corresponding r_i and t_i values. To answer this question, we defined two separate metrics: the ratios r_i/s_i and s_i/t_i, and averaged these ratios over the 100 build schedules. The 100 build schedules were generated randomly with equal proportions of the four types of jobs and each of the three algorithms were executed against each of the build schedules. In other words, the same build schedule was used for one execution of each of the three algorithms. The first ratio, the fraction of the optimization's value the rule of thumb approach was able to achieve, had a mean of 84.5% (with a minimum of 74.1%, a maximum of 93.2%, and a standard deviation of 4.1%). The second ratio, the fraction of the objective function upper bound value we were able to obtain using the optimization for the given flow matrix, had a mean of 91.6% (with a minimum of 83.6%, a maximum of 95.3%, and a standard deviation of 2.1%).

The end-state optimization algorithm consistently outperformed the rule of thumb policy, and generally showed strong performance relative to the objective function upper bound for the given flow matrix. In addition, as mentioned earlier, the variance in value was highest under the rule of thumb policy. This policy not only performed worse than the end-state optimizer policy, but had greater variation in performance

as well.

3.5 Conclusions

In this chapter, we introduced a new modeling approach for the End-State optimization problem that expanded the line topologies that can be handled and also showed dramatic improvement in terms of the computational requirements of the dynamic programming algorithm when applied to a serial line. Previously, the end-state optimizer was only able to handle serial lines. With the introduction of the novel hierarchical approach to modeling the production line, we can now handle assembly splits and merges as well as sub-assembly line merges, common topological structures of automotive manufacturing lines. Through a change to the way we assess time penalties, we removed the maximum shutdown time from the dynamic programming state.

The next main extension for the end-state optimization model and algorithm is to incorporate the ability to better handle stochasticity. Machine breakdowns and uncertain repair times are a fact of life within the complicated environment of automotive manufacturing. The ability to better handle this environment during the shutdown process would be a tremendous advance. This topic is covered in the next chapter.

CHAPTER IV

A Learning-based Approach to the Stochastic End-State Problem

4.1 Introduction

Having achieved positive results from both the simple dynamic program applied to a serial line as well as the more complicated hierarchical extension of the simple dynamic program, we now move on to exploring ways to explicitly handle stochasticity in the algorithm. We previously addressed stochasticity by using a deterministic algorithm to find an open-loop policy and applying it within a stochastic setting. Now we attempt to integrate information about stochasticity into the policy development process and compare the performance of the resulting policy against the policies from both the deterministic algorithm as well as a rolling horizon optimization that uses the deterministic algorithm.

Unfortunately, the explicit consideration of stochasticity dramatically complicates what previously was a reasonably straightforward model. One of the major complications is that the policy should be time dependent, due to the fact that the production line evolves in a non-deterministic manner over time. While finding an optimal policy proves intractable, we explore two different approaches for finding heuristic policies: rolling horizon optimization and an objective function approximation technique that uses an extension of Sampled Fictitious Play.

This chapter makes the following contributions:

- Explores a rolling horizon approach that periodically updates the shutdown policy in effect and is similar to the strategy used by an actual plant manager.
- Develops an objective function approximation approach that uses Sampled Fictitious Play (SFP) and identifies a shutdown policy that is better in the face of stochasticity than a policy yielded by the simple deterministic dynamic program.
- Demonstrates how this SFP approach has wider applicability as a method of intelligently searching a space of mathematical models to identify one that is best suited for a particular problem instance.
- Applies the SFP search algorithm to a representation of a production line at General Motors and demonstrates performance nearing that of a rolling horizon approach with far more information at its disposal.

Fictitious Play, introduced by Brown (1951) and Robinson (1951), was originally developed as a technique for finding the Nash equilibria of finite, two-person non-cooperative games. While Shapley (1964) showed that there exist problems for which fictitious play will not converge, Monderer and Shapley (1996) proved that fictitious play converges under the condition that the players share a common objective function. Building upon this result, Lambert III et al. (2005) developed a sampled version of the fictitious play algorithm that possesses specific convergence properties while reducing the computational requirements of each iteration of the algorithm.

In Chapters II and III, we introduced two flavors of the End-State optimization problem in a deterministic setting and developed two deterministic algorithms for finding their optimal shutdown policies. These algorithms performed appreciably

better than a rule-of-thumb policy modeled after shutdown strategies followed by plant managers, but still left some value on the table when compared to a policy with hindsight in a stochastic setting.

In this chapter, we merge the deterministic algorithms of the previous two chapters with Sampled Fictitious Play to develop a method for finding shutdown policies that maximize a fictional manager's objective function. Due to the difficulty in directly optimizing this manager's objective function, we develop an approximation that makes use of the speed with which we can solve the deterministic DP. Throughout this chapter, we use the model of Chapter III, (3.6). However, the approach is also applicable to the model of Chapter II, (2.1).

Section 4.2 describes the challenges associated with incorporating stochasticity into the mathematical model and why finding an optimal policy poses such difficulties. The following two sections present two different heuristic approaches to the End-State problem that make use of the ability to quickly solve the deterministic dynamic program. Section 4.3 presents a rolling horizon optimization approach to dealing with stochasticity that is based upon techniques frequently applied in practice. Section 4.4 introduces an objective function approximation technique that attempts to modify the parameters used by the deterministic algorithm to find a shutdown policy that performs better in a stochastic environment than the original policy. Computational experiments, presented in Section 4.5, compare the performance of the two algorithms when applied to a realistic setting. The last section concludes with a summary of the contributions and suggestions for future work.

4.2 Challenges of a Stochastic Model

In this section, we discuss the challenges of incorporating stochasticity into the mathematical model. In a manufacturing plant, stochasticity is experienced in two ways: (1) line element breakdowns, and (2) uncertain repair times. Throughout the discussion, we assume the cycles between failures of each line element and the times to repair are independent exponential random variables. We first discuss why modeling the problem as a sequential decision problem makes sense and then what makes this modeling approach so complicated.

4.2.1 Modeling Approach Necessitated by Stochasticity

The introduction of stochasticity changes the End-State problem such that an optimal solution to the problem must be based upon a sequential decision process. If such an approach were not taken, as when a policy is determined at the beginning of the time horizon and applied until the production line is shutdown, no guarantee can be made about optimality. As time progresses and random events occur, what was once an optimal policy based upon a deterministic view of the world may quickly be rendered obsolete. To solve to optimality, whatever algorithm is developed must in some way have the ability to develop "closed-loop" policies that evolve over time as random events occur.

A typical approach for sequential decision problems is to identify a state of the system that exhibits the Markov property. In other words, if we know the current state of the system, we need not know how the system evolved to that state. An attempt to identify this collection of information for the stochastic End-State problem reveals its dramatic complexity. First, one would need to know the condition of each line element: up, down, or stopped. "Up" means a line element is able to perform

work, "down" means a line element is broken and being repaired, and "stopped" means that a line element can no longer accept jobs from upstream. If we did not know this information, we would not be able to determine whether a line element was eligible to be stopped or not, nor could we differentiate between an empty up line element and a down line element. Note that a line element is shutdown once both it and its successor are both stopped.

The next piece of information required would be the content of the line elements, as well as the remaining cycletime for any line elements with work-in-process (WIP). Collectively, we refer to these two pieces of information as the status of the production line. As with the earlier deterministic algorithms, we would need to know the content of the line elements to evaluate the value due to goals of the current state. The remaining cycletime is a necessary piece of information due to the fact the cycletimes are deterministic values rather than memoryless random variables like the cycles between failures and the times to repair. If we did not have information about the remaining cycletime for jobs in-process, we would not be able to compute the probabilities associated with the time until an up machine breaks down.

The final piece of information necessary for the state of the sequential decision problem is the current time. The time is necessary to compute the penalties associated with lost production or overtime. In the case of the deterministic algorithms, we did not explicitly have time in the dynamic programming state. However, this information was unnecessary as we could look up the shutdown time of a line element based upon its index and the decision made at that line element. Once we incorporate stochasticity, we no longer have this luxury as the flow matrix, the matrix of times at which each job enters each line element, is now random.

In summary, the state would need to have three main components: time, line

condition, and line status. We next discuss why such a defined state would prove computationally intractable.

4.2.2 The Computational Challenges of a Sequential Decision Model

The example production lines considered in Chapters II and III are small portions of an entire production line, but would still require intractably large state spaces with a state as defined above. Consider a production line with 50 line elements, each with a cycletime of 60 seconds. Further, suppose the portion of the build schedule under consideration has 200 jobs. Also assume that time has been discretized and the horizon is 3600 seconds (1 hour). Under these very conservative assumptions, we would find the number of states to be exceedingly large:

$$3600 * 3^{50} * 60^{50} \geq 10^{100}, \tag{4.1}$$

where the above expression does not even include a factor for the contents of the line elements.

While the above results are discouraging for finding an optimal solution to the stochastic End-State problem, we can explore heuristic alternatives. In the following two sections, we consider two different heuristic approaches that take advantage of the fact that we can solve the deterministic problem (3.6) via an extremely efficient dynamic programming approach. The policy that results from applying the dynamic programming solver of Chapter III while using the flow matrix constructed under the assumption that no breakdowns or repairs occur is called the **deterministic dynamic programming policy**. The associated flow matrix, called the **deterministic flow matrix**, is denoted by z' in the following sections. Denote by $h(\mathbf{j}, z')$ the objective function value of (3.6) if policy $\mathbf{j}$ is applied to flow matrix z' (The flow matrix impacts the specifics of function $F(\mathbf{j}, T_{\max})$). The policy yielded by the

deterministic dynamic program is then:

$$\operatorname*{argmax}_{\mathbf{j}} h(\mathbf{j}, z'). \tag{4.2}$$

We first consider a rolling horizon optimization algorithm and then introduce an objective function approximation technique backed by an extension to the existing version of Sampled Fictitious Play.

4.3 Rolling Horizon Optimization

In this section we present a rolling horizon optimization approach to deal with a stochastic environment. This technique simply applies the deterministic solver periodically to update the current shutdown policy. This technique is quite similar to what one would find in practice.

4.3.1 Description of Rolling Horizon Optimization Procedure

Rolling horizon optimization is a multi-period, finite horizon technique for identifying a heuristic policy for a sequential decision process. In each time period, the planner solves a T-period horizon problem, and uses the action from the first time period of this policy as the action to use in the current time period.

Rolling horizon procedures have been used to generate approximate solutions to infinite horizon problems (cf. Alden and Smith, 1992), but we use them here in a finite horizon problem as a way to respond to the stochastic evolution of a production line. Our deterministic dynamic program is used to solve for a shutdown policy at the beginning of each time interval of length T. During the time intervals between successive optimizations, we use the most recent policy as the shutdown policy. If, during the interval, the policy calls for stopping a line element, we then carry out that decision and that particular line element being stopped becomes a hard constraint for any future optimizations that we do.

4.3.2 Rolling Horizon as Proxy for a Plant Manager

The rolling horizon optimization procedure, as described above, bears similarity to the method used by plant managers to shutdown a production line. Typically, a plant manager will view the locations of jobs, the buffer levels, and any current breakdowns, and develop a shutdown policy based on experience. As time passes, other line elements will break down and some broken line elements will be fixed. The plant manager receives information about these events, and updates the shutdown plan. The rolling horizon procedure mimics this idea of periodic updates to the shutdown policy, while having the added advantage of using the deterministic dynamic program to develop this policy.

4.4 Objective Function Approximation

In this section, we present a heuristic approach to the End-State problem that relies upon a technique for approximating a fictional plant manager's objective function. This approach takes advantage of the fast deterministic DP solver to find a policy that performs better within a stochastic environment than the original deterministic DP policy.

4.4.1 Parameterized Objective Function

The parameters of the objective function of (3.6) — the lost production and overtime penalty parameters as well as the goal reward and penalty parameters — capture a manager's preferences. We can extend the objective function notation to include this parameter vector. Denote the objective function of a manager with parameter vector p by $h_p(\mathbf{j}, Z)$, where $\mathbf{j}$ is a shutdown policy and Z is a random flow matrix. Suppose the manager in charge of developing a shutdown schedule has the parameter vector $\bar{p}$. This manager would like to select a shutdown policy that

maximizes the expected value of the objective function with parameter vector $\bar{p}$. More formally, this specific manager's objective function is:

$$g(\mathbf{j}) = E[h_{\bar{p}}(\mathbf{j}, Z)]. \tag{4.3}$$

4.4.2 Objective Function Approximation

While we can quickly solve the deterministic DP, this policy may not be the optimal policy for the manager's objective function (4.3). Ideally, the manager would like to find the policy $\mathbf{j}$ that maximizes $g(\mathbf{j})$. However, finding an optimal policy is difficult due to the expectation in expression (4.3) as well as the size of the vector $\mathbf{j}$. However, we can obtain a policy by solving the deterministic DP with the objective function parameterized by parameter vector p (which may be different from $\bar{p}$):

$$\mathbf{j}(p) = \underset{\mathbf{j}}{\operatorname{argmax}}\, h_p(\mathbf{j}, z'). \tag{4.4}$$

Here h_p has an additive form as presented in (3.6), so we can apply the deterministic DP developed in Chapter III to find $\mathbf{j}(p)$.

The solution from the deterministic DP may not be optimal for the manager's objective function, but we hope to find a parameter vector p that yields a corresponding policy $\mathbf{j}(p)$ that comes as close as possible to maximizing the manager's original objective function:

$$g(\mathbf{j}(p)) = g(\underset{\mathbf{j}}{\operatorname{argmax}}\, h_p(\mathbf{j}, z')) \approx g(\underset{\mathbf{j}}{\operatorname{argmax}}\, g(\mathbf{j})) = \max_{\mathbf{j}} g(\mathbf{j}). \tag{4.5}$$

Putting the above together, we seek the following p^*:

$$p^* = \underset{p}{\operatorname{argmax}}\, g(\mathbf{j}(p)) = \underset{p}{\operatorname{argmax}}\, E[h_{\bar{p}}(\mathbf{j}(p), Z)]. \tag{4.6}$$

To solve this problem, we need an algorithm that can approximate the expectation and that makes use of the ability to quickly solve for $\mathbf{j}(p)$. Through an extension of the

existing Sampled Fictitious Play algorithm, we are able to replace the expectation with a sample average and take advantage of the speed of finding $\mathbf{j}(p)$. We next present background on Sampled Fictitious Play followed by the extension of Sampled Fictitious Play to accommodate the expectation in the objective function.

4.4.3 Sampled Fictitious Play Background

We provide background on Sampled Fictitious Play following the notational conventions of Lambert III et al. (2005). Fictitious Play originates with the work of Brown (1951) and Robinson (1951). Consider a game in strategic form with the set of players $N = \{1, 2, \ldots, n\}$, where each player has a finite set of strategies, denoted $\mathcal{Y}^i$ for player $i \in N$. Let $\mathcal{Y} = \mathcal{Y}^1 \times \mathcal{Y}^2 \times \cdots \times \mathcal{Y}^n$. The utility function of player $i \in N$ is $u^i : \mathcal{Y} \to \mathbb{R}$.

The idea of mixed strategies is one that appears frequently within game theory literature. For an arbitrary player i, a **mixed strategy** is a probability distribution over the strategies in $\mathcal{Y}^i$. We can formally denote the set of mixed strategies for player $i \in N$ by Δ^i:

$$\Delta^i = \left\{ f^i : \mathcal{Y}^i \to [0,1] : \sum_{y^i \in \mathcal{Y}^i} f^i(y^i) = 1 \right\}. \tag{4.7}$$

Each $f^i \in \Delta^i$ assigns a non-negative probability to each and every element of $\mathcal{Y}^i$. Set $\Delta = \Delta^1 \times \Delta^2 \times \cdots \times \Delta^n$.

Having established the idea of a mixed strategy, we need to extend the definition of player i's utility function, $i \in N$. Consider $u^i : \Delta \to \mathbb{R}$:

$$\begin{aligned} u^i(f) &= u^i(f^1, f^2, \ldots, f^n) \\ &= \sum_{y \in \mathcal{Y}} u^i(y^1, y^2, \ldots, y^n) f^1(y^1) f^2(y^2) \cdots f^n(y^n). \end{aligned} \tag{4.8}$$

This extended definition of u^i has the intuitive interpretation of being the expected

utility of player i when players select strategies in accordance with the probability distributions specified by mixed strategies $f^1, f^2, \ldots, f^n$. Implicit in the construction of Equation (4.8) is the fact that players choose strategies independently.

To support the upcoming convergence analysis, we introduce several equilibrium and convergence concepts. For a belief vector $g \in \Delta$ and $\varepsilon \geq 0$, g is said to be an **ε-equilibrium** if for each $i \in N$:

$$u^i(g) \geq u^i(f^i, g^{-i}) - \varepsilon \qquad \forall f^i \in \Delta^i,$$

where $(f^i, g^{-i}) = (g^1, \ldots, g^{i-1}, f^i, g^{i+1}, \ldots, g^n)$. A **Nash equilibrium** is a special case with $\varepsilon = 0$, and is henceforth simply referred to as an equilibrium. Denote the set of all ε-equilibria by K_ε, and the set of all equilibria of the game Γ by K.

To be able to discuss the proximity of two belief vectors to one another, we must select a norm on Δ. Denote the Euclidean norm on Δ by $\|\cdot\|$. The set of all belief vectors within a distance $\delta > 0$ of at least one equilibrium of Γ is defined as follows:

$$B_\delta(K) = \{g \in \Delta : \min_{f \in K} \|g - f\| < \delta\}.$$

We can now consider a sequence of belief vectors, $(f(t))_{t=1}^\infty \in \Delta$, also referred to as a **belief path**, and define a notion of convergence for this sequence. A belief path $(f(t))_{t=1}^\infty$ is said to **converge to equilibrium** if for every $\delta > 0$, there exists an integer T such that $f(t) \in B_\delta(K)$ for all $t \geq T$.

We can draw a connection between a sequence in $\mathcal{Y}$ and a belief path. Define a path in $\mathcal{Y}$ as a sequence of elements of $\mathcal{Y}$, $(y(t))_{t=1}^\infty$. We can associate a belief path $(f_y(t))_{t=1}^\infty$ with a path $(y(t))_{t=1}^\infty$ as follows:

$$f_y(t) = \frac{1}{t} \sum_{s=1}^{t} y(s) \qquad \forall t \geq 1, \tag{4.9}$$

where the $y(s)$ vectors should be interpreted as members of Δ. A path $(y(t))_{t=1}^{\infty}$ is said to **converge in beliefs to equilibrium** if the associated belief path $(f_y(t))_{t=1}^{\infty}$ defined by Equation (4.9) converges to equilibrium.

The final concept we must introduce is that of a fictitious play process. Intuitively, a path $(y(t))_{t=1}^{\infty}$ is a **fictitious play process** if for every $t \geq 1$, $y^i(t+1)$ is a best response of player i to the mixed strategies of the other players. We now formally characterize a fictitious play process. For $i \in N$ and $f \in \Delta$, define:

$$v^i(f) = \max\{u^i(g^i, f^{-i}) : g^i \in \Delta^i\}.$$

Thus, $v^i(f)$ is the payoff of player i's best response to the strategies of the other players, f^{-i}. We then define a path $(y(t))_{t=1}^{\infty}$ to be a fictitious play process if for every $i \in N$:

$$u^i(y^i(t+1), f_y^{-i}(t)) = v^i(f_y(t)) \text{ for every } t \geq 1. \tag{4.10}$$

Games of Identical Interests

We now sharpen our focus to specifically consider games of identical interests. This class of games is characterized by the fact that all players share the same utility function, $u^1 = u^2 = \cdots = u^n = u$. Every fictitious play process of a finite game with identical interests converges to equilibrium (Monderer and Shapley, 1996). A formal statement of the fictitious play algorithm can then be formulated as follows:

Fictitious Play Algorithm (Lambert III et al., 2005)

Initialization: Set $t = 1$ and select $y(1) \in \mathcal{Y} = \mathcal{Y}^1 \times \mathcal{Y}^2 \times \ldots \times \mathcal{Y}^n$ arbitrarily; set $f_y(1) = y(1)$.

Iteration $t \geq 1$**:** Given $f_y(t)$, find

$$y^i(t+1) \in \underset{y^i \in \mathcal{Y}^i}{\operatorname{argmax}}\, u^i(y^i, f_y^{-i}(t)),\ i = 1, \ldots, n. \tag{4.11}$$

Set $f_y(t+1) = f_y(t) + \frac{1}{t+1}(y(t+1) - f_y(t))$ and increment t by 1.

The primary problem with the above algorithm is that it is likely to be too computationally burdensome. The high computational requirements of the above algorithm motivated work on a sampled version of the fictitious play algorithm. The idea behind the sampled version of fictitious play is that players best respond to a sample from players' histories.

The Fictitious Play Algorithm is updated as follows to incorporate sampling:

Sampled Fictitious Play Algorithm (Lambert III et al., 2005)

Initialization: Set $t = 1$ and select $y(1) \in \mathcal{Y} = \mathcal{Y}^1 \times \mathcal{Y}^2 \times \ldots \times \mathcal{Y}^n$ arbitrarily; set $f_y(1) = y(1)$.

Iteration $t \geq 1$**:** Given $f_y(t)$, select a sample size $k_t \geq 1$, and draw i.i.d. random samples $Y_j(t),\ j = 1, \ldots, k_t$, from the distribution given by $f_y(t)$. Using the above samples, find

$$y^i(t+1) \in \underset{y^i \in \mathcal{Y}^i}{\operatorname{argmax}} \{\bar{u}^i_{k_t}(y^i, f_y^{-i}(t))\},\ i = 1, \ldots, n, \tag{4.12}$$

where $\bar{u}^i_{k_t}(y^i, f_y^{-i}(t))$ is the realization of $\bar{U}^i_{k_t}(y^i, f_y^{-i}(t))$ as defined by (16) in (Lambert III et al., 2005). Set $f_y(t+1) = f_y(t) + \frac{1}{t+1}(y(t+1) - f_y(t))$, increment t by 1 .

One of the primary results from Lambert III et al. (2005) in which we are interested is the convergence of a sampled fictitious play process:

Theorem 5 (Lambert III et al., 2005). *Let Γ be a finite game in strategic form with identical payoffs. Then, any sampled fictitious play process y(t) with sample sizes $k_t = \lceil Ct^\beta \rceil$ for $\beta > 1/2$ and $C > 0$ converges in beliefs to equilibrium with probability 1.*

We extend both the sampled fictitious play algorithm and the corresponding convergence proof for the case where there is an additional non-optimizing "Nature" player that simply selects a strategy at random according to a given probability distribution. We first discuss how the presence of this additional player changes the objective function of the optimization problem. Then, we develop an updated sampled fictitious play algorithm that includes the Nature player. Finally, we present a new convergence result for sampled fictitious play processes that include a Nature player.

4.4.4 Extension to Sampled Fictitious Play

The addition of the Nature player changes the objective function from a simple utility function to an expectation. If we denote the Nature player's strategy by the random element Z with the cumulative distribution function G, the optimization problem we are now trying to solve is of the form:

$$\max\{E_Z[u(y, Z)] : y = (y^1, y^2, \ldots, y^n) \in \mathcal{Y}^1 \times \mathcal{Y}^2 \times \cdots \times \mathcal{Y}^n, Z \sim G\}. \tag{4.13}$$

One can view the Nature player as introducing "noise" to the objective function that leads to errors in the best response computations. In the first set of computational experiments, presented in the next section, an outcome of the Nature random element is a randomly generated flow matrix.

We follow a similar approach to that taken in Lambert III et al. (2005) and write the best response computation of the fictitious play algorithm as an expected value, which we then replace by a sample average computation. To write the best response computation as an expected value, we first recognize that each element of the belief vector $f_y(t)$ can be interpreted as a probability distribution over the finite set of strategies of one of the non-Nature players. The best response computation,

Equation (4.11), can then be written:

$$\max_{y^i \in \mathcal{Y}^i} \{E_{Y^{-i},Z}(u(y^i, Y^{-i}, Z))\},$$

where Y^{-i} is a random vector whose components, $Y^j,\ j \neq i$, have probability distribution described by $f_y^j(t)$, respectively, and Z is a random element with probability distribution G.

We write the function $\bar{U}_k^i(\cdot, f_y^{-i}(t), G) : \mathcal{Y}^i \to \mathbb{R}$ to be used as a sample average to replace the expectation calculation:

$$\bar{U}_k^i(y^i, f_y^{-i}(t), G) = \sum_{j=1}^{k} \frac{u^i(y^i, Y_j^{-i}(t), Z_j)}{k}, \tag{4.14}$$

where $Y_j^{-i}(t),\ j = 1, \ldots, k$, are i.i.d. random vectors with the distribution given by $f_y^{-i}(t)$ and $Z_j,\ j = 1, \ldots, k$ are i.i.d. random elements with distribution given by G. The expression $\bar{U}_k^i(y^i, f_y^{-i}(t), G)$ has the intuitive meaning of being the sample average of player i's utility when employing strategy y^i.

We proceed towards updating the Sampled Fictitious Play Algorithm to include the Nature player. The modification to the algorithm entails sampling from Nature's distribution in addition to sampling from $f_y(t)$. For a given iteration t, we have a given vector of beliefs $f_y(t) \in \Delta$. We draw an independent sample of size $k_t \geq 1$ for both the players and Nature. The players' sample, the elements of which are denoted $Y_j(t),\ j = 1, \ldots, k_t$, are drawn from the distribution given by $f_y(t)$. Nature's sample, denoted $Z_j, j = 1, \ldots, k_t$, is drawn from distribution G.

Given the random sample of size k_t, we can move on to describe how the best response is computed. Define $\bar{u}_{k_t}^i(y^i, f_y^{-i}(t), G)$ to be a realization of $\bar{U}_{k_t}^i(y^i, f_y^{-i}(t), G)$ for $i = 1, \ldots, n$. Players select a best response to the sample by identifying the strategy that maximizes their sample average (as opposed to an expectation). More formally, player i's best response, denoted $y^i(t+1)$ is chosen such that $y^i(t+1) \in$

$\text{argmax}\{\bar{u}^i_{k_t}(y^i, f_y^{-i}(t), G) : y^i \in \mathcal{Y}^i\}$. The belief vector, $f_y(t)$, is updated with each player's best response. One of the key advantages of the sampled version of fictitious play over traditional fictitious play is that the mixed strategy that results from the random sample will have a limited number of positive elements. This reduces the computational burden of the best reply computations.

We can now present the formal statement of the updated Sampled Fictitious Play Algorithm to include Nature:

Sampled Fictitious Play With Nature Algorithm

Initialization: Set $t = 1$ and select $y(1) \in \mathcal{Y} = \mathcal{Y}^1 \times \mathcal{Y}^2 \times \cdots \times \mathcal{Y}^n$ arbitrarily; set $f_y(1) = y(1)$.

Iteration $t \geq 1$**:** Given $f_y(t)$ and G, select a sample size $k_t \geq 1$, and draw a random sample $Y_j(t),\ j = 1, \ldots, k_t$, from the distribution given by $f_y(t)$, and $z_j,\ j = 1, \ldots, k_t$, from the distribution given by G. Using the above sample, find

$$y^i(t+1) \in \underset{y^i \in \mathcal{Y}^i}{\text{argmax}}\{\bar{u}^i_{k_t}(y^i, f_y^{-i}(t), G)\},\ i = 1, \ldots, n, \tag{4.15}$$

where $\bar{u}^i_{k_t}(y^i, f_y^{-i}(t), G)$ is the realization of $\bar{U}^i_{k_t}(y^i, f_y^{-i}(t), G)$ as defined by (4.14). Set $f_y(t+1) = f_y(t) + \frac{1}{t+1}(y(t+1) - f_y(t))$, increment t by 1 .

The sequence $(y(t))_{t=1}^{\infty}$ is a stochastic process, which we call a **sampled fictitious play process with Nature**. We denote the associated stochastic belief process $(F_y(t))_{t=1}^{\infty}$. Recall that in the description of iteration t of the Sampled Fictitious Play with Nature Algorithm, we stated that $f_y(t) \in \Delta$ was given. In reality $f_y(t)$ will be a result of a partial realization of the sequence $(y(t))_{t=1}^{\infty}$. Thus, the sample mean computed in the best response calculation, Equation (4.15), is in fact conditional upon $F_y^{-i}(t) = f_y^{-i}(t)$.

With the updated algorithm established, we can now move on to the proof of convergence of a sampled fictitious play process with Nature. As with the theorem in Lambert III et al. (2005), we can similarly guarantee convergence of a sampled fictitious play process with Nature provided the sample sizes grow sufficiently. Specifically, we require that $k_t = \lceil Ct^\beta \rceil$ with $\beta > \frac{1}{2}$ and $C > 0$, where $\lceil x \rceil$ is the smallest integer greater than or equal to x.

Theorem IV.1. *Let Γ be a finite game in strategic form with identical payoffs and assume that $\sup_{f\in\Delta, z\in\Omega} |u(f,z)|$ is finite, where Ω is the sample space of the random element Z. Then any sampled fictitious play process with Nature $y(t)$ with sample sizes $k_t = \lceil Ct^\beta \rceil$ for $\beta > \frac{1}{2}$ and $C > 0$ converges in beliefs to equilibrium with probability 1.*

Proof: See Appendix B.

An astute reader may notice that due to the increasing size of the sample taken, the number of elements of the mixed strategy associated with the sample that are positive may increase over time. However, as $f_y(t)$ tends toward its limiting distribution, this number should stabilize and tend toward the number of positive components of the limiting equilibrium strategy.

The reader should observe the similarity in structure between Equations (4.6) and (4.13). Each parameter of the parameterized objective function can be viewed as an optimizing player while the random flow matrix is the lone non-optimizing Nature player. An action of a parameter player is a specific parameter value, while an action of the Nature player is a realization of a flow matrix.

To evaluate the performance of both the rolling horizon and the objective function approximation approaches, we now evaluate each in a series of computational experiments.

4.5 Experiments

In this section, we conduct computational experiments and compare the performance of the policy yielded by Sampled Fictitious Play with nature against the rolling horizon optimization procedure and the deterministic dynamic program algorithm. We return to the serial line of Chapter II to focus on stochasticity. Recall, however, that the mathematical model of Chapter III is used in this chapter. Due to the fact that we are dealing with a serial line, the mathematical model would simplify as described in Section 3.3.4.

In these experiments, we fixed the value of the overtime penalty, p_o, at 1 and adjusted the other parameter values accordingly so that the parameters have the same relative values as in Chapter II. The parameter values in Chapter II were 5, 10, 20, 5, 1, 7, 3, and 1, respectively, for the overtime penalty, lost production penalty, high priority goal reward, medium priority goal reward, low priority goal reward, high priority goal penalty, medium priority goal penalty, and low priority goal penalty. Recall that the overtime penalty of 5 applied to entire line of 66 line elements, rather than a single line element. Converting this value to a penalty for a single line element yields $5/66 = 0.0758$. To scale this value up to 1, we must multiply it by 13.2. We apply this same scaling factor to all of the parameter values to get 1, 2, 264, 66, 13.2, 92.4, 39.6, and 13.2. These are the parameter values used for both the deterministic DP and the rolling horizon optimization, and are used as a starting point for the parameter value ranges for the players in the Sampled Fictitious Play algorithm.

4.5.1 Scenario Description

The scenario we use here makes use of the same tasks and associated goals as in Chapter II. This scenario attempts to portray tasks and goals related to a plant preparing to introduce a new product. We use the same eight objectives for these computational experiments as we did in Section 2.4. We again use a planning horizon of one hour, implying $T_d = 3600$ seconds. One difference from the experiments of Chapter II is the fact that time penalties are computed on an individual line element basis, as in (3.6). We can now summarize the experiments and the results we obtained.

4.5.2 Experimental Results

The purpose of the experiments is to evaluate the performance of the Sampled Fictitious Play with Nature algorithm against both the rolling horizon optimization and the basic deterministic dynamic programming optimization when applied to the end-state problem. In addition, we seek to determine how well the objective function approximation (4.5) works. To evaluate these algorithms we seek to measure how well each performs in comparison to an instance of the deterministic dynamic programming algorithm with perfect hindsight knowledge of the flow matrix. Throughout these experiments, the build schedule remains fixed. The build schedule matches the one from the first set of experiments in Section 2.4. We select a parameter vector $\bar{p} = (1, 2, 264, 66, 13.2, 92.4, 39.6, 13.2)$ to represent the preferences of a fictional plant manager. This vector is chosen so that the relative parameter values are similar to those in Chapter II.

The deterministic dynamic programming algorithm can be thought of as a serial line special case of the algorithm from Chapter III. To develop a shutdown policy,

we generate the flow matrix that assumes a reliable line, and use this flow matrix to solve for a shutdown policy, $\mathbf{j}(\bar{p})$. This shutdown policy is applied to 100 different flow matrix realizations, and the value of the shutdown policy is evaluated using $h_{\bar{p}}(\mathbf{j}(\bar{p}), z)$, where z represents a flow matrix realization. Denote these values by $d_i, i = 1, \ldots, 100$.

The perfect information value is computed using the same procedure, except that the shutdown policy is developed with full knowledge of the flow matrix realization rather than using z'. In other words, if we define $\mathbf{j}(p, z)$ as follows:

$$\mathbf{j}(p, z) = \underset{\mathbf{j}}{\operatorname{argmax}}\, h_p(\mathbf{j}, z), \tag{4.16}$$

then the perfect information value shutdown policy is $\mathbf{j}(\bar{p}, z)$. The value of the perfect information policy then equals $h_{\bar{p}}(\mathbf{j}(\bar{p}, z), z)$. The value of this policy is computed for each of the same 100 flow matrices used earlier. The perfect information value associated with flow matrix realization z_i is denoted p_i.

For each flow matrix, the percent attainment of the perfect information value by the deterministic DP policy is computed as $\frac{d_i}{p_i}$. Across the 100 flow matrices, the deterministic DP solver achieved 54.1% of the perfect information value on average (minimum of 6.5%, maximum of 80.5%, standard deviation of 17.9%). The performance of the DP solver relative to the perfect information value varies widely, in the worst case attaining only 6.5% of the perfect information value available. Even on average, it can only obtain just over half of the perfect information value.

The rolling horizon optimization procedure is similar. The process described for the deterministic DP is repeated every 10 minutes during the horizon: we compute the flow matrix under the assumption of a reliable line, solve for the corresponding shutdown policy, and apply this policy to a stochastically generated flow matrix. The remaining downtime and cycletime are noted at the re-optimization point and

Player	Parameter Values
Overtime	1
Lost production	2, 10, 20
High reward	280, 400, 600, 800, 1000
Medium reward	70, 100, 150, 200
Low reward	14, 20, 30, 40, 50
High penalty	100, 200, 300, 400, 500
Medium penalty	42, 50, 75, 100
Low penalty	14, 16, 18, 20, 25

Table 4.1: Parameter value sets for each "player"

the line elements are initialized with that information before conducting the next optimization. A key point to note here is that during the 10 minutes between policy computations, the policy currently in effect is implemented. Thus, if the policy calls for stopping line element 3 when job 25 enters, and this event happens during the time this policy is in effect, we carry out this decision (stop line element 3) and this becomes a hard constraint for the remainder of the horizon. The rolling horizon optimization procedure was able to attain over 98% of the perfect information value. Due to the high reliability of the production line in these experiments, the rolling horizon optimization is able to achieve nearly all of the available value.

The Sampled Fictitious Play with Nature experiments are quite a bit more complicated. The experiments mimic the algorithm as described in Section 4.4.4. The non-Nature players' histories are initialized with a randomly chosen action (parameter value) from the player's action space (set of allowable parameter values). Table 4.1 shows the list of allowable parameter values for each player. For each of 10 iterations, we sample players' histories and then compute best responses. More specifically, a sample of size 1 is drawn for each player (including Nature). The non-Nature players sample from their history while Nature samples from its distribution. Call the vector that contains the players' sampled parameter values plus Nature's sampled value the sample vector and denote it by $v_{p,z}$, where p is the parameter vector created by the

sampled parameters and z is the sampled flow matrix.

Once the sample is generated, each non-Nature player computes its best response. If the player that is computing its best response is identified by index i, with action set A_i, denote by $v_{p,z}(a, i)$ the sample vector for which the i^{th} element (the parameter value corresponding to player i) has been replaced by $a \in A_i$. The modified parameter vector, call it p', is used to determine $\mathbf{j}(p')$, the policy from the deterministic DP. The value of this action a is then $h_{\bar{p}}(\mathbf{j}(p'), z)$. The value of each action is evaluated, and the action with the best value (known as the best response) is added to the player's history. Note also that the p' with the best $h_{\bar{p}}$ value across all iterations is tracked. The policy associated with this parameter combination, $\mathbf{j}(p')$, is the shutdown policy to implement.

Once the SFP with Nature policy is determined, the value of this policy is evaluated against the perfect information policy value in the same way the deterministic DP policy was evaluated. If the policy from SFP with Nature is denoted $\hat{\mathbf{j}}$, the value of this policy is $h_{\bar{p}}(\hat{\mathbf{j}}, z)$ when applied to flow matrix realization z. The value of the SFP with Nature policy when applied to flow matrix z_i is denoted s_i.

Using the same flow matrices as in the deterministic DP case, the percent attainment of the perfect information value by the SFP policy was computed as $\frac{s_i}{p_i}$. Across the 100 flow matrices, the SFP policy was able to achieve 88.9% of the perfect information value on average (minimum of 42.3%, maximum of 99.1%, standard deviation of 11.0%). The SFP-based algorithm shows both much stronger average performance and a lower standard deviation than the deterministic DP.

To evaluate the effect of the time horizon, we executed the same experiments as above, but ran the optimization fifteen minutes prior to shutdown (implying $T_d = 900$ seconds). With this shorter time horizon, and the additional information that comes

Approach	Time Until T_d	Mean	Min	Max	Std Dev
DP solver	1 hour	54.1%	6.5%	80.5%	17.9%
Obj Func Approx	1 hour	88.9%	42.3%	99.1%	11.0%
DP solver	15 minutes	87.4%	30.9%	100.0%	15.4%
Obj Func Approx	15 minutes	98.3%	64.9%	100.0%	5.3%
Obj Func Approx (with $\tilde{Z}$)	15 minutes	98.8%	58.4%	100.0%	4.6%

Table 4.2: Comparison of the DP Solver and the Objective Function Approximation

with it, the deterministic DP performed much better, achieving 87.4% of the perfect information value (minimum of 30.9%, maximum of 100.0%, standard deviation of 15.4%). The SFP with Nature algorithm improved as well, achieving 98.3% of the perfect information value (minimum of 64.9%, maximum of 100.0%, standard deviation of 5.3%). The SFP with Nature algorithm matched the performance of the rolling horizon algorithm in spite of having less information available. In addition, the ability of the SFP with Nature algorithm to achieve over 98% of the perfect information value suggests that our objective function approximation technique worked well.

The final experiment evaluates the effect of changing the meaning of Nature's sample point. Rather than representing a single flow matrix, we now view Nature's sample point as a vector of 30 flow matrices. Denote this modified definition of Z by $\tilde{Z}$. The function $h_p(\mathbf{j}, Z)$ is modified to be a sample average estimate of $g(j)$ based on the sample of size 30. This modified function is denoted by $\tilde{h}_p(\mathbf{j}, \tilde{Z})$. Changing the definition of Nature's sample point in this way enables a more accurate best response computation as the value of an action (parameter value) is now a sample average. This version of the algorithm, with a fifteen minute horizon, achieved 98.8% of the perfect information value (with a minimum of 58.4% , a maximum of 100.0% and a standard deviation of 4.6%). This modification improved the average performance and the standard deviation over the earlier, alternate definition of Nature's sample point. Table 4.2 summarizes the results of the deterministic DP and Objective

Function Approximation experiments.

The Sampled Fictitious Play with Nature algorithm achieved superior results to the policy from the deterministic dynamic program using the fictional manager's parameter values. In addition, with shorter time horizons, the SFP with Nature algorithm is able to achieve nearly all of the value attainable under perfect information.

4.6 Conclusions

In this chapter we explored two heuristic approaches to including stochasticity in the development of a shutdown policy for the End-State problem. As part of an objective function approximation technique, we introduced an extension to the Sampled Fictitious Play Algorithm that includes a Nature player. Intuitively, this Nature player adds noise to the objective function which introduces errors in the best responses. After outlining the updated algorithm, we presented an updated convergence result similar to that in Lambert III et al. (2005).

We applied this modified algorithm to the End-State problem and demonstrated how it could be used to learn a manager's objective function and find a shutdown policy that improves upon the original policy from the deterministic dynamic program. The analysis in Section 4.4 assumed that the manager's parameter vector, $\bar{p}$, was given and the objective was to find a new parameter vector that yielded an improved policy. The relative values of these parameters provide information about the manager's ranking of the relative importance of goals as well as stopping on time. However, as discussed earlier, the parameters within the parameter vector $\bar{p}$ are imprecise or hard to quantify.

While some of these values such as the overtime penalty can be determined with

a reasonable degree of confidence, others may be more difficult to justify. However, we expect that when presented with two competing shutdown policies, the manager could choose one in preference to the other. This becomes important in the best response computation stage of the Sampled Fictitious Play With Nature Algorithm. Rather than using the function g to evaluate $\mathbf{j}(p)$, we could instead ask the manager to choose the best among several possible shutdown policies. In this way, the algorithm can learn the manager's preferences and identify a parameter vector p^* that will yield a better policy.

This technique has broad applicability when viewed as an approach to identify the best mathematical model for a problem. Typically one develops a mathematical model which is fixed when an optimization is carried out. In this case, we acknowledge that our ability to correctly specify the mathematical model is hindered by the lack of data available to accurately compute the parameter coefficients in the objective function. Through Sampled Fictitious Play With Nature in combination with the deterministic dynamic programming solver, we get a fast solver that learns the mathematical model underlying a plant manager's preferences.

One significant aspect of the End-State problem we have not addressed is dealing not only with machine breakdowns and repairs, but also resequencing of jobs. This is a common occurrence in the paint shop, and is the major barrier that must be overcome to be able to address the End-State problem in the paint shop setting. Once this has been accomplished, we can move towards an optimizer that covers the entire plant.

CHAPTER V

Conclusions

While automotive manufacturing plants are complex, we were able to successfully develop a computational model that enabled the identification of shutdown policies that significantly improve on current practices. We first considered a simple serial line, then moved on to a line with certain types of non-serial topologies, and finally integrated stochasticity into the model.

Chapter II introduced the End-State problem as well as an approach based upon a network model of a production line. Using a dynamic programming approach, we took advantage of the restrictions the capacity and ordering constraints place on the set of feasible decisions. Through computational experiments, we were able to show the superiority of this dynamic programming algorithm to a rule of thumb based upon a common shutdown method used in practice.

One shortcoming of the original dynamic programming algorithm is the fact that it can only handle serial lines. The topic of Chapter III was a hierarchical decomposition approach that enabled the extension of the analysis to certain types of non-serial lines such as the split/merge topology seen in the general assembly area. This new approach not only expanded the topologies we could handle, but also reduced the computational complexity of the solver when applied to serial lines, handled ad-

ditional types of goals, and was more effective at pushing work in process further downstream.

We returned to analyze a serial line in Chapter IV, but considered stochasticity in the development of the model. While a truly optimal model proved to be intractable, we were able to develop a heuristic model based upon Sampled Fictitious Play. We extended the Sampled Fictitious Play algorithm to the case where the objective function is noisy, captured by the introduction of a "Nature" player that selects actions at random from a given distribution. After presenting a convergence result for this algorithm, we applied it to the End-State problem discussed throughout this dissertation. The algorithm was able to improve upon the policy from the deterministic dynamic program to create a policy that performed better within a stochastic environment.

The one main area not addressed in this dissertation is the resequencing of jobs. This happens infrequently in both the body shop and general assembly, but happens regularly within the paint shop. To be able to extend the algorithms to the paint shop, and develop a solver that can optimize the entire plant, this hurdle must be overcome.

This dissertation focused on an application to automotive manufacturing, but the results have broader applicability. Other potential applications include project management and aircraft and crew positioning. In the case of project management, suppose that a company has multiple projects that follow the same general process: design, prototyping, product launch, and support. If a new version of software needs to be installed, the company may want projects to be at certain stages prior to the software upgrade to minimize the disruption to the projects. Aircraft and crew positioning refers to the problem of determining where aircraft and crew should end

up at the end of each day to facilitate the following day's operations while satisfying a large number of constraints including the airline's schedule and various union rules. The application of the end-state approach to this problem would address operation of the airline on a day-to-day basis, as opposed to crew-recovery research that deals with recovering from catastrophic events (Yu et al., 2003). In the same way that certain end of shift states are better than others in manufacturing, the same could be said about end of day states for airlines.

APPENDICES

APPENDIX A

Sequential Decision Model

The purpose of this appendix is to explore in greater detail an approach for explicitly modeling the end-state problem as a sequential decision process. In both Chapters II and III, we applied solutions to the deterministic problem to a stochastic setting. Our objective now is to formulate a sequential decision model that explicitly includes stochasticity. We first discuss the sequential decision process state, subsequently describe the decisions that are made, and then move on to how the sequential decision process state transitions are governed. This appendix concludes with an explanation of the expected cost to go from an sequential decision process state and the development of the functional equation.

A.1 Sequential Decision Process State

The sequential decision process state is governed by the tuple (t, κ, σ), where t represents the current (relative) time, κ is the condition of the line (to be described more fully shortly), and σ represents the status of the production line. After describing each of the members of the sequential decision process state in turn, we later use the variable x as shorthand for the state.

The relative time, t, denotes how close we are to the desired shutdown time, T_d. A lost production penalty is assessed at each line element shutdown prior to T_d, at

a rate of p_l per unit time. Similarly, an overtime penalty of p_o is incurred for each unit of overtime.

In the discussion of the next two elements of the state, vectors κ and σ, we must first define and distinguish between the condition of the line elements and the status of the production line. The **condition of a line element** refers to whether the line element is up, down, or stopped. An up line element is functional, and may or may not be actively working on a job. A down line element is broken and being repaired. A stopped line element can no longer perform work, and will remain in this condition through the end of the horizon. The condition vector κ represents the condition of each line element, with κ_n representing the condition of line element n. The condition of an up, down, or stopped line element is given by 1, -1, or 0, respectively.

The **status of the production line** refers to the locations of jobs and the remaining cycletime for those jobs in process. One way in which the production line status could be specified is by a combination of an $(N+1)$-dimensional vector with the last job to enter each line element and the last job to exit the last line element, as well as an N-dimensional vector with the remaining cycletime for each job currently being processed at each line element. In the event that a workstation is empty or shutdown, this value could simply be set to zero by default. To simplify exposition, we assume that we have an N-dimensional vector that characterizes the production line status, denoted $\sigma \in \mathcal{S}$, where $\mathcal{S}$ is the space of all production line status vectors that satisfy the capacity and ordering constraints implied by the production line topology and the ordered list of jobs to be built. The status of line element n is σ_n.

The cycles between failures and time to repair of line elements are exponential random variables, resulting in a time to the next random event for the entire pro-

duction line that is an exponential random variable. If we denote the time until the condition of line element n changes as a result of a random event (breakdown or completed repair) by τ_n, then τ_n is an exponential random variable with rate λ_n. We say λ_n is equal to 0 if the condition of line element n is stopped. If line element n is not stopped, λ_n represents the appropriate rate based upon whether line element n is up or down.

We can then denote by $\tau_r = \min\{\tau_1, \tau_2, \ldots, \tau_N\}$, the time until the next random event occurs within the production line. τ_r is an exponential random variable with rate equal to $\lambda = \lambda_1 + \lambda_2 + \cdots + \lambda_N$. Again, $\lambda_n = 0$ if line element n is stopped. The condition of the line will either change due to the occurrence of a random event or due to an operator-controlled line element stoppage.

A.2 Decision Epochs, Decisions, and State Transitions

Operator-controlled stoppages may occur at any **decision epoch**. We define a decision epoch as a point in time at which any of the following occur: the condition vector κ changes due to a random event, a job enters a line element, a job finishes processing at a line element, or the time in the sequential decision process state is T_d. At each decision epoch, we must decide which line elements that are neither stopped nor in mid-cycle should be stopped. The actions associated with whatever decisions are made are taken immediately, leaving no possibility that a breakdown will occur or a repair will be completed to modify the state.

We digress momentarily to discuss why the decision epochs we have chosen are sufficient. More detail will be provided about the quantitative evaluation of a policy later, but for now we simply state without justification that a decision epoch must provide control over the contents of the line elements and the time at which each line

element is shutdown. Control over the contents of the line elements follows directly from the inclusion of job entering events in the list of valid decision epochs.

In Chapter II we showed that the contents of the line could be completely specified by controlling the last job to exit each line element, and the same logic applies to the last job to enter.

A more detailed analysis is required of the time at which a line element is shutdown. A line element is shutdown when it and its successor line element are both stopped. Since we can stop a line element at any decision epoch, we simply need to show that we would never want to shutdown a line element at some other time. The analysis below is broken up based upon the condition of the line element: up, down, or stopped.

An up element can be described in more detail as busy, blocked, or starved. We do not allow the shutdown of a line element in the middle of a busy period, meaning a line element can be shutdown only on one of the two boundaries of a busy period. We will then show that the only time we would shutdown a blocked or starved line element at a non-boundary time is if the time T_d, a valid decision epoch, is in the middle of the period.

First we consider a busy period of an up line element. As mentioned, the only points in this period at which a decision is allowed are the beginning and end of the period. Since one of the decision epochs is the time at which a line element completes processing, clearly the end of the busy period is covered. The beginning of the busy period is captured by the fact that this epoch must coincide with the end of another period.

Next we examine a period during which an up line element is blocked. Our purpose is to show that we will never want to shutdown a line element in the middle

of a blocked period at a time that is not a decision epoch. We temporarily ignore any decision epochs in the middle of the blocked period other than the occurrence of time T_d. If time T_d does not occur in the middle of the period, we would want to shutdown the line element at the boundary of the period that is closest to time T_d. If T_d does occur in the middle of the time period,

Throughout, we denote a decision by $u \in \{0,1\}^N$, and $U(x)$ is the set of feasible decisions that can be made while in state x. The N elements of u indicate line element stoppages to carry out, where a zero corresponds to a decision to stop a line element, while a one does not modify the condition of the line element. Conveniently, the condition vector can be updated by way of a component-wise product of the old vector and the decision applied, u. So as to not abuse notation, we will represent the component-wise vector product by the function ρ, which takes two arguments and returns a new vector representing the component-wise product of its arguments.

If a decision is made to stop at least one line element, we first carry out this decision prior to allowing the line to evolve. If the decision made does not stop a line element, we immediately allow the line to evolve. The **evolution of the line** is a function that takes the current production line status and a duration and returns an updated production line status, under the assumption that the condition of each line element remains constant through the entire duration. We can formally denote this function by $e_\kappa : \mathcal{S} \times \mathbb{R}^+ \to \mathcal{S}$. This function has built into it knowledge of the current condition of each line element (i.e., it is parameterized by the condition vector κ of the line). The computation performed by a given e_κ is deterministic, as the cycletimes of the line elements are deterministic under the assumption that the condition of all line elements remains constant. The evolution function enables the determination of the earliest time after the current time that a job enters a line

element. In combination with the pre-defined value T_d and the fact that the time until the next failure is an exponential random variable, we can probabilistically characterize the time until an sequential decision process state transition occurs.

We are now in a position to more formally describe the sequential decision process state update process, given that we start in state $x = (t, \kappa, \sigma)$. If we decide to stop at least one line element, let κ' be the condition vector κ with the condition of the line elements to be stopped changed from up to stopped. The sequential decision process state will be updated to $x' = (t, \kappa', \sigma) = (t, \rho(\kappa, u), \sigma)$ prior to its continued evolution. If we do not stop any line elements, the line will evolve from the original state x.

As described above, the transition to the next sequential decision process state will be due to the first of the following events to occur: a condition change due to a machine breakdown or a completed repair, a job-related event (either a job completing processing or entering a line element), or reaching time T_d. We will now formally characterize each of these three possibilities. In the write-up that follows, we assume the line evolution starts from sequential decision process state $x = (t, \kappa, \sigma)$ to simplify notation.

First, let's consider an sequential decision process state transition due to a condition change. Denote by κ^n the condition vector that differs from κ only in the condition of line element n, under the assumption that a random event at line element n changed its condition. In other words, for $k \in \{1, 2, \ldots, N\}$:

$$\kappa_k^n = \begin{cases} \kappa_k & \text{if } k \neq n \\ -\kappa_k & \text{if } k = n \end{cases} \tag{A.1}$$

In Equation (A.1), we make use of the fact that a condition of 1 indicates an up line element, while a down line element has a condition of -1. A random event at line

element n simply toggles the condition at that line element between up and down.

If the condition change occurs at line element n, the new sequential decision process state will be $(t + \tau_n, \kappa^n, e_\kappa(\sigma, \tau_n))$. Recall that τ_n is a random variable that represents the time until the next condition change at line element n.

Next, we characterize an sequential decision process state transition due to a job-related event. We can deterministically compute the next time a job enters a line element or completes processing starting the line from state (t, κ, σ) under the assumption that the condition vector of the line remains constant. Denote this time by t_j.

The function $\mathcal{L}$ returns a vector containing the list of jobs in each line element, and ignores the remaining cycletime information. The new sequential decision process state, when the transition is due to a job-related event, is then $(t_j, \kappa, e_\kappa(\sigma, t_j - t))$.

The final possibility is the event that we hit time T_d, the desired shutdown time. In this case, the new sequential decision process state will be $(T_d, \kappa, e_\kappa(\sigma, |t|))$.

Now, we develop the probability of making each of these three sequential decision process state transitions. Given the current sequential decision process state, a state transition will be due to either a random event that causes a change in condition or the earlier of a job-related event or the time T_d. The probability of a random event leading to an sequential decision process state transition is $P(\tau_r < \min\{t_j, T_d\})$:

$$P(\tau_r < \min\{t_j, T_d\}) = \sum_{n \in \mathbf{N}} \int_0^{\min\{t_j, T_d\}} P(\tau_m \geq \tau, \forall m \in \mathbf{N} \setminus \{n\}) h_{\tau_n}(\tau) d\tau \tag{A.2}$$

In Equation (A.2), the function h_{τ_n} is the probability density function of the time to the next random event at line element n, where we use the convention that $h = 0$ for all finite τ if line element n is already stopped.

Otherwise, with probability $P(\tau_r \geq \min\{t_j, T_d\})$, the next sequential decision process state transition will be due to the earlier of the first job-related event or

reaching time T_d.

Using the notation of Bertsekas (2005), we now summarize the state transitions. We assume that we start in state $x = (t, \kappa, \sigma)$, make decision $u \in U(x)$, and experience random outcome w. For our purposes here, assume w yields the shortest time until a random event occurs, and without loss of generality that this random event takes place at line element n. The state update function f accepts a state, decision, and random outcome as inputs, and returns a new state:

$$f(x,u,w) = \begin{cases} (t+w, \rho(\kappa,u)^n, e_{\rho(\kappa,u)}(\sigma, w)) & \text{random event} \\ (t_j, \rho(\kappa,u), e_{\rho(\kappa,u)}(\sigma, t_j - t)) & \text{job entered event} \\ (T_d, \rho(\kappa,u), e_{\rho(\kappa,u)}(\sigma, |t|)) & \text{time } T_d \end{cases} \tag{A.3}$$

Having established the allowed decision epochs, what decisions can be made, and how the sequential decision process state transitions occur, we can now move on the discussing the costs incurred as a result of a decision.

A.3 Costs

The costs incurred include both immediate costs that directly result from the decision made as well as expected costs based upon the subsequent state. To discuss the immediate costs, we must first discuss what we mean by shutting down a line element. A line element is said to be shutdown when it and its successor line element are both stopped. At this time, both time penalties and any rewards or penalties due to goals can be computed for the shutdown line element.

To consider what the immediate costs will be, let's consider a simple case in which we start in state $x = (t, \kappa, \sigma)$ and enact a decision $u \in U(x)$ that shuts down line element n (it ensures that line element n is stopped). Then the immediate costs are

given by:

$$\begin{aligned} g(x,u) &= \left[p_o(t-T_d)^+ + p_l(t-T_d)^-\right] \\ &\quad + \left[\sum_{\gamma\in\Gamma_n} r_\gamma \mathbf{1}_{c_n\in C_\gamma} - p_\gamma \mathbf{1}_{c_n\notin C_\gamma}\right] \end{aligned} \tag{A.4}$$

In Equation (A.4), we assumed that the actions taken to shutdown line element n did not also result in a shutdown of another line element. If they did, we would also need to include time penalties and goal rewards and penalties for each line element shutdown as a result of the current decision.

The first line in Equation (A.4) represents the time penalties due to a shutdown time that deviates from T_d, while the second line represents the rewards and penalties due to goals. The term Γ_n is a set consisting of all goals defined at line element n. The reward and penalty associated with goal γ are r_γ and p_γ, respectively. Finally, c_n represents the contents of line element n at shutdown, while C_γ represents the contents of line element n that satisfy goal γ.

Following Bertsekas (2005), we can now develop a recursive equation in terms of the immediate cost and state update function:

$$J_{k+1}(x) = \min_{u\in U(x)} g(x,u) + E_w\left\{J_k(f(x,u,w))\right\} \tag{A.5}$$

The difficulty we face is that we have an infinite number of states and decision epochs (although the feasible controls at a given state are finite). In this case, we may incur cases in which value iteration and policy iteration do not converge, or an optimal policy may not even exist.

APPENDIX B

Analysis of Sampled Fictitious Play with Nature Algorithm

Proof of Theorem IV.1

Proof: Let $(y(t))_{t=1}^{\infty}$ be a sampled fictitious play with nature process with sample sizes $k_t = \lceil Ct^{\beta} \rceil$, and $(F_y(t))_{t=1}^{\infty}$ be the associated belief process. We begin by establishing a bound on the random variable

$$\bar{U}_{k_t}^i(y^i, F_y^{-i}(t), G) - u(y^i, F_y^{-i}(t), G)$$

for an arbitrary $y^i \in \mathcal{Y}^i$. Fix t, let $y^i \in \mathcal{Y}^i$, and define

$$X_j(t) = u(y^i, Y_j^{-i}(t), Z_j) - u(y^i, F_y^{-i}(t), G),\ j = 1, \ldots, k_t,$$

where the $Y_j^{-i}(t)$ are random vectors with distribution $F_y^{-i}(t)$ and Z_j are random numbers with distribution G. The $X_j(t)$'s are not independent random variables as they are described by functions of the same random variable $F_y^{-i}(t)$; also the $Y_j^{-i}(t)$'s are dependent both on $F_y^{-i}(t)$ and on each other. However, for a fixed value of t and conditional on $F_y^{-i}(t) = f_y^{-i}(t)$, vectors $Y_j^{-i}(t),\ j = 1, \ldots, k_t$ are i.i.d. with distribution $f_y^{-i}(t)$ and $u(y^i, f_y^{-i}(t), G)$ is a constant. With such conditioning, therefore, $X_1(t), \ldots, X_{k_t}(t)$ are i.i.d. random variables with mean 0 (by the law of the unconscious statistician).

Let $L = \sup_{f \in \Delta, z \in \Omega} |u(f, z)|$, where Ω is the sample space of the random element

Z, and let A denote the event that $F_y^{-i}(t) = f_y^{-i}(t)$. Then, $E[X_j(t)^4|A] \leq (2L)^4$ and $E[X_j(t)^2|A] \leq (2L)^2$ for all $j = 1, \ldots, k_t$.

Let $S(t) = \sum_{j=1}^{k_t} X_j(t)$. Then

$$\begin{aligned} E[S(t)^4|A] &= k_t E[X_1(t)^4|A] + \binom{4}{2}\binom{k_t}{2} E[X_1(t)^2 X_2(t)^2|A] \\ &\leq k_t(2L)^4 + (3{k_t}^2 - 3k_t)E[X_1^2|A]E[X_2^2|A] \leq 3(2L)^4 {k_t}^2. \end{aligned}$$

Neither of the above bounds depended on $f_y^{-i}(t)$, therefore unconditionally,

$$E\left[\frac{S(t)^4}{{k_t}^4}\right] \leq \frac{3(2L)^4 {k_t}^2}{{k_t}^4} = \frac{3(2L)^4}{\lceil Ct^\beta \rceil^2} \leq \frac{3(2L)^4}{C^2 t^{2\beta}}. \tag{B.1}$$

By the Markov inequality, for any $\delta > 0$,

$$P\left\{\frac{S(t)^4}{{k_t}^4} > \delta t^{0.5-\beta}\right\} \leq \frac{E\left[\frac{S(t)^4}{{k_t}^4}\right]}{\delta t^{0.5-\beta}}.$$

Combining this with (B.1),

$$\sum_{t=1}^{\infty} P\left\{\frac{S(t)^4}{{k_t}^4} > \delta t^{0.5-\beta}\right\} \leq \sum_{t=1}^{\infty} \frac{E\left[\frac{S(t)^4}{{k_t}^4}\right]}{\delta t^{0.5-\beta}} \leq \sum_{t=1}^{\infty} \frac{3(2L)^4}{C^2 \delta t^{\beta+0.5}} < \infty.$$

By the Borel-Cantelli lemma (which does not require independence of events, see Ross, 1996), we have, with probability 1, $(S(t)^4/{k_t}^4) \to 0$ at an asymptotic order of $1/t^{\beta-0.5}$. The previous argument implies that, with probability 1, $(S(t)/k_t) \to 0$ at an asymptotic order of $1/t^\alpha$, where $\alpha = (\beta - 0.5)^{0.25}$. Since

$$\frac{S(t)}{k_t} = \bar{U}_{k_t}^i(y^i, F_y^{-i}(t), G) - u(y^i, F_y^{-i}(t), G),$$

we conclude that $\bar{U}_{k_t}^i(y^i, F_t^{-i}, G) - u(y^i, F_y^{-i}(t), G)$ converges to 0 at an asymptotic rate of $1/t^\alpha$, where $\alpha = (\beta - 0.5)^{0.25}$, for any $y^i \in \mathcal{Y}^i$, with probability 1.

For the sampled fictitious play with nature process $(y(t))_{t=1}^{\infty}$, define

$$\varepsilon_t^i = v^i(F_y(t), G) - u(y^i(t+1), F_y^{-i}(t), G) \geq 0$$

for $i \in N$ and $t \geq 1$. (As before, the vectors $(\varepsilon_t)_{t=1}^{\infty}$ can be interpreted as the errors in the players responses at iteration t as reflected by the utility function. In this case the errors are brought on by the players optimizing the sample means of their payoff functions instead of the true payoff functions, and hence form a stochastic process.) We will show that, with probability 1, for any player $i \in N$, $\varepsilon_t^i \to 0$ as $t \to \infty$ at an asymptotic order of $t^{-\alpha}$, with $\alpha = (\beta - 0.5)^{0.25} > 0$. By Theorem 4 of Lambert III et al. (2005) this would imply that the path $(y(t))_{t=1}^{\infty}$ converges in beliefs to equilibrium with probability 1.

Let $\tilde{y}^i \in \operatorname{argmax}_{y^i \in \mathcal{Y}^i} u^i(y^i, f_y^{-i}(t), G)$. Conditioning on the event $F_y^{-i}(t) = f_y^{-i}(t)$,

$$
\begin{aligned}
0 \leq \varepsilon_t^i &= v^i(f_y(t), G) - u(y^i(t+1), f_y^{-i}(t), G) \\
&= u(\tilde{y}^i, f_y^{-i}(t), G) - u(y^i(t+1), f_y^{-i}(t), G) \\
&= u(\tilde{y}^i, f_y^{-i}(t), G) - \bar{u}_{k_t}^i(\tilde{y}^i, f_y^{-i}(t), G) \\
&+ \left(\bar{u}_{k_t}^i(\tilde{y}^i, f_y^{-i}(t), G) - \bar{u}_{k_t}^i(y^i(t+1), f_y^{-i}(t), G)\right) \\
&+ \left(\bar{u}_{k_t}^i(y^i(t+1), f_y^{-i}(t), G) - u(y^i(t+1), f_y^{-i}(t), G)\right) \\
&\leq \left(u(\tilde{y}^i, f_y^{-i}(t), G) - \bar{u}_{k_t}^i(\tilde{y}^i, f_y^{-i}(t), G)\right) \\
&+ \left(\bar{u}_{k_t}^i(y^i(t+1), f_y^{-i}(t), G) - u(y^i(t+1), f_y^{-i}(t), G)\right),
\end{aligned}
$$

where the last inequality follows since $y^i(t+1)$ is chosen to maximize $\bar{u}_{k_t}^i(y^i, f^{-i}(t), G)$. The above bound did not depend on a particular realization $f_y^{-i}(t)$ of $F_y^{-i}(t)$; therefore, we have unconditionally

$$
\begin{aligned}
0 \leq \varepsilon_t^i &\leq \left(u(\tilde{y}^i, F_y^{-i}(t), G) - \bar{U}_{k_t}^i(\tilde{y}^i, F_y^{-i}(t), G)\right) \\
&+ \left(\bar{U}_{k_t}^i(y^i(t+1), F_y^{-i}(t), G) - u(y^i(t+1), F_y^{-i}(t), G)\right).
\end{aligned}
$$

Applying the derived asymptotic rate of convergence to the two terms of the above bound, we conclude that, with probability 1, ε_t^i converges to 0 at an asymptotic order of $1/t^\alpha$, where $\alpha = (\beta - 0.5)^{0.25}$, establishing the desired result. ∎

BIBLIOGRAPHY

BIBLIOGRAPHY

J. M. Alden and R. L. Smith. Rolling horizon procedures in nonhomogeneous markov decision processes. *Operations research*, 40(Supplement 2: Stochastic Processes):S183–S194, May - Jun. 1992.

D. P. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific, Belmont, Mass., 2005.

G. W. Brown. Iterative solution of games by fictitious play. In *Activity analysis of production and allocation*, New York, 1951. Wiley.

K.-Y. Cheung, C.-W. Hui, H. Sakamoto, K. Hirata, and L. O'Young. Short-term site-wide maintenance scheduling. *Computers and Chemical Engineering*, 28:91–102, 2004.

D. I. Cho and M. Parlar. A survey of maintenance models for multi-unit systems. *European journal of operational research*, 51:1–23, 1991.

E. V. Denardo. *Dynamic programming : models and applications*. Prentice-Hall, Englewood Cliffs, N.J., 1982.

S. Duffuaa, A. Raouf, and J. D. Campbell. *Planning and control of maintenance systems : modeling and analysis*. Wiley, New York, 1999.

General Motors Corporation. Follow a truck down the assembly line, Accessed: July 22, 2008a. URL `http://www.gm.com/experience/education/misc_images/map_body_shop.gif`.

General Motors Corporation. Follow a truck down the assembly line, Accessed: July 22, 2008b. URL `http://www.gm.com/experience/education/promos/2006/mfg_images/body1.jpg`.

General Motors Corporation. Follow a truck down the assembly line, Accessed: July 22, 2008c. URL `http://www.gm.com/experience/education/promos/2006/mfg_images/paint2.jpg`.

General Motors Corporation. Follow a truck down the assembly line, Accessed: July 22, 2008d. URL `http://www.gm.com/experience/education/promos/2006/mfg_images/chassis1.jpg`.

S. B. Gershwin. *Manufacturing systems engineering*. PTR Prentice Hall, Englewood Cliffs, N.J., 1994.

D. A. Kimber, X. Zhang, P. H. Franklin, and E. J. Bauer. Modeling planned downtime. *Bell Labs technical journal*, 11(3):7–19, 2006.

T. J. Lambert III, M. A. Epelman, and R. L. Smith. A fictitous play approach to large-scale optimization. *Operations Research*, 53(3):477–489, May-June 2005.

J. J. McCall. Maintenance policies for stochastically failing equipment: A survey. *Management science*, 11(5):493–524, 1965.

D. Monderer and L. Shapley. Fictitious play property for games with identical interests. *Journal of Economic Theory*, 68:258–265, 1996.

N. C. Nelson. Downtime procedures for a clinical information system: A critical issue. *Journal of critical care*, 22(1):45–50, 2007.

W. P. Pierskalla and J. A. Voelker. A survey of maintenance models: The control and surveillance of deteriorating systems. *Naval research logistics quarterly*, 23:353–388, 1976.

J. Robinson. An iterative method of solving a game. *The Annals of Mathematics*, 54(2): 296–301, September 1951.

P. Samaranayake, G. S. Lewis, E. R. A. Woxvold, and D. Toncich. Development of engineering structures for scheduling and control of aircraft maintenance. *International journal of operations & production management*, 22(8):843–867, 2002.

L. S. Shapley. *Some Topics in Two-Person Games*, volume 52 of *Advances in game theory*, page 679. Princeton University Press, Princeton, 1964.

Y. S. Sherif and M. L. Smith. Optimal maintenance models for systems subject to failure: A review. *Naval research logistics quarterly*, 28:47–74, 1981.

C. Valdez-Flores and R. M. Feldman. A survey of preventive maintenance models for stochastically deteriorating single-unit systems. *Naval research logistics*, 36(4):419–446, 1989.

G. Yu, M. Argüello, G. Song, S. M. McCowan, and A. White. A new era for crew recovery at continental airlines. *Interfaces*, 33(1):5–22, January-February 2003.