Pruning Redundancy in Answer Set Optimization Applied to Preventive Maintenance Scheduling

Anssi Yli-Jyrä[0000−0003−0731−2114],
Masood Feyzbakhsh Rankooh[0000−0001−5660−3052], and
Tomi Janhunen[0000−0002−2029−7708]

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

Abstract. Multi-component machines deployed, e.g., in paper and steel industries, have complex physical and functional dependencies between their components. This profoundly affects how they are maintained and motivates the use of logic-based optimization methods for scheduling preventive maintenance actions. Recently, an abstraction of maintenance costs, called miscoverage, has been proposed as an objective function for the preventive maintenance scheduling (PMS) of such machines. Since the minimization of miscoverage has turned out to be a computationally demanding task, the current paper studies ways to improve its efficiency. Given different answer set optimization encodings of the PMS problem, we motivate constraints that prune away some sub-optimal and otherwise redundant or invalid schedules from the search space. Our experimental results show that these constraints may enable up to ten-fold speed-ups in scheduling times, thus pushing the frontier of practically solvable PMS problem instances to longer timelines and larger machines.

1 Introduction

Multi-component machines deployed, e.g., in paper and steel industries, have complex physical and functional dependencies between their components forming an entire production line. Moreover, the machinery should be kept constantly in operation to maximize production. These aspects profoundly affect the ways in which such machines can be maintained in the first place. Besides this, further constraints emerge from resources available for carrying out particular maintenance actions; see, e.g., [3] for taking such features into account. There are two main-stream *maintenance policies*, viz. *corrective* and *preventive*. The former is failure-driven and typically supersedes the latter that is complementary by nature and aims to ensure the full operationality of a machine in the long run.

The multitude of concerns, as briefed above, calls for highly flexible scheduling methods. Such potential is offered by logic-based methods supporting optimization, including *constraint optimization problems* (COPs) [11, S. 7.4], *mixed-integer programming* (MIP) [11, S. 15.4], *maximum satisfiability* (MaxSAT) [9],

and *answer set optimization* (ASO) [12]. Since answer set programming [2] is known to be well-suited for solving scheduling and resource allocation problems [1, 4, 10], we continue the development of ASO-based encodings [14] for solving *preventive maintenance scheduling* (PMS) problems in this work. These encodings abstract on maintenance costs and aim at *covering* the timeline of each component by maintenance actions that should occur with recommended component-specific intervals. Delays in this respect give rise to *under-coverage* while servicing too often denotes *over-coverage*. Based on this correspondence, the minimization of the respective objective function penalizes for overall *miscoverage*. Since the machine is assumed to be inoperable during service breaks, the breaks form a central resource to be utilized maximally for maintenance, as far as excessive servicing is avoidable. For the same reason, components cannot be maintained independently of each other and, in addition, we assume the omission of all components serviceable during the normal operation of the machine. The number of service breaks is treated as a parameter whose value is to be minimized implicitly while optimizing schedules otherwise.

The previous ASO-based encodings from [14] can be understood as *golden designs* that provide us with the basis for solving PMS problem instances correctly with respect to their formal definition. Since the global minimization of miscoverage tends to be computationally demanding in the numbers of both service breaks and components, the current paper studies ways to improve the efficiency of encodings by incorporating constraints that prune away sub-optimal and otherwise redundant or invalid schedules from the search space. In addition to expressing some known properties of optimal solutions to remove sub-optimal solutions directly, such pruning constraints may also be used to break *symmetries* present in the search space. Symmetries have been studied in the context of constraint programming [8] as well as ASP [5]. Symmetry breaking constraints are often incorporated by programmers themselves, but there is also recent work aiming at their automated detection [13]. Regardless how such constraints are devised, they intensify optimization by removing both (i) equally good solutions that are uninteresting modulo symmetries and (ii) invalid (partial) solutions.

Our experimental results indicate that pruning constraints may enable up to ten-fold speed-ups in scheduling times, thus pushing the frontier of practically solvable PMS problem instances to longer timelines and larger machines. The main contributions of our work are as follows:

- The characterization of miscoverage-based PMS [14] in terms of new constraints that prune the search space for optimal schedules.
- The respective correctness arguments for the pruning constraints, indicating that at least one globally optimal schedule will be preserved.
- Experimental analysis revealing the effects of the pruning constraints on the performance of the Clingo system in the computation of optimal schedules.

In this way, we demonstrate the extensibility of the original framework with additional constraints; the focus being on performance improvement whereas other constraints affecting the quality of schedules are left as future work.

The plot for this article is as follows. In Section 2, we recall the formal definitions of multi-component machines and their preventive maintenance schedules, including the coverage-based objective functions that are relevant for optimization. The respective baseline encodings [14] of preventive maintenance scheduling (PMS) problems are summarized in Section 3. Then, we are ready to present novel constraints that can be used to prune the search space for optimal schedules in various respects. Besides the proofs of correctness, we encode these constraints in terms of rules that are compatible with the baseline encodings, enabling straightforward combinations. The experimental part in Section 5 investigates the effect of the new constraints on the performance of the Clingo system when solving PMS problems optimally. Section 6 concludes this work.

2 Definitions

In this section, we recall the preventive maintenance scheduling problem from [14]. The definitions abstract away from the complications of practical maintenance on purpose, ignoring such aspects as the shutdown/restarting costs, the duration of maintenance, the availability of maintenance resources, and the feasible combinations of maintenance actions. The abstracted problem is founded on the notions of a multi-component machine (Def. 1) and its preventive maintenance schedule (Def. 2). These tightly related concepts are explained as follows.

Definition 1. *A multi-component machine* $\mathcal{M} = \langle C, \iota, \rho \rangle$ *comprises of a set of* components C*, an* initial lifetime *function* $\iota : C \to \mathbb{N}$*, and a* recommended maintenance interval *function* $\rho : C \to \mathbb{N}$*, satisfying* $\iota(c) < \rho(c)$ *for each* $c \in C$*.*

The failure rate of each component c will start to grow rapidly when the time passed since the most recent maintenance action of the component reaches the recommended maintenance interval $\rho(c) > 0$. The initial lifetime $\iota(c)$ of a component c aims to capture, e.g., situations where the component has been maintained just a moment before the schedule starts. In that case, the component's lifetime at the beginning of the schedule is just a single time step less than the recommended maintenance interval, i.e., $\iota(c) + 1 = \rho(c)$. But it is also possible that the component is older, in which case its initial lifetime is even smaller. Thus we have the constraint $\iota(c) < \rho(c)$. If $\iota(c) = 0$, the recommendation is that c is serviced as soon as possible. For simplicity, we assume that $\iota(c) < h$ for all $c \in C$, because otherwise the component c need not be scheduled for maintenance.

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

- $h \in \mathbb{N} \setminus \{0\}$ – *the last time step called the* horizon*,*
- $A : C \times \{1, \ldots, h\} \to \{0, 1\}$ – service selection function *over the components and the time steps (implying the set of maintenance breaks* $B = \{t \mid c \in C,\ 1 \leq t \leq h,\ \text{and } A(c,t) = 1\}$ *for the whole machine),*
- $b \in \{0, \ldots, \ell\}$ – *the* upper bound for the number of maintenance breaks*,* $|B|$*,*

– $\ell \in \{1, \ldots, h\}$ – *the* upper bound for the last break time, $\max B \cup \{1\}$.

The schedule will impose a machine-wide *break* $t \in B$ at those time steps where at least one component is maintained, but it cannot break the run of the machine for maintenance more than b times or after the time step ℓ. For each component $c \in C$, we define $B_c = \{t \mid A(c,t) = 1\}$ as the set of time steps $1 \leq t \leq h$ when c is maintained by some *maintenance action*. The effects of all scheduled maintenance actions are uniform in the following sense: if a maintenance action is applied to a component c at time step t, the component c becomes immediately as good as new, meaning that its next due time for maintenance is as far away as the recommended maintenance interval $\rho(c)$ suggests.

The purpose of a schedule is to determine the maintenance plan for a given *timeline* $1..h$ and to keep the machine in a healthy state during the whole timeline. To support this goal, the maintenance times B_c of each component in the PMS try to *cover* the whole timeline of each component c in the following sense:

1. The initial lifetime $\iota(c)$ of a component c *covers* the time steps $1..\iota(c)$ once.
2. The maintenance of a component c at time step t *covers* the time steps $t... \min\{h, t + \rho(c) - 1\}$ once.
3. A particular time step can be covered more than once in relation to a component c, if c is maintained during its initial lifetime or c is subject to two or more maintenance actions closer than $\rho(c)$ time steps to one another.

The covering of the component-wise timelines cannot be done just by maximizing the maintenance of components at every possible maintenance break but it has to be done optimally, with respect to some cost function. Our work focuses on a cost function based on *miscoverage*. The fully general form of this function was originally given in [14], but here we confine ourselves to a specific version that is practically sufficient. The specific version divides into three components:

1. The *super-coverage* $sc(\mathcal{M}, S)$. For reasons explained in [14, Lemma 3], we immediately ignore all schedules where a component is maintained in such a way that a time step is covered more than twice. This property of the schedule corresponds to a single, but unnecessary cost that makes ignored schedules bad enough to be completely thrown away during the optimization.
2. The *under-coverage* $uc(\mathcal{M}, S)$ of schedules is a measure that amounts to the total number of component-wise time steps neither covered by initial lifetime nor by the set of time steps when each components is maintained.
3. The *over-coverage* $oc(\mathcal{M}, S)$, the dual notion for the under-coverage, is trickier to define. We rely on a *(specific) over-coverage* $oc(\mathcal{M}, S)$ that is definable, thanks to super-coverage above, as the total number of component-wise time steps covered exactly twice.

The sum of the super-coverage, under-coverage and the (specific) over-coverage costs gives us the (specific) *miscoverage function* $mc(\mathcal{M}, S) = sc(\mathcal{M}, S) + uc(\mathcal{M}, S) + oc(\mathcal{M}, S)$ for schedules S, the only cost function studied in this paper. By Lemma 3 in [14], the optimal schedules of the specific miscoverage function coincide with the optimal schedules of the (general) miscoverage function, since $sc(\mathcal{M}, S) = 0$. Therefore, we will not distinguish the two in the sequel.

Listing 1. A PMS problem instance

```
1 comp(1,5,2).    comp(3,7,0).    comp(5,9,0).    comp(7,5,4).
2 comp(2,10,0).   comp(4,4,3).    comp(6,11,2).   comp(8,8,0).
```

Definition 3. *Let f be a cost function mapping multi-component machines and their schedules to integer-valued costs. An* optimizing PMS problem *P_f is then a function problem 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 problem P_f is a schedule $S = \langle h, A, b, \ell \rangle$ that minimizes the value $f(\mathcal{M}, S)$.*

Definition 4. *The MISCOVERAGE PMS problem is an optimizing PMS problem that uses the miscoverage $mc(\mathcal{M}, S)$ as the cost for each schedule S.*

3 An ASO-Based Implementation

In what follows, we give a baseline ASP encoding called *Elevator* for the MISCOVERAGE PMS problem in the language fragment of the GRINGO grounder as described by Gebser et al. in [7]. The Elevator encoding is one of the four encodings introduced and published in connection to [14]. In addition, we announce a new encoding, called Mixed. Since all the five baseline ASP encodings will be used later in our experiments, we give brief characterizations of them.

- Elevator is an AI Planning -style encoding with a generic set of *states* $0, \ldots, n$ for each component subject to restriction $n = 2$.
- *2-Level* is a related encoding for states 1 and 2 only represented by separate predicates. This encoding is already explained in detail in [14].
- *Compact* is a compacted version of the 2-Level encoding written with one predicate for both states.
- *1-Level* is a transition-less encoding to recognize the occurrence of the state combinations $\{1, 2\}$ and $\{2\}$ from which all three states can be derived.
- *Mixed* is a synthesis of a 1-Level and 2-Level encodings. The state 1 is computed without transitions, whereas the encoding for the state 2 uses a transition from state 1.

In the ASP framework, a machine is encoded in terms of the predicate `comp/3`. For example, the atom `comp(6,11,2)` specifies that $\iota(c_6) = 2$ and $\rho(c_6) = 11$ for the component c_6. Listing 1 has an 8-component machine, with recommended maintenance intervals of the components in the range 4–11, and the initial lifetimes in the range 0–4. The scheduling parameters h, b, and ℓ are encoded as ASP constants `h`, `b`, and `l`, and the maintenance actions $\langle c, t \rangle$ with $A(c, t) = 1$ as atoms `serv`(c, t). In addition, the experiments of the current paper assume that $\ell = h$, although ℓ can also be set freely in the problem encoding.

The encoding is inspired by approaches to AI Planning where we have a set of states, a set of actions, and state transitions enforced by the actions. In Line 1,

Listing 2. PMS encoding: counting maintenance coverage (Elevator)

```
1  time(0..h). comp(C) :- comp(C,_,_). val(0..2).
2  { break(T): time(T), T>0, T<=l } <= b.
3  1 <= { serv(C,T): comp(C) } :- break(T).
4
5  inc(C,T)    :- serv(C,T).
6  dec(C,T+R)  :- serv(C,T), comp(C,R,_), time(T+R).
7  dec(C,L+1)  :- comp(C,R,L), 0<L, time(L+1).
8
9  scnt(C,0,0) :- comp(C,R,0).
10 scnt(C,0,1) :- comp(C,R,L), 0<L.
11
12 scnt(C,T,V+1) :- inc(C,T), not dec(C,T), scnt(C,T-1,V),
13                    comp(C), time(T), val(V), val(V+1).
14 scnt(C,T,V-1) :- not inc(C,T), dec(C,T), scnt(C,T-1,V),
15                    comp(C), time(T), val(V), val(V-1).
16 scnt(C,T,V)   :- scnt(C,T-1,V), not inc(C,T), not dec(C,T),
17                    comp(C), time(T), val(V), time(T-1).
18 scnt(C,T,V)   :- scnt(C,T-1,V), inc(C,T), dec(C,T),
19                    comp(C), time(T), val(V), time(T-1).
20 #minimize { 1@1,C,T: scnt(C,T,0), T>0;
21             1@1,C,T: scnt(C,T,2);      }.
```

we define the timeline using `time/1` as a domain predicate, extract the identities of the components and restrict, using the predicate `val/1`, the component-wise state-space based on three states that encode 0-, 1-, and 2-fold coverage for each component and each time step. As to be argued later, this restriction is sufficient to implement miscoverage in the sense of Section 2. In Line 2, we select with predicate `break/1` at most `b` time steps for maintenance breaks that occur no later than the time step `l` (the parameter ℓ). Then, for each break, at least one component is selected for maintenance using the predicate `serv/2` in Line 3. These definitions induce the search space of all feasible schedules.

The rest of the encoding describes the miscoverage of the schedule and optimization based on the resulting cost function. We have defined two *derived* action predicates, `inc/2` and `dec/2`, taking the component `C` and the time step `T` as arguments. These actions can co-occur or be both absent, so that the true set of actions is the powerset of {`inc`,`dec`} for each `C` and `T`. The increment action (`inc` in Line 5) encodes the change where the maintenance of a component `C` at a time step `T` covers a set of time steps starting from `T`, possibly incrementing the state counter that indicates how many times the time step is being covered. The decrement action (`dec` in Lines 6–7) indicates that either a recommended maintenance interval or the initial lifetime has just ended and the respective state counter must be decremented to reflect the required change in the count. Note that there is no chance that coverage due to the initial lifetime and coverage due to some maintenance action could end simultaneously.

The service state of the machine at each time step is encoded with the (functional) predicate `scnt(C,T,N)` mapping the component `C` and the time step `T` to the state `N` ranging from `0` to `2`, as restricted by the `val` predicate. The state `0` indicates that the time step of the component is not covered at all. The state `1` indicates that the time step is covered exactly once, while the state `2` indicates a double coverage. In Lines 9–10, the state of the component at the moment

before the beginning of the timeline is declared to be either `0` or `1`, depending on the initial lifetime of the component.

Lines ranging from 12 to 19 define state transitions that depend on the combination of the elementary actions `inc` and `dec` at each time step 1...`h`. From the state of the previous time step `T-1`, the component moves deterministically to one of the target states, depending on the combination of the elementary actions at the current time step `T`. The set of actions {`inc`} is treated by Line 12, {`dec`} in Line 14, {} in Line 16, and {`inc`,`dec`} in Line 18. These four cases, respectively, correspond to increasing, decreasing, and (the last two) maintaining coverage through inertial transitions that do not change the state of the component. As a combined effect of the initialization of the state (either `0` or `1`) and the subsequent transitions, the `scnt/3` predicate will map each component and each time step to one of the three states. Only the states of the time steps 1...`h` matter when in comes to the computation of miscoverage, i.e., the sum of the time steps where a component's state is either `0` or `2`. This sum is to be minimized by Lines 20–21.

4 Pruning Constraints and Correctness Proofs

In this section, we present new constraints that can be used to prune the search space for optimal schedules. To this end, let $\mathcal{M} = \langle C, \iota, \rho \rangle$ be a multi-component machine and $S = \langle h, A, \ell, b \rangle$ a PMS for $\mathcal{M}$. For every $c \in C$, recall the set of time steps B_c at which preventive maintenance actions are used to service component c according to S. For each component c, and time step $1 \leq i \leq h$, we define a function analogous to the `scnt/3` predicate of Listing 2 as

$$cnt_S(c,i) = min\Big(\big|\{j \in B_c | j \leq i < j + \rho(c)\}\big| + \big|\{1 | \iota(c) > 0, i \leq \iota(c)\}\big|, 2\Big). \quad (1)$$

The miscoverage $mc(S, \mathcal{M})$ can be alternatively computed as

$$mc(S,\mathcal{M}) = \big|\{\langle c,i\rangle | c \in C, 1 \leq i \leq h,\ cnt_S(c,i) \neq 1\}\big|. \quad (2)$$

For $k \geq 1$ and $1 \leq t_1 < t_2 \leq h+1$, we also define

$$\sigma(S,c,t_1,t_2,k) = \big|\{i | t_1 \leq i < t_2,\ cnt_S(c,i) = k\}\big|. \quad (3)$$

We now define six properties that can be used for pruning the search space.

Definition 5. *Let $\mathcal{M} = \langle C, \iota, \rho \rangle$ be a multi-component machine, $S = \langle h, A, \ell, b \rangle$ be a solution to the MISCOVERAGE PMS problem with $\mathcal{M}$ as the input, and $t \in B$ be a scheduled maintenance break of S. The schedule S is deemed*

- *lagging at t iff for all components $c \in C$ we have $cnt_S(c, t-1) = 0$;*
- *congested at t iff for all components $c \in C$, we have $cnt_S(c,t) = 2$;*
- *under-serving for a component $c \in C$ at t iff $A(c,t) = 0$ and $\sigma(S,c,t,t',0) > \sigma(S,c,t,t',1)$, where $t' = min(t + \rho(c), h)$;*
- *over-serving for a component $c \in C$ at t iff $A(c,t) = 1$ and $\sigma(S,c,t,t',2) \geq \sigma(S,c,t,t',1)$, where $t' = min(t + \rho(c), h)$;*

– *under-tight for a component* $c \in C$ *at* t *iff* $cnt_S(c,t) = 0$*;*
– *over-tight for a component* $c \in C$ *at* t *iff* $cnt_S(c,t-1) = 2$*.*

For the sake of simplicity, Definition 5 assumes that ℓ is equal to h. However, if this assumption does not hold, it suffices to require t of Definition 5 to be strictly smaller than ℓ. In what follows, we prove that at least one globally optimal schedule will be preserved after pruning all schedules that have any of the properties of Definition 5. Let $\mathcal{M} = \langle C, \iota, \rho \rangle$ be a multi-component machine, $\langle h, \ell, b \rangle$ be a triple of scheduling parameters, and $\mathcal{P}$ be the MISCOVERAGE PMS problem given $\mathcal{M}$ and $\langle h, \ell, b \rangle$ as the input. For every solution $S = \langle h, A, \ell, b \rangle$ to $\mathcal{P}$, we define a measure $\|\cdot\|_{\mathcal{M}}$ for *service delay* by setting

$$\|S\|_{\mathcal{M}} = \sum_{c \in C,\ 1 \le t \le h,\ A(c,t)=1} t. \tag{4}$$

Lemma 1 shows that congested schedules are redundant and can be eliminated by delaying service actions without increasing miscoverage. In other words, any congested schedule is either symmetric to a non-congested one, or is sub-optimal and therefore can be eliminated from the search space.

Lemma 1. *There is a solution to* $\mathcal{P}$ *that is not congested at any time step.*

Proof. Let $S = \langle h, A, \ell, b \rangle$ be a solution to $\mathcal{P}$. Assume that i is a time step at which S is congested. Now i cannot be equal to h, since otherwise we can improve the miscoverage by setting $A(c,i) = 0$ for $c \in C$. Moreover, if $A(c,i) = 1$, then $A(c,i+1) = 0$, otherwise we can improve the miscoverage of S by setting $A(c,i) = 0$, which contradicts the optimality of S. Also, if $A(c,i) = 1$, then $cnt_S(c, i + \rho(c)) = 1$, otherwise we can improve the miscoverage by setting $A(c,i) = 0$ and $A(c,i+1) = 1$. Considering these properties, we transform S to a schedule $S' = \langle h, A', \ell, b \rangle$ by setting $A'(c,i) = 0$ for all $c \in C$, $A'(c, i+1) = 1$ for all $c \in C$ such that $A(c,i) = 1$ or $A(c,i+1) = 1$, and $A'(c,t) = A(c,t)$ for all $c \in C$ and $1 \le t \le b$ such that $t \neq i$ and $t \neq i+1$. This transformation moves all service actions at time step i to the time step $i+1$ and we have $mc(S', \mathcal{M}) = mc(S, \mathcal{M})$. Note that the number of maintenance breaks does not increase by this transformation. Therefore, S' is also a solution to $\mathcal{P}$. Furthermore, we have $\|S\|_{\mathcal{M}} < \|S'\|_{\mathcal{M}}$. Since $\|\cdot\|_{\mathcal{M}}$ is bounded from above, by repetition of this transformation one can produce a solution to $\mathcal{P}$ that is not congested at any time step. □

We now show that over-tight schedules are also redundant, because they are either symmetric to schedules that are not over-tight, or can be pruned as sub-optimal. The main observation here is that one can delay service actions in over-tight schedules without increasing miscoverage.

Lemma 2. *There is a solution to* $\mathcal{P}$ *that is not over-tight for any component at any time step.*

Proof. Let $S = \langle h, A, \ell, b\rangle$ be a solution to $\mathcal{P}$. Assume that i is the smallest number such that S is over-tight for component $c \in C$ at time step i. Then, there must exist time step $j < i$ such that $A(c, j) = 1$ and $A(c, k) = 0$ for $j < k < i$. We have $cnt_S(c, k) = 2$ for $j \leq k < i$. Moreover, we have $cnt_S(c, \rho(c) + k) = 1$ for $j < k < i$, otherwise we can improve miscoverage by setting $A(c, j) = 0$ and $A(c, i) = 1$, which contradicts the optimality of S. Moreover, $A(c, i) \neq 1$, otherwise we can decrease miscoverage by setting $A(c, j) = 0$. We transform S to a schedule $S' = \langle h, A', \ell, b\rangle$, by setting $A'(c, j) = 0$ and $A'(c, i) = 1$, and $A'(c', t) = A(c', t)$ for $\langle c', t\rangle \neq \langle c, i\rangle$. Then S' is not over-tight for c at any time step $j \leq i$, and we have $mc(S', \mathcal{M}) = mc(S, \mathcal{M})$. By repetitive application of the mentioned transformation, one can produce a solution to $\mathcal{P}$ that is not over-tight for any component at any time step. □

Lemma 3 shows that elimination processes of congested and over-tight schedules explained in the proofs of Lemmas 1 and 2 do not conflict with each other.

Lemma 3. *There is a solution to $\mathcal{P}$ that (i) is not congested at any time step; and (ii) has no component making the solution over-tight at any time step.*

Proof. The transformations used in the proofs of Lemmas 1 and 2 increase $\|\cdot\|_{\mathcal{M}}$. The proof is complete by considering that $\|\cdot\|_{\mathcal{M}}$ is bounded from above. □

In analogy to congested and over-tight schedules, lagging and under-tight schedules can either be eliminated as symmetric to other schedules or pruned as sub-optimal by preceding the service actions without increasing miscoverage. We show by the following two lemmas that this is possible without reproducing congested or over-tight schedules.

Lemma 4. *There is a solution to $\mathcal{P}$ that is not lagging nor congested at any time step, nor is over-tight for any component at any time step.*

Proof. According to Lemma 3, there is a solution $S = \langle h, A, \ell, b\rangle$ to $\mathcal{P}$ that is not congested at any time step, nor is over-tight for any component at any time step. Assume that i is a time step at which S is lagging. By definition, i cannot be equal to 1. Also, if $A(c, i) = 1$, then $i + \rho(c) - 1 \leq h$ and $cnt_S(c, i + \rho(c) - 1) = 1$, otherwise we can improve miscoverage by setting $A(c, i) = 0$ and $A(c, i - 1) = 1$. Considering these properties, we transform S to a schedule $S' = \langle h, A', \ell, b\rangle$, by setting $A'(c, i) = 0$ for every $c \in C$, $A'(c, i - 1) = 1$ for every $c \in C$ such that $A(c, i) = 1$, and $A'(c, t) = A(c, t)$ for every $c \in C$ and $1 \leq t \leq b$ such that $t \neq i - 1$ and $t \neq i$. We have $mc(S', \mathcal{M}) = mc(S, \mathcal{M})$. This transformation moves all service actions at time step i to time step $i-1$ in the case that no component is being serviced at time step $i - 1$. Therefore, the number of maintenance breaks does not increase, and also, for each $c \in C$ and $1 \leq t \leq h$, $cnt_{S'}(c, t) = 2$ only if $cnt_S(c, t) = 2$. We can conclude that S' is not congested at any time step, nor over-tight for any component at any time step. Furthermore, we have $\|S\|_{\mathcal{M}} > \|S'\|_{\mathcal{M}}$. Since $\|\cdot\|_{\mathcal{M}}$ is bounded from below, by repetition of the this transformation one can produce a solution to $\mathcal{P}$ that is not lagging nor congested at any time step, nor is over-tight for any component at any time step. □

Listing 3. Pruning constraints

```
:- comp(C), break(T), scnt(C,T-1,2).
:- comp(C), break(T), scnt(C,T,0).

:- comp(C), serv(C,T1),
   0 >= #sum{ -1,T2: scnt(C,T2,2), time(T2), T2>=T1, T2<T1+R;
               1,T2: scnt(C,T2,1), time(T2), T2>=T1, T2<T1+R }.
:- comp(C), not serv(C,T1), break(T1),
   0 >  #sum{ -1,T2: not scnt(C,T2,1), time(T2), T2>=T1, T2<T1+R;
               1,T2: scnt(C,T2,1),     time(T2), T2>=T1, T2<T1+R }.

:- break(T),  scnt(C,T,2): comp(C).
:- T>1, break(T),  not scnt(C,T-1,1): comp(C).
```

Lemma 5. *There is a solution to $\mathcal{P}$ that is not congested at any time step, nor is under-tight nor over-tight for any component at any time step.*

Proof. According to Lemma 3, there is a solution $S = \langle h, A, \ell, b\rangle$ to $\mathcal{P}$ that is not congested at any time step, nor is over-tight for any component at any time step. Assume that i is the smallest number such that S is under-tight for component $c \in C$ at time step i. Then, there must exist time step $j > i$ such that $A(c, j) = 1$ and $cnt_S(c, k) = 0$ for $i \leq k < j$, otherwise we can improve miscoverage by setting $A(c, i) = 1$, which contradicts the optimality of S. Moreover, by definition, i cannot be equal to 1. We transform S to a schedule $S' = \langle h, A', \ell, b\rangle$, by setting $A'(c, j) = 0$ and $A'(c, i) = 1$, and $A'(c', t) = A(c', t)$ for any $\langle c', t\rangle \neq \langle c, i\rangle$. The schedule S' is not under-tight for c at any time step $j \leq i$, and we have $mc(S', \mathcal{M}) = mc(S, \mathcal{M})$. This transformation moves the service action of c from time step j to time step i where c is not being serviced at any time step between i and j. Therefore, for any time step $1 \leq t \leq b$, $cnt_{S'}(c, t) = 2$ only if $cnt_S(c, t) = 2$. Thus, S' is not congested at any time step, nor over-tight for any component at any time step. By iterating this transformation, one can produce a solution to $\mathcal{P}$ that gets never congested, nor is under-tight nor over-tight for any component at any time step. □

We now show that if a schedule is over-serving for some component, then a service action for that component can be removed without increasing miscoverage, and this is possible without reproducing congested or over-tight schedules.

Lemma 6. *There is a solution to $\mathcal{P}$ that is not congested at any time step, nor is over-tight nor over-serving for any component at any time step.*

Proof. According to Lemma 3, there is a solution $S = \langle h, A, \ell, b\rangle$ to $\mathcal{P}$ that is not congested at any time step, nor is over-tight for any component at any time step. If S is over-serving for a component $c_i \in C$ at time step t_1, we can transform it to the schedule $S' = \langle h, A', \ell, b\rangle$ by setting $A'(c, t) = 0$, for any $\langle c, t\rangle = \langle c_i, t_1\rangle$, and $A'(c, t) = A(c, t)$, otherwise. Then we have $mc(S, \mathcal{M}) - mc(S', \mathcal{M}) = \sigma(S, c_i, t_1, t_2, 2) - \sigma(S, c_i, t_1, t_2, 1) \geq 0$, where $t_2 = min(t_1 + \rho(c_i), h)$. This transformation removes the preventive maintenance action used to service component c_i at time step t_1, and therefore does not cause S' to be congested at any time

step, nor be over-tight for any component at any time step. By repetitive application of this transformation, one can produce a solution to $\mathcal{P}$ that is not congested at any time step, nor is over-tight nor over-serving for any component at any time step. □

Note that according to the proof of Lemma 6, S is symmetric to S' if $\sigma(S, c_i, t_1, t_2, 2) = \sigma(S, c_i, t_1, t_2, 1)$. Also, S is sub-optimal if $\sigma(S, c, c_i, t_1, t_2, 2) > \sigma(S, c_i, t_1, t_2, 1)$. In both cases, S can safely be eliminated from the search space.

Similarly, we show that if a schedule is under-serving for some component, then a service action for that component can be added to the schedule, which decreases miscoverage. In other words, under-serving schedules can be safely pruned as sub-optimal ones.

Lemma 7. *No solution to $\mathcal{P}$ is under-serving for any component at any time step.*

Proof. Assume the contrary, i.e., let $S = \langle h, A, \ell, b\rangle$ be a solution to $\mathcal{P}$ such that for some $c_i \in C$, we have $A(c_i, t_1) = 0$ and $\sigma(S, c_i, t_1, t_2, 0) > \sigma(S, c_i, t_1, t_2, 1)$, where $t_2 = min(t_1 + \rho(c), h)$. Let $S' = \langle h, A', \ell, b\rangle$, where $A'(c, t) = 1$ for $\langle c, t\rangle = \langle c_i, t_1\rangle$, and $A'(c, t) = A(c, t)$, otherwise. We have $mc(S, \mathcal{M}) - mc(S', \mathcal{M}) = \sigma(S, c_i, t_1, t_2, 0) - \sigma(S, c_i, t_1, t_2, 1) > 0$, contradicting the optimality of S. □

We are now ready to present the main theoretical result of this paper. Theorem 1 shows that at least one optimal schedule will exist after pruning all schedules that have any of the properties introduced by Definition 5.

Theorem 1. *There is a solution to $\mathcal{P}$ that is not congested nor lagging at any time step, nor is under-tight, over-tight, over-serving, nor under-serving for any component at any time step.*

Proof. According to Lemma 3, there is a solution $S = \langle h, A, \ell, b\rangle$ to $\mathcal{P}$ that is not congested at any time step, nor is over-tight for any component at any time step. By Lemma 7, S is not under-serving for any component at any time step. The transformations used in the proofs of Lemmas 4, 5, and 6 decrease the measure $\|\cdot\|_{\mathcal{M}}$. The proof is complete by noting that $\|\cdot\|_{\mathcal{M}}$ is bounded from below. □

Listing 3 shows the ASP constraints for encoding our pruning constraints when the Elevator encoding explained in Section 3 has been used as the baseline encoding. In the case of a different baseline encoding, minor modifications might be necessary. Lines 1 and 2 eliminate answer sets representing over-tight and under-tight schedules from the search space, respectively. Lines 4–6 and 7–9 guarantee that no answer set represents an over-serving nor an under-serving schedule, respectively. Lines 11 and 12 forbid congested and lagging schedules to be represented by any answer set, respectively. Thus, adding these constraints to the baseline ASP encoding is safe, as it does not cause all optimal answer sets to be pruned.

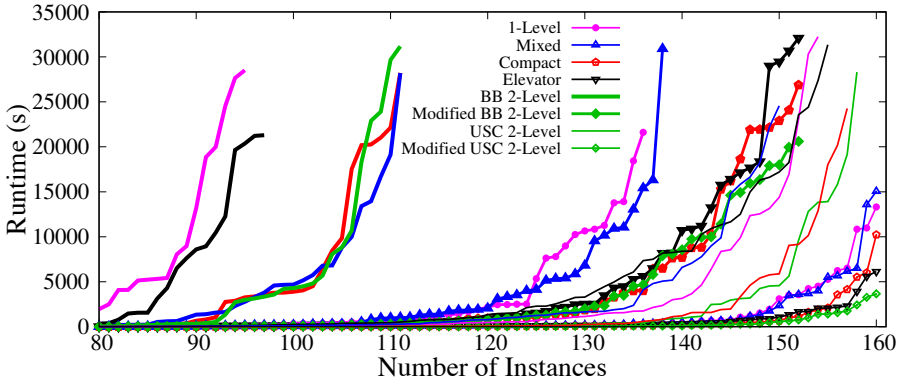


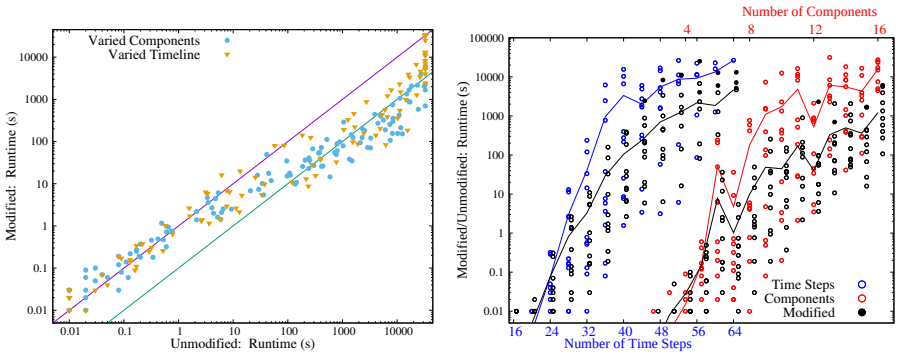
Fig. 1. Cactus plot of the execution times of different encodings, with and without modifications (points) and with the BB (thick line) and USC optimization strategies

5 Experiments

Our experiments were conducted on a SLURM-based cluster containing 140 nodes with on Intel® Xeon® CPUs (E5-2620 v3 at 2.40GHz) and over 3000+ cores. The search for optimal schedules was implemented on this cluster using the version 3.3.8 of the Clasp solver [6]. The specific goal of the experiments was to quantify the execution time differences between baseline encodings and their extensions by pruning constraints. For the scaling experiments, we generated ten random machines for each machine size from 1 to 16 components, thus giving rise to 160 test instances in total. In addition, we generated ten random 8-component machines and used these machines to check the effects of larger timelines on the execution time. All experiments respected the timeout at 9 hours.

The first experiment measured the runtimes of ten different ASO encodings and two optimization strategies. The results of the experiment are shown in Fig. 1. In this cactus plot, there are five baseline encodings – 1-Level, 2-Level, Mixed, Compact 2-Level, and Elevator – and their counterparts extended by pruning constraints. For each encoding, optimization based on both branch-and-bound (BB) and unsatisfiable core (USC) strategies were tried out to explore the potential combined effects of pruning constraints and optimization strategies. Thus, in the figure, there are five baseline encodings plotted in five different colors, and four solving methods indicated by four line shapes.

We observe clearly notable differences between the runtimes. The first observation highlights the strategy: the BB strategy handles 95–111 instances with various baseline encodings, and 137–153 instances with extended encodings. On the contrary, the USC strategy handles 150–158 instances with baseline encodings and 159–160 instances with extended encodings. Thus, regardless of the encoding used, the USC strategy handles more instances than the BB strategy. Therefore, we decided to use the USC strategy in the later experiments only. The second observation is that the encodings extended by pruning constraints help especially when the BB strategy is used, but they also help to treat more



(a) Unmodified vs Modified (b) Scalability wrt Parameters

Fig. 2. Runtime of the Elevator encoding without and with pruning modifications

difficult instances when the USC strategy is used. In general, the improvement with the USC strategy is slightly greater than the effect of pruning alone, but the fastest method is obtained when we combine the USC strategy with pruning.

The remaining observations concern the effect of pruning constraints when added to the baseline encodings. First, the baseline Elevator is among the slowest encodings under both BB and USC strategies, but its modified variant runs up to the proximity of 2-Level encoding. Second, 2-Level and Compact 2-Level encodings are fast and behave pretty much alike under all four solving methods. Yet the modified 2-level encoding seems to tackle difficult instances in less time than the Compact encoding. Third, we observe that optimization strategy and pruning constraints are less effective with the Mixed encoding than with the other encodings: it is among the fastest baseline encodings under the BB strategy, while others benefit more from pruning constraints and the USC strategy.

A different kind of experiment focused on the Elevator encoding. The scatter plot in Fig. 2a summarizes two such experiments where the first one tested the versions of the encoding with a fixed timeline $h = 32$ and random machines whose component sizes varied in the range from 1 to 16, and the second one tested the versions of the encoding with ten random 8-component machines and timelines ranging from 16 to 64 time steps. In these experiments, each parameter value was tested with 10 random machines. The diagonal lines indicate that the runtimes of the modified encoding art an order of magnitude better than the baseline in both experiments. The same experiments have been viewed from another perspective in Fig. 2b. This scatter plot and the average lines show how the parameter of each experiment correlates with the running time. By them, the running time grows exponentially according to both the timeline size and the machine size. The effect of increasing the number of time steps seems to be moderated, probably because of the small number of breaks with which the over-coverage can be fully avoided. Furthermore, we observe that the baseline encodings modified by adding the pruning constraints improve the runtimes

roughly by a factor of 10, and a few solid circles indicate that the modified encoding can solve more instances than the unmodified encoding.

Based on our observations, we recap that the experiments support a significant advantage when extending the baseline encodings with pruning constraints. The effect of this modification seemed to be largely independent of the choice of the optimization strategy and the combination of both techniques gives the best result. Moreover, these improvements turned out to depend on the base encoding used, but similar for all. Regardless of the choice of these techniques, 2-Level and Compact 2-Level encodings kept their leading positions consistently. In our tests, scalability with respect to both components and timeline appear very similar. As a further reflection, we note that the present experiments lancer more rigorous practice in comparison to the earlier work [14]. All tests were carried out with random machines, while allowing test reiteration, and averages were replaced with matched scatter plots. We also extended the scalability tests to the effect of timeline, an important dimension of the problem instances. However, larger and new kinds of experiments to separate scalability with respect to components and timelines seem to be necessary in the future.

6 Conclusion

In this article, we continue the previous research [14] on an intractable optimization problem, viz. the minimization of miscoverage in schedules devised for the preventive maintenance of a multi-component machine. In this follow-up study, we design additional constraints for pruning the search space and evaluate their effect on the search of optimal solutions. The original preventive maintenance scheduling (PMS) problem and the novel pruning constraints are both encoded as logic programs in the framework of answer set programming (ASP) and optimization (ASO). In this way, we follow the overall methodology set up in [14]. As regards technical results, we developed a principled approach to motivate pruning on the basis of the problem structure. In particular, we established seven lemmas about the properties of *pruning constraints* and their correctness, and drew these lemmas together in a cap-stone theorem, Theorem 1.

To quantify the effect of the pruning constraints on optimization time, we carried out several experiments with random multi-component machines and ranges of parameters that included machine size (1–16 components) and timeline length (16–64 time steps) as basis for scalability analysis. These machines are used to instantiate five different (baseline) encodings of the PMS problem, subsequently solved through two different optimization strategies, viz. branch and bound (BB) and unsatisfiable core (USC), as implemented in the Clingo system. According to our experiments in general, the pruning constraints often improve the efficiency of all encodings by a factor of 10, except when problem instance get so small that the overhead of introducing the pruning constraints dominates the time taken by optimization.

It is worth emphasizing that pruning constraints designed in this work retain at least one optimal schedule in the search space. This is possible since the

solutions to the optimization problem are not further constrained by other constraints. However, if the problem is extended by *resource constraints*, e.g., concerning the availability and expertise of service personnel, pruning constraints may have to be treated as soft constraints or as secondary components in the objective function. Although this might cancel some of the speed-up perceived in the experiments reported in this work, pruning constraints nevertheless demonstrate that a tightened representation of the search space can push the practical solvability limits of the PMS optimization problem forward.

In the future, the interaction between pruning constraints and supplementary resource constraints in the problem formulation calls for further attention. Yet another dimension is provided by the stochastic aspects of PMS which can be used to enrich the models of preventive maintenance.

References

1. Banbara, M., Inoue, K., Kaufmann, B., Okimoto, T., Schaub, T., Soh, T., Tamura, N., Wanko, P.: *teaspoon*: solving the curriculum-based course timetabling problems with answer set programming. Annals of Operation Research **275**, 3–37 (2019)
2. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011)
3. 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)
4. Dodaro, C., Maratea, M.: Nurse scheduling via answer set programming. In: LPNMR 2017. pp. 301–307. Springer (2017)
5. Drescher, C., Tifrea, O., Walsh, T.: Symmetry-breaking answer set solving. AI Commun. **24**(2), 177–194 (2011)
6. Gebser, M., Kaminski, R., Kaufmann, B., Romero, J., Schaub, T.: Progress in clasp series 3. In: LPNMR 2015. pp. 368–383 (2015)
7. Gebser, M., Kaminski, R., Ostrowski, M., Schaub, T., Thiele, S.: On the input language of ASP grounder gringo. In: Logic Programming and Nonmonotonic Reasoning. pp. 502–508. Springer (2009)
8. Heule, M., Walsh, T.: Symmetry in solutions. In: AAAI 2010. pp. 77–82 (2010)
9. Li, C.M., Manyà, F.: MaxSAT, hard and soft constraints. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 613–631. IOS Press (2009)
10. Luukkala, V., Niemelä, I.: Enhancing a smart space with answer set programming. In: RuleML 2010. pp. 89–103. Springer (2010)
11. Rossi, F., van Beek, P., Walsh, T.: Constraint programming. In: Handbook of Knowledge Representation, Foundations of Artificial Intelligence, vol. 3, pp. 181–211. Elsevier (2008)
12. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2), 181–234 (2002)
13. Tarzariol, A.: A model-oriented approach for lifting symmetry-breaking constraints in answer set programming. In: IJCAI 2022. pp. 5875–5876 (2022)
14. Yli-Jyrä, A., Janhunen, T.: Applying answer set optimization to preventive maintenance scheduling for rotating machinery. In: Declarative AI (RuleML) (2022), accepted for publication