

Applying Answer Set Optimization to Preventive Maintenance Scheduling for Rotating Machinery

Anssi Yli-Jyrä^[0000–0003–0731–2114] and Tomi Janhunen^[0000–0002–2029–7708]

Tampere University, Tampere, Finland
anssi.yli-jyra@tuni.fi, tomi.janhunen@tuni.fi

Abstract. Preventive maintenance (PM) of manufacturing units aims at maintaining the operable condition of the production line while optimizing the maintenance timing and the loss of productivity during maintenance operations. The lesser studied type of preventive maintenance understands a production line as a single machine with multiple components of different maintenance needs. This is relevant when rotating machinery is deployed, e.g., in the paper and steel industries, to mass production of raw materials consumed by other businesses. A failure in any stage of the production line has the potential of making the entire machine inoperable and enforcing a shutdown and corrective maintenance costs. This work gives an abstract formalization of PM scheduling for multi-component machines as an optimization problem. To provide a lower bound for the complexity of the optimization problem, we prove that the underlying decision problem is NP-complete for varying-size multi-component machines and scheduling timelines. Besides the formalization, the second main contribution of the paper is due to the practical need to solve the problem in industrial applications: the work gives the first encoding of the PM scheduling problem using Answer Set Optimization (ASO). Some preliminary experiments are conducted and reported to set the scene for further algorithm development.

1 Introduction

Rotating machinery is commonly deployed and maintained, e.g., in paper and steel industry, for the mass production of raw materials like newsprint and rolled steel consumed by respective branches of business. Such a *multi-component machine* consists of a large number of rotating components, also abbreviated *rotors* in general, operating in synchrony to form a continuous *production line*, such as a paper machine (see Fig. 1). Maintenance of any stage of this rotating machinery often requires a complete shutdown as maintenance potentially makes the entire production line inoperable. Most breaks incur loss in production, but abrupt servicing that focuses on *corrective maintenance* of failed components bring along extra costs and delays. Therefore, *preventive maintenance* (PM) and scheduled servicing play key roles when it comes to ensuring resource-efficient and timely production as demanded by global manufacturing and resilient industry.

Besides the practical importance, digitized *PM scheduling* (PMS) is a challenging computational problem and has been studied a lot in multi-machine settings. However, not enough attention has been spent on PMS of multi-component machines.

For example, a forthcoming survey [13] summarizes previous work on *integrated production, maintenance and resource scheduling*, but its focus is on multi-machine environments rather than in multi-component machines. Nevertheless, PMS for multi-machine systems is somewhat adaptable to paper plants. In this respect, we mention Do et al. [6] who study preventive maintenance of multi-component systems with serially connected components. In addition, there is some work in the context of paper mills (see, e.g., [11, 14, 16]).

In this work, we focus on PMS of abstract multi-component machines. This task is to increase the overall reliability and productivity of the machinery throughout the scheduling timeline with properly scheduled preventive maintenance actions. Unlike [13], we neither integrate production and maintenance scheduling explicitly nor study concrete machines. Instead, our general goal is to investigate options when it comes to the formalization of *features* that have not been formalized before in PMS of multi-component machines:

- We propose a penalty-driven flexible scheduling method that minimizes the *under-coverage* (delayed preventive maintenance actions) and *over-coverage* (advanced preventive maintenance actions). These can be used, to some extent, to represent the concern of continuous production in PMS.
- We do not insist on commonly used *periodic* maintenance that is based on recommended maintenance intervals of components. Instead, we try to synchronize preventive maintenance actions, at least partially, to reduce the need for separately scheduled maintenance breaks. Partial synchronization does not exclude the possibility that ultimately periodic schedules emerge.
- According to the survey [13], most approaches to PMS allow no exceptions beyond designated time intervals and flexibility windows. However, the approaches of [1, 17] are more versatile enabling variance in this respect. Our target for maintaining a particular component is extremely flexible: the next target time is determined by the previous preventive maintenance action and the recommended maintenance interval of the component, while any deviations are subject to penalty.

The paper claims three major contributions to the area: (1) Our definition of the PMS problem for multi-component machines is a new combination of PMS ideas not aiming at maximal parallel throughput or job-shop type scheduling but keeping the whole machine maximally in a good condition and in use. (2) We formalize some of the related decision and optimization problems and characterize their computation complexity with lower and the upper bounds. (3) We present

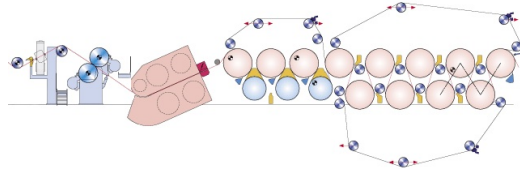


Fig. 1: A Paper Machine (KnowPap.com)

a prototypical implementation, i.e., encodings of PMS for multi-component machines using logic programming and optimization, including basic experiments and performance results of its practical behavior.

Previous implementations of PMS have deployed a myriad of AI techniques, including *genetic algorithms* (GAs) [5], *mixed-integer programming* (MIP) [4], and formulations as *constraint satisfaction problems* (CSPs) [9]. To our knowledge, there is no prior PMS implementation based on *answer set programming* (ASP, see [3] for an overview) that offers a rule-based language for knowledge representation. A typical *encoding* of a search or optimization problem in ASP is a logic program that captures the solutions of the problem as answer sets. It is already known that ASP is well-suited for the formalization of various scheduling problems (see, e.g., [7, 8]). The current work concentrates on interval-based scheduling of maintenance operations rather than timetabling individual events.

The rest of this article is organized as follows. In Section 2, we provide the formal definition of a multi-component machine and of its maintenance schedule, and study some objective functions that are relevant for the optimization of schedules. These definitions are crucial building blocks in Section 3 where we formalize PMS for multi-component machines from a number of perspectives. The computational complexity of the resulting decision and function problems is then roughly characterized in the same section. To move from the theoretical analysis to practice, we present an ASP encoding of the PMS optimization problem in Section 4. The performance obtained by this encoding for PMS is preliminarily studied further in Section 5 using the CLINGO system to implement the actual search for optimal schedules. Section 6 concludes the paper.

2 Definitions of Machines and Schedules

In this section, we provide an abstraction of a multi-component machine and a schedule specifying preventive maintenance actions for its components in the scheduling timeline. These definitions pave the way for the definition of the scheduling problem whose variants will be studied further in the next section.

Definition 1. A multi-component machine is a triple $\mathcal{M} = \langle C, \iota, \rho \rangle$ consisting of a finite set of components C , an initial lifetime function $\iota : C \rightarrow \mathbb{N}$, and a recommended maintenance interval function $\rho : C \rightarrow \mathbb{N} \setminus \{0\}$.

Fig. 1 illustrates how the values of the functions in Definition 1 are to be interpreted. The initial lifetime function ι tells how many time moments are covered by maintenance performed before the intended PM scheduling. The recommended maintenance interval function ρ tells a similar measure for every round of preventive maintenance action, but with the difference that it specifies the number of time moments that are covered by one preventive maintenance action. Every time t when a component is maintained it becomes as good as new. The recommended maintenance interval (RMI) of that component starts at the time t of preventive maintenance action and includes it, and all non-maintained

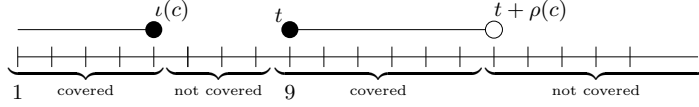


Fig. 2. Illustration of $\iota(c) = 5$ and $\rho(c) = 6$ with a Service at Time Point $t = 9$

components continue aging during the time step t . The preventive maintenance actions are specified by schedules:

Definition 2. A preventive maintenance schedule (PMS, or schedule for short) for machine \mathcal{M} is represented as a quadruple $S = \langle h, \ell, b, A \rangle$ where

- $h \in \mathbb{N} \setminus \{0\}$ marks the horizon of the scheduling timeline,
- $\ell \in \{1, \dots, h\}$ is the limit for the last possible time moment of a preventive maintenance action,
- $b \in \{0, \dots, \ell\}$ is the maximum size of a set $B \subseteq \{1, \dots, \ell\}$ of time points, also called scheduled maintenance breaks, during which any preventive maintenance action must take place,
- $A : C \times \{1, \dots, \ell\} \rightarrow \{0, 1\}$, called the service selection function, is a characteristic function indicating, for each component $c \in C$ of the machine \mathcal{M} , the set $B_c = \{t \mid A(c, t) = 1\} \subseteq B$ of time moments of preventive maintenance actions used to service component c .

The limit ℓ controls how far towards the horizon the breaks can be allocated, having an compressing effect on the schedule. If $\ell = h$, any preventive maintenance action performed at the time moment h is mainly a wasted investment to the future without a significant effect on the scheduling timeline. The schedule is *empty* if $b = 0$. Four evaluation functions for the quality of component-wise schedules are defined. Their definitions are facilitated by two auxiliary functions: let $\pi(c) = \{(s, t) \in B_c \times B_c \mid s < t, \text{ and } s < u < t \text{ for no } u \in B_c\}$ be the pairs of consecutive maintenance times of component $c \in C$ and $\delta : \mathbb{Z} \rightarrow \mathbb{N}$ a filter function defined in such a way that $\delta(x) = 0$ when $x < 0$ and $\delta(x) = x$ otherwise.

Definition 3. For each schedule S , we define the component-wise over-coverage function $oc : C \rightarrow \mathbb{N}$, the component-wise under-coverage function $uc : C \rightarrow \mathbb{N}$, the component-wise miscoverage function $mc : C \rightarrow \mathbb{N}$, and the component-wise action count function $ac : C \rightarrow \mathbb{N}$ as follows.

If B_c is empty, we have $oc(c) = ac(c) = 0$, $uc(c) = mc(c) = h - \iota(c)$. Otherwise, the values of the componentwise functions are given by

$$\begin{aligned}
 oc(c) &= \delta(\iota(c) - \min B_c + 1) + \sum_{(s,t) \in \pi(c)} \delta(\min(s + \rho(c), h + 1) - t), \\
 uc(c) &= \delta(\min B_c - \iota(c) - 1) + \sum_{(s,t) \in \pi(c) \cup \{(\max B_c, h+1)\}} \delta(t - (s + \rho(c))), \\
 mc(c) &= oc(c) + uc(c), \quad ac(c) = |B_c|.
 \end{aligned}$$

Intuitively, the value $oc(c)$ indicates the number of time points during which the implementation of a due preventive maintenance action of component c is advanced earlier from the time suggested by the recommended maintenance interval $\rho(c)$. The value $uc(c)$ indicates, for component c , the number of time points that are neither covered by the initial lifetime nor a recommended maintenance interval started by a preventive maintenance action. The miscoverage $mc(c)$ simply combines these two quality evaluation functions into a sum, and the action count $ac(c)$ tells the number of preventive maintenance actions of component c .

Lemma 1 links the under-coverage, the over-coverage and the number of preventive maintenance actions to each other. Lemma 2 demonstrates that over-coverage is potentially much larger than under-coverage. Finally, Lemma 3 shows that servicing too often does not help to reduce the under-coverage.

Lemma 1. *Let $c \in C$ and assume that $\ell + \rho(c) \leq h$. Then $uc(c) = h - \iota(c) - ac(c)\rho(c) + oc(c)$.*

Proof. Define first the sequence $B_c^0, B_c^1, \dots, B_c^{ac(c)}$, in such a way that $B_c^0 = \emptyset$, $B_c^1 = \min B_c$, and $B_c^k = B_c^{k-1} \cup \{t \mid (\max B_c^{k-1}, t) \in \pi(c)\}$. This gives us a growing sequence of service times $t_1 = \max B_c^1, \dots, t_{ac(c)} = \max B_c^{ac(c)}$.

The lemma is now proven by induction on k , $0 \leq k \leq ac(c)$.

$$\begin{array}{ll}
k = 0: & oc_0(c) = 0, \\
& uc_0(c) = h - \iota(c) - k\rho(c) + oc_0(c). \\
k = 1: (i) \quad \iota(c) < t_1: & oc_1(c) = 0, \\
& uc_1(c) = h - \iota(c) - k\rho(c) + oc_1(c). \\
(ii) \quad \iota(c) \geq t_1: & oc_1(c) = t_1 - \iota(c) + 1, \\
& uc_1(c) = h - \iota(c) - k\rho(c) + oc_1(c). \\
k > 1: (i) \quad t_{k-1} + \rho(c) \leq t_k: & oc_k(c) = oc_{k-1}(c), \\
& uc_k(c) = h - \iota(c) - k\rho(c) + oc_k(c). \\
(ii) \quad t_{k-1} + \rho(c) > t_k: & oc_k(c) = oc_{k-1}(c) + (t_{k-1} + \rho(c) - t_k), \\
& uc_k(c) = h - \iota(c) - k\rho(c) + oc_k(c).
\end{array}$$

Thus $uc_k(c) = h - \iota(c) - k\rho(c) + oc_k(c)$ for all $k, 1 \leq k \leq ac(c)$. \square

Lemma 2. *Let $c \in C$, and assume that $\ell + \rho(c) - 1 \leq h$ and $\iota(c) = 0$. Then $0 \leq uc(c) \leq h$, and $0 \leq oc(c) \leq (\ell - 1)(\rho(c) - 1)$.*

Proof. Clearly, $uc(c) \geq 0$. On one hand, the value $uc(c)$ reaches 0 when the services of the component c occur at time points ℓ and $t = \iota(c) + 1 + k\rho(c)$, such that $k \geq 0$ and $t < \ell$. In this way, $uc(c) = h - (\ell + \rho(c) - 1)$. If $\ell = h - \rho(c) + 1$, we reach $uc(c) = h - (h - \rho(c) + 1 + \rho(c) - 1) = 0$. If $\ell > h - \rho(c) + 1$ and $B_c = \{1, \dots, h\}$, we still have $uc(c) = 0$. On the other hand, the value $uc(c)$ is the greatest when $B_c = \emptyset$. In this case, $uc(c) = h$, and $oc(c) = 0$.

Clearly, $oc(c) \geq 0$. In fact, $oc(c) = 0$ when $B_c = \emptyset$, but the value $oc(c)$ is the greatest when $B_c = \{1, \dots, \ell\}$. In this case, $oc(c) = (\ell - 1)(\rho(c) - 1)$. \square

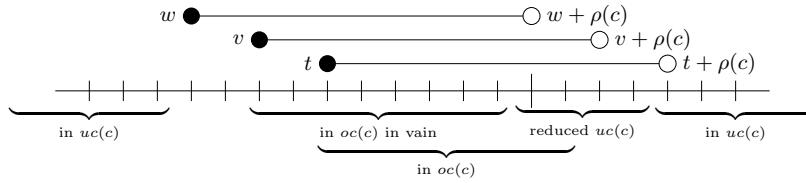


Fig. 3. Servicing Too Often Does Not Pay Off

Lemma 3. *Let w and t such that $w + 2 \leq t \leq w + \rho(c)$ be two maintenance times of component c , contributing $w + \rho(c) - t$ to $oc(c)$. Adding a maintenance time v , such that $w + 1 \leq v \leq t - 1$, increases $oc(c)$ without decreasing $uc(c)$. \square*

Proof. Fig. 3 shows a situation described in the lemma. The service at time point v has only an increasing effect on the over-coverage of the schedule. \square

The component-wise evaluation functions are lifted for a machine as follows:

Definition 4. *For every schedule $S = \langle h, \ell, b, A \rangle$, there are associated measures: the over-coverage $oc(C) = \sum_{c \in C} oc(c)$, the under-coverage $uc(C) = \sum_{c \in C} uc(c)$, the miscoverage $mc(C) = uc(C) + oc(C)$, and action count $ac(C) = \sum_{c \in C} ac(c)$.*

In the rest of the paper, when solving PMS problems, we employ no other measures of quality than under-coverage and miscoverage. The following two lemmas identify corner cases for these measures.

Lemma 4. *Let $\mathcal{M} = \langle C, \iota, \rho \rangle$ be a multi-component machine. For any horizon $h \in \mathbb{N} \setminus \{0\}$, and limit $\ell \in \{1, \dots, h\}$, there is a schedule $S = \langle h, \ell, b, A \rangle$ such that the over-coverage $oc(C)$ associated with the schedule is 0.*

Proof. The over-coverage $oc(C)$ of the empty schedule is 0. \square

Lemma 5. *Let $\mathcal{M} = \langle C, \iota, \rho \rangle$ be a multi-component machine. For any horizon $h \in \mathbb{N} \setminus \{0\}$, and breakset size $b \geq \lceil h / \min_c \rho(c) \rceil$ there is a schedule $S = \langle h, \ell, b, A \rangle$ such that the under-coverage $uc(C)$ associated with S is 0.*

Proof. Assume without loss of generality that $\iota(C) = \{0\}$. Construct a schedule $\langle h, \ell, b, A \rangle$ such that $B_c = \{1 + i \times (\min_c \rho(c)) \mid i = 0, \dots, b - 1\}$ for all $c \in C$. These preventive maintenance actions are enough to cover the scheduling timeline, giving under-coverage $uc(C) = 0$. \square

3 Basic PMS Problems and Their Complexities

In the following, we define some variants of the PMS problem and study their computational complexities.

Definition 5. *The EXACT MISCOVERAGE PMS problem is a decision problem that assumes, as its input, a multi-component machine $\mathcal{M} = \langle C, \iota, \rho \rangle$ and a quadruple $T = \langle h, \ell, b, m \rangle$ of scheduling parameters, and poses the question whether the machine has a schedule $S = \langle h, \ell, b, A \rangle$ such that $mc(C) = m$.*

The NP membership of EXACT MISCOVERAGE PMS will be shown under the assumption that all the elements of the scheduling timeline $\{1, \dots, h\}$ are separate units of the input, i.e., h is essentially encoded in unary. The following lemma gives an upper bound for the computation of the miscoverage.

Lemma 6. *Let $\mathcal{M} = \langle C, \iota, \rho \rangle$ be an arbitrary multi-component machine and $S = \langle h, \ell, b, A \rangle$ be one of its schedules. The miscoverage $mc(C)$ associated with S can be computed in $O(|C|h)$ time and $O(|C|h)$ space.*

Proof. Assume accessing of A takes $O(1)$ time steps. We compute the miscoverage of the schedule S with a loop that runs over C , and with an inner loop over all time points $\{1, \dots, h\}$ in $O(|C|h)$ time: For each component c , the inner loop starts from time point 1 and keeps track of the initial lifetime and the RMI that starts at a preventive maintenance action. (i) Every extra RMI covering the time point contributes one over-coverage to $mc(C)$. (ii) Each time point not covered by any RMI contribute one under-coverage to $mc(C)$. In addition, it is safe to say that the space requirement of the computation is $O(|C|h)$. \square

Theorem 1. *The EXACT MISCOVERAGE PMS problem is in NP.*

Proof. Let an EXACT MISCOVERAGE PMS problem instance consist of a multi-component machine $\mathcal{M} = \langle C, \iota, \rho \rangle$ and parameters $T = \langle h, \ell, b, m \rangle$. Since $\iota(c) < \rho(c)$ for all $c \in C$, and $h \geq \ell \geq b$, the length of the input is $\Omega(|C| \max_c \rho(c) + h + \log m)$. A certificate to the EXACT MISCOVERAGE PMS problem consists of a schedule $S = \langle h, \ell, b, A \rangle$ that requires $O(|C|h)$ space, which is polynomial in the input size. Let S be an arbitrary certificate. To verify it, we only need to check that the miscoverage $mc(C)$ associated with S equals m . By Lemma 6, $mc(C)$ can be computed in a polynomial time and space. \square

The SUBSET SUM problem (SSP) is an example of an NP-complete problem [10]. In the sequel, it will be shown to be at most as hard as the EXACT MISCOVERAGE PMS problem by using an appropriate reduction.

Definition 6. *The SSP is a decision problem that assumes as its instance a pair $\langle N, s \rangle$ where $N = \{c_1, c_2, \dots, c_n\}$ is a multiset of positive integers $c_i \in \mathbb{N} \setminus \{0\}$, $i = 1, \dots, n$, and $s \in \mathbb{N} \setminus \{0\}$ is the target sum for a subset of these integers. The problem is to decide whether there is a subset $N' \subseteq N$ such that the target s is obtainable as the sum of the elements of N' , i.e., $\sum N' = s$.*

Theorem 2. *The EXACT MISCOVERAGE PMS problem is NP-hard.*

Proof. Let $P = \langle N, s \rangle$, $N = \{c_1, c_2, \dots, c_n\}$ be an arbitrary instance of SSP. Without loss of generality, assume that $c_i \leq c_{i+1}$, for $1 \leq i \leq n - 1$. This SSP instance reduces by a poly-time function to an instance of EXACT MISCOVERAGE PMS given by machine $\mathcal{M} = \langle C, \iota_0, \rho \rangle$ and scheduling parameters $T = \langle \max N, 1, 1, n \max N - s \rangle$, such that $C = \{1, \dots, n\}$, and $\iota_0(i) = 0$, $\rho(i) = c_i$ for all $i \in C$. Since all preventive maintenance actions are enforced to occur at time step 1, we have $oc(C) = 0$ for all schedules S . If S is the empty schedule, then $uc(C)$ reaches its maximum value $n \max N$.

Let $N' \subseteq N$ be a multiset subset with sum s , being a solution to the SSP instance $\langle N, s \rangle$. In the reduction, each integer $c_i \in N'$ is encoded by $A(i, 1) = 1$, while $A(i, 1) = 0$ encodes that $c_i \notin N'$. This gives a schedule $\langle \max N, 1, 1, A \rangle$ whose associated under-coverage $uc(C)$ is $n \max N - s$. This schedule is a solution to the EXACT MISCOVERAGE PMS instance $\langle \mathcal{M}, T \rangle$.

Conversely, let $S = \langle \max N, 1, 1, A \rangle$ be a solution to the EXACT MISCOVERAGE PMS instance $\langle \mathcal{M}, T \rangle$. The associated under-coverage is $n \max N - s$. The schedule encodes the multiset subset $N' = \{c_i \mid i \in C, A(i, 1) = 1\} \subseteq N$ with sum s . This subset is a solution to the SSP instance $\langle N, s \rangle$. \square

Definition 7. *The BOUNDED MISCOVERAGE PMS problem is a decision problem that assumes, as its input, a multi-component machine $\mathcal{M} = \langle C, \iota, \rho \rangle$ and a quadruple $T = \langle h, \ell, b, m \rangle$ of bounds, and poses the question whether the machine has a schedule $S = \langle h, \ell, b, A \rangle$ such that $mc(C) \leq m$.*

Theorem 3. *The BOUNDED MISCOVERAGE PMS problem is in NP.*

The proof of Theorem 3 is similar to Theorem 1 and thus left to the reader.

Function problems can be defined almost in a similar way as decision problems. The following function problems concern the optimization of schedules with respect to particular measures associated with them.

Definition 8. *The UNDER-COVERAGE PMS and MISCOVERAGE PMS are optimizing function problems whose inputs consists of a multi-component machine $\mathcal{M} = \langle C, \iota, \rho \rangle$ and a triple $\langle h, \ell, b \rangle$ of scheduling parameters. The solution to the UNDER-COVERAGE PMS problem is a schedule $S = \langle h, \ell, b, A \rangle$ such that the under-coverage $uc(C)$ associated with the schedule S is minimized, and the solution to the MISCOVERAGE PMS problem is a schedule $S = \langle h, \ell, b, A \rangle$ such that the miscoverage $mc(C)$ associated with the schedule S is minimized.*

4 An ASO-Based Implementation

In what follows, we present an ASP encoding of the PMS problem. The encoding is presented in the language fragment of the GRINGO grounder. Thus existing ASP solvers such as CLASP [12] and WASP [2] can be readily used to implement the search for optimal schedules in practice.

Listing 1 illustrates the representation of an eight-component machine using the domain predicate `comp/3` whose arguments give the identity `C`, the recommended maintenance interval `R`, and the initial lifetime `L` due to a preventive maintenance action before the scheduling timeline. The actual PMS encoding has been divided into three sections given in Listings 2–4.

In Listing 2 (Line 2), the parameters `h`, `l` and `b` for the number of time steps, the limit for the last maintenance break, and the number of scheduled maintenance breaks, respectively, are set to their default values. In Line 3, we define `time/1` as a domain predicate for representing time steps. Moreover, the identities of components are extracted as the extension of the `comp/1` predicate

Listing 1. A PMS Problem Instance

1	<code>comp(1,5,2).</code>	<code>comp(3,7,0).</code>	<code>comp(5,9,0).</code>	<code>comp(7,5,4).</code>
2	<code>comp(2,10,0).</code>	<code>comp(4,4,3).</code>	<code>comp(6,11,2).</code>	<code>comp(8,8,0).</code>

Listing 2. PMS Encoding: Parameters, Domains, Choices, and Actions

1	<code>% Parameters and domains</code>
2	<code>#const h=32. #const l=32. #const b=3.</code>
3	<code>time(1..h).</code>
4	<code>comp(C) :- comp(C,_,_).</code>
5	
6	<code>% Breaks and the allocation of components for service</code>
7	<code>{ break(T): time(T), T <= l } <= b.</code>
8	<code>1 <= { serv(C,T): comp(C) } :- break(T).</code>
9	
10	<code>% End of maintenance interval</code>
11	<code>emi(C,T+R) :- comp(C,R,L), serv(C,T), time(T+R).</code>
12	<code>emi(C,L+1) :- comp(C,R,L), L>0, time(L+1).</code>

in Line 4, recall `comp/3` in Listing 1. Then we are ready to choose time steps for scheduled maintenance breaks, as formalized by the choice rule in Line 7. In our basic encoding, at most `b` breaks are picked directly and for each scheduled maintenance break at least one component is selected for maintenance in Line 8. This is represented in terms of the `serv/2` predicate. In Line 11, we define when the respective RMI ends using the `emi/2` predicate. For components `C` with some initial lifetime, the analogous time point is defined in Line 12.

Based on the predicates introduced so far, we may define how points of time are covered by RMIs on a component-by-component basis, see Listing 3. This leads to an encoding ('**2-level**') with recursive definitions of two predicates `cov1/2` and `cov2/2` denoting that a component `C` is covered by exactly one or two RMIs, respectively. Naturally, the target is that `cov1(C,T)` would hold for every `C` and `T`, indicating that `C` is neither under nor over serviced over time. The base cases for `cov1` are given in Lines 2–3: either `C` is being maintained at time `T` or it has some initial lifetime `L` in the beginning due to recent servicing. Based on this, we may infer that `cov1(C,T+1)` holds for the following time step `T+1`: either `C` is not maintained again nor the RMI ends (Line 5), or `C` is maintained again but the preceding RMI ends at the same time (Line 6). The third possibility is that `C` is maintained again within the same RMI, a reason for making `cov2(C,T+1)` true in Line 7. The *inertial* rules for `cov2/2` in Lines 9–10 are analogous to the ones of `cov1/2` in Lines 5–6. The final possibility is formalized by Line 11: there is an end of a RMI falsifying `cov2(C,T+1)` but making `cov1(C,T+1)` true again. Note that according to answer set semantics, all instances of `cov1(C,T)` and `cov2(C,T)` are *false by default*. Thus, we need not specify the cases when they are falsified.

Finally, Listing 4 sets the scene for solving the optimization problem in question. Firstly, in Line 2, more than two overlapping RMIs are banned in the spirit

Listing 3. PMS Encoding: Counting Overlap of Intervals

```

1 % Component-specific coverage of time points by RMIs
2 cov1(C,T) :- serv(C,T).
3 cov1(C,1) :- comp(C,R,L), L>0.
4
5 cov1(C,T+1) :- cov1(C,T), not serv(C,T+1), not emi(C,T+1), time(T+1).
6 cov1(C,T+1) :- cov1(C,T), serv(C,T+1), emi(C,T+1), time(T+1).
7 cov2(C,T+1) :- cov1(C,T), serv(C,T+1), not emi(C,T+1), time(T+1).
8
9 cov2(C,T+1) :- cov2(C,T), not serv(C,T+1), not emi(C,T+1), time(T+1).
10 cov2(C,T+1) :- cov2(C,T), serv(C,T+1), emi(C,T+1), time(T+1).
11 cov1(C,T+1) :- cov2(C,T), not serv(C,T+1), emi(C,T+1), time(T+1).

```

Listing 4. PMS Encoding: Constraints and Objective Function

```

1 % Deny (excessive) overlaps of RMIs
2 :- cov2(C,T), serv(C,T+1), not emi(C,T+1), time(T+1).
3
4 % Optimization with respect to miscoverage
5 #minimize {1,C,T: not cov1(C,T), comp(C), time(T);
6           1,C,T: cov2(C,T), comp(C), time(T) }.

```

of Lemma 3. This is also why predicates `cov1/2` and `cov2/2` are sufficient to keep track of overlaps in the first place. Secondly, the objective function penalizes for under servicing (time points not covered) and over servicing (time points covered twice) on equal basis. Note that the mutual exclusion of these two cases partially counts on equal tuples $1, C, T$ in the set representation in Lines 5–6. For the sake of illustration, we have depicted two globally optimal schedules for the 8-component machine from Listing 1. The one in Fig. 4a is based on the minimization of under-coverage (white cells) only. This leads to a substantial overlap of RMIs and over-coverage indicated in red. A form of a periodic pattern of two scheduled maintenance breaks maintaining together all components of the machine is ultimately emerging, but certain components like number 6 are maintained selectively, thus better meeting the component-specific RMI. However, if over-coverage is penalized equally, far better schedules result as illustrated in Fig. 4b. The risks from under service are slightly higher but the costs resulting from unnecessary preventive maintenance actions are decreased substantially. The number of individual preventive maintenance actions is also decreased from 44 to 36. Interestingly, the dimensions of the schedules reflect the space complexity of ground logic programs obtained from our encoding (cf. Lemma 6). The effect of the bound b is negligible.

Proposition 1. *The size of the ground program resulting from Listings 2–4 is $\mathcal{O}(|C|h)$ for a set of components C and the time horizon h .*

Theorem 4. *Let $\mathcal{M} = \langle C, \iota, \rho \rangle$ be a machine, $\langle h, \ell, b \rangle$ the triple of scheduling parameters, and $P_{\mathcal{M}}$ their representation as a ground logic program based on Listings 2–4 and a set of facts encoding \mathcal{M} .*

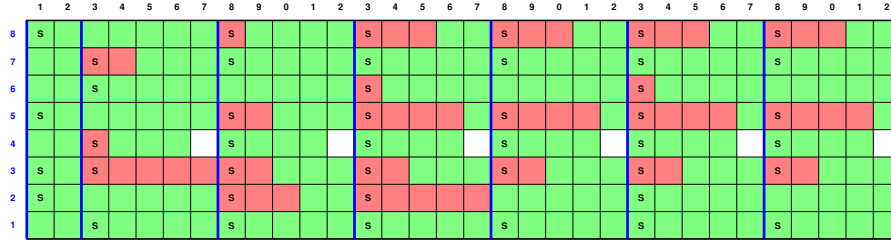
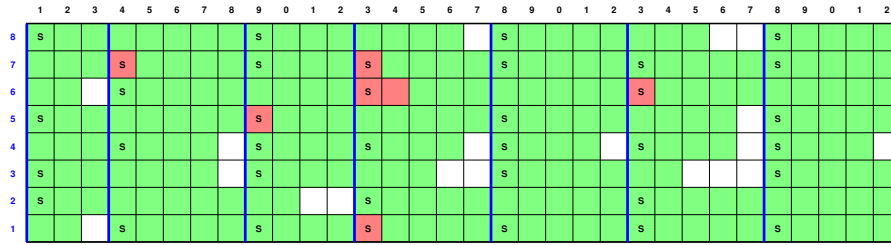
(a) Schedule for $b = 7$ (Under-coverage)(b) Schedule for $b = 7$ (Miscoverage)

Fig. 4. Globally optimal schedules for the machine of Listing 1. Blue lines indicate scheduled maintenance breaks and individual preventive maintenance actions are marked with letter “s”.

1. If X is an (optimal) answer set of $P_{\mathcal{M}}$, then there is an (optimal) solution $S_X = \langle h, \ell, b, A_X \rangle$ to the MISCOVERAGE PMS problem $\langle \mathcal{M}, \langle h, \ell, b \rangle \rangle$.
2. If a schedule $S = \langle h, \ell, b, A \rangle$ is an (optimal) solution to the MISCOVERAGE PMS problem $\langle \mathcal{M}, \langle h, \ell, b \rangle \rangle$, then there is an (optimal) answer set X_S of $P_{\mathcal{M}}$.

Proof (Sketch). Due to space restrictions, we concentrate on describing the one-to-one correspondence between answer sets and PMSes as follows. Firstly, given an answer set X of $P_{\mathcal{M}}$, the respective schedule $S_X = \langle h, \ell, b, A_X \rangle$ where for a component $c \in C$ and a time point $1 \leq i \leq h$, $A_X(c, i) = 1 \iff \mathbf{serv}(c, i) \in X$.

Secondly, given a PMS $S = \langle h, \ell, b, A \rangle$ for \mathcal{M} subject to scheduling parameters $\langle h, \ell, b \rangle$, we may calculate for each component and a point of time $1 \leq i \leq h$,

$$\mathit{cnt}(c, i) = |\{j \in B_c \mid j \leq i < j + \rho(c)\}| + |\{1 \mid \iota(c) > 0, i \leq \iota(c)\}|, \quad (1)$$

i.e., the number of recommended maintenance intervals covering i while maintaining $c \in C$. Based on these, the respective answer set X_S of $P_{\mathcal{M}}$ contains

- $\mathbf{comp}(c, \rho(c), \iota(c))$ and $\mathbf{comp}(c)$ for every $c \in C$;
- $\mathbf{time}(i)$ for every $1 \leq i \leq h$;
- $\mathbf{break}(i)$ for every $1 \leq i \leq h$ such that $A(c, i) = 1$ for some $c \in C$;
- $\mathbf{serv}(c, i)$ for every $c \in C$ and $1 \leq i \leq h$ such that $A(c, i) = 1$;
- $\mathbf{emi}(c, i + \rho(c))$ for every $c \in C$ and $1 \leq i \leq h$ with $A(c, i) = 1$ and $i + \rho(c) \leq h$;

- $\text{emi}(c, \iota(c) + 1)$ for every $c \in C$ with $\iota(c) > 0$ and $\iota(c) + 1 \leq h$;
- $\text{cov1}(c, i)$ for every $c \in C$ and $0 \leq i \leq h$ such that $\text{cnt}(c, i) = 1$; and
- $\text{cov2}(c, i)$ for every $c \in C$ and $0 \leq i \leq h$ such that $\text{cnt}(c, i) = 2$.

The idea is that $X_{(S_X)} = X$ and $S_{(X_S)} = S$ hold in the bijective correspondence. Moreover, the measures $uc(c) = |\{i \mid \text{cov1}(c, i) \notin X, \text{cov2}(c, i) \notin X\}|$ and $oc(c) = |\{i \mid \text{cov2}(c, i) \in X\}|$ can be read off from answer sets X . Thus, the minimality of $mc(C) = uc(C) + oc(C)$ matches with the optimality of X . \square

It is known that function problems corresponding to optimization problems that can be formalized in terms of disjunction-free logic programs (as above) are FP^{NP} -complete function problems [15], i.e., only polynomially many calls to an NP-oracle are needed in the worst case. As a consequence, the computational complexities of MISCOVERAGE PMS and UNDER-COVERAGE PMS are bounded from above and the corresponding decision problems reside in Δ_2^P .

Corollary 1. *Given a machine $\mathcal{M} = \langle C, \iota, \rho \rangle$ and the scheduling parameters $\langle h, \ell, b \rangle$, the respective function problems MISCOVERAGE PMS and UNDER-COVERAGE PMS are in FP^{NP} .*

Hardness results in this respect are left as future work.

5 Experiments

In this section, we evaluate the performance of the encoding from Listings 2–4. In the experiments, we use GRINGO (v. 5.2.2) as the grounder and CLASP (v. 3.3.4) as the solver. All runs are executed on an Intel(R) Core i7-8750H CPU with a 2.20GHz clock rate under Linux operating system. Our preliminary screening showed that the 8-component machine from Listing 1 is already sufficient to create great variance with respect to runtimes. Thus, we concentrate on analyzing performance of CLASP on this particular instance when the number of scheduled maintenance breaks b varies from 1 to 15 for schedules in a scheduling timeline $h = 32$. The time required for grounding is negligible and omitted altogether. The runtime behavior of CLASP as the back-end solver can be inspected from Fig. 5. All graphs are smoothed by using the *sbezier* option of GNU PLOT.

Minimization of under-coverage is quite easy. The lowest two plots in Fig. 5a concern optimization based on two different strategies known as *branch-and-bound* (**BB**) and *unsatisfiable cores* (**USC**). At first, the BB strategy is faster but becomes slower than USC when almost fully covered schedules become feasible at $b = 9$. The upper two plots relate to the minimization of miscoverage which seems to be far more difficult task from the computational point of view. Now the USC strategy performs much better. We think that this is due to the fact that USC approaches optimal solutions from below and since the values of the objective function are relatively small in this example, the optimal value can be reached soon. The BB strategy, however, uses upper bounds and finally, when the optimality of a found schedule is to be proved, a potentially high number of other candidates—not improving the objective value—have to be excluded

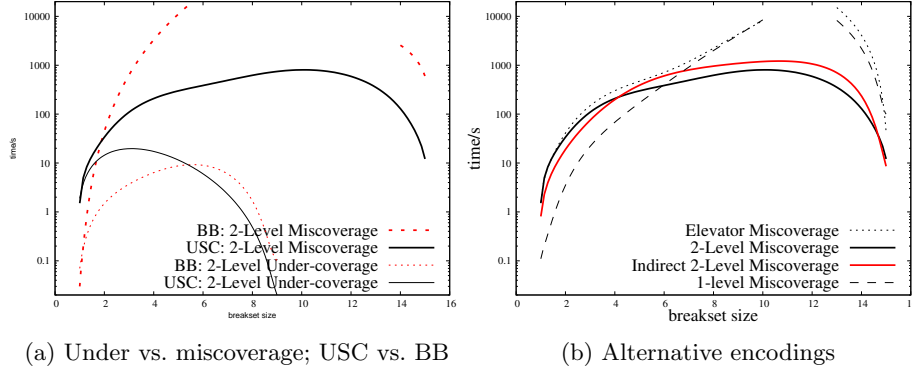


Fig. 5. Runtimes in seconds for $b = 1 \dots 15$ on logarithmic scale

by the solver. Indeed, the last stage of optimization dominates for BB, and the runtimes for $b = 7 \dots 13$ are clear outliers far above the values visible in Fig. 5a.

We have developed several variants of the ASP encoding and include results for some of them in Fig. 5b. The first alternative encoding (**'Indirect'**) formalizes the choice of scheduled maintenance breaks (Line 7 in Listing 2) *indirectly*, i.e., the scheduled maintenance breaks are inferred from selected preventive maintenance actions (recall Line 8 in Listing 2) and their number is constrained accordingly. However, the performance is slightly worse than our basic encoding presented above, spending roughly double time cumulatively. Yet another encoding (**'1-Level'**) infers coverage and over-coverage information *straight* from preventive maintenance actions, e.g., if `serv(C,T)` is made true (Line 8) then `cov(C,T)`, `cov(C,T+1)`, \dots , `cov(C,T+R-1)` are inferred for the length R of the RMI. Analogous rules for over-coverage `ocov/2` are no longer linear in scheduling timeline h . It performs very well for small values of b , but degrades soon such that values $b = 11, 12, 13$ turn out to be clear outliers falling outside the illustration. Finally, we mention our early encoding (**'Elevator'**) based on up-and-down *counting* of the number of overlapping maintenance intervals according to (1). In Listing 3, the predicates `cov1/2` and `cov2/2` are the counterparts of `cnt(C,T,1)` and `cnt(C,T,2)` used in that encoding, and there is no pendant for `cnt(C,T,0)` expressible via default negation. The respective plot in Fig. 5b indicate that such a systematic saving in the number of predicates pays off.

To further assess the scalability of the method, we tested the 2-Level encoding and the USC strategy with 10 randomly generated machines per each of the sizes from 1 to 16 components. Fig. 6 shows the averages of the running times in this experiment. When optimizing the miscoverage, a majority of the larger machine sizes hit the timeout of 1200s. This highlights that without approximations and further optimizations, the problem cannot be solved on a large scale.

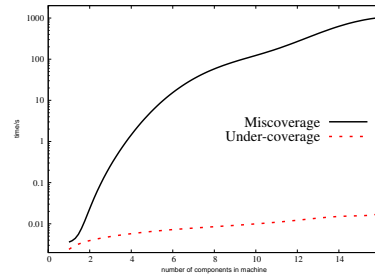


Fig. 6: Effect of Machine Size

6 Conclusion

In this paper, we have focused on a lesser studied type of PMS that involves one continuously operating, multi-component machine instead of multiple machines that can be serviced separately. Our formalization focuses on maintenance timing while the duration of maintenance is abstracted away. The current model has also taken a simplistic view to the end of the timeline. This was enabled by the combination of horizon h and limit ℓ that was used to parameterize the way the residual lifetimes of components are taken into account in cost functions. Alternative approaches could assume that the schedule becomes ultimately periodic, or that the distant future and its associated cost fades out by discounting in exchange for the closer moments whose scheduling costs are more tightly optimized, and clearly dominating when it comes to decision making.

After formalizing the problem as a search problem, we analyzed the computational complexity of the problem and presented an ASP encoding of the optimization problem that minimizes the schedule with respect to a cost function. The experiments on minor variants of the encoding demonstrate that the MISCOVERAGE PMS problem is indeed a challenging one. Although changes in encoding make a huge difference, the current experiments handle only small problem sizes. Thus, since practical multi-component machines can have orders of thousands serviceable components and the time line of PMS is expected to cover months or years (i.e., hundreds of time points), the quest for performance improvement is still substantial. It is worth noting, however, that we computed globally optimal schedules, and approximations in this respect are possible.

It is also within our interests to study the PMS problem in context of additional constraints, which are usually easy to incorporate into ASP encodings in an orthogonal way. In this respect, specific constraints on *availability* may determine when certain components of the system can be maintained. If such constraints become strict, some maintenance operations must be assigned to specific points of time. In less strict settings, however, availability constraints determine allowed time intervals within which the maintenance operations must be scheduled; see, e.g., [17, 18] where these intervals depend on workload or the current condition of the machine.

References

1. Ali, M.B., Sassi, M., Gossa, M., Harrath, Y.: Simultaneous scheduling of production and maintenance tasks in the job shop. *International Journal of Production Research* **49**, 3891–3918 (2011)
2. Alviano, M., Dodaro, C., Leone, N., Ricca, F.: Advances in WASP. In: *LPNMR 2015*. pp. 40–54 (2015)
3. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. *Commun. ACM* **54**(12), 92–103 (2011)
4. Chansombat, S., Pongcharoen, P., Hicks, C.: A mixed-integer linear programming model for integrated production and preventive maintenance scheduling in the capital goods industry. *International J. of Production Research* **57**(1), 61–82 (2019)
5. Chen, X., An, Y., Zhang, Z., Li, Y.: An approximate nondominated sorting genetic algorithm to integrate optimization of production scheduling and accurate maintenance based on reliability intervals. *Journal of Manufacturing Systems* **54**, 227–241 (2020)
6. Do, P., Vu, H.C., Barros, A., Bérenguer, C.: Maintenance grouping for multi-component systems with availability constraints and limited maintenance teams. *Reliability Engineering & System Safety* **142**, 56–67 (2015)
7. Dodaro, C., Maratea, M.: Nurse scheduling via answer set programming. In: *LPNMR 2017*. pp. 301–307. Springer (2017)
8. Eiter, T., Geibinger, T., Musliu, N., Oetsch, J., Skocovský, P., Stepanova, D.: Answer-set programming for lexicographical makespan optimisation in parallel machine scheduling. In: *KR 2021*. pp. 280–290 (2021)
9. Frost, D., Dechter, R.: Optimizing with constraints: A case study in scheduling maintenance of electric power units. In: *CP 1998*. p. 469. Springer (1998)
10. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman (1979)
11. Garg, H., Rani, M., Sharma, S.P.: Preventive maintenance scheduling of the pulping unit in a paper plant. *Japan Journal of Industrial Applications of Mathematics* **30**, 397–414 (2013)
12. Gebser, M., Kaminski, R., Kaufmann, B., Romero, J., Schaub, T.: Progress in clasp series 3. In: *LPNMR 2015*. pp. 368–383 (2015)
13. Geurtsen, M., Didden, J.B., Adan, J., Atan, Z., Adan, I.: Production, maintenance and resource scheduling: A review. *European Journal of Operational Research* (2022)
14. Sachdeva, A., Kumar, D., Kumar, P.: Planning and optimizing the maintenance of paper production systems in a paper plant. *Computers & Industrial Engineering* **55**, 817–829 (2008)
15. Simons, P., Niemelä, I., Soinen, T.: Extending and implementing the stable model semantics. *Artificial Intelligence* **138**(1-2), 181–234 (2002)
16. Wannawiset, S., Tangjitsitharoen, S.: Paper machine breakdown reduction by FMEA and preventive maintenance improvement: A case study. In: *ICRAIEM 2018. IOP Conferences in Materials Science and Engineering*, vol. 530 (2019)
17. Youssef, H., Brigitte, C.M., Noureddine, Z.: Lower bounds and multiobjective evolutionary optimisation for combined maintenance and production scheduling in job shop. In: *IEEE 2003 Conference on EFTA*. vol. 2, pp. 95–100 (2003)
18. Zheng, Y., Lian, L., Mesghouni, K.: Comparative study of heuristics algorithms in solving flexible job shop scheduling problem with condition based maintenance. *Journal of Industrial Engineering and Management* **7**(2 Special Issue), 518–531 (2014)