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

Shih-Fen Cheng[*][†] Blake E. Nicholson[‡] Marina A. Epelman[§]
Daniel J. Reaume[¶] Robert L. Smith[‖]

Abstract

In modern production systems, it is critical to perform maintenance, calibration, installation, and upgrade tasks during planned downtime. Otherwise, the systems become unreliable and new product introductions are delayed. For reasons of safety, testing, and access, task performance often requires the vicinity of impacted equipment to be left in a specific "end state" when production halts. Therefore, planning the shutdown of a production system to balance production goals against enabling non-production tasks yields a challenging optimization problem. In this paper, we propose a mathematical formulation of this problem and a dynamic programming approach that efficiently finds optimal shutdown policies for deterministic serial production lines. An event-triggered re-optimization procedure that is based on the proposed deterministic dynamic programming approach is also introduced for handling uncertainties in the production line for the stochastic case. We demonstrate numerically that in these cases with random breakdowns and repairs, the re-optimization procedure is efficient and even obtains results that are optimal or nearly optimal.

Keywords: Manufacturing systems, shutdown planning, auto industry, dynamic programming

[*]Corresponding author

[†]School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902, Republic of Singapore, `sfcheng@smu.edu.sg`

[‡]Revenue Management Systems, Delta Air Lines, 1030 Delta Boulevard, Atlanta, GA 30320-6001, USA, `blaken@umich.edu`

[§]Department of Industrial and Operations Engineering, University of Michigan, 1205 Beal Avenue, Ann Arbor, MI 48109-2117, USA, `mepelman@umich.edu`

[¶]Operations Research Activity, General Motors R&D and Strategic Planning M/C 480-106-359, 30500 Mound Road, Warren, MI 48154, USA, `daniel.reaume@gm.com`

[‖]Department of Industrial and Operations Engineering, University of Michigan, 1205 Beal Avenue, Ann Arbor, MI 48109-2117, USA, `rlsmith@umich.edu`

1 Introduction

Maximizing equipment utilization is essential to the profitability of capital-intensive production processes. Although many researchers addressed the question of 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.

Planned downtime is useful for a variety of critical tasks including preventive maintenance, calibrations, installations, and upgrades, that can be performed only when a work 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, consider the task of upgrading a particular station in a production line consisting of stations separated by buffers. 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. Without an aid of an appropriate decision support tool, the problem of achieving as many such requirements (called *end-state goals* in the rest of the paper) as possible while trading off potential lost production time or overtime costs presents a challenge to the line managers even if the production line is assumed to be deterministic. As a result, managers usually resort to simple rules of thumb in making shutdown decisions, leading to significantly suboptimal shutdown policies. For the stochastic case, the development of a decision support tool is itself a significant challenge, since an exact representation of a stochastic production line would require a Markov Decision Process model whose state space explodes to an unmanageable size.

In this paper we address both of the above challenges. We present a mathematical model and a dynamic programming approach that solves the deterministic version of the problem efficiently. Moreover, in the stochastic setting, we can utilize this deterministic dynamic programming model within an efficient event-triggered re-optimization procedure that obtains solutions which in our numerical experiments were optimal or near-optimal. Among its contributions, this paper:

- Develops an efficient dynamic programming (DP) 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.

- Proposes a DP-based, event-triggered re-optimization procedure that effectively handles unexpected breakdowns and repairs at stations. We demonstrate numerically that our procedure produces results that are optimal or near optimal, by comparing them to shutdown policies obtained under assumption of perfect hindsight.
- Applies the mathematical model to data taken from an actual production line of a major automotive manufacturer, demonstrating that the model and the algorithm have real-world utility. We use simulation experiments to demonstrate that the shutdown policy computed by the algorithm significantly improves upon a typical rule-of-thumb approach used in practice.

This paper also charts new territory in looking at how to optimally control a production line in the time leading up to a scheduled downtime. To the best of our knowledge, this topic has been largely unaddressed in the literature. Most research related to maintenance either focuses on maintenance scheduling 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), or on 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., 1998; Samaranayake et al., 2002). There are some works on chemical production plants (Cheung et al., 2004) and clinical information systems (Nelson, 2007) that discuss how one can optimize the shutdown plans in order to minimize negative impacts. However, even among those studies, we cannot find one that explicitly considers both production and maintenance goals. Although we discuss an automotive assembly application in this paper, this methodology is applicable to a variety of systems involving work-in-process inventory. Examples include oil refineries, chemical processing plants, semiconductor manufacturing, transactional back-office operations, and new product development and introduction.

This paper is organized as follows. Section 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 3 presents an efficient dynamic programming formulation for finding an optimal shutdown policy in a deterministic environment. An event-triggered re-optimization procedure is proposed in Section 4 to handle various uncertainties. Section 5 presents computational experiments. Finally, Section 6 summarizes lessons learned.

2 A Model of the End-State Planning Problem

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

2.1 A Network Representation 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 work in progress (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 configuration, a serial line, for the rest of this paper. Figure 1 illustrates a serial line configuration. 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 to the head of the line. Jobs, numbered sequentially from 1 to J, thus enter the line at element N, proceed through the line in order, and exit at element 1. The rationale for this numbering will become clear later on.

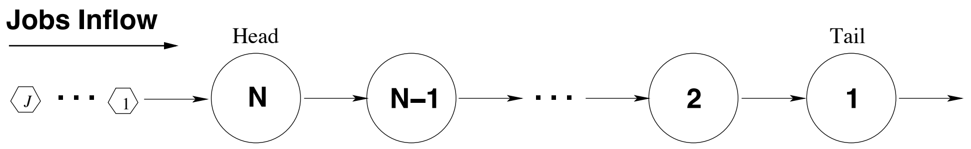


Figure 1: A serial production line. Jobs enter at line element N, and exit at line element 1.

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 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 line in order, $i_n^m = i_n^1 + (m-1)$. If $m = 0$, the line element is empty.
- Let $\mathbf{K}_n = \{1, \ldots, k_n\}$ denote the index set of end-state goals associated with line element n, $n \in \mathbf{N}$. If $\mathbf{K}_n = \emptyset$, no end-state goal is associated with line element n.
- Let $\mathbf{R}_n^k,\ k \in \mathbf{K}_n$, be the set of all tuples of WIP that would satisfy the end-state goal k for line element n.
- Let v_n^k be the value awarded if, at shutdown, $r_n \in \mathbf{R}_n^k$, and let p_n^k be the penalty assessed if, at shutdown, $r_n \notin \mathbf{R}_n^k$.
- Let T_d be the desired line shutdown time, e.g., the end of the normal shift. A cost associated with overtime or lost production time 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 cost per unit time associated with lost production 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 The Formal Definition of the End-State Planning Problem

An end-state goal is, broadly speaking, a description of the desired contents of a line element (for example, the desired quantity or combination of jobs) when the production line shuts down, which would enable performance of a particular task during downtime. The objective of the end-state planning problem 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 not meeting end-state goals, overtime, or lost production time.

Although build schedules are known to the managers, figuring out the above-mentioned trade-off is non-trivial in general, because: 1) multiple goals specified for an individual line element may conflict with one another; 2) the build schedule may cause conflicts between line elements; 3) given a set of end-state goals that can be jointly satisfied, the line might have to be run beyond (or stopped before) the desired shutdown time, causing excessive overtime (or lost production).

A 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, a shutdown policy will be specified 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. One reason 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, job-based shutdown policy guarantees that the same set of end-state goals will be achieved even if the production system is subject to uncertainty (e.g., line elements may break down at random and need to be repaired).

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: since j_{n+1} is the last job to enter n from $n+1$, while $j_n + 1$ is the first job in n, we can determine r_n as a function of j_n and j_{n+1}, namely

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

where line element n is empty if $j_n = j_{n+1}$. (To handle line element N, 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}.) If $r_n(j_n, j_{n+1}) \in R_n^k$, i.e., the corresponding end-state goal k is met, a reward of value v_n^k is awarded; otherwise, a penalty of value p_n^k 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^k 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^k will be described using predicates $\leq$, $=$, and $\geq$ on the number of jobs, or jobs of particular types, and containment $r_n(j_n, j_{n+1}) \in R_n^k$ will be checked simply by verifying that the tuple $r_n(j_n, j_{n+1})$ satisfies the resulting constraints.

In contrast to end-state goal satisfaction, with the shutdown policy specified in the above form, the shutdown time of each line element and the overall shutdown time is not easily computable. Even in a perfectly predictable, i.e., **deterministic** line with known job cycle times at 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 recursive procedure is developed for the deterministic environment (see Section 3). With this recursive procedure, we can identify the shutdown time for each line element. 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. Based on the above definition, the shutdown time of the line can be defined to be a function of shutdown decisions, $T_s(\boldsymbol{j})$, where $\boldsymbol{j} = (j_1, \ldots, j_{N+1})$.

Given the shutdown time of the line, the associated overtime or lost production time cost is easily computed. Recall that T_d denotes the desired stopping time. When $T_s(\boldsymbol{j}) > T_d$, overtime cost is incurred at the rate of p_o per unit time. Otherwise, when $T_s(\boldsymbol{j}) < T_d$, 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 time-related cost definitions.)

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 planning problem can be formally defined as:

$$\max_{\boldsymbol{j} \in \tilde{\mathbf{J}} \subset \mathbf{J}^{N+1}} \quad \sum_{n \in \mathbf{N}} \sum_{k \in K_n} \left[I_n^k v_n^k - (1 - I_n^k) p_n^k \right] - p_o (T_s(\boldsymbol{j}) - T_d)^+ - p_l (T_d - T_s(\boldsymbol{j}))^+ \tag{2}$$

$$\text{s.t.} \quad I_n^k = \begin{cases} 1, & r_n(j_n, j_{n+1}) \in R_n^k \\ 0, & \text{o/w} \end{cases}, \ \forall\, k \in K_n, \ \forall\, n \in \mathbf{N}$$

$$T_s(\boldsymbol{j}) \le T_{\max}.$$

Note that (2) only captures deterministic problem instances; once uncertainties are introduced, the production line shutdown time, $T_s(\cdot)$, becomes a random variable and (2) is no longer an adequate model. The handling of such uncertainties is deferred to Section 4, in which an event-triggered re-optimization procedure is introduced.

3 Deterministic Dynamic Programming Formulation

In this section we develop an efficient dynamic programming formulation of the end-state planning problem (2) in a deterministic setting. Specifically, we assume that the equipment is reliable and there are no unpredictable breakdowns. This dynamic programming formulation will serve as the foundation for building the re-optimization procedure in Section 4, in which we will relax the deterministic assumption and deal with uncertainties.

Toward providing an efficient algorithm for solving (2), 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. As such, as long as we know the decision made at the immediate previous line element, all decisions made at other line elements would provide no further information. In other words, the decision at the immediate previous line element summarizes 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), which is known to be extremely efficient for sequential decision problems (for more detail, refer to Denardo, 1982).

3.1 Feasible Shutdown Policies

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 nor the capacities of the line elements. In particular, 1) for two consecutive line elements $n+1$ and n, j_{n+1} must be at least j_n; 2) since $r_n(j_n, j_{n+1}) = (j_n + 1, \ldots, j_{n+1})$ for $n \leq N$, we require that $j_{n+1} - j_n \leq m_n$. Summarizing the above two observations, the values for j_n are constrained as follows:

$$j_n \in \begin{cases} \mathbf{J}, & n = 1, \\ \{j_{n-1}, j_{n-1}+1, \ldots, j_{n-1}+m_{n-1}\}, & n > 1. \end{cases} \tag{3}$$

3.2 Computing the Shutdown Time from the Shutdown Policy

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

Let $t_{j,n}$ be the processing time, 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 have already exited line element $n+1$, which occurs at time $e_{j,n+1}$, and completed 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 have already exited line element n, which occurs at time $e_{j-1,n}$, and job j must subsequently have been 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
        $e_{j,n} = \max\{e_{j,n+1} + t_{j,n}\ ,\ e_{j-1,n} + t_{j,n}\ ,\ e_{j-m_{n-1},n-1}\}$    (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 $T_s = \max_{n \in \mathbf{N}}\{e_{j_n,n}\}$. Alternatively, if we denote by T_n the maximum shutdown time of line elements 1 through n, the production line shutdown time could be computed recursively as

$$T_0 = 0,\ T_n = \max\{e_{j_n,n},\ T_{n-1}\},\ n = 1, \ldots, N, \text{ and } T_s = T_N. \tag{5}$$

3.3 Dynamic Programming Model

Based on the above discussion, problem (2) can be cast as a sequential decision process, where a decision is made at each line element, starting from line element 1. From Equation (3), 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 (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 using Equation (5) as soon as j_n is chosen.

From the above description, the 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 goals specified for line element $n-1$ is obtained by first computing r_{n-1} according to Equation (1), then checking to see if $r_{n-1} \in R^k_{n-1}$, where $k \in K_{n-1}$ denotes goal k defined at line element $n-1$. If so, the decision garners a reward of v^k_{n-1}, otherwise it incurs a penalty of p^k_{n-1}. Summing over all $k \in K_{n-1}$ then gives the aggregate reward/penalty at line element $n-1$. (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 (1)). 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 time 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 through $n-1$.

In the initial stage $n = 1$, there is only one state, namely $(n, T) = (1, 0)$.

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

$$j_n \in \begin{cases} \mathbf{J}, & n = 1 \\ \{j, j+1, \ldots, j + m_{n-1}\}, & n > 1. \end{cases} \tag{6}$$

- State transition functions are as described above.
- Reward function at state (n, j, T) with decision j_n:

$$V(1,0) = 0, \text{ and } V(n, j; j_n) = \sum_{k \in K_{n-1}} \left[I_{n-1}^k v_{n-1}^k - (1 - I_{n-1}^k) p_{n-1}^k \right], \ 2 \le n \le N+1, \tag{7}$$

where

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

Note that a reward/penalty is assessed at stage n for meeting the end-state goals at line element $n-1$, as discussed above.

- Terminal cost:

$$L(T) = p_o(T - T_d)^+ + p_l(T_d - T)^+, \tag{8}$$

where T is the shutdown time of the line. The terminal cost represents the cost due to overtime or lost production 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). Then for $n = 1$:

$$f(1, 0) = \max_{j_1 \in \mathbf{J}} \{ f(2, j_1, e_{j_1,1}) \}, \tag{9}$$

for $2 \le n \le N$:

$$f(n, j, T) = \max_{j_n \in \{j, j+1, \ldots, j+m_{n-1}\}} \{ V(n, j; j_n) + f(n+1, j_n, \max\{T, e_{j_n,n}\}) \}, \tag{10}$$

for $n = N + 1$:

$$f(N+1, j, T) = \max_{j_{N+1} \in \{j, j+1, \ldots, j+m_N\}} \{V(N+1, j; j_{N+1}) - L(T)\}. \tag{11}$$

Note that when computing $f(n, j, T)$ using Equation (10), any j_n that would lead to next state with undefined functional equation will be pruned from consideration.

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

3.3.1 Pruning DP States: Dealing with Initial Content, $T_{\max}$, and Pre-Existing Shutdowns

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, in practice 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 planning problem is being considered, and we are given the 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 $n(k) = N + 1$). Since job k starts at line element $n(k)$, none of the line elements upstream — with ID greater than $n(k)$ — can 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.

Besides pruning DP states, we also need to update the flow matrix to reflect the starting positions of these jobs. Obviously, if job k starts at $n(k)$, it cannot visit any of the upstream line elements; therefore, for each job k with $n(k) \neq N + 1$ (jobs with $n(k) = N + 1$ have not entered the production line yet), the first condition defined in section 3.2 should be refined as:

$$e_{k,n} \geq \begin{cases} -M, & n > n(k) \\ t_{k,n}, & n = n(k) \\ e_{k,n+1} + t_{k,n}, & \text{o/w} \end{cases},$$

where $-M$ is any sufficiently negative number used to nullify the first of the three terms in (4) for $n(k)$'s upstream line elements.

We can use a similar approach to ensure that the shutdown time resulting from the decision $(j_1, \ldots, j_N, j_{N+1})$ does not exceed $T_{\max}$ by pruning all DP states (n, j, T) with T exceeding $T_{\max}$. Furthermore, for every remaining state, if any feasible decision j_n would lead to a next state with T exceeding $T_{\max}$, j_n has to be removed.

Finally, if some line elements are already shut down, we should limit the feasible decision sets for these line elements to be singletons containing only the given shutdown decisions. For every remaining state, if any decision j_n would lead to a next state with undefined functional equation, j_n has to be removed from the feasible set.

3.3.2 Computational Complexity of the DP

Here we compute an upper bound on the computational effort required to solve the above dynamic programing model by the standard backward induction 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 integers.

Assume that no state pruning is performed and the size of the state space is $(N \cdot J \cdot T_{\max})$. In almost all cases, the end-state planning is not performed until the final hour; this implies that reasonable upper bounds can be defined for J and $T_{\max}$. If the values of J and $T_{\max}$ are fixed to their respective upper bounds, the computational effort is linear in the number of line elements N.

To get a rough idea of the empirical performance of our DP solver, we use the execution time of the numerical example presented in Section 5 as a benchmark. In this scenario, $J = 200$, $T_{\max} = 4800$ (seconds), and $N = 66$. On the server equipped with a 3.16GHZ Intel Xeon CPU, it takes only 1.3 seconds on average to solve the resulting instance of the dynamic programming model. Since the computational effort is linear in N, even for production line that has ten times more line elements, we expect be able to obtain optimal solutions within tens of seconds.

4 Handling Uncertainty: Event-Triggered Re-optimization

4.1 An Intractably Large MDP Formulation

The DP model described in section 3.3 is extremely efficient. Even for instances of realistic sizes, optimal solutions can be found within seconds. Unfortunately, due to its deterministic nature, if the identified solution is implemented in an uncertain environment its performance will deteriorate. The fulfillment of the end-state goals will be largely unaffected by the uncertainty in the production line. However, the production line shutdown time could be delayed notably due to unexpected breakdowns and repairs, and such delays could incur significant increases in overtime costs (and, if production time exceeds $T_{\max}$, affect fulfillment of the end-state goals as well). Such surge in overtime costs is the major reason for deterioration of solution performance.

A traditional approach to handling uncertainties in a sequential decision problem is to model it as a Markov decision process (MDP). To achieve this we would need to identify states of the system that exhibit the Markov property (i.e., the computation of the optimal action for a particular state requires no information on how that state is reached). To properly handle uncertainties in the end-state planning problem, such state should at least contain three pieces of information: 1) the condition of each line element, e.g, "up", "down", or "stopped"; 2) the content of each line element and its corresponding remaining cycle time; and 3) current time. This state definition overlaps with the DP state defined earlier in section 3.3, but is significantly larger. To illustrate just how intractably large such a state space is, we consider the same numerical instance mentioned in section 3.3.2: a production line with 66 line elements, 200 jobs, 4800 time periods (each time period is 1 second). For simplicity, we assume the cycle time to be 60 seconds for each and every line element. The size of the state space based on this moderate example is:

$$4800 \cdot 3^{66} \cdot 60^{66} > 10^{152} \text{ states,}$$

and we have not even incorporated the contents of the line elements yet. The sheer size of such a state space will make any straightforward implementation effort infeasible.

To develop an approximate solution method for the end-state planning problem with uncertainty, we first make the following observations: 1) Uncertainties of the production line will not affect the fulfillment of the end-state goals (unless production time exceeds $T_{\max}$), but will delay the shutdown time, leading to increase in penalty for overtime. 2) If a line manager monitors the status of the production line, and can make adjustments to the shutdown schedule, the manager would give up some end-state goals if their rewards compensate for the induced overtime cost. From the above two observations, we conclude that, if real-time status of the production line (e.g., breakdowns and repairs) can be observed, the shutdown policy could be updated to react to the consequences of these events. Using this approach a manager can prevent excessive overtime in situations when attainment of some end-state goals becomes unprofitable as a result of unforeseen delays on the production line.

To incorporate real-time production line information, one could periodically take a "snapshot" of the line and update and re-solve the DP. However, to ensure sufficient resolution in capturing the occurrences of unforeseen events, the frequency of such updates must be fairly high, and this might lead to redundant computations. We thus propose an event-triggered re-optimization procedure that will only update and re-solve the DP when specific significant disruptive events occur on the production line. In particular, we should only focus on events that would affect the DP. These events and the re-optimization procedure are described in the next subsection.

4.2 An Event-Triggered Re-Optimization Procedure

As noted in the previous subsection, only events that could potentially change the DP should be considered in our re-optimization procedure. One such event is a breakdown of a line element. Additionally, inaccurate estimates of line element repair time should also be monitored, since they affect the *effective cycle time*, which we define as the sum of the actual cycle (i.e., processing) time and the repair time for the corresponding line element. Inaccurate repair time estimates can be detected under two circumstances: *overestimation* and *underestimation*. The formal definitions and the necessary follow-up steps for the above three events are as follows:

Breakdown This event occurs when a line element breaks down. When a line element breaks down, its repair time is estimated and used in the estimate of the effective cycle time. I.e., if the line element n is processing job j at the time of its breakdown, its effective cycle time for job j is calculated as $(t_{j,n} + \hat{r}_{j,n})$, where $\hat{r}_{j,n}$ is an estimate of the repair time. The breakdown event and the above mentioned statistics can be captured and measured by most Computer

Integrated Manufacturing (CIM) systems (e.g., see Davies (2006)).

Underestimation of repair time This event is detected if a faulty line element is still under repair after its estimated repair time lapses. If this line element n was processing job j at the time of its breakdown, a *new* value of the repair time estimate $\hat{r}_{j,n}$ will be generated and used to replace the previous estimate in the calculation of the effective cycle time for job j.

Overestimation of repair time This event is detected if a faulty line element is repaired earlier than estimated. If the repaired line element n was processing job j at the time of breakdown, its effective cycle time for job j should be updated to $(t_{j,n} + r_{j,n})$, where $r_{j,n}$ is the actual observed repair time, which replaces the latest estimate $\hat{r}_{j,n}$.

We can now formally describe the event-triggered re-optimization procedure. At the beginning of the planning horizon ($t = 0$), an initial shutdown plan is generated by solving the DP with default cycle times (these cycle times can take into account expected breakdowns and associated repair times). This initial plan is taken to be the **incumbent** shutdown plan. The line manager will execute the incumbent shutdown plan until one of the three aforementioned disruptive events is detected at one of the line elements. As soon as an event occurs, the manager should update the effective cycle time at the corresponding line element and re-optimize. (Note that all previously shut down line elements should remain shut down during this re-optimization, which can be achieved by applying steps described in Section 3.3.1.) This new shutdown plan will become the incumbent plan, which the manager will implement until the next event occurs, or the line is shut down.

Although line managers are being assigned the duties of monitoring events, executing re-optimization, and implementing the incumbent plan in the above description, all three tasks could be automated if the plant is equipped with sufficiently capable CIM system. The critical capabilities required for such automation are the data link for real-time production line statues update (for event monitoring) and the programmable controller that can be controlled remotely (for executing shutdown command). Both functionalities are available in most modern CIM systems, including the one installed at our partner's plant

Despite its simplicity, this event-triggered re-optimization approach is very effective; as we will demonstrate in the next section using real-world-inspired numerical cases, *in many instances, it can discover shutdown policies that are as good as the ones obtained with perfect hindsight.*

5 Computational Experiments

A hypothetical yet realistic end-state scenario from an assembly plant of a major automotive manufacturer is presented in this section. The line configuration and parameters such as cycle times and capacities in this scenario reflect those observed on a production line in this plant, and the build schedule is randomly generated according to the proportion of job types produced there. The end-state requirements are constructed based upon discussions with plant personnel about typical situations that they experience. To preserve the confidentiality of the plant operations, the values of parameters v_n, p_n, p_o and p_l used in the scenario are rescaled and made unitless. However, these values are chosen in an effort to preserve the relative proportions among corresponding parameters.

In discussions with plant personnel, 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 and quantifying 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 numerical values and penalties with the goals at hand. To estimate the reward 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, and 4) the likelihood and cost of delaying the launch of a new product resulting from late installation or calibration of production equipment. At present, 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 based upon experience. 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 time. Despite this, these computational examples demonstrate that significant improvement in goal attainment can be achieved with minimal sacrifice of production time. (Moreover, in our experiments the performance of our procedure did not prove to be particularly sensitive to small changes in these parameter values.) 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.

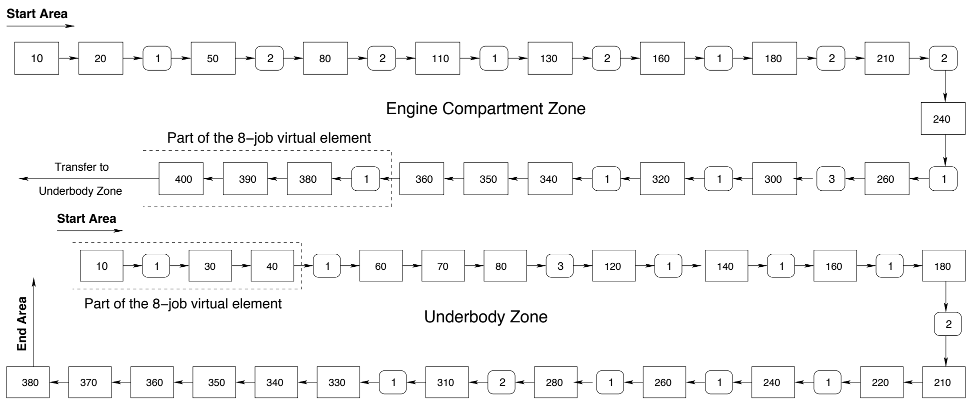
The rest of this section is organized as follows. We first introduce the background and description of a typical shutdown scenario. Subsequently, we define the *rule-of-thumb* shutdown policy as a comparison baseline and solve the end-state planning problem in a deterministic setting, pointing out operational insights one could obtain with our model. Finally, we make the scenario stochastic by introducing probabilistic breakdowns and repairs. The stochastic version of the problem is solved by the event-triggered re-optimization procedure described in Section 4, and its effectiveness is measured by comparing its performance to the performance of a shutdown schedule determined with perfect hindsight.

5.1 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 Figure 2. In Figure 2, 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 line configuration, cycle times, and capacities of that line segment. Together these two zones will be treated as a serial production line.



Start Area
Engine Compartment Zone
Part of the 8–job virtual element
Transfer to Underbody Zone
Start Area
Part of the 8–job virtual element
Underbody Zone
End Area

Figure 2: Schematic graph for the engine compartment and underbody zones.

We define eight high-level objectives we would like to achieve during a downtime period and describe end-state goals based on each of these objectives. The goals are designated 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, 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 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 and 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 20 minutes of overtime are allowed, so $T_{\max} = 4{,}800$ 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 (7)), 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 compare different solution approaches.

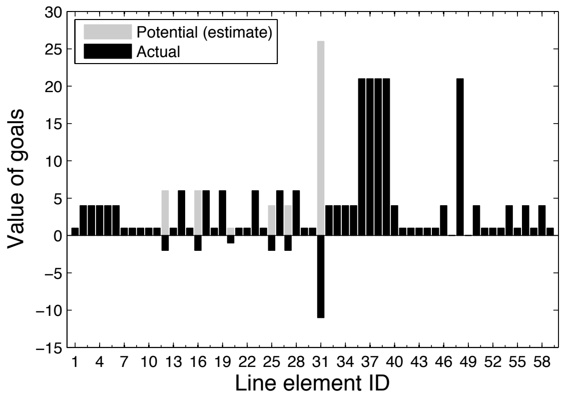
5.2 Experimental Results: Deterministic Case

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

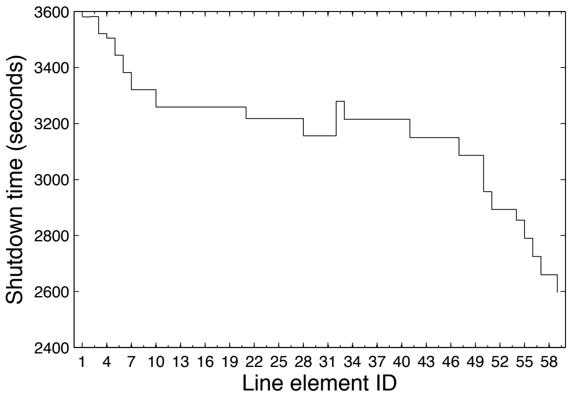
To generate a problem instance, we created a build schedule by sampling jobs at random in accordance with the given percentages of the four job types. The resulting problem was solved by the DP algorithm described in Section 3. The optimal policy found stops the production line at T_s =3,705 seconds (105 seconds later than the desired time) and achieves 68 of the 92 goals defined. The 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 cost due to a shutdown time that deviates from the desired shutdown time, was 197.25.

Recall that, 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 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 the specifics of the build schedule. Figure 3a shows these estimates of potential achievable 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. A visible 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 3b 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



(a)



(b)

Figure 3: (a) Estimate of potential achievable value, and value achieved by the optimal policy, for each line element. (b) Shutdown time for each line element.

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 to a representative policy that is commonly adopted by managers. 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 rely on rule-of-thumb (ROT) guidelines. For example, 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 you can." 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 you can" 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 value of the formal ROT policy will provide an upper bound on the 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 time costs to extremely high values. The resultant shutdown plan would meet goals optimally subject to having T_s as close to T_d as possible, and its 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 overall value of the policy 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.

5.2.2 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 structure and performance of any shutdown policy depend on the build schedule defining the problem instance. To assess whether the quality of the results we reported in the previous subsection is affected by the build schedule, we extended the experiments 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.

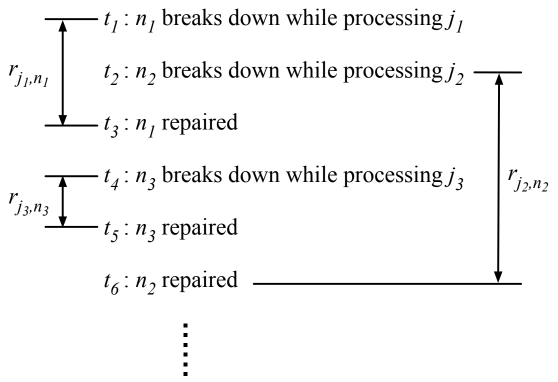
5.3 Experimental Results: Stochastic Case

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 configuration 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. We will model the number of cycles between breakdowns for each line element 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 plant which served as a basis for the example discussed in Section 5.1.

The experiment was conducted as follows. A build schedule was generated, and an initial 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 this 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 generated 100 sampled scenarios of operations by sampling breakdown and repair times according to the probability distributions derived from the real-world data. A sampled scenario of operations is essentially a list of time-stamped breakdowns and repair completions (see Figure 4). This information, together with the original cycle times, allows us to compute the effective cycle times and subsequently, a new flow matrix corresponding to the sampled scenario.

The performance of the basic policy in a particular sampled scenario can be evaluated using the corresponding flow matrix. In most cases, the set of satisfied goals will remain the same for different flow matrices, while the shutdown time will change. However in some cases line elements will have to be shut down at $T_{\max}$ rather than upon releasing the job specified by the policy.

As a point of comparison, we also solved the DP for each of the flow matrices corresponding to each sampled scenario of operations. Such **optimal policy with hindsight** (OPH) provides an upper bound on the value that can be attained by *any policy* in this stochastic setting, since it assumes full hindsight, i.e., a priori knowledge of the timing of breakdowns and repair times of line elements during the production process. For each sampled scenario, a ratio between the values



$t_1 : n_1$ breaks down while processing j_1
$t_2 : n_2$ breaks down while processing j_2
$t_3 : n_1$ repaired
$t_4 : n_3$ breaks down while processing j_3
$t_5 : n_3$ repaired
$t_6 : n_2$ repaired
r_{j_1,n_1}
r_{j_3,n_3}
r_{j_2,n_2}

Figure 4: An example of a sampled scenario of operations.

of the basic policy and the OPH was computed. The average of these ratios (over 100 sampled scenarios) characterizes the performance of the basic policy for this build schedule.

The above experiment was repeated for 50 different build schedules. For the fifty build schedules examined, the basic policy on average achieved 52.7% to 88.4% (with a mean of 72.3%) of the value attained by the OPH. The formal ROT policy, if we grant it full hindsight on the same sampled scenarios, on average achieved 74.4% to 89.0% (with a mean of 83.3%) of the value attained by the OPH. As predicted, the performance of the basic policy deteriorates due to the fact that it sometimes generates significant overtimes due to unexpected breakdowns and repairs.

Finally, we executed the event-triggered re-optimization (ER) procedure for each sampled scenario. By simply updating the flow matrix and re-solving the DP when disruptive events occurred, the performance of the shutdown policy improved significantly: *it on average achieved 91.8% to 98.5% (with a mean of 95.6%) of the value attained by the OPH.* Moreover, in 47.7% of the cases (build schedule/sampled scenario combinations), the ER procedure produced a result that matched the value of the OPH for that case. Performance of the three approaches is summarized in Table 1.

Table 1: Performance of the basic, formal Rule-of-Thumb, and event-triggered re-optimization policies, (as percentage of value attained by the optimal policy with hindsight), on 50 build schedules.

	Avg.	Std. Dev.	Min.	Max.	Matches OPH[1]
Basic policy	72.3%	7.3%	52.7%	88.4%	1.9%
ROT w/ hindsight	83.3%	3.1%	74.4%	89.0%	3.8%
ER procedure	95.6%	1.7%	91.8%	98.5%	47.7%

[1] Percentage of cases in which the policy attains the same value as OPH.

5.3.1 Why Does Re-Optimization Work?

The success of the event-triggered re-optimization procedure is due to its ability to dynamically adjust the shutdown schedule in order to avoid exceedingly late shutdown times due to unexpected breakdowns and long repairs. To illustrate this, we present a snapshot from a typical sampled scenario of operations for one of the built schedules (see Table 2). This scenario contains 53 events (namely, 24 breakdowns, and 22 overestimations and 7 underestimations of repair times). In Table 2, we present performance estimates of various policies calculated at three time points in this scenario. In row 1 of the table, we report performance estimate of the basic policy as calculated in the first step of the re-optimization procedure based on default cycle times at time $t = 0$. In rows 2 and 3 we report performance estimates of the basic policy and the event-based re-optimization (ER) policy, respectively, as calculated immediately after Event 32. Finally, in rows 4 and 5 we report the final performance characteristics of these two policies as calculated upon line shutdown. (Rows 6 and 7 contain values of the two perfect hindsight policies — formal ROT and OPH — considered in this paper.) Comparing values attained at shutdown to the initial estimate (row 1), we see that the basic policy suffered greatly due to unexpected breakdowns (a loss of almost 34%), while the ER procedure resulted in performance loss of only about 4%. Despite the fact that the ER approach is reactive, in this scenario it performed as well as the optimal policy with hindsight. By plotting estimates of shutdown time calculated after each event (see Figure 5), we

Table 2: Example: Evolution of performance estimates for a particular scenario of operations.

#		Value from Goals	Shutdown Time	Value from Time	Total Value
1	Basic policy (at $t = 0$)	164	3,582	-3	161
2	Basic policy (after Event 32)	164	4,292	-57.7	106.3
3	ER procedure (after Event 32)	156	3,606	-0.5	155.5
4	Basic policy (at shutdown)	164	4,292	-57.7	106.3
5	ER procedure (at shutdown)	156	3,625	-2.1	153.9
6	ROT w/ Hindsight	140	3,600	0	140
7	OPH	156	3,625	-2.1	153.9

can see that estimated performance of the basic policy deteriorated the most between Event 27 and Event 32 (the shutdown time was delayed by 16% in the span of these 6 events). Event 27 was a major breakdown at line element 10, lasting approximately 500 seconds, resulting in three

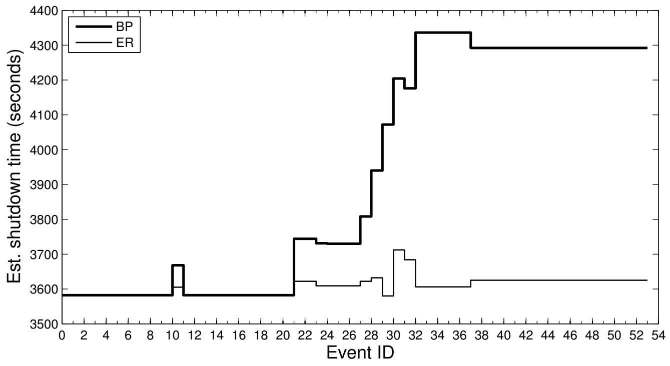
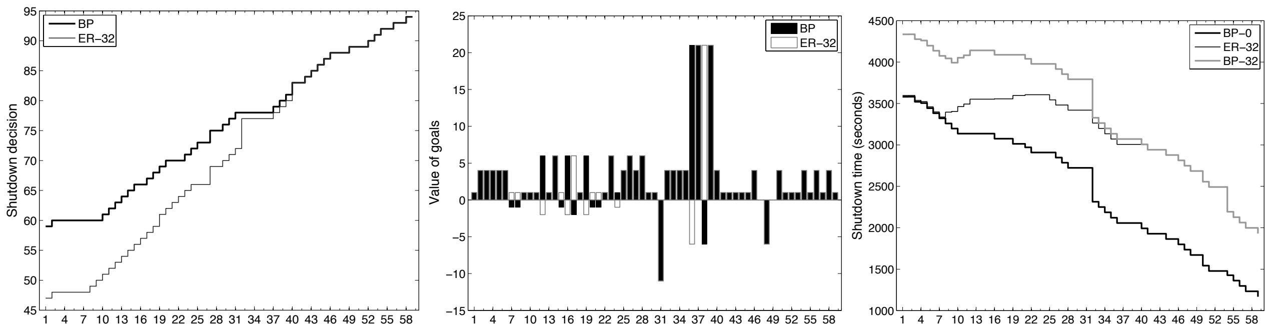


Figure 5: Estimates of shutdown times under basic policy (BP) and event-based re-optimization policy (ER), updated after each event

underestimations and one overestimation of its repair time. Event 32 was another major breakdown at line element 12, lasting about 225 seconds. Although estimated performance of the ER policy temporarily deteriorated during the same time period, the re-optimization procedure was able to make the necessary adjustments to bring the shutdown time back, nearly matching its desired value. From Figure 6a, we can see that this was achieved by shutting down line elements 1 through 32 earlier by approximately 6 to 12 jobs. Such adjustments resulted in lesser goal values (as shown in Figure 6b), but avoided late shutdown times for line elements 1 through 32 (see Figure 6c).



(a) Shutdown decisions. (b) Value from goals. (c) Shutdown times.

Figure 6: In all three charts, line element IDs are plotted on the horizontal axis. BP and ER refer to the “basic policy” and “event-triggered re-optimization policy,” respectively. BP-32 and ER-32 refer to performance estimates of these policies calculated immediately after Event 32.

6 Conclusions

We have shown that a dynamic programming model can be used in making complex decisions involved in shutting down elements of a production line, considering end-state goal fulfillment and costs of overtime and lost production time. Although this model is deterministic in nature, it can be used within an even-triggered re-optimization procedure, producing superior numerical results even in situations where line elements are subject to stochastic breakdowns and repairs. A software implementation based in part on the work described in this paper was developed and is in the process of being deployed as part of a pilot installation at an assembly plant belonging to a major automotive manufacturer.

This work lays the foundation for additional future research. One of the most important areas of future exploration is the consideration of non-serial production lines. A model that considers non-serial production will be able to account for line configurations such as merging of sub-assemblies into a main line and parallel production, and is the subject of our current research.

Yet another opportunity exists in a joint optimization of the shutdown policy and the build schedule. Section 5 demonstrated 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.

7 Acknowledgments

This work was supported in part 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. The authors would like to thank Michael Wellman for his detailed comments and suggestions.

References

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

Cho, D. I. and Parlar, M. (1991) A survey of maintenance models for multi-unit systems. *European Journal of Operational Research* **51**, 1–23.

Davies, S. (2006) Listening to the factory. *Computing and Control Engineering* **17**(6), 38–43.

Denardo, E. V. (1982) *Dynamic Programming.* Prentice-Hall.

Duffuaa, S. O. , Raouf, A. , and Campbell, J. D. (1998) *Planning and Control of Maintenance Systems: Modeling and Analysis.* John Wiley & Sons, Inc.

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

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

Pierskalla, W. P. and Voelker, J. A. (1976) A survey of maintenance models: The control and surveillance of deteriorating systems. *Naval Research Logistics Quarterly* **23**, 353–388.

Samaranayake, P. , Lewis, G. , Woxvold, E. , and Toncich, D. (2002) Development of engineering structures for scheduling and control of aircraft maintenance. *International Journal of Operations & Production Management* **22**(8), 843–867.

Sherif, Y. S. and Smith, M. L. (1981) Optimal maintenance models for systems subject to failure: A review. *Naval Research Logistics Quarterly* **28**, 47–74.

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