Production-maintenance scheduling in complex machinery

João Dionísio[1,2], Ambros M. Gleixner[3,4], João Pedro Pedroso[1,2], and Ksenia Bestuzheva[3]

[1] Faculdade de Ciências, Universidade do Porto, rua do Campo Alegre s/n, 4169–007 Porto, Portugal
[2] CMUP, Centro de Matemática da Universidade do Porto
[3] Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
[4] Hochschule für Technik und Wirtschaft Berlin, Germany

Abstract. Long-term maintenance of machines with costly components has been studied throughout the years. However, research on combining long-term maintenance decisions with short-term production scheduling to satisfy a predetermined demand is less prevalent, especially that which study complex machinery with multiple interacting components. Our work contributes to this subject by studying an abstract mixed-integer nonlinear programming model, where an increase in production leads to faster degradation of components and thus creates the need for more maintenance. We will study a Dantzig-Wolfe decomposition to solve this difficult problem, with heuristics to speed up the pricing problem.

Keywords: Production-maintenance scheduling · Dantzig-Wolfe decomposition · MINLP · Resource allocation

KB: to make git diffs more meaningful, it helps to write each sentence on its own line. You don't have to change everything now, but you can change this in the parts that you edit anyway. JD@KB:Done.

KB: your use of commas is often excessive; it might come from a difference between punctuation rules in your language and English. I am removing some unneeded commas as I go, but this is something you might want to watch out for generally.

1 Introduction

In this article, we present a MINLP with the objective of minimizing the maintenance cost of machines composed of multiple components, that must satisfy a predetermined continuous demand of a product. We also present a Dantzig-Wolfe reformulation. Additionally, we develop heuristics to aid in the solving of these difficult models.

Production-maintenance scheduling ?????? EXPAND ON THE BACKGROUND ????? This problem belongs to the class of production-maintenance scheduling problems with parallel machines - see Geurtsen et al. [12] for a comprehensive literature review. Many variants of this problem exist and have been

studied for decades, albeit less than other more classical problems. The majority of the literature looks at linear variants of this problem (see [15] for a resource-production exception), mostly because physical considerations tend to be ignored in more abstract models. Other authors avoid the nonlinearities by modeling machine failure with a random distribution [6, 25], sometimes using Markov Chains to model a discrete set of conditions [18, 21, 24]. In these stochastic models, a few authors perform Bender's decomposition [13, 23].

In his thesis, Pries focuses on decomposition approaches for joint production-maintenance scheduling problems [23], using both Bender's and Dantzig-Wolfe decomposition. As opposed to our work, this one considers the machines as single structures, and that they are all identical. Furthermore, the nonlinearities we use to model the degradation, are instead replaced by random failures.

Jahromi et al. heuristically decouple the maintenance and the production decisions, to make the model more tractable [3]. Not performing these decisions at the same time allows the decision horizon for maintenance to be longer, but loses reliability as a result of the assumptions on production.

Our work differs from the ones we could find, since we are not assuming that our parallel machines are inseparable units, but rather consider them structures composed of interacting components, which can be independently maintained. Furthermore, even working with abstract machines, we admit the possibility of nonlinearities in the degradation of the components, which brings the models closer to reality, and increases their difficulty, as they are mixed-integer nonlinear programs.

Maintenance in practice Industrial maintenance tends to rely on two heuristics for maintenance scheduling: Time Based Maintenance (TbM), where maintenance is planned considering the time since the last maintenance action, and Condition Based Maintenance (CbM), where the asset is regularly tested and when some predefined threshold is reached, maintenance is scheduled ([1] elaborates on this in the context of power transformers).

Research on methods for employing just-in-time maintenance is plentiful, with many of these methods resorting to machine learning, with the use of real-world data [2]. The idea is to perform maintenance at the precise point in time when any delay in the maintenance action would result in the failure of a component. As will be shown in Example 1 from Section 5, this can lead to suboptimal solutions.

Applications This model can be used for minimizing maintenance costs of power transformer fleets, where the "product" is electricity, or in water treatment stations, where the "product" is clean water. More abstractly, it can also be understood as production-maintenance scheduling in factories working in parallel with different interacting machines. In this last example, the factories take on the roll of the (pottentially different) machines, and the machines of the (pot-

tentially interacting) components.

This article is organized as follows: Section 1 frames the problem and provides a light background on production-maintenance problems and on maintenance practices, besides presenting the related work. In Section 2, the compact formulation is presented. Section 3 shows the Dantzig-Wolfe reformulation of this compact model, with implementation details appearing in Section 4. Due to the difficulty of this model we need some heuristics and consider some simplifications in Section 5. Section 6 presents the experimental setup, along with the results and corresponding discussion. We conclude the paper in Section 7.

This work was funded by the Portuguese Foundation for Science and Technology - FCT under ????, and by Zuse Institute Berlin - ZIB.

2 Model

JD: I made the tables bigger. What do you think?

Parameter	Parameter Meaning	Parameter Range
C^k	Cost of maintaining component k	$\mathbb{R}^+$
D^k	Duration of component k's maintenance	$\mathbb{N}$
E_t	Demand at period t	$\mathbb{R}^+$
$\mathcal{K}$	Components to be maintained	
M	Big constant to dominate constraint	$\mathbb{R}^+$
$\mathcal{N}$	Set of machines	$\mathbb{N}$
Q^k	Maximum permissible production	$\mathbb{R}^+$
R^k	Maximum condition of component k	$\mathbb{R}^+$
$\mathcal{T}$	Set of periods	$\mathbb{N}$
$\mathcal{Z}$	Set of maintenance implications	$\mathcal{K}^n \times \mathcal{K}^n$

Table 1: Description of the parameters used in the model

Variable	Variable Meaning	Variable Type
$x^k_{n,t}$	Maintenance of component k of machine n at period t	Binary
$y_{n,t}$	Production of machine n at period t	Continuous
$r^k_{n,t}$	Condition of component k of machine n at period t	Continuous

Table 2: Description of the variables used in the model

Below we present a general formulation of the production-maintenance scheduling problem. The variables are represented by lowercase letters (x, y, r), parameters by uppercase letters with a different font (C, Q, M, etc.), and sets by uppercase letters with a third font ($\mathcal{N}, \mathcal{K}, \mathcal{Z}$). Variables and parameters will have subscripts and superscripts. For example, $r^k_{n,t}$ is the variable r for component k

of machine n at the time period t, and this nomenclature is standardized across the model.

$$\underset{x,\,y}{\text{minimize}} \quad \sum_{n\in\mathcal{N}}\sum_{t\in\mathcal{T}}\sum_{k\in\mathcal{K}^n} C^k \cdot x^k_{n,t} \tag{1a}$$

subject to

$$\sum_{n\in\mathcal{N}} y_{n,t} \geq \mathsf{E}_t \qquad \forall t \in \mathcal{T}\, \forall n \in \mathcal{N}, \tag{1b}$$

$$r^k_{n,t} \leq f_{n,k}(r^k_{n,t-1}, y_{n,t}; r^1_{n,t-1}, \ldots, r^{|\mathcal{K}_n|}_{n,t-1}) + \mathsf{M} \cdot x^k_{n,t} \qquad \forall t, n, k, \tag{1c}$$

$$y_t \leq g_{n,k}(r^k_{n,t}) \qquad \forall t\, \forall n \in \mathcal{N}\, \forall k \in \mathcal{K}, \tag{1d}$$

$$y_{n,t} \leq (1 - x^k_{n,t}) \cdot \mathsf{Q}^k \qquad \forall t\, \forall n \in \mathcal{N}\, \forall k \in \mathcal{K}, \tag{1e}$$

$$x^k_{n,i} \geq x^k_{n,t} - x^k_{n,t-1} \qquad \forall n, k, t \leq i \leq t + \mathsf{D}^k, \tag{1f}$$

$$x^k_{n,t} \leq x^{k'}_{n,t} \qquad \forall t \in \mathcal{T}\, \forall (k,k') \in \mathcal{Z}, \tag{1g}$$

$$x^k_{n,t} \in \{0,1\} \qquad \forall t\, \forall n \in \mathcal{N}\, \forall k \in \mathcal{K}, \tag{1h}$$

$$y_{n,t} \geq 0 \qquad \forall t\, \forall n \in \mathcal{N}, \tag{1i}$$

$$0 \leq r^k_{n,t} \leq \mathsf{R}^k_n \qquad \forall t\, \forall n \in \mathcal{N}\, \forall k \in \mathcal{K} \tag{1j}$$

The objective of the model is to minimize the maintenance cost (added over all components of all machines in all time periods) while guaranteeing that the machines satisfy the demand in Constraints [1b]. Constraints [1c] model the evolution of the condition of the components. A big-M term represents the maintenance action which, in conjunction with the binary variable $x^k_{n,t}$ will simulate the replacement of the component. Function f will model the (possibly nonlinear) degradation of the component, which depends on its previous condition, the current production, and possibly the condition of the other components of the same machine. Constraints [1d] enforce a (possibly nonlinear) upper bound on the production of a machine, based on the condition of its components. A machine with damaged components will not be able to produce as much as a new one. Constraints [1e] stipulate that the machines stop production while one of their components is being maintained, and Constraints [1f] impose a predetermined duration for replacing the components. Finally, Constraints [1g] represent the interactions between components regarding maintenance, where the replacement of some components forces the maintenance of others.

Since a maintenance action cannot be started if it does not have enough time to be finished, the following is a valid cut: $x^k_{n,t} \leq x^k_{n,|\mathcal{T}|-\mathsf{D}^k}, \forall k, t \geq |\mathcal{T}| - \mathsf{D}^k$.

3 Dantzig-Wolfe Reformulation

The formulation above exhibits a block diagonal structure with a set of complicating constraints. These are Constraints [1b], requiring that the demand must

be satisfied, while the blocks are composed of the constraints describing the functioning of the independent machines. Dantzig-Wolfe decomposition is a method for solving problems with these attributes [8]. The nonlinearities in the formulation will appear in the pricing problem, thus not directly interfering with the algorithm. Example of other works with nonlinear pricing problems can be found in [4, 5, 14].

In the master problem below, we aggregate subproblems if they correspond to identical machines, as they share the same set of possible solutions. With this aggregation, we expect that the usefulness of the reformulation will increase with the number of identical machines, given its symmetry-breaking characteristics. Thus, the value of $\lambda_{z,i}$ will tell us how many machines in the subgroup z of identical machines will use solution i. The notation $\mathsf{y}_{z,i,t}$ refers to the production in solution i (from the subgroup z) at time t, and y uses a different font to emphasize that in the master model it is a parameter.

$$\underset{\lambda}{\text{minimize}} \quad \sum_{z \in \mathcal{Z}} \sum_{i \in \mathcal{X}_z} c^\intercal \lambda_{z,i} \tag{2a}$$

subject to

$$\sum_{z \in \mathcal{Z}} \sum_{i \in \mathcal{X}_z} \lambda_{z,i} \cdot \mathsf{y}_{z,i,t} \geq \mathsf{E}_t, \quad \forall t \in \mathcal{T} \quad (\pi), \tag{2b}$$

$$\sum_{i \in \mathcal{X}_z} \lambda_{z,i} \leq |\mathcal{Z}_z|, \qquad \forall z \in \mathcal{Z} \quad (\pi_0), \tag{2c}$$

$$0 \leq \lambda_{z,i} \leq |\mathcal{Z}_z|, \qquad \forall z \in \mathcal{Z} \tag{2d}$$

Constraints [2b] stipulate that the chosen solutions must satisfy the production demand. Computationally speaking, it makes more sense to use an inequality rather than an equation, as it tends to allow for a faster stabilization of the corresponding dual values. Constraints [2c] enforce that each subgroup of identical machines can have at most *size of the subgroup* solutions.

This reformulation is solved with column generation, which will provide new production-maintenance patterns for each of the different sets of machines, looking for the ones with the lowest reduced cost. The dual variables from constraints [2b] will incentivize solutions with a higher production, while the maintenance cost will prevent solutions with too many maintenance actions. The generation of new patterns is obtained by the pricing problem below.

$$\underset{x,\, y}{\text{minimize}} \quad C^\intercal x - \pi^\intercal y - \pi_0 \tag{3a}$$

subject to

$$\text{Constraints [1c]- [1g] being satisfied,} \tag{3b}$$

$$y_t \leq \mathsf{E}_t \, \forall t \in \mathcal{T} \tag{3c}$$

Since the production is a variable in this subproblem, y uses the variable font.

As the production demand is a clear upper bound on the production of any specific machine, Constraint [3c] is a valid cut.

The pricing problem can also be interpreted in the context of machine production. The objective function is equivalent to the maximization of the profit of a machine (where the selling price of the product at time t is given by the corresponding dual value). The added valid inequality enforces a cap on production, after which the product cannot be sold, simulating markets where there is limited demand. A special case of this problem to power transformers was introduced by Dionísio and Pedroso in [9].

As the pricing problem remains a MINLP, it is prohibitively expensive to solve it to optimality in every iteration.

4 Extended Formulation Implementation

KB: you should be consistent about the capitalisation of the section titles (i.e. same level headings should have similar capitalisation). JD@KB:Done.

KB: I would add some introductory text here. JD@KB: Done.

Implementing Branch-and-Price requires a lot of effort. Fortunately SCIP [7], the solver used in our work, is equipped with a Branch-and-Price framework. Nevertheless, there remains some problem-specific details which need to be implemented, described in this Section.

Initial columns generation We use Farkas' Pricing [11, p. 43] for initializing the RMP and for fixing infeasibilities during the Branch-and-Price process. This technique uses Farkas' Lemma [10] by finding a master variable that breaks the lemma's statement and adding it to the RMP.

Choice of subproblem As the subproblems can be very difficult when solved exactly, we will use the results of the heuristic described in Section 5 to decide the order in which they should be solved. Subproblems with a more promising heuristic result will be solved first. After getting a column with a negative reduced cost, we advance to the next iteration in order to get more up-to-date dual values. As the master problem is not difficult, in contrast with the pricing problem, using the best possible dual values is preferable.

Branching Branching on fractional master variables is inefficient, and branching on pricing variables is not possible due to the aggregation of identical subproblems - two identical machines might pick different columns in the optimal solution.

After solving an iteration of the RMP, we sum the columns to find a maintenance decision x with a fractional value. Adding the master variables that perform this maintenance action, the branching decision is to enforce that these variables must not exceed $\lfloor k \rfloor$ on the left branch, or that they must at least sum

to $\lceil k \rceil$ on the right branch, where k is the (necessarily fractional) sum of these master variables.

Mathematically, the left branch adds the constraint $\sum_{\lambda|x_\lambda=1} \lambda \leq \lfloor k \rfloor$, where x_λ represents the value of a specific maintenance variable x in the pattern associated to master variable λ.

KB: make sure you define x_λ somewhere. JD@KB: Done.

Dual bound ?????? WORK IN PROGRESS ????? The following is a dual bound for the master problem: ??? WHAT IS THE NAME? NEED REFERENCE ???

$$z^*_{\text{RMP}} + \sum_{n \in \mathcal{N}} |\mathcal{N}_n \cdot min(0, z^*_{\text{PP}_n})| \leq z^*_{\text{MP}}$$

As one of the approaches will not always solve the pricing problems to optimality, the optimal objective value of the pricing problem can be replaced by its dual bound, whenever available.

Parallelism ?????? I WILL MOST LIKELY NOT BE IMPLEMENTING THIS

In decomposition approaches, solving the different subproblems in parallel can yield large speedups (in the order of T/t, where T is the time to solve all subproblems, and t is the time to solve the most difficult subproblem). While it is also possible to use parallelism to increase the speed of the compact formulation (see [20] for a review), this would take considerably more effort, both from a technical and a theoretical perspective. For this reason, it is our understanding that applying parallelism only to the decomposition approach is not unfair.

5 Heuristic & Simplifications

Realistic instances can quickly become intractable due to the size of the problem, having many integer variables and potentially many nonlinear constraints.

KB: put this right after the first sentence of the section, since this is background. The paper's contribution would better go next. JD@KB: Done.

An iterative refinement heuristic that would be applicable to this problem can be found in a previous work of ours [9], whose idea is limiting the number of maintenance possibilities and expanding them with every iteration. As this heuristic did not yield a large improvement, we omit it from this paper.

KB: I find this paragraph to be too vague to be helpful to the reader. You could clarify/expand a bit.

For that reason, we developed a heuristic that tries to deal with this. It is applied to the column generation process, and remains applicable if it is used to initialize the RMP (in other words, when solving the Farkas iterations).

The production-fixing heuristic This heuristic tries to take advantage of the impact of the production variable on the model. Given a fixed production vector, the idea is to perform maintenance at the last possible moment, where if any maintenance action were to be delayed by one time period, the solution would

be infeasible. This maintenance strategy is referred to as just-in-time maintenance. For a fixed production vector, Constraints [1d] will fix the condition of the components, sans the maintenance. Heuristically, the fixed production vector is chosen given the Farkas' duals or the actual duals of the current iteration. Assuming a normal distribution with mean and standard deviation provided by these duals, the production will benefit more the more unlikely high its dual is.

KB: this assumption is important, so I would move it closer to the beginning of the section. And maybe mention whether the heuristic would still produce any meaningful results for the non-concave case. JD@KB: Done

This heuristic assumes the concavity of function f for a fixed y. However, we were unable to create a realistic example where this would not be the case, in the context of production-maintenance scheduling. It seems sensible to assume that if a component's condition affects another's degradation, then the worse the condition of the former, the higher the impact on the latter. In the event of a nonconcave relationship, the heuristic will incorrectly mark more production vectors as unattainable, and will also provide worse solution the less concave the degradation. In this case the heuristic still works, but its benefit will be limited.

However, even if function f is concave for a fixed y, and we select the best possible production vector, this heuristic is not optimal. This is due to the interaction between components when it comes to their degradation, but also to the maintenance implication constraints ([1g]). See the following example:

Example 1. Consider machines with two components, A and B, and suppose that the maintenance of the B implies the maintenance of component A. Assume also that we have the best possible production for this heuristic. It may happen that component A is maintained in year x, and component B (and consequently, A) in year $y > x$. However, it is possible that maintaining B in year x also leads to a feasible solution, which would save us one maintenance of component A. For example, if the total number of periods is $y + 1$, it is not unreasonable that both components could have been maintained at x and still reached the end without failing. This anticipation of maintenance actions results in a solution that is strictly better than the one we started with.

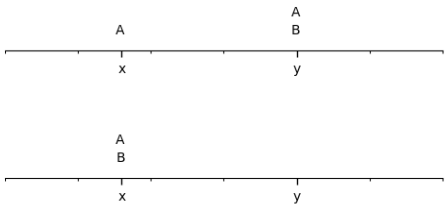

Fig. 1: The anticipation of the B-A maintenance in the top solution leads to the bottom solution. If feasible, the bottom solution is strictly better.

KB: I would expand the caption to briefly explain what in the figure stands for what. JD@KB:Done.

Note that if the bottom solution is feasible, then it is guaranteed to be an optimal solution (for this fixed production vector). By construction, the original solution tells us that component B requires one maintenance action at y at the latest, and component A also requires one at x at the latest.

The study of these types of operations is interesting, but out of scope for the present paper. In a future iteration of this heuristic, they may be used to improve it.

Below follows the production-fixing algorithm used in this work.

Algorithm 1: PF-heuristic

Data: π

production $\leftarrow$ getProduction(π);
$y_t \leftarrow$ production[t];
$r_0^k \leftarrow \mathsf{R}^k, \forall k$;
$x_t^k \leftarrow 0, \forall k, t$;
while $t \leq |\mathcal{T}|$ **do**
  for $k \in \mathcal{K}^n$ **do**
    if $x_t^k = 1$ **then**
      continue;
    $r_t^k \leftarrow f_{n,k}(r_{n,t-1}^k, y_{n,t}; r_{n,t-1}^1, \ldots, r_{n,t-1}^{|\mathcal{K}_n|})$;
    if $y_t > g_k(r_t^k)$ **then**
      $t \leftarrow \min(t, |\mathcal{T}| - \mathsf{D}^k + 1)$;
      for $t' \in [t..t + D^k]$ **do**
        $y_{t'} \leftarrow 0$;
        $x_{t'}^k \leftarrow 1$;
        $r_{t'}^k \leftarrow \mathsf{R}^k$;
      maintenanceDependencies(r, y, m);
    $t \leftarrow t - 1$;
    break;
  t $\leftarrow$ t+1;
variables $\leftarrow [y, m, r]$;
result $\leftarrow \sum_{t \in \mathcal{T}} \sum_{k \in \mathcal{K}^n} x_t^k \cdot \mathsf{C}^k$;
return variables, result;

Starting with the description of the auxiliary procedure getProduction, we assume that the dual values follow a normal distribution with mean and standard deviation equal to the average and the standard deviation of the dual values, and calculate the cumulative probability of each. Dual values that are unexpectedly high will result in a higher production.

Onto the main procedure, PF-heuristic, we start by calling getProduction to obtain the production vector, using the dual values given by the current iteration of the master problem. Iterating first over the components and then over the time periods (otherwise the impact that the components have on each others' condition would be evaluated incorrectly), we compute the condition of the current component for the current time period, as every variable of Constraint 1c

Algorithm 2: getProduction

Data: π
1 mean $\leftarrow$ mean(π);
2 std $\leftarrow$ std(π);
3 **for** $i \in \mathcal{T}$ **do**
4 | production[i] $\leftarrow$ probability[t] $\cdot$ E[t]/$|\mathcal{N}|$;
5 | production[i] $\leftarrow$ min(production[i], Q^k), $\forall k$;
6 **return** production;

Algorithm 3: maintenanceDependencies

Data: r, y, x
1 **do**
2 | addedMaintenance $\leftarrow$ FALSE;
3 | **for** $k' \in \mathcal{K}^n$ **do**
4 | | **if** $\exists k' \mid (k', k) \in \mathcal{Z} \wedge x_t^{k'} < x_t^k$ **then**
5 | | | addedMaintenance $\leftarrow$ TRUE;
6 | | | **for** $t' \in [t..t + D^{k'}]$ **do**
7 | | | | $y_{t'} \leftarrow 0$;
8 | | | | $x_{t'}^{k'} \leftarrow 1$;
9 | | | | $r_{t'}^{k'} \leftarrow \mathsf{R}^{k'}$;
10 **while** addedMaintenance = TRUE;
11 **return** r, m, y;

is fixed, sans maintenance. (And here is the assumption of concavity of f for a fixed y, as otherwise setting the component's condition to a value lower than its upper bound might be better.) If the resulting condition is incompatible with the production (see Constraint 1d), then maintenance is scheduled. This means setting production to 0, the condition to its optimal value, and the maintenance variables to 1, for the duration of component k's maintenance (lines $12 - 15$). Line 11 ensures that a component does not start maintenance that it would not be able to finish.

Due to Constraints 1g, we need to ensure that the maintenance implications are satisfied. To this end, Algorithm 3 iterates over all components k, and if it finds a component k' such that $(k', k) \in \mathcal{Z}$ (and so $x_t^{k'} \leq x_t^k$) which is not maintained, it schedules its maintenance. Then it repeats the procedure, as the scheduling of $x_t^{k'}$ might imply the maintenance of other components (if $\exists k'' \mid (k'', k') \in \mathcal{Z} \wedge x_t^{k''} = 0$). The algorithm terminates when no more maintenance actions are added.

After scheduling these maintenance actions, we need to recompute the condition of the components that suffered production damage in this time period. Since the production was set to 0 only when analyzing component k, the components that preceded it were incorrectly marked as damaged. Line 17 computes the correct values.

Some details were excluded to make the heuristic more readable. For example, if when reaching line 12 $x_{t'}^{k}$ had already been set to 1, then we know that this heuristic cannot provide a feasible solution, since a component in optimal condition without any production will still require maintenance. This can be either because the other components have a big impact on the degradation, which this heuristic does not consider, or because the instance itself is not feasible.

The choice of the production vector in Algorithm 2 can be more substantiated, and the results mentioned in the beginning of the section regarding maintenance anticipation should also provide better results, meaning that this heuristic can be improved upon.

5.1 Simplifications

KB: this is still what you are doing in the implementation, right? I would include this as a paragraph or two in the extended formulation implementation section, but I agree that this doesn't seem to be big enough for a separate subsection

PROBABLY I'LL REMOVE THIS SUBSECTION

This model may quickly become very large, depending on the number of periods being considered, but even for average-sized instances, it can be quite difficult to solve.

To cope with this, we assume that there are different time periods for production and maintenance decisions. For example, while production decisions are made every hour, maintenance might be planned on a biyearly basis. This forces minor modifications to the formulations, and some more significant ones, as the constraints that combine both y and x variables (like [1c]) need to use the proper indices.

Despite having a big impact on the tractability of the models, this simplification also falls in line with real-world applications, since maintenance decisions are naturally made on a much more granular scale than production decisions.

6 Computational Experiments

6.1 Benchmark instances used

KB: no need for a separate section to describe the instances. You can make one section for the computational experiments and include this as a subsection. JD@KB:Done.

In this section, we will detail the experimental setup that was used to validate the model, as well as the metrics used. All the code and experiment results can be found in [17].?????? STILL NEED TO PUSH

The instances were randomly generated. They are divided by the number of time periods (either 10, 20, or 50), by the number of machines (with some being identical), and they can have "high" or "low" complexity, meaning that the corresponding machines will have a higher or lower number of components with more or less interaction between them. The complexity of the components also affects the amount of nonlinearity in the model. For each of these combinations, we generate 10 different instances.

It is easy to create instances that favor each of the models, as having more or less identical machines favors the extended or the compact formulation respectively. For this reason, we have opted for creating three sets of instances, with one of them having several identical machines (one group of 20 machines), one other having many dissimilar machines (4 groups of 2), and an intermediate set (one group of 5 and one group of 2).

KB: please double check that this is indeed what you meant: I rephrased the sentence. JD@KB: Done. Rephrased a bit as well.

The varying combinations of time periods, complexity and machine groups yield $10 \cdot 3 \cdot 2 \cdot 3 = 180$ different instances.

For every instance, we ran three formulations: the compact formulation, the extended formulation with exact pricing, and the extended formulation with heuristic pricing. In the result subsection represented by "Compact", "DW", and "DW+PF" respectively.

KB: specify how these formulations are denoted in the tables. JD@KB: Done.

Each of the results in Section 6.2 represents the average of the 10 randomly generated instances according to the relevant specification (all with the same complexity, time periods, number of machines). The Time column will show the average time to optimality for the instances where optimality is achieved, and the Gap (%) column will show the average gap for the instances where it was not. The gap is the relative gap: $\frac{|\text{primal bound} - \text{dual bound}|}{\min(|\text{primal bound}|, |\text{dual bound}|)}$.

The experiments were run on two 6-Core Intel Core i7 with 64 GB running at 3.20GHz with a 30 minute time limit. The models were solved with the SCIP solver, version 9.0.0 [7]. For the implementation, we used SCIP's Python interface, PySCIPOpt [19], version 5.0. Some scripts (most importantly visualization with Matplotlib [16]) were implemented in Python version 3.11.4.

6.2 Results and Analysis

??? FIGURES > TABLES? HOW MANY INSTANCES? WHAT TO VARY?

Table 3: Comparing effect of time

Table 4: Results for $\mathcal{T} = 20$

Model	$\mathcal{N}$	Complexity	Time(s)	Gap (%)
Compact	???	???	???	???
DW	???	???	???	???
DW + PF	???	???	???	???

Table 5: Results for $\mathcal{T} = 50$

Model	$\mathcal{N}$	Complexity	Time(s)	Gap (%)
Compact	???	???	???	???
DW	???	???	???	???
DW + PF	???	???	???	???

KB: don't you already have different complexities in the previous tables? JD@KB: Yeah, I still need to think how I want to present the results...

As expected, the decomposition approaches appear advantageous in comparison to the compact model when increasing the number of identical machines. On the other hand, having more machines increases the number of pricing problems, and longer time-horizons increase their difficulty, so the compact formulation seems better suited for these instances.

Table 6: Comparing effect of different machines

Table 7: Results for $\mathcal{N}$ =???

Model	$\mathcal{T}$	Complexity	Time(s)	Gap (%)
Compact	???	???	???	???
DW	???	???	???	???
DW + PF	???	???	???	???

Table 8: Results for $\mathcal{N}$ =???

Model	$\mathcal{T}$	Complexity	Time(s)	Gap (%)
Compact	???	???	???	???
DW	???	???	???	???
DW + PF	???	???	???	???

Table 9: Comparing effect of machine complexity

Table 10: Results for low complexity

Model	$\mathcal{N}$	$\mathcal{T}$	Time(s)	Gap (%)
Compact	???	???	???	???
DW	???	???	???	???
DW + PF	???	???	???	???

Table 11: Results for high complexity

Model	$\mathcal{N}$	$\mathcal{T}$	Time(s)	Gap (%)
Compact	???	???	???	???
DW	???	???	???	???
DW + PF	???	???	???	???

An increase in machine complexity also increases the difficulty of the pricing problems....

7 Conclusion and Future Work

SLIGHTLY OUTDATED

This article presented a MINLP for a general production-maintenance scheduling problem. Besides the compact model, we also developed a Dantzig-Wolfe reformulation, and an additional heuristic due to the model's difficulty.

The model is able to solve multiple medium-sized instances to optimality, but on larger instances, the difficulty grows rapidly. The heuristic has trouble reaching optimality but can obtain good solutions very quickly, and in larger instances, it even gets better results that the original model. These results leave us optimistic that we can decrease the granularity of small and medium-sized instances, and still obtain good solutions.

The heuristic proved to be a good complement to exact pricing. Besides the comments at the end of the heuristic's description, we are considering taking better advantage of how fast the heuristic is. Our idea is doing multiple iterations of the heuristic with different production vectors, while adding benders-like optimality and feasibility cuts, which should provide significantly better columns.

A different extended formulation shows promising results, by reducing the number of costly pricing rounds. The idea is to generate maintenance patterns and production patterns (patterns being a concrete assignment to the corresponding variables) in different pricing problems. The formulation we arrived at requires more theory than this one, as it requires the simultaneous generation of columns and rows. See [22] for more details on column-and-row generation.

On the modeling side, more than one type of maintenance action can make the model more interesting. It increases its difficulty, but it also makes it more

realistic, as complete replacement of a component is usually not the only option. The model can also become more general. For example, there might be non-critical components that may be maintained while still allowing for some production of the machine. Another example is the production upper bound in Constraints[1d], which might depend not only on the condition of individual components, but also on the joint condition of multiple components.

Finally, the application of these techniques in real-world scenarios is also desirable.

References

1. *Guide for Transformer Maintenance. CIGRÉ*, Paris, 2011.
2. *Artificial Intelligence Application for Just in Time Maintenance*, volume Day 1 Mon, November 09, 2020 of *Abu Dhabi International Petroleum Exhibition and Conference*, 11 2020. D011S018R004. arXiv:https://onepetro.org/SPEADIP/proceedings-pdf/20ADIP/1-20ADIP/D011S018R004/2378389/spe-202869-ms.pdf, doi:10.2118/202869-MS.
3. Amir Abiri-Jahromi, Masood Parvania, François Bouffard, and Mahmud Fotuhi-Firuzabad. A two-stage framework for power transformer asset maintenance management—part i: Models and formulations. *IEEE Transactions on Power Systems*, 28(2):1395–1403, 2013. doi:10.1109/TPWRS.2012.2216903.
4. Muhamed Aganagic and Safoura Mokhtari. Security constrained economic dispatch using nonlinear dantzig-wolfe decomposition. *IEEE Transactions on Power Systems*, 12, 02 1997. doi:10.1109/59.574929.
5. Andrew Allman and Qi Zhang. Branch-and-price for a class of nonconvex mixed-integer nonlinear programs. *Journal of Global Optimization*, 81(4):861–880, May 2021. URL: http://dx.doi.org/10.1007/s10898-021-01027-w, doi:10.1007/s10898-021-01027-w.
6. Thomas Bittar, Pierre Carpentier, Jean-Philippe Chancelier, and Jérôme Lonchampt. A decomposition method by interaction prediction for the optimization of maintenance scheduling. *Annals of Operations Research*, 316(1):229–267, Sep 2022. doi:10.1007/s10479-021-04460-y.
7. Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. The scip optimization suite 9.0, 2024. arXiv:2402.17702.
8. George B. Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960. arXiv:https://doi.org/10.1287/opre.8.1.101, doi:10.1287/opre.8.1.101.
9. João Dionísio and João Pedro Pedroso. An optimization model for power transformer maintenance. In João Paulo Almeida, Filipe Pereira e. Alvelos, Jorge Orestes Cerdeira, Samuel Moniz, and Cristina Requejo, editors, *Operational Research*, pages 63–74, Cham, 2023. Springer Nature Switzerland.

10. Julius Farkas. Theorie der einfachen ungleichungen. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1902(124):1–27, 1902. URL: https://doi.org/10.1515/crll.1902.124.1 [cited 2024-01-24], doi:doi:10.1515/crll.1902.124.1.
11. Gerald Gamrath. *Generic Branch-Cut-and-Price*. PhD thesis, 03 2010.
12. M. Geurtsen, Jeroen B.H.C. Didden, J. Adan, Z. Atan, and I. Adan. Production, maintenance and resource scheduling: A review. *European Journal of Operational Research*, 305(2):501–529, 2023. URL: https://www.sciencedirect.com/science/article/pii/S0377221722002673, doi:10.1016/j.ejor.2022.03.045.
13. Mageed Ghaleb, Sharareh Taghipour, Mani Sharifi, and Hossein Zolfagharinia. Integrated production and maintenance scheduling for a single degrading machine with deterioration-based failures. *Computers & Industrial Engineering*, 143:106432, 2020. URL: https://www.sciencedirect.com/science/article/pii/S0360835220301662, doi:10.1016/j.cie.2020.106432.
14. Ambros Gleixner, Stephen J. Maher, Benjamin Müller, and João Pedro Pedroso. Price-and-verify: a new algorithm for recursive circle packing using dantzig–wolfe decomposition. *Annals of Operations Research*, 284(2):527–555, December 2018. URL: http://dx.doi.org/10.1007/s10479-018-3115-5, doi:10.1007/s10479-018-3115-5.
15. Alexander Grigoriev and Marc Uetz. Scheduling jobs with time-resource tradeoff via nonlinear programming. *Discrete Optimization*, 6(4):414–419, 2009. URL: https://www.sciencedirect.com/science/article/pii/S1572528609000334, doi:10.1016/j.disopt.2009.05.002.
16. J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi:10.1109/MCSE.2007.55.
17. Joao-Dionisio. Production-maintenance-scheduling-in-complex-machinery. https://github.com/Joao-Dionisio/Production-maintenance-scheduling-in-complex-machinery, 2024.
18. Zhenglin Liang and Ajith Parlikad. A markovian model for power transformer maintenance. *International Journal of Electrical Power & Energy Systems*, 99:175–182, 2018. URL: https://www.sciencedirect.com/science/article/pii/S0142061517321312, doi:10.1016/j.ijepes.2017.12.024.
19. Stephen Maher, Matthias Miltenberger, João Pedro Pedroso, Daniel Rehfeldt, Robert Schwarz, and Felipe Serrano. PySCIPOpt: Mathematical programming in python with the SCIP optimization suite. In *Mathematical Software – ICMS 2016*, pages 301–307. Springer International Publishing, 2016. doi:10.1007/978-3-319-42432-3_37.
20. Stephen J. Maher, Ted K. Ralphs, and Yuji Shinano. Assessing the effectiveness of (parallel) branch-and-bound algorithms. *CoRR*, abs/2104.10025, 2021. URL: https://arxiv.org/abs/2104.10025, arXiv:2104.10025.
21. Dragan Banjevic Maliheh Aramon Bajestani and J. Christopher Beck. Integrated maintenance planning and production scheduling with markovian deteriorating machine conditions. *International Journal of Production Research*, 52(24):7377–7400, 2014. doi:10.1080/00207543.2014.931609.
22. İbrahim Muter, Ş. İlker Birbil, and Kerem Bülbül. Simultaneous column-and-row generation for large-scale linear programs with column-dependent-rows. *Mathematical Programming*, 142(1):47–82, Dec 2013. doi:10.1007/s10107-012-0561-8.
23. Sven Henrik Pries. *Selected topics on integrated production-scheduling and maintenance-planning problems*. PhD thesis, Staats-und Universitätsbibliothek Hamburg Carl von Ossietzky, 2022.

24. Adriaen Verheyleweghen, Himanshu Srivastav, Anne Barros, and Johannes Jäschke. Combined maintenance scheduling and production optimization. pages 499–506, 01 2019. doi:10.3850/978-981-11-2724-3_0253-cd.
25. Zhicheng Zhu, Yisha Xiang, and Bo Zeng. Multi-component maintenance optimization: A stochastic programming approach, 2019. arXiv:1907.00902.