Maintenance Scheduling for Modular Systems: Modeling and Algorithms

Retsef Levi,[1] Thomas Magnanti,[2,3] Jack Muckstadt,[4] Danny Segev,[5] Eric Zarybnisky[6]

[1]*Sloan School of Management, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, Massachusetts 02139*

[2]*Institute Professor, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, Massachusetts 02139*

[3]*President, Singapore University of Technology and Design, 20 Dover Drive, Singapore 138682*

[4]*School of Operations Research and Information Engineering, Cornell University, Ithaca, New York 14853*

[5]*Department of Statistics, University of Haifa, Haifa 31905, Israel*

[6]*Operations Research Center, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, Massachusetts 02139*

Received 18 July 2012; revised 11 July 2014; accepted 13 July 2014
DOI 10.1002/nav.21597
Published online 31 July 2014 in Wiley Online Library (wileyonlinelibrary.com).

Abstract: We study new models of scheduled maintenance management for modular systems, consisting of multiple components with respective cycle limits. The cycle limit of each component specifies the time interval in which this component must be repaired or replaced. The goal is to compute a feasible maintenance schedule that minimizes the cost associated with component maintenance. Applications of these models arise in Air Force aircraft maintenance as well as in other arenas with required preventive maintenance. The typical cost structures that arise in practical settings are submodular, which make the resulting models computationally challenging. We develop two efficient and operationally tenable approximation algorithms. We prove constant factor worst-case guarantees for both algorithms, and present computational experiments showing that these algorithms perform within a few percent of optimality on operationally relevant instances. © 2014 Wiley Periodicals, Inc. Naval Research Logistics 61: 472–488, 2014

Keywords: approximation algorithms; maintenance scheduling; submodular costs; cycle-limited components

1. INTRODUCTION

1.1. Maintenance in Modular Systems

Many modern systems consist of multiple components arranged in a modular design. Performing maintenance on (or replacing) a component requires removal of the module containing the component. Once the component is repaired and returned to a serviceable condition, the repaired component and module are reinserted into the system. In many cases, the system design dictates that the removal of one module requires removal of other modules. For instance, the F100 engine, a U.S. Air Force engine used in fighter aircraft, is composed of five major modules: The fan module, the core module, the fan drive turbine module, the augmentor/exhaust module, and the gearbox module. If maintenance is required on a component in the fan drive turbine module, the augmentor/exhaust module must be removed to access the fan drive turbine module [6]. Such maintenance actions are costly and time consuming due to the required teardown, maintenance, reassembly, and testing. For the F100 engine, this process can take 12 to 21 days without considering inventory or manpower delays [1]. For an aircraft engine, frequent or extended system downtimes have a negative effect on mission capability rates for the corresponding aircraft. In addition, maintenance activities can require specialized manpower and equipment which are limited.

We consider the management of scheduled maintenance activities for modular systems, such as the aforementioned

Correspondence to: Danny Segev (segevd@stat.haifa.ac.il)

aircraft engine, in which every system component has a cycle limit that specifies the maximum number of cycles it can be used between successive maintenance actions. For example, a cycle for the starter system in an aircraft engine could be one startup sequence. For components in an aircraft braking system, a cycle could be one landing sequence. Each component can be used for a certain number of cycles and then must be repaired or replaced due to safety or failure concerns. These cycle limits are determined through a number of methods including physical testing, simulation, and analytical assessment. Although it is possible that components fail prior to their cycle limits, due to the conservative nature of these cycle limits, such events are extremely rare. As a result, it is common to assume that a component is operational until its cycle limit is reached and that after maintenance it has a full cycle limit once again.

While the systems we consider operate in continuous time, a sortie or day of usage will consume a given number of cycles. In most cases, the rate by which cycles are used has very little variability. Accordingly, we assume, by normalization if necessary, that the cycle limits are all integer multiples of a given time epoch, which we call a period. Without loss of generality, we assume that after normalization one cycle occurs in each period. With these definitions in place, the goal is to devise a maintenance policy of minimum cost over a planning horizon of finitely, or infinitely, many discrete periods. Put differently, the objective is to determine the subset of components to be maintained in each time period such that all components are maintained within their respective cycle limits and such that the total cost (or long run average cost) is minimized. The specifics of our models, along with some necessary notation, are provided in Section 2.

1.2. Cost Structure

The missing ingredient in the above discussion is an explanation regarding the perperiod costs we incur throughout the planning horizon. Instead of focusing on specific functions, and potentially losing the ability to draw wider conclusions later on, we use a general (and realistic) way to model the cost of maintaining a subset of components, through cost functions that are nonnegative, increasing, and submodular.[1]

Submodular cost structures arise in several maintenance contexts in the U.S. Air Force. As explained in Section 5.1, one way to model the maintenance costs in modular systems is through an out-rooted directed tree. Figure 1 shows a partial example for the F100 engine. Directed arcs in the tree correspond to the modular dependencies that exist between the connected nodes, that is, the parent node must be removed to repair any of its child nodes. A dependency path from the root to a leaf describes the order in which modules must be removed and replaced to maintain a specific component. Each node has an associated cost that reflects the time, manpower, and/or inventory utilization required to perform the specific maintenance activity. To perform maintenance on a component, all modules on its dependency path must be removed in the order specified by the path. Therefore, the cost of a maintenance activity for a component is the sum of the costs of all nodes on the dependency path for that component. Once all parent modules have been removed and the maintenance on the component is completed, the modules are reassembled in the reverse order of the dependency path. Due to the modular construction, we assume, without loss of generality, that maintenance is required only on leaves of the dependency tree,[2] corresponding to cycle-limited components. As we note later, the resulting cost function is indeed submodular.

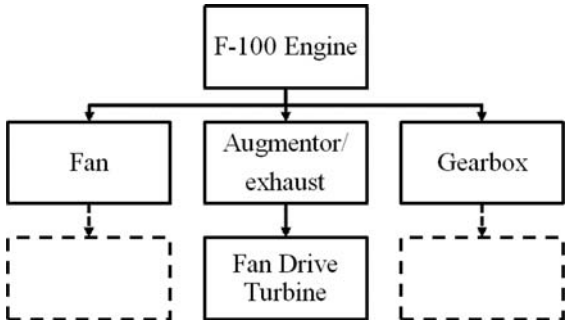

Figure 1. Partial dependency tree for the F100 engine.

Another submodular cost function of interest arises in situations, where the cost incurred by maintaining a subset of components in a given time period is the maximum cost of all the components included in the subset. Returning to the earlier tree example, if the node costs represent time, the time required for a maintenance action is computed by considering the length of the longest path between the root and each of the leaves (components) included in the maintenance action. In particular, the length of time required for the maintenance action is equal to the length of the longest path. This cost function implicitly assumes that maintenance actions occur in parallel for unconnected modules and components.

1.3. Practical Significance

Next, we discuss several real-life settings within the U.S. Air Force in which the models and algorithms studied in this article could be of use. During the design and testing phase of a modular system, a development team could use the models and algorithms we propose to explore the short

[1] See Definition 2.1 in Section 2.

[2] This assumption can be easily enforced by connecting each internal node that originally requires maintenance to a dummy leaf node, which inherits its cycle limit (with zero cost).

and long term implications of changing various design parameters such as early investment in maintenance capability to reduce perincident maintenance costs or increasing component cycle limits to decrease the future sustainment costs for the system. Once a system has been fielded, operational considerations or challenges might affect the maintenance philosophy codified during the development phase. During this operational phase, actual maintenance costs or costs from an auxiliary optimization problem (i.e., linear programming dual variables) could be used to produce feasible and low cost maintenance schedules. Our models and algorithms could also be used as building blocks within more comprehensive operational maintenance and inventory management models.

1.4. Additional Real-Life Applications

While the motivation for this work is grounded in current Air Force issues, there are numerous other examples of complex systems that require preventive maintenance (due to safety concerns or high uptime requirements) and have a modular structure (or, more generally, submodular costs). One such example is the generators used for electrical power production in hydroelectric dams or other power generation facilities. For such systems, the downtime late at night serves as an opportunity to perform preventive maintenance to ensure high service levels during times of peak power demand. Another example is large factory production equipment that is preventively maintained in preparation for high demand seasons.

1.5. Our Results

As explained in Sections 2 and 3, which provide a formal definition of our model and a detailed literature review, the modular maintenance scheduling problem bears some similarity to a number of inventory management models, where finding the optimal policy appears to be computationally challenging. As demonstrated by numerical experiments, the optimal policy can be very complex, and its structure might not provide the operational simplicity that is essential for implementation in many practical settings. A natural approach that is often considered in practice is to make use of cyclic (or frequency-based) policies that maintain each component at a fixed frequency. When one is interested in adopting this approach, the overall hope is that it would be possible to efficiently compute such policies, either exactly or approximately, and to bound their cost in comparison to arbitrarily-structured policies.

The main contribution of this article is in developing two efficient and operationally tenable approximation algorithms that compute provably-good cyclic policies for the modular maintenance scheduling problem with submodular costs. We derive constant-factor worst-case guarantees for both algorithms, and present computational experiments showing that these algorithms perform within a few percent of optimality on operationally relevant instances. These algorithms provide easily implementable schedules that meet the operational considerations of the maintenance community. Our main results can be briefly summarized as follows:

- **Lower bound**. In Section 4, we develop a general method to evaluate the worst-case performance of cyclic policies, based on a comparison to a lower bound on the cost of any feasible policy, which will be used to derive the approximation guarantees of our algorithms. In essence, this method describes a charging scheme that allows one to attain an upper bound on the resulting performance guarantees in terms of the maximum (multiplicative) increase in the maintenance frequency of any component. It is important to mention that our lower bound is crucially dependent on having time-invariant costs; when this model assumption does not hold, we prove in Appendix A that the problem becomes set-cover hard to approximate.
- **Cycle rounding algorithm**. In Section 5, we first consider (and define more formally) the dependency-tree models discussed earlier, with either path-additive or longest-path costs. For this setting, we develop the cycle rounding algorithm, in which the maintenance frequencies of the various components are computed by iteratively rounding their cycle limits. This algorithm has a worst-case performance guarantee of two for finite and infinite horizon instances. That is, for any instance, the resulting solution is guaranteed to have a cost at most twice the cost of an optimal policy, which is not necessarily cyclic.
- **Shifted power-of-two algorithm**. In Section 6, we proceed by considering general submodular cost functions, and develop a second algorithm, that computes shifted power-of-two policies, where each component is maintained at a frequency that is a power of 2 times a specified base (or shifting parameter). We identify a small class of shifted power-of-two policies (of size less than or equal to the number of components) that depend only on the cycle limits and not on the cost parameters. Moreover, the least expensive of these policies is guaranteed to have a cost within $1/\ln(2) \approx 1.44$ of optimal for infinite horizon instances. This is in fact true for *all* possible increasing submodular cost functions. An alternative way to derive an approximation ratio of $1/\ln(2)$ is given in Appendix B.
- **Experimental results**. In Section 7, we conduct extensive computational experiments, where the cycle rounding and shifted power-of-two policies are

shown to perform within a few percent of optimal for many realistic instances. As a by-product, we demonstrate that cyclic policies indeed perform in practice significantly better than their theoretical worst-case guarantees.

2. THE MODULAR MAINTENANCE SCHEDULING PROBLEM

Consider a system with a finite set of components, denoted by $\mathcal{C}$, that must be maintained over a discrete-time planning horizon of either T or infinite number of periods. Each component $i \in \mathcal{C}$ has a component-specific (integer) cycle limit f_i that specifies the maximum allowable number of periods between two successive maintenance actions of this component. Each time a component is repaired/replaced, it starts with a full cycle limit. We assume without loss of generality that $1 < f_i < T$ and that the components are numbered in nondecreasing order of their cycle limits (i.e., $f_1 \leq \cdots \leq f_{|\mathcal{C}|}$). To justify the former assumption, note that, if $f_i = 1$ for some component i, then this component can be scheduled for maintenance in every time period over the planning horizon and removed from the problem. In addition, if $f_i \geq T$ for some component i, then this component does not need to be maintained during the planning horizon and can be eliminated as well.

We assume that all components have a full cycle life at the beginning of the first period in the planning horizon. At the beginning of each period, a decision is made about which components, if any, should be maintained in that period. Our assumption is that maintenance occurs instantaneously and the associated costs are incurred, justified by the practical observation that such actions are typically much shorter than the duration of a single (normalized) time period.[3] This assumption follows directly from the system requirement that all components should be functional. After any maintenance actions have been carried out, the system is used for one period and the remaining cycles for all components are decremented by one.

The objective is to compute a feasible maintenance schedule that minimizes the total cost for finite horizon problems, or long run average cost for infinite horizon problems. More specifically, a feasible schedule is one where component i maintenance actions are scheduled no more than f_i periods apart, for every $i \in \mathcal{C}$. In addition, the cost of a given schedule is defined as the sum of maintenance costs incurred in each period over the planning horizon, or the long run average cost in infinite horizon models. It remains to explain how perperiod costs are structured. To this end, for each subset of components $S \subseteq \mathcal{C}$, let $K(S)$ be the total cost of maintaining these components in a given time period. We assume that $K : 2^{\mathcal{C}} \to \mathbb{R}_+$ is a nondecreasing submodular cost function (see definition below, and more generally, [17, Chap. 44]), for which, without loss of generality, $K(\emptyset) = 0$.

DEFINITION 2.1 (Submodular function): A real-valued function K defined on subsets of a ground set $\mathcal{C}$ is submodular if, for any two subsets S and S',

$$K(S) + K(S') \geq K(S \cup S') + K(S \cap S').$$

An equivalent definition focuses on economies of scale. From this perspective, we say that K is submodular if, for every $S \subset S' \subset \mathcal{C}$ and $i \in \mathcal{C} \setminus S'$,

$$K(S \cup \{i\}) - K(S) \geq K(S' \cup \{i\}) - K(S').$$

3. LITERATURE REVIEW

Numerous researchers have examined maintenance models for multicomponent systems with economic and/or structural dependencies. In models with economic dependencies, the cost of maintaining a set of components does not equal the sum of individual maintenance costs for these components. In models with structural dependencies, components form a module that forces several related components to be maintained together. The existing literature in this field is extensive, and it is beyond the scope of this article to discuss all the relevant research contributions. Instead, we refer the reader to related surveys [7, 20] for a comprehensive review of previous work, and concentrate on highlighting a number of directly related articles below.

3.1. Maintenance of Stochastically Failing Systems

Most of the relevant literature focuses on infinite horizon problems with stochastically failing components, and attempts to find optimal maintenance schedules that minimize either downtime or maintenance costs. Two widely studied models in this context are the k-out-of-n model and a model with n constant failure rate (CFR) components and one increasing failure rate (IFR) component. The first model assumes the system consists of n identical components of which at least k must be functional for the system to operate. Most of the work in this area has focused on the case where $k < n$ and on developing policies that balance the downtime and preventive maintenance costs. A relatively small fraction of the work considers n-out-of-n systems with IFR components. In the second model, the decision to be made is when to perform preventive maintenance on the single IFR component, either when a CFR component fails or when the

[3] Alternative models, where maintenance actions are not necessarily instantaneous, and their computational hardness, are discussed in Section 8.

IFR component has reached a certain time in service. In this model, the goal is to minimize maintenance costs, which might differ depending on the state of the IFR component when maintenance is initiated.

3.2. Maintenance of Multicomponent Systems

In contrast, despite its operational relevance, very little progress has been made for more general settings, such as the maintenance of multicomponent systems with multiple setup activities. In fact, Kobbacy et al. [7] specifically state in reference to models that consider multiple setup activities over a finite horizon "We have found one article in this category...this is the first attempt to model a maintenance problem with a hierarchical set-up structure." In this article, van Dijkhuizen [18] studied the problem of grouping preventive maintenance actions in a multistep, multicomponent production system. He models the multicomponent production system as a tree with leaves representing components that require maintenance within specified frequencies. The model he develops assumes that maintenance actions can occur only at integer multiples of a selected, part-dependent, preventive maintenance frequency. Under this assumption, the preventive maintenance schedule repeats itself over the finite planning horizon. An integer programming formulation is then used to find the best cyclic policy. van Dijkhuizen and van Harten [19] proposed a combinatorial algorithm for the same model. For instances with a single common setup activity (i.e., when the underlying dependency tree is a star), they developed a polynomial-time dynamic programming algorithm to compute the optimal solution. For instances with multiple shared setup activities, they develop a branch-and-bound algorithm that terminates with the optimal policy but is not polynomial.

3.3. Relation to Inventory Models

The models we investigate in this article, as well as the shifted power-of-two algorithm proposed in Section 6, have some basic similarities with inventory models that have been studied in the 1970s and through the 1990s. Somewhat informally, each component can be thought of as a separate commodity, that is consumed at a fixed rate, while scheduling a maintenance action corresponds to placing an order. The cycle limits, restricting the number of time periods between successive maintenance actions, are equivalent to stating that the items ordered have (time-invariant) expiration dates. Here, we briefly describe a selected portion of the existing literature, whereas in Section 8, we point out several fundamental differences between our maintenance models and algorithmic methods and those studied in the context of inventory management.

The complexity of continuous-time infinite-horizon inventory models with stationary demand rates in a multistage production/inventory environment has prompted the development of specialized algorithms that focus on imposing certain structure on the solution space, specifically restricting attention to stationary-nested policies, with varying performance guarantees [4]. The effectiveness of power-of-two policies were first established in the seminal articles of Roundy [12] and Maxwell and Muckstadt [9]. They showed how to compute power-of-two policies that are within 1.06 of optimal, when one does not optimize the base (shift parameter), and within 1.02 when the shift parameters could be optimized. The analysis is based on a nonlinear relaxation of the problem, and the resulting policies highly depend on the respective cost parameters. Federgruen and Zheng [5] studied the lot-sizing/inventory models above, with a similar upper bound constraint on the size of the reorder interval, and showed, using techniques similar to the initial work of Roundy [12], that power-of-two policies are again within 1.06 of optimal. That being said, their model retains the additive cost structure as well as the lot sizing characteristic, where inventory can be held at intermediate nodes and thus higher level items can have reorder intervals that are longer than lower level items. With an additive cost structure, Roundy [13] studied the above inventory models with a lower bound capacity constraint that restricts the reorder interval from being too small, which is the "opposite" of our model, in the sense that we ask the reorder interval to be small enough. Interestingly, he showed that power-of-two policies are within $1/\ln(2)$ of optimal, obtaining a similar worst-case guarantee as we obtained for maintenance models with general submodular costs and upper bounds on the reorder interval. More recently, Teo and Bertsimas [17] considered similar models and replicated the aforementioned results with some generalizations. Unlike the classical results listed earlier, the techniques introduced by Teo and Bertsimas, with some observations and modifications, can be alternatively used to derive the $1/\ln(2)$ bound attained by the best-possible shifted power-of-two policy. We provide additional details on this issue in Appendix B.

4. PRELIMINARIES

In this section, we present a general method to lower bound the worst-case performance of any feasible policy, which will be subsequently used to derive the approximation guarantees of our algorithms. We also discuss the possibility of devising continuous-time schedules and converting them into discrete-time schedules without meaningful deterioration in their performance ratio.

4.1. Bounding the Approximation Ratio

4.1.1. Lower Bound

We begin by introducing a charging scheme that is used to allocate the total cost of each maintenance action among the

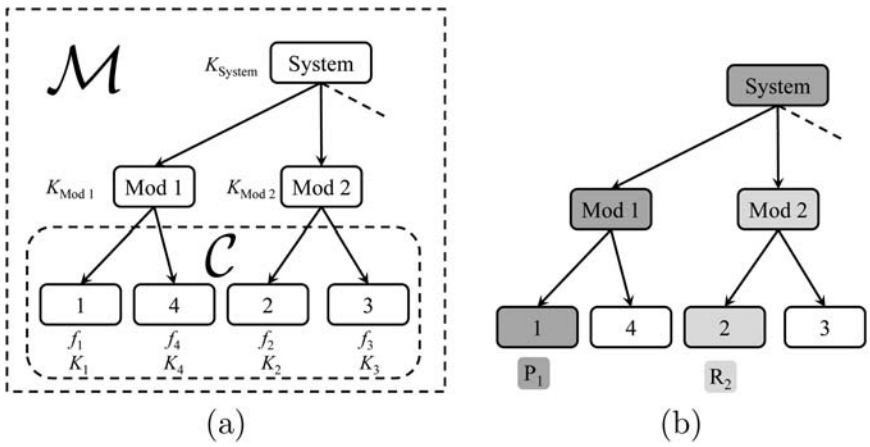

Figure 2. a: A schematic illustration of the sets $\mathcal{C}$ and $\mathcal{M}$, cycle limits, and maintenance costs for a system modeled by a tree. b: The dependency path P_1 of component 1 consists of the heavily shaded nodes, whereas the residual path R_2 of component 2 (see Section 5.2) consists of the light-shaded nodes.

components included in that action. This leads to a decomposition of the total cost of any policy by allocating it to components, and bares some similarity to solution concepts in cooperative game theory, such as the Shapley value of a coalition game [15, 11].

For a maintenance action including the set of components $S \subseteq \mathcal{C}$ and for each $i \in S$, let $S_i = S \cap \{1, 2, \ldots, i-1\}$ be the subset of lower indexed components that are included in this maintenance action, which in particular have a cycle limit that is smaller than or equal to the cycle limit of component i (recall that $f_1 \leq \cdots \leq f_{|\mathcal{C}|}$). Component i is then charged with the marginal cost of adding component i to a maintenance action including only S_i, and this quantity is denoted by $\psi_S(i)$. That is, i is charged $\psi_S(i) = K(S_i \cup \{i\}) - K(S_i)$. Since $K(\cdot)$ is nondecreasing, it follows that $\psi_S(i) \geq 0$, and moreover, since $K(\emptyset) = 0$, the full cost of the maintenance action S is charged to its components as

$$K(S) = \sum_{i \in S} (K(S_i \cup \{i\}) - K(S_i)) = \sum_{i \in S} \psi_S(i).$$

We also define the *residual cost* for component i, denoted by K^i, to be $K(\{1, 2, \ldots, i\}) - K(\{1, 2, \ldots, i-1\})$. Since the cost function $K(\cdot)$ is submodular, this is the minimum amount component i can be charged for any maintenance action, regardless of the other components maintained in conjunction with component i. Also, note that the definition of residual costs implies the existence of at least one component $m \leq i - 1$ for which

$$K^i = K(\{1, 2, \ldots, m\} \cup \{i\}) - K(\{1, 2, \ldots, m\}). \quad (1)$$

We use $m(i)$ to denote the minimal (index-wise) such component, and refer to it as the predecessor of i. For example, in Fig. 2b, Section 5.1, we have $m(4) = 1$.

With these definitions in hand, we establish lower bounds on the optimal cost for both finite and infinite horizon problems. These are stated in the following lemma.

LEMMA 4.1: The cost of an optimal solution, denoted by OPT, satisfies:

$$\begin{cases} \text{OPT} \geq \sum_{i \in \mathcal{C}} \left\lfloor \frac{T}{f_i} \right\rfloor \cdot K^i & \text{Finite horizon} \\ \text{OPT} \geq \sum_{i \in \mathcal{C}} \frac{K^i}{f_i} & \text{Infinite horizon.} \end{cases}$$

PROOF: For both lower bounds, we will use the observations that the residual cost K^i is the minimum amount component i can be charged when it is maintained, and that every feasible schedule must maintain each component i at least once every f_i periods. Specifically, given a finite planning horizon consisting of T periods, we can partition it into $\lfloor T/f_i \rfloor$ disjoint intervals, each of length at least f_i. In each of these intervals, component i should be maintained one or more times, incurring the minimum charge K^i per maintenance action, which yields the finite horizon result. In the infinite horizon setting, simply dividing by T and taking the limit as T goes to infinity yields the corresponding bound. □

Note that the lower bounds in Lemma 4.1 are also valid for continuous time models in which a maintenance action can take place at any point in time. Accordingly, the approximation results shown later in Sections 5 and 6 are valid even when we allow maintenance actions to occur at non-integral periods. From an operational standpoint, this allows us to consider relatively coarse time periods without significantly impacting the quality of the resulting maintenance schedule.

4.1.2. Nested Schedules

We define a nested schedule to be one in which, whenever any component i is maintained, it is charged exactly its residual cost, K^i. That is, for every component $i \in \mathcal{C}$ and for every maintenance action $S \subseteq \mathcal{C}$ in this schedule, we have

$$\psi_S(i) = K(S_i \cup \{i\}) - K(S_i) = K^i.$$

As shown next, the infinite horizon lower bound (given in Lemma 4.1) will permit us to obtain worst-case performance guarantees for nested and cyclic schedules by bounding the maximum ratio over all components between the frequency that component i is maintained and the minimum possible frequency, $1/f_i$.

THEOREM 4.2: Consider a cyclic nested policy with maintenance cycles $\hat{f}_i \leq f_i$ for each component $i \in \mathcal{C}$. In the infinite horizon model, the long run average cost of this schedule is at most $\max_{i\in\mathcal{C}}(f_i/\hat{f}_i)$ times the optimal long run average cost OPT.

PROOF: The maintenance frequency for each component i is $1/\hat{f}_i$. Since the schedule is nested, component i is charged exactly K^i per maintenance action and thus contributes $K^i/\hat{f}_i$ to the long run average cost. Therefore, the total long run average cost of this schedule is

$$\sum_{i\in\mathcal{C}} \frac{K^i}{\hat{f}_i} = \sum_{i\in\mathcal{C}} \frac{f_i \cdot K^i}{f_i \cdot \hat{f}_i} \leq \max_{i\in\mathcal{C}} \frac{f_i}{\hat{f}_i} \cdot \sum_{i\in\mathcal{C}} \frac{K^i}{f_i} \leq \max_{i\in\mathcal{C}} \frac{f_i}{\hat{f}_i} \cdot \text{OPT},$$

with the final inequality resulting from the lower bound on OPT, obtained in Lemma 4.1. □

4.2. Moving to Continuous Time and Back

Note that the worst-case performance guarantee in Theorem 4.2 is based solely on the scheduled maintenance frequencies and not affected by the cost parameters. Accordingly, we develop two approximation algorithms that compute cyclic and nested policies. While the first algorithm works in discrete time, we describe the second algorithm assuming that time is continuous. Therefore, the resulting maintenance schedules can be fractional, whereas in the original problem formulation maintenance actions may be scheduled only at discrete time periods. However, we can convert the resulting continuous time (fractional) maintenance schedule into a discrete time schedule with no increase in cost. For this purpose, consider a cyclic and nested continuous time schedule with maintenance actions scheduled for component i every $\hat{f}_i$ time periods. The corresponding discrete time schedule for component i will be

$$\left\lceil \hat{f}_i \right\rceil, \left\lceil 2 \cdot \hat{f}_i \right\rceil, \left\lceil 3 \cdot \hat{f}_i \right\rceil, \ldots.$$

This rounded schedule is feasible since the time between consecutive maintenance actions is at most f_i, as

$$\left\lceil (k+1) \cdot \hat{f}_i \right\rceil - \left\lceil k \cdot \hat{f}_i \right\rceil \leq \left\lceil k \cdot \hat{f}_i + f_i \right\rceil - \left\lceil k \cdot \hat{f}_i \right\rceil = \left\lceil k \cdot \hat{f}_i \right\rceil + f_i - \left\lceil k \cdot \hat{f}_i \right\rceil = f_i.$$

The first inequality holds since $\hat{f}_i \leq f_i$ and the first equality follows from the integrality of f_i. Since the fractional schedule was nested, the new discrete schedule is also nested. Consequently, at every maintenance action that includes component i, it will be charged only its residual cost K^i. In addition, the rounding procedure does not increase the number of maintenance actions for each component i. Therefore, the cost of the resulting schedule does not increase.

5. THE CYCLE ROUNDING ALGORITHM

In this section, we begin by formally defining the dependency-tree models mentioned in Section 1, with either path-additive or longest-path costs. For this setting, we propose a cycle rounding algorithm, that generates easy-to-compute cyclic policies, and is shown to provide provably-good maintenance schedules in the finite horizon setting. Specifically, the cost of these policies is guaranteed to be within a factor of two of the optimal cost, uniformly for all instances. It is worth noting once again that the optimal policy might not be cyclic and thus hard to compute and implement in practice.[4]

5.1. Dependency-Tree Models

EXAMPLE 1: Dependency tree with path-additive costs.

As previously explained, one very common submodular cost function can be defined on an out-rooted tree where the root corresponds to the entire system, the leaves correspond to components, and the other nodes in the tree correspond to modules, as demonstrated in Fig. 2. Let $\mathcal{M}$ denote the set of all modules and all components (leaves), including the entire system, and let P_i denote the dependency path for component i. That is, the path from the root to (and including) a leaf describes the order in which modules must be removed and replaced to maintain a specific component. Specifically, a module j belongs to P_i if and only if module j must be removed to perform maintenance on component i. Let $K_j \geq 0$ denote the cost to remove and replace module $j \in \mathcal{M} \setminus \mathcal{C}$ or to repair component $j \in \mathcal{C}$. In this case, the cost of maintaining a subset S of components is equal to the sum of module

[4] See additional discussion regarding computational hardness in Appendix A and Section 8.

removal/replacement and component maintenance costs over all modules and components that reside in the union of the dependency paths $\{P_i : i \in S\}$, that is,

$$K(S) = \sum_{j \in \cup_{i \in S} P_i} K_j. \quad (2)$$

It is easy to verify that this specialized cost function is submodular, nonnegative, and satisfies $K(\emptyset) = 0$. A more general version of this model, which is investigated in a follow-up article [8], could include systems that are represented by a directed acyclic graph, in which a module might depend on multiple higher level modules.

EXAMPLE 2: Dependency tree with longest-path costs.

Another example mentioned earlier for a submodular cost function of particular interest is the overall downtime cost. This could be defined on a tree. The dependency path P_i will be the same as defined above. However, in this setting, the cost K_j stands for the time required to remove and replace module $j \in \mathcal{M} \setminus \mathcal{C}$ or to repair component $j \in \mathcal{C}$. The cost of maintaining a subset of components S is equivalent to the longest path from the root to some component $i \in S$, that is,

$$K(S) = \max_{i \in S} \sum_{j \in P_i} K_j. \quad (3)$$

Again, it is not difficult to verify that this function is submodular.

5.2. Algorithm and Analysis

In what follows, we focus on the dependency-tree model with path-additive costs (Example 1), and then explain how to extend our results to longest-path costs (Example 2). The performance ratio we obtain holds in the finite horizon setting, which appears to be slightly more difficult to approximate than its infinite horizon counterpart. In fact, it is easy to verify that, for any fixed $\varepsilon > 0$, a performance guarantee of α in the finite horizon setting implies an $(\alpha + \varepsilon)$-approximation in the infinite horizon case.[5] Therefore, these results are incomparable with the ones for arbitrary submodular functions, as our shifted power-of-two algorithm (see Section 6) works only in the infinite setting. In addition, the cycle rounding algorithm offers a number of advantages from a practical point of view, as explained toward the end of this section, and seems to perform better in computational experiments (see Section 7).

5.2.1. The Algorithm

We describe a simple procedure, called cycle-rounding, that schedules maintenance actions iteratively, starting with component 1. As we shall show, the algorithm computes a nested, cyclic policy with a total cost at most twice the optimal cost in the finite horizon setting. Recall that the components are numbered in nondecreasing order of their cycle limits (i.e., $f_1 \le \cdots \le f_{|\mathcal{C}|}$), and that P_i denotes the dependency path for component i, connecting this node to the root of the dependency tree.

To describe the algorithm, define the residual path R_i of component $i \in \mathcal{C}$ to be component i and all modules that are on the dependency path P_i, excluding those that reside on dependency paths of the lower indexed components, $1, \ldots, i-1$. That is, $R_i = P_i \setminus (\cup_{i' < i} P_{i'})$. From the definition of R_i, we have $R_i = P_i \setminus P_{m(i)}$, where $m(i)$ is the predecessor of component i, as defined in Section 4.1 [see the discussion of Eq. (1)].

Let f_i^{CR} denote the respective maintenance cycles of the cycle rounding algorithm. That is, component i will be maintained every $f_i^{\mathrm{CR}} \le f_i$ periods. Next, we explain how to iteratively set the values of f_i^{CR} for all $i \in \mathcal{C}$. For $i = 1$, set $f_1^{\mathrm{CR}} = f_1$. For all components $i \ge 2$ in order, consider $f_{m(i)}^{\mathrm{CR}}$, which has already been computed since $m(i) \le i - 1$, and set f_i^{CR} to be the largest multiple of $f_{m(i)}^{\mathrm{CR}}$ that is still smaller or equal to f_i. That is,

$$f_i^{\mathrm{CR}} = \left\lfloor \frac{f_i}{f_{m(i)}^{\mathrm{CR}}} \right\rfloor \cdot f_{m(i)}^{\mathrm{CR}}.$$

At this point, one final tweak is needed. For each component i, if its last maintenance action is redundant (i.e., if the just-before-last action occurs at time $T - f_i + 1$ or later), we eliminate i from this action.

5.2.2. Analysis

Observe that for the dependency-tree model with path-additive costs, the residual cost K^i equals $\sum_{j \in R_i} K_j$, which is exactly what component i would be charged, given that its predecessor $m(i)$ is also maintained in the same time period. Moreover, by construction, the cycle rounding algorithm schedules maintenance actions for component i only in periods where component $m(i)$ is also maintained, since f_i^{CR} is a multiple of $f_{m(i)}^{\mathrm{CR}}$. Therefore, the cycle rounding algorithm generates a cyclic schedule which is also nested, as the charge of component i is always K^i. We proceed by showing that the cost of the schedule produced by the cycle rounding algorithm is at most twice the optimal cost. This proof is complemented by an example, demonstrating that our analysis is tight.

THEOREM 5.1: The cycle rounding algorithm guarantees an approximation ratio of 2 in the finite horizon setting.

PROOF: We have already argued that the schedule generated by the cycle rounding algorithm is nested. Therefore, due to the lower bound of OPT $\geq \sum_{i\in\mathcal{C}} \lfloor T/f_i \rfloor \cdot K^i$ given in Lemma 4.1, it suffices to show that every component $i \in \mathcal{C}$ is maintained at most $2 \cdot \lfloor T/f_i \rfloor$ times.

To this end, we first observe that $f_i^{\mathrm{CR}} > f_i/2$, since

$$f_i - f_i^{\mathrm{CR}} = f_i - \left\lfloor \frac{f_i}{f_{m(i)}^{\mathrm{CR}}} \right\rfloor \cdot f_{m(i)}^{\mathrm{CR}} = \left(\frac{f_i}{f_{m(i)}^{\mathrm{CR}}} - \left\lfloor \frac{f_i}{f_{m(i)}^{\mathrm{CR}}} \right\rfloor \right) \cdot f_{m(i)}^{\mathrm{CR}} < f_{m(i)}^{\mathrm{CR}} \leq f_i^{\mathrm{CR}}.$$

Having established this inequality, note that the planning horizon $1, \ldots, T$ can be divided into $\lfloor T/f_i \rfloor$ disjoint intervals of length exactly f_i, and an additional interval strictly less than f_i, located at the end of the schedule. Since $f_i^{\mathrm{CR}} > f_i/2$, during the first $\lfloor T/f_i \rfloor$ intervals, the cycle rounding algorithm will schedule at most $2 \cdot \lfloor T/f_i \rfloor - 1$ maintenance actions for component i. The remaining interval has a length strictly less than f_i, and thus will include at most one additional maintenance action (or otherwise, if there were two, component i should have been eliminated from the last action). □

LEMMA 5.2: For any fixed $\varepsilon > 0$, there are instances where the cycle rounding algorithm computes a maintenance schedule of cost at least $(2 - \varepsilon) \cdot \mathrm{OPT}$.

PROOF: Consider an instance of the modular maintenance scheduling problem with two components and one module. The module maintenance cost is $\in$ as is the maintenance cost for component 1. However, the maintenance cost for component 2 is 1. The cycle limit for component 1 is 2^κ and the cycle limit for component 2 is $2^{\kappa+1} - 1$, for some positive integer κ. Figure 3 illustrates this example.

We first observe that the long run average cost of the optimal solution, OPT, is upper bounded by that of the solution where one schedules component 1 every 2^κ periods and component 2 every $2^{\kappa+1} - 1$ periods. Therefore,

$$\mathrm{OPT} \leq \frac{2\varepsilon}{2^\kappa} + \frac{1+\varepsilon}{2^{\kappa+1}-1} - \frac{\varepsilon}{2^\kappa \cdot (2^{\kappa+1}-1)}.$$

The cycle rounding algorithm will schedule both components every 2^κ periods and will incur $1 + 2\varepsilon$ units of cost every 2^κ periods, which yields a long run average cost of

$$\frac{1+2\varepsilon}{2^\kappa}.$$

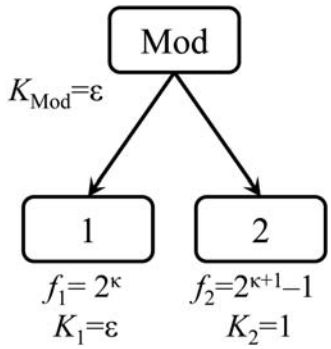

Figure 3. Bad example for the cycle rounding algorithm.

Consequently, the asymptotic cost ratio between the cycle rounding solution and the optimal solution, when the parameter κ tends to infinity, can be upper bounded by

$$\lim_{\kappa\to\infty} \frac{\frac{1+2\varepsilon}{2^\kappa}}{\frac{2\varepsilon}{2^\kappa} + \frac{1+\varepsilon}{2^{\kappa+1}-1} - \frac{\varepsilon}{2^\kappa\cdot(2^{\kappa+1}-1)}} = \lim_{\kappa\to\infty} \frac{1+2\varepsilon}{2\varepsilon + \frac{1+\varepsilon}{2-1/2^\kappa} - \frac{\varepsilon}{2^{\kappa+1}-1}} = \frac{2(1+2\varepsilon)}{1+5\varepsilon} \geq 2 - 6\varepsilon. \quad \square$$

5.3. Additional Remarks

5.3.1. *Some Practical Advantages*

Observe that the cycle rounding algorithm produces a 2-approximate schedule that depends only on the cycle limits and the tree structure but not on the maintenance costs. This property is attractive in practice since the costs associated with maintenance can be hard to define, difficult to estimate accurately, and/or they can change over time. In particular, the maintenance costs might include man hours, use of specialized tooling, holding inventory, and the use of specialized maintenance facilities. Since the schedule does not depend on maintenance costs and has well defined, intuitive scheduling rules, this algorithm could be implemented in a maintenance setting without requiring frequent updates or the execution of complex operational plans.

5.3.2. *Robustness*

Another benefit of the cycle rounding algorithm is its robustness to changes in the component cycle limits. The cycle limit for component i can be reduced by $f_i - f_i^{\mathrm{CR}}$ and increased by $f_{m(i)}^{\mathrm{CR}} - \left(f_i \bmod f_{m(i)}^{\mathrm{CR}}\right)$ without changing the resulting schedule. Intuitively, the allowable decrease is based on the feasibility of f_i^{CR} and the increase on $f_i - \left\lfloor f_i/f_{m(i)}^{\mathrm{CR}} \right\rfloor \cdot f_{m(i)}^{\mathrm{CR}}$ remaining less than $f_{m(i)}^{\mathrm{CR}}$. During the system's design and testing phase, this robustness can prove

valuable. Designers will be able to tailor margins of safety and engineers will be able to develop appropriate test plans to set the cycle limit of a component appropriately considering the tradeoff between near term investments and longer term maintenance costs. Even in an operational setting, such robustness properties can become relevant with time, since an improved understanding of component failure patterns could potentially result in updated cycle limits.

5.3.3. *Extension to Longest-Path Costs*

The cycle rounding algorithm can also be applied to the downtime cost function in which the cost incurred for any maintenance action is the maximum cost among all components included in the maintenance action, as described in Example 2 (see Section 5.1). For component i, the predecessor $m(i)$ will be the component for which $\left[K_i - K_{m(i)}\right]^+ = K^i$ where K_i, as defined earlier, is the cost of maintaining component i. By the previous analysis, the resulting schedule is cyclic, nested, and has a maximum maintenance frequency increase of 2. This implies that the cycle rounding algorithm is a 2-approximation for the downtime cost function. However, its reliance on the tree structure or the ability to identify a single predecessor $m(i)$ for each component, precludes using the algorithm for more general submodular cost functions. One such submodular cost function that cannot be addressed with the cycle rounding algorithm is additive costs on a directed acyclic graph (instead of a tree). This cost function could arise if two or more modules need to be removed simultaneously (instead of sequentially) to access a lower level module or component.

6. THE SHIFTED POWER-OF-TWO ALGORITHM

In what follows, we consider general submodular cost functions, and focus our attention on computing shifted power-of-two policies, where each component is maintained at a frequency that is a power of 2 times a specified base (or shifting parameter). We identify a small class of shifted power-of-two policies that depend only on the cycle limits and not on the cost parameters. The least expensive of these policies is guaranteed to have a cost within $1/\ln(2) \approx 1.44$ of optimal for infinite horizon instances.

6.1. Some Intuition

While the cycle rounding algorithm is an efficient approximation algorithm that yields nested cyclic schedules, it is not well defined for more general submodular cost functions. In addition, there might be operational considerations that necessitate a more restricted cyclic maintenance schedule in which a system is repaired in a fully nested manner, in which component i is maintained only if components $1, 2, \ldots, i-1$ are also maintained. Clearly, any fully nested schedule is also nested. Such considerations might include equipment availability, manpower scheduling, and inventory management.

Building on the worst case instance described in Lemma 5.2, we consider a more refined rounding scheme. The key insight from this tight example and from Theorem 4.2 is that the performance guarantee is strongly affected by the amount a cycle limit is rounded down from f_i. If, by shifting the rounding point, we can reduce the maximal amount of rounding that occurs, we might be able to improve the quality of the solution. We begin by considering a continuous time relaxation of the problem in which maintenance actions can be scheduled in fractional time periods. Following the discussion in Section 4.1, we will then round the solution to obtain a feasible discrete solution with no higher cost.

6.2. Shifted Power-of-Two Policies

We focus attention on a family of cyclic policies called shifted power-of-two policies, in which the rounded cycle limits for all components are powers of 2 times a base. Specifically, we parameterize this class of policies through a parameter $\delta \in [1, 2)$. For a given value of δ, the rounded cycle limits, f_i^δ, are set to be $f_i^\delta = \delta \cdot 2^\kappa$ where κ is the unique integer for which $\delta \cdot 2^\kappa \le f_i < \delta \cdot 2^{\kappa+1}$.

For any value of $\delta \in [1, 2)$, the resulting schedule is obviously cyclic and is also fully nested, since a component i is scheduled for maintenance only in periods in which components $1, 2, \ldots, i-1$ are also scheduled. Thus, the worst-case analysis below is focused on bounding the ratio f_i / f_i^δ. Before analyzing the best-possible shifted power-of-two policy, we derive an immediate result.

THEOREM 6.1: Any shifted power-of-two policy is guaranteed to have a cost that is at most twice the cost of the optimal solution in the infinite horizon setting.

PROOF: Let $\delta \in [1, 2)$ denote the rounding parameter for the shifted power-of-two policy. For any component i, note that its cycle limit can be written as $f_i = \beta_i \cdot 2^{\kappa_i}$ for some $\beta_i \in [1, 2)$ and positive integer κ_i. Also,

1. If $\delta \le \beta_i$, then the rounded frequency for component i is $f_i^\delta = \delta \cdot 2^{\kappa_i}$, and we have $f_i / f_i^\delta = \beta_i / \delta \le 2$.
2. If $\delta > \beta_i$, then $f_i^\delta = \delta \cdot 2^{\kappa_i - 1}$. In this case, $f_i / f_i^\delta = 2 \cdot \beta_i / \delta \le 2$.

Since these bounds are valid for all components i, Theorem 4.2 implies the result. □

6.3. Algorithm and Analysis

6.3.1. Randomizing the Shifting Parameter

The analysis in Theorem 6.1 assumes a worst-case choice of the shifting parameter δ. Next, we discuss how δ can be chosen "optimally" and demonstrate that this leads to an improved worst-case performance guarantee. To analyze the best choice for the parameter δ in the shifted power-of-two algorithm, we will first analyze an algorithm in which δ is chosen randomly from a specific distribution over the interval [1,2). This will yield an improved worst-case guarantee in expectation. We will then show how to deterministically find the optimal shifting parameter, δ^*, that attains this worst-case guarantee.

LEMMA 6.2: Let U be uniformly distributed over the interval $[0, \ln(2))$ and let $\delta = e^U$. Then, the expected cost of the resulting shifted power-of-two policy is at most $1/\ln(2)$ times that of an optimal policy.

PROOF: Since any choice of δ yields a (fully) nested and cyclic schedule, for each component i the contribution to the long run average cost is K^i/f_i^δ. Therefore, we have the following result:

$$\frac{\mathbb{E}_\delta\left[\sum_{i\in\mathcal{C}}\frac{K^i}{f_i^\delta}\right]}{\text{OPT}} \le \frac{\mathbb{E}_\delta\left[\sum_{i\in\mathcal{C}}\frac{K^i}{f_i^\delta}\right]}{\sum_{i\in\mathcal{C}}\frac{K^i}{f_i}} = \frac{\sum_{i\in\mathcal{C}}\mathbb{E}_\delta\left[\frac{K^i}{f_i^\delta}\right]}{\sum_{i\in\mathcal{C}}\frac{K^i}{f_i}} \le \max_{i\in\mathcal{C}}\frac{\mathbb{E}_\delta\left[\frac{K^i}{f_i^\delta}\right]}{\frac{K^i}{f_i}} = \max_{i\in\mathcal{C}}\mathbb{E}_\delta\left[\frac{f_i}{f_i^\delta}\right].$$

The first inequality follows from the infinite horizon lower bound obtained in Lemma 4.1. The second inequality follows from the fact that we are summing over the same sets (in the numerator and denominator) which implies the ratio of the sums is bounded by the maximum ratio of individual terms.

Now, for each $i \in \mathcal{C}$ we can write $f_i = \beta_i \cdot 2^{\kappa_i}$ for some $\beta_i \in [1,2)$ and positive integer κ_i. If $\delta \le \beta_i$, then $f_i^\delta = \delta \cdot 2^{\kappa_i}$ and if $\delta > \beta_i$, then $f_i^\delta = \delta \cdot 2^{\kappa_i - 1}$. These cases directly relate to the events in which the random variable $U \in (0, \ln(\beta_i)]$ and $U \in (\ln(\beta_i), \ln(2))$, respectively. In these cases, we can define the ratio between f_i and f_i^δ as follows:

$$\frac{f_i}{f_i^\delta} = \begin{cases} \frac{\beta_i \cdot 2^{\kappa_i}}{e^U \cdot 2^{\kappa_i}} = \frac{\beta_i}{e^U}, & U \in (0, \ln(\beta_i)]; \\ \frac{\beta_i \cdot 2^{\kappa_i}}{e^U \cdot 2^{\kappa_i - 1}} = \frac{2\cdot\beta_i}{e^U}, & U \in (\ln(\beta_i), \ln(2)). \end{cases}$$

If we take the expectation over U we have the following upper bound for all components:

$$\begin{aligned}\mathbb{E}_U\left[\frac{f_i}{f_i^\delta}\right] &= \int_0^{\ln(2)} \frac{f_i}{f_i^\delta}\cdot\frac{1}{\ln(2)}\cdot du \\ &= \int_0^{\ln(\beta_i)} \frac{\beta_i}{e^u}\cdot\frac{1}{\ln(2)}\cdot du + \int_{\ln(\beta_i)}^{\ln(2)} \frac{2\cdot\beta_i}{e^u}\cdot\frac{1}{\ln(2)}\cdot du \\ &= -\frac{\beta_i}{\ln(2)}\left(e^{-u}|_0^{\ln(\beta_i)} + 2\cdot e^{-u}|_{\ln(\beta_i)}^{\ln(2)}\right) \\ &= -\frac{\beta_i}{\ln(2)}\left(\frac{1}{\beta_i} - 1 + 2\cdot\frac{1}{2} - 2\cdot\frac{1}{\beta_i}\right) = \frac{1}{\ln(2)}.\end{aligned}$$

The result then follows from Theorem 4.2. □

6.3.2. Derandomization

In Lemma 6.2, we analyze a randomized algorithm that chooses δ from a specified distribution, and guarantees an expected cost of at most $1/\ln(2)\cdot$ OPT. Therefore, there has to be some specific value $\delta \in [1,2)$ for which the resulting δ-shifted power-of-two policy has a cost of at most $1/\ln(2)\cdot$OPT as well. Lemma 6.3 below shows that we can in fact compute the best shifted power-of-two policy deterministically, by limiting the search space for the aforementioned value δ to a small finite set. By testing each of the potential values as the best value for δ, we obtain the same worst-case guarantee of $1/\ln(2)$ without the use of randomization.

LEMMA 6.3: For any shifted power-of-two policy with parameter δ, there exists another shifted power-of-two policy with the same or lower cost which has a rounding parameter δ^*, satisfying $f_i^{\delta^*} = f_i$ for some component $i \in \mathcal{C}$.

PROOF: Suppose that for some rounding parameter δ, we have

$$f_i^\delta = \delta\cdot 2^\kappa < f_i < \delta\cdot 2^{\kappa+1}$$

for every $i \in \mathcal{C}$, that is, the first inequality above is strict for all components. When δ is increased by a sufficiently small amount, the rounded maintenance frequencies f_i^δ will remain feasible (i.e., smaller or equal to f_i) while the time between successive maintenance actions will increase for all components, which clearly leads to improved average per-period cost. Accordingly, we can increase δ until $f_i^\delta = f_i$ for some component i. □

For each component i, we can write $2^{\kappa_i} \le f_i = \beta_i\cdot 2^{\kappa_i} < 2^{\kappa_i+1}$ for some number $\beta_i \in [1,2)$ and positive integer κ_i. Lemma 6.3 implies that the best (optimal) shifted power-of-two policy has a rounding parameter δ^* satisfying $f_i^{\delta^*} = f_i$ for some component i. That is, $\delta^* \in \{\beta_1, \ldots, \beta_{|\mathcal{C}|}\}$. It follows

that one can identify the best shifted power-of-two policy efficiently by considering only policies defined by these $|\mathcal{C}|$ shifting parameters. We have now obtained the following theorem.

THEOREM 6.4: The best shifted power-of-two policy, where $\delta^* \in \{\beta_1, \ldots, \beta_{|\mathcal{C}|}\}$, is guaranteed to have a cost of at most $1/\ln(2)$ times that of an optimal policy in the infinite horizon setting.

Recall that a shifted power-of-two policy can schedule maintenance in fractional time periods. However, by applying the rounding procedure described in Section 4.1, we can convert this policy into a discrete time policy with no higher cost. Note that in the resulting discrete time policy, maintenance actions might not be evenly spaced over the planning horizon but the spacing between subsequent maintenance actions varies by at most one period.

6.3.3. *Matching Worst-Case Performance*

We conclude the analysis by demonstrating that the prior distribution chosen for δ is best possible and that the $1/\ln(2)$ bound cannot be improved by choosing a different distribution for δ.

LEMMA 6.5: For any fixed $\varepsilon > 0$, there are instances where any shifted power-of-two policy generates a maintenance schedule of cost at least $1/(\ln(2) + \varepsilon) \cdot \mathrm{OPT}$.

PROOF: Our example consists of a dependency tree with one module and 2^κ components, where κ is an integer parameter. The maintenance cost for the module is ϵ and is 1 for each of the components. The cycle limits for the components are $2^\kappa, 2^\kappa + 1, \ldots, 2^{\kappa+1} - 1$. Figure 4 illustrates this example.

On the one hand, the long run average cost of the optimal solution, $\mathrm{OPT}_{\kappa,\varepsilon}$, is upper bounded by that of the schedule where each component is maintained exactly at its cycle limit, which implies

$$\begin{aligned}\mathrm{OPT}_{\kappa,\varepsilon} &\le \left(\frac{1}{2^\kappa} + \frac{1}{2^\kappa + 1} + \cdots + \frac{1}{2^{\kappa+1} - 1}\right) + \varepsilon \\ &= H_{2^{\kappa+1}-1} - H_{2^\kappa - 1} + \varepsilon = \ln(2) + \lambda_{2^\kappa} + \varepsilon,\end{aligned}$$

where $H_t = \sum_{k=1}^{t} \frac{1}{k}$ is the t-th harmonic number, and $|\lambda_t| = \Theta(1/t)$. The last equality holds since $|H_t - \ln t| = \gamma + O(1/t)$, where γ is the Euler–Mascheroni constant [21].

On the other hand, from Theorem 6.4 we know that the optimal shifted power-of-two rounding parameter δ^* will be of the form $(2^\kappa + d)/2^\kappa$ for an integer $d \in [0, 2^\kappa - 1]$. For

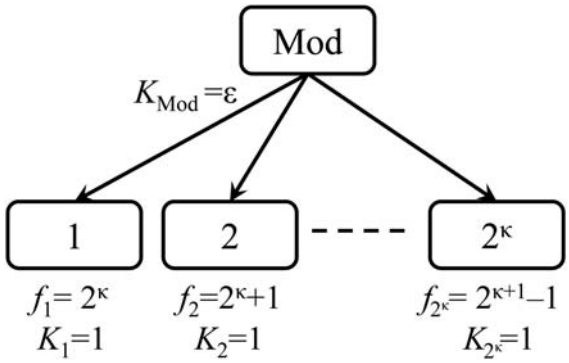

Figure 4. Shifted power-of-two bad example.

$d = 0$, $f_i^\delta = f_1$ for all components i, which implies the long run average cost of the solution will be

$$\frac{K^1}{f_1} + \sum_{i=2}^{2^\kappa} \frac{K^i}{f_1} = \frac{1+\varepsilon}{2^\kappa} + (2^\kappa - 1) \cdot \frac{1}{2^\kappa} = 1 + \frac{\varepsilon}{2^\kappa}.$$

For any positive choice of d, the long run average cost of the solution will be

$$\begin{aligned}&\frac{K^1}{f_1^\delta} + \sum_{i=2}^{d} \frac{K^i}{f_i^\delta} + \sum_{i=d+1}^{2^\kappa} \frac{K^i}{f_i^\delta} \\ &\quad = \frac{2+2\varepsilon}{2^\kappa + d} + (d-1) \cdot \frac{2}{2^\kappa + d} + (2^\kappa - d) \cdot \frac{1}{2^\kappa + d} \\ &\quad = \frac{2^\kappa + d + 2\varepsilon}{2^\kappa + d} = 1 + \frac{2\varepsilon}{2^\kappa + d}.\end{aligned}$$

This implies that the cost ratio between any solution provided by the shifted power-of-two algorithm and the optimal solution tends to $1/(\ln(2) + \varepsilon)$, as the parameter κ tends to infinity. □

6.4. Additional Remarks

6.4.1. *Robustness Issues*

As we noted previously, in attaining an improved approximation guarantee, the best shifted power-of-two policy is dependent on the cost parameters and the component cycle limits. While the cost parameters do not affect the set of shifted power-of-two policies considered, a change in the cost parameters might change the optimal choice of β_i and the resulting shifted power-of-two maintenance schedule. Also, if the cycle limit of a component changes, the β_i coefficient associated with that component might change and so might the best choice of the rounding parameter δ^* from the set of β_i's. Accordingly, there is no a priori bound on the allowable change in the cycle limits as there is for the cycle

rounding algorithm. Rather, each possible cycle limit under consideration during the design phase must be evaluated individually.

6.4.2. Yet Another Algorithm

An alternative way to derive an approximation ratio of $1/\ln(2)$, by altering the randomized rounding algorithm of Teo and Bertsimas [17] for resource-constrained lot-sizing problems, is given in Appendix B.

7. COMPUTATIONAL RESULTS

While the worst-case analyses for the cycle rounding and shifted power-of-two algorithms ensure that the solutions they provide will be no worse than 2 and $1/\ln(2)$ times the value of the optimal solution, the problem instances for their worst-case examples are significantly different than the instances we would expect to find in practice. Accordingly, we examine the typical empirical performance of these algorithms by testing them on a set of mid-sized problem instances. These instances are large enough to stress the algorithms but small enough to allow for qualitative solution comparison. The approximation algorithms were implemented in MATLAB R2010b run on a standard laptop computer with a 1.9 GHz dual-core processor and 2 GB of RAM.

7.1. Construction of Tested Instances

We conducted extensive computational tests of 15,000 infinite horizon problem instances over a variety of tree structures, cycle limits, and maintenance costs for both the path-additive and longest-path (downtime) cost functions. In these test instances, dependency trees were generated based on specific structures of interest related to practical settings at the Air Force. This collection of instances can be divided into three prototypes:

- The first dependency tree was created with one base module, two submodules, one sub-submodule, and eight components. One of the components depended only on the base module, two depended on the first submodule, two depended on the other submodule, and three depended on the sub-submodule.
- The second dependency tree was created with one base module, four submodules, and six sub-submodules. Then, 50 components were distributed almost evenly at each level (e.g., the same number of components depended on each module). In addition, the base module as well as all submodules each had one or two more components. This model roughly matches the F100 engine (see Section 1).
- The third dependency tree was created by uniformly choosing the number of components, between 1 and 50, and the number of modules (again, uniformly) between 1 and the previously picked number of components. Further dependencies between modules and components were also defined randomly, ensuring that we did not end up with cycles.

Consequently, the maintenance hierarchy contained up to 50 components, with depth ranging from rather shallow (up to three indentures) to very detailed (up to five levels of indenture). That is, maintaining a component requires the removal of either just the system, or the system and up to three levels of modules. The cycle limits for each component were uniformly selected from the integer set [2,50]. Components with a cycle limit of one were not allowed, as discussed in Section 2.

These tests were broken down into three groups based on the maintenance costs. In all three cases, the maintenance costs for modules (i.e., the K_i value of each internal node) were selected uniformly over the set of integer values [1,100]. In the first group the costs for components were also selected uniformly over the set of integer values [1,100]. In the second group, the maintenance costs increased linearly as the component cycle limit increased. In particular, the cost to maintain component i was chosen uniformly from the integer set $[1, f_i]$. For the third group, the component maintenance costs increased exponentially as the component cycle limit increased, namely selected uniformly over the integer set $[1, \lceil (1.25)^{f_i} \rceil]$. For the path-additive cases, Eq. (2) in Section 5.1 was used to compute the cost function, whereas for the downtime cases, we made use of Eq. (3). Technically speaking, linear and exponential cost functions are representative examples that have actually been observed with respect to the previously mentioned settings at the Air Force. It is worth pointing out that these examples do not cover all possible cases, and others could have been used as well.

7.2. Evaluating Solution Quality

With this description in hand, it remains to address one basic question: Once we execute one of the algorithms on a given infinite horizon instance, how do we evaluate the ratio between the average perperiod cost obtained and that of the optimal maintenance policy? Clearly, the former cost can be easily computed, as the schedules generated by both algorithms are cyclic and nested. However, computing the optimal cost efficiently is exactly the seemingly intractable problem we are trying to tackle. Therefore, our method to efficiently compute a lower bound is to make use of Lemma 4.1, which exploits our charging scheme to bound the cost of any feasible schedule (and, in particular, the optimal one). Therefore, the findings presented in the remainder of this

Table 1. Average solution ratios for 15,000 randomly chosen infinite horizon instances

Objective	Algorithm	Uniform costs	Cost proportional in cycle limit	Cost exponential in cycle limit
Additive	Cycle Rounding	1.13	1.11	1.09
	Shifted Power-of-two	1.24	1.16	1.02
Downtime	Cycle Rounding	1.06	1.25	1.4
	Shifted Power-of-two	1.06	1.23	1.11

section are upper bounds on the performance guarantees of the algorithms considered, and in fact, it is quite possible that the actual performance is even better.

7.3. Results and Discussion

Over the test set of instances mentioned above, the cycle rounding algorithm had an average runtime of approximately 4×10^{-3} s, and yielded solutions no more than 30% above the lower bound for 92% of instances and within 10% percent of the lower bound for 35% of instances. The shifted power-of-two algorithm had an average running time of approximately 4×10^{-4} s, with 91% and 33% of solutions no more than 30% and 10% above the lower bound, respectively. For each algorithm and test group, Table 1 shows the ratio between the long run average additive cost and its lower bound, as well as the corresponding results for the downtime cost.

Under the additive cost function, the cycle rounding algorithm dominated in 98% of the uniform cost instances while the shifted power-of-two algorithm dominated in 80% of the exponential cost instances. This demonstrates the power of exploiting the cost data to determine the maintenance frequency of components whenever there are disparate costs for maintaining components. The shifted power-of-two algorithm makes use of the cost data to balance the amount of rounding with the cost implications. Conversely, a plausible explanation for why the cycle rounding algorithm performs better in the uniform cost instances could be that it is not limited to a single rounding point as the shifted power-of-two algorithm is. Rather, it rounds each component only as much as is needed to incur the residual cost. For instances where the costs are proportional to the cycle limit, the cycle rounding algorithm produced solutions that had a lower long run average cost than the shifted power-of-two algorithm in 80% of the instances.

Another statistic of interest is the magnitude of the difference in performance between the cycle rounding and shifted power-of-two algorithms. The uniform and proportional cost instances yielded no more than 25% variation between the cycle rounding and shifted power-of-two solutions. However, the results were much more drastic in the exponential cost case. In one instance, the shifted power-of-two algorithm had a solution within 1% of the lower bound while the cycle rounding solution was 98% higher than the lower bound. This again demonstrates the importance of considering maintenance costs when there are highly disparate costs for maintaining each component.

While there were major differences between the algorithms for the additive cost function in all tested groups, for the downtime objective the outcomes in the uniform and proportional costs are comparable. In 40% of the uniform cost instances, the cycle rounding algorithm and the shifted power-of-two algorithm obtained identical costs. In only 10% of the uniform and proportional cost instances did the cycle rounding and shifted power-of-two solutions vary by more than 15%. That said, in the exponential case, the differences in performance are even more noticeable than for additive costs. Here, the shifted power-of-two algorithm outperformed the cycle rounding algorithm in 99% of the tested instances, even though there were no extreme cases similar to the instance discussed above.

As a consequence of these computational experiments, and based on additional practical experience, one of our main managerial insights is the value (or lack thereof) of fully quantifying the maintenance costs for a given system. In many cases, the required maintenance frequencies are an artifact of the design and implementation, often determined to support the overall maintenance program. In contrast, costs are much harder to quantify, as it could be very expensive and time consuming to perform surveys, in depth audits, etc. Therefore, when maintenance costs are believed to be uniform or proportional to the cycle limits, it may not be worth the cost to fully quantify them. In such cases, the cycle rounding algorithm can be implemented with the indicated performance guarantees. Conversely, when maintenance costs are thought to be exponential, it is likely worthwhile to quantify those costs, so that the shifted power-of-two algorithm can be used. In addition, when the system dependencies cannot be represented by a tree, but rather through a more general submodular cost function, then the shifted power-of-two algorithm would be used, which necessitates quantification of the maintenance costs.

8. CONCLUDING REMARKS

We studied several models focused on the maintenance of modular systems with component-specific cycle limits. We proposed two easily implementable and conceptually simple cyclic policies. These policies have provable worst-case

guarantees, and they perform very well in computational experiments on a large set of relevant instances. As a by-product of our worst-case analysis, we showed that one can efficiently compute cyclic policies that approximate arbitrarily-structured policies to within small constant factors. This finding is particularly important, since such policies are operationally intuitive and usually easier to implement in practice. We conclude by explaining the main differences between our approach and that of existing techniques, addressing some practical issues, and discussing intractability questions.

8.1. Comparison to Inventory-Related Algorithms

Next, we summarize the major differences between our work and the inventory management papers discussed in Section 3. We focus attention on power-of-two policies, and explain how existing work in this context differs in terms of both the nature of the results obtained and the techniques being used.

1. **Lower bound**. The relevant inventory papers derive a lower bound on the continuous-time infinite-horizon variants of the problems studied by solving a nonlinear convex program. In contrast, our lower bound holds for both the finite and infinite horizon settings. While being (seemingly) looser, it is derived through direct arguments based on a cost decomposition and a charging scheme that distributes the cost of each maintenance action between its components. Quite surprisingly, we demonstrate that our "looser" bound actually leads to best-possible shifted power-of-two policies.
2. **Algorithms**. The relevant inventory papers are based on various rounding methods, applied to the optimal solution of the convex programs mentioned above. They then show that the resulting policy is provably good. In contrast, we show how to compute an optimal shifted power-of-two policy, by identifying a small set of policies, such that for any input instance, one of them is guaranteed to be optimal among all shifted power-of-two policies. The algorithm is combinatorial, and does not require solving any convex relaxations. In addition, we also show how to take fractional (continuous-time) schedules and convert them to integral (discrete-time) schedules that have identical performance guarantees; these ideas can also be applied in the finite-horizon case, where we would incur an extra additive term in the approximation ratio. Moreover, the small set of policies we identify depends only on the cycle limits and not on any other problem parameter. Therefore, the worst-case guarantees we obtain are in fact a priori, in the sense that the precomputed set of policies guarantees these worst-case bounds for all possible problem instances. In contrast, previous algorithms do not necessarily provide an optimal policy (within the class of policies considered), and also depend on the specific instance in question.

8.2. Practical Concerns: Corrective Maintenance and Rolling Horizon Settings

As mentioned in Section 1, the models studied in this article are not meant to be operational by themselves, but rather to be used in planning, testing, or along with other operational models. Therefore, if corrective maintenance occurs, its impact will strongly depend on the submodular cost function in question, the module dependency structure, and the cycle limits of the affected components. These practical concerns, along with additional real-life issues, are addressed in Eric Zarybnisky's thesis [22].

One particular extension that can be easily handled, with a relatively small deterioration in the worst-case approximation ratio, is that of rolling-horizon settings. To this end, assuming that all components have a full cycle life at the very beginning is clearly not a concern in the infinite horizon setting. There, we could simply schedule a maintenance action for all components $\mathcal{C}$ in period 1, and this has no effect on the long run average cost, since the additional term $K(\mathcal{C})/T$ tends to zero. In the finite horizon setting, given an α-approximation algorithm (with the full cycle assumption), this modification increases the performance guarantee to at most $\alpha + 1/\lfloor \min_i (T - \ell_i)/f_i \rfloor \leq \alpha + 1$, where ℓ_i is the initial number of cycles remaining for component i. This holds since each component i has to be maintained at least $\lfloor \min_i (T - \ell_i)/f_i \rfloor$ times, and since K is submodular.

8.3. Computational Hardness

Our main results show that the modular maintenance scheduling problem admits efficient constant-factor approximation algorithms. However, in spite of our best efforts, we were still unable to resolve the question of whether this problem is NP-hard or not, which seems highly nontrivial. For this reason, it would be interesting to investigate such complexity issues as part of future research. A first step in this direction has already been taken by Telha [19, Section 4.5], who proved that the problem of computing an optimal cyclic policy is in fact integer-factorization hard, even for the special case of path-additive costs (see Section 5.1, Example 1) on a dependency tree with only three components.

8.4. Noninstantaneous Maintenance

One of our basic model assumptions was that maintenance actions occur instantaneously, justified by the observation

that such actions are typically much shorter than the duration of a single (normalized) time period. Without this assumption, the resulting computational problem appears to be significantly harder to deal with. The first question in this context is how to sensibly incorporate downtime into the existing model. We looked into several options, such as those where:

- The system has to be operational at least some given proportion of the planning horizon.
- Some given number of time periods has to elapse between successive maintenance actions.
- There are upper bounds on the number of maintenance actions for each component (or on its frequency) during the planning horizon.

At present time, we can show that such settings can be formulated as integer programming problems, or as high-dimensional dynamic programs. However, beyond that, the question of how to efficiently design provably-good schedules in still wide open.

APPENDIX A: HARDNESS OF NONSTATIONARY COSTS

The models and algorithms developed in this article assume stationary costs, where the function $K(\cdot)$ is dependent only on the subset of components to be maintained, regardless of the time period in which this action occurs. One natural extension would be to allow these costs to vary over time (with a finite horizon), meaning that we could have a different function $K_t(\cdot)$ for each period t. With this extension, however, the modular maintenance scheduling problem does not admit a constant factor approximation, even for the special case of having a dependency tree with path-additive costs (see Example 1 in Section 5.1). Specifically, the next lemma provides an approximation-preserving reduction from the set cover problem.

LEMMA A1: The modular maintenance scheduling problem with nonstationary costs cannot be approximated within factor $c \ln n$ unless P = NP, for some constant $c > 0$. Here, n stands for the underlying number of components.

PROOF: Consider an instance of the set cover problem, with a ground set of elements $\mathcal{C} = \{e_1, \ldots, e_n\}$ and a family of subsets $\mathcal{S} = \{S_1, \ldots, S_m\} \subseteq 2^{\mathcal{C}}$. The objective is to compute a minimum cardinality subset $\mathcal{S}' \subseteq \mathcal{S}$ such that the sets in $\mathcal{S}'$ collectively cover all elements in $\mathcal{C}$. By plugging the proof system of Raz and Safra [10] or Arora and Sudan [2] into the reduction of Bellare et al. [3], the former authors showed that set cover is NP-hard to approximate within a factor of $c \ln n$, for some constant $c > 0$.

Given a set cover instance, we create a corresponding instance of the modular maintenance scheduling problem as follows:

- The planning horizon is of length $m + 1$, where the time periods are numbered $0, \ldots, m$.
- There is one component for every element in $\mathcal{C}$, each with a cycle limit of m.
- For every time period, the cost of the single shared module is 1. Also, for all components $i \in \mathcal{C}$:
  1. We make sure that none of the components are maintained at period 0, by setting $K_i^0 = \infty$.
  2. We create a one-to-one correspondence between maintaining components corresponding to the elements in S_t at time period t and between picking the subset S_t in the set cover instance. This is achieved by setting $K_i^t = 0$ for every t such that $e_i \in S_t$, and $K_i^t = \infty$ otherwise.

Note that since components cannot be maintained at period 0 without incurring infinite cost, and since the cycle limit of every component is m, each component must be maintained exactly once over the planning horizon between period 1 and m, as one unit of its cycle limit is inevitably exhausted in period 0. With this observation in place, we can see that any feasible cover can be easily translated to a feasible maintenance schedule with no greater cost, and vise versa. This implies that the modular maintenance scheduling problem with nonstationary costs cannot be approximated within a factor of $c \ln n$, for some constant $c > 0$. □

APPENDIX B: AN ALTERNATIVE WAY TO DERIVE 1/LN(2)

In light of the discussion in Sections 4.1 and 6, expert readers may be able to extract a matching worst-case guarantee of $1/\ln(2)$ by altering the randomized rounding algorithm of Teo and Bertsimas [17] for resource-constrained lot-sizing problems. Even though a model similar to ours has not been described (or even considered) in their original article, we proceed by giving a sketch of the necessary modifications:

- The lower bounding method of Teo and Bertsimas is based on computing an optimal solution to a convex relaxation of the resource-constrained lot-sizing problem. Here, one can modify this relaxation by: (1) excluding the holding costs, which play no role in maintenance scheduling; (2) setting the objective function to compute reorder intervals $T_1, \ldots, T_n$ that minimize $\max_i (K^i/T_i)$, where K^i is the residual cost of component i, as defined in Section 4.1 (rather than the original K^i variables, that were tied together by polymatroid constraints); and (3) adding a constraint that forces the reorder intervals to be small enough (i.e., $T_i \leq f_i$).
- As a consequence of the above modifications, one can argue that $T_i^* = f_i$ is an optimal solution to the resulting program. This relieves us from the need to use a nonlinear solver, but still, T^* does not correspond to a nested schedule. A close inspection of the randomized algorithm proposed by Teo and Bertsimas to obtain a nested schedule $\tilde{T}$ reveals that, while T^* may be rounded either up or down (and in this context, up is the wrong direction, since we end up with $\tilde{T}_i > f_i$), the factor by which any reorder interval blows up is uniformly bounded. Therefore, another necessary alteration is scaling the prerounding solution T^* by an appropriate factor.

It is worth mentioning that our analysis addresses a couple of natural questions, relevant to both articles: Why should we make use of the particular distribution function stated in Lemma 6.2 and in [20, Algorithm B]? Could we improve on the $1/\ln(2)$ ratio by considering some other distribution? These are answered in Lemma 6.5, where we demonstrate that the prior distribution chosen is actually best-possible, and that the analysis cannot be improved by choosing some other distribution.

ACKNOWLEDGMENTS

We would like to thank the associate editor and two anonymous reviewers for their very detailed comments on an early version of this article. It is also a pleasure to thank Gen.

George Babbitt, U.S. Air Force retired, and David Shmoys for stimulating discussions and insights into the problem, and Yaron Azrieli for a number of literature pointers.

The research of Retsef Levi is partially supported by NSF grants DMS-0732175 and CMMI-0846554 (CAREER Award), an AFOSR award FA9550-08-1-0369, an AFOSR grant FA9550-11-1-0150, an SMA grant and the Buschbaum Research Fund of MIT. The research of Thomas Magnanti and Eric Zarybnisky is partially supported by an AFOSR award FA9550-08-1-0369.

The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the U.S. Government.

REFERENCES

[1] M. Amouzegar, L. Galway, and A. Geller, Alternatives for jet engine intermediate maintenance, Technical Report RB-82-AF, RAND, Santa Monica, CA, 2002.

[2] S. Arora and M. Sudan, Improved low-degree testing and its applications, Combinatorica, 23 (2003), 365–426.

[3] M. Bellare, S. Goldwasser, C. Lund, and A. Russell, "Efficient probabilistically checkable proofs and applications to approximations," in: Proceedings of the 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, USA, 1993, pp. 294–304.

[4] W. Crowston, A. Henshaw, and M. Wagner, A comparison of exact and heuristic routines for lot-size determination in multi-stage assembly systems, AIIE Trans 4 (1972), 313–317.

[5] A. Federgruen and Y. Zheng, Optimal power-of-two replenishment strategies in capacitated general production/distribution networks, Manage Sci 39 (1993), 710–727.

[6] J. Forbes and P. Wyatt, Optimal replacement policy for the F-15 aircraft engine modules. Master's thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, August 1975.

[7] K. Kobbacy, D. Murthy, R. Nicolai, and R. Dekker, "Optimal maintenance of multi-component systems: A review," in H. Pham (Editor), Complex system maintenance handbook, Springer Series in Reliability Engineering, Springer, London, 2008, pp. 263–286.

[8] R. Levi, T. Magnanti, D. Segev, and E. Zarybnisky, Mathematical programming analysis of a graph visiting problem, Manuscript (2012).

[9] M. Maxwell and J. Muckstadt, Establishing consistent and realistic reorder intervals in production-distribution systems, Oper Res 33 (1985), 1316–1341.

[10] R. Raz and S. Safra, "A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np," in: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, STOC '97, ACM, New York, NY, 1997, pp. 475–484.

[11] A.E. Roth (Editor), The shapley value: Essays in Honor of Lloyd S. Shapley, Cambridge University Press, 1988.

[12] R. Roundy, 98%-effective integer-ratio lot-sizing for one-warehouse multi-retailer systems, Manage Sci 31 (1985), 1416–1430.

[13] R. Roundy, Rounding off to powers of two in continuous relaxations of capacitated log sizing problems, Manage Sci 35 (1989), 1433–1442.

[14] A. Schrijver, Combinatorial optimization—Polyhedra and efficiency, Springer-Verlag, Berlin, 2003.

[15] L.S. Shapley, "A value for n-person games," in: H. Kuhn and A. Tucker (Editors), Contributions to the theory of games, Vol. 28, Annals of mathematical studies, Princeton University Press, Princeton, New Jersey, 1953, pp. 307–317.

[16] C. Telha, Algorithms and hardness results for the jump number problem, the joint replenishment problem, and the optimal clustering of frequency-constrained maintenance jobs, PhD thesis, Massachusetts Institute of Technology, 2012.

[17] C. Teo and D. Bertsimas, Multistage lot sizing problems via randomized rounding, Oper Res 49 (2001), 599–560.

[18] G. van Dijkhuizen, "Maintenance grouping in multi-step multi-component production systems," in: M. Ben-Daya, S. O. Duffuaa, and A. Raouf (Editors), Maintenance, modeling, and optimization, Klumer Academic, Norwell, MA, 2000, pp. 283–306.

[19] G. van Dijkhuizen and A. van Harten, Optimal clustering of frequency-constrained maintenance jobs with shared set-ups, Eur J Oper Res 99 (1997), 552–564.

[20] H. Wang and H. Pham, Reliability and optimal maintenance, Springer Verlag, London, 2006.

[21] R.M. Young, Euler's constant. Math Gaz 75 (1991), 187–190.

[22] E. Zarybnisky, Maintenance scheduling for modular systems–models and algorithms, PhD thesis, Massachussetts Institute of Technology, 2011.