Actor-based Large Neighborhood Search for Weekly Maintenance Scheduling

Christian Brunbjerg Jespersen[a], Kristoffer Sigsgaard Wernblad[a], Thomas Jacob Riis Stidsen[b], Kasper Barslund Hansen[a], Jingrui Ge[a], Simon Didriksen[a], Niels Henrik Mortensen[a]

[a] *DTU Construct, Technical University of Denmark, Anker Egelundsvej 1, Kongens Lyngby, 2800, Hovedstaden, Denmark*
[b] *DTU Management, Technical University of Denmark, Anker Egelundsvej 1, Kongens Lyngby, 2800, Hovedstaden, Denmark*

Abstract

Many planning problems in operations research are challenging due to their inherent uncertainty and dynamic nature. Methods such as Stochastic Programming (Birge and Louveaux, 2011), fuzzy logic (Zadeh, 1988), robust optimization (Ben-Tal et al., 2009), dynamic optimization (Yang et al., 2013), and human-guided search optimization (Talbi, 2009) have been proposed to address these issues. However, these methods often assume static data and problem settings. Maintenance scheduling exemplifies a dynamic problem where information is continually updated, requiring frequent rescheduling. While traditionally viewed as an operations management task, this paper argues that general maintenance scheduling approaches can be developed by integrating optimization solutions into business processes guided by operations management principles (e.g., plan-do-check-act) (Deming, 1982). This paper proposes a novel actor-based optimization approach using a large neighborhood search metaheuristic to address dynamic scheduling problems characterized by real-time data updates and external inputs influencing optimization with future research focusing on integrating interdependent decisions by multiple actors with differing objectives. The metaheuristic is tested on real-world maintenance scheduling for offshore platforms operated by Total Energies in the North Sea and demonstrates general applicability to other operational planning problems.

Keywords: Large Neighborhood Search, Actor Framework, Maintenance scheduling, Real-time Optimization, Human-centered Computing,

Decision Support Systems, Interactive Optimization.

---

1. Introduction

Maintenance scheduling is part of a class of operational problems that have proven hard to solve in practice (maintenance scheduling problems are usually modelled as resource-constrained project scheduling problems, knapsack formulations, machine scheduling problems, etc. which are all NP-hard problems (Garey and Johnson, 1979)). To be effective in dynamic and uncertain environments where maintenance scheduling is carried out, optimization must be tightly integrated with existing IT infrastructure. This integration ensures that the tacit knowledge of decision-makers can seamlessly influence the planning process. The planning process often involves multiple decision-makers operating at different business levels. As a result, responsibility for decision-making is typically assigned to individuals who represent only a small segment of the overall process. These multiple smaller planning processes are often difficult to map to a single mathematical model describing the whole system as elaborated by (Barthélemy et al., 2002). Solving operation research problems that are operational in nature have additional requirements over more typical static problems: they have to be responsive to changing parameters; able to be assimilated into the decision-makers workflow; allow for integration with dynamic data sources such as databases and APIs (Meignan et al., 2015b). Operational aspects of operation research, as opposed to higher level strategic and tactical ones, are characterized by extensive amounts coordination and negotiation on proposed schedules often in a short amount of time (Palmer, 2019a).

The lack of integration into the schedulers and supervisors workflow and lack of responsiveness can lead to a situation where solutions are not directly implemented in practice but instead only provides initial suggestions (Meignan et al., 2015b). Theses initial suggestions are then iterated on elsewhere in the scheduling process usually through much more manual means. In (Barthélemy et al., 2002) the authors argue that many problems that operation research aim to solve are often composed of a group of individuals whose decisions are consolidated into an "epistemic subject" for which a mathematical model can be formulated and solved, with many scheduling problems being good examples. However, often multiple actors have different views on what constitutes an optimal schedule hence resulting in multiple-objectives. Even if multi-objective optimization (Ehrgott and Gandibleux,

2002) is applied to find a Pareto Front (Pareto, 1897) a negotiating process still is needed between the actors to select the final schedule.

This paper proposes a solution method that will allow for real-time optimization based on actor/user interaction and a connection to a dynamic data source, effectively managing the changes to the parameter space as they occur. The proposed solution method will be tested on the weekly maintenance scheduling problem (Palmer, 2019b) which closely resembles a variant of the multi-compartment multi-knapsack problem (MCMKP) (do Prado Marques and Arenales, 2007). It should be noted that the scientific maintenance scheduling literature can deviate significantly from its practical implementation which is detailed in (Palmer, 2019a). The solution method is based on the large neighborhood search (LNS (Shaw, 1998)) metaheuristic and is in this paper described as actor-based large neighborhood search (AbLNS). LNS was chosen due to its properties of naturally being able to work with and fix infeasible solutions and because of its state of the art performance on various scheduling problems (gen, 2019).

To understand the need for actor-based methods some background knowledge will be required about the maintenance scheduling process. Figure 1 illustrates the general setup of a maintenance planning and scheduling system. The system's actors have the following responsibilities: the planner generates the work orders that are to be scheduled; each scheduler creates weekly schedules for a set of work orders; based on the weekly schedule the supervisors assign work order activities to technicians; the technicians executes the work. Each planning problem is matched by a corresponding optimization problem, for the scheduler, it is a variation of the multi-compartment multi-knapsack problem, for the supervisor it is a variation of the assignment problem and for the technicians it is a single machine scheduling problem.

The concept of "ownership" of a work order is fundamental to understanding the need for actor-based approaches. During the scheduling process, each work order is assigned to a specific actor, who alone has the authority to modify or execute it.

This means that a single model approach is very difficult to implement in practice as a work order is modelled differently depending on the actor that currently owns it. This highlights another point in maintenance scheduling: that the stochastic nature of the maintenance scheduling process can be handled using a change of model, each with different levels of aggregation and different sets of constraints, opposed to more academic approaches such as fuzzy logic and stochastic optimization. When the inherrent uncertainties

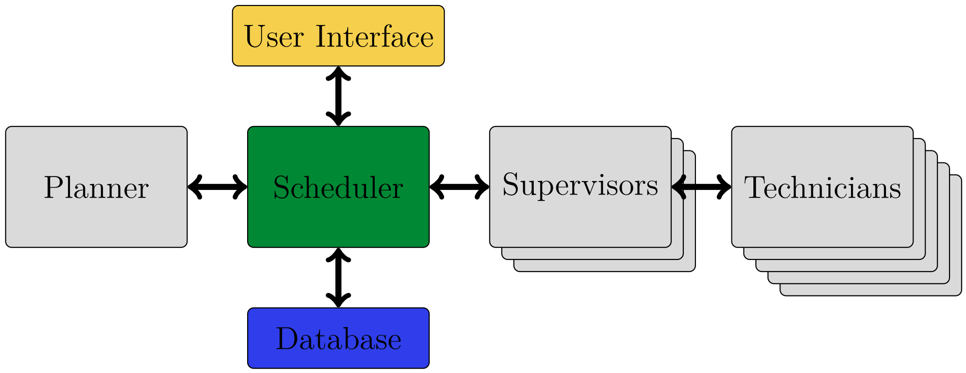

Fig 1: Simple overview of a maintenance scheduling process with its primary types of actors. The planner, the scheduler, the supervisor(s), and the technicians. The green color highlights the scheduler as it the actor in the maintenance scheduling process that is modelled in this paper.

manifest themselves during planning or execution, work orders are rescheduled by moving between the different actors, meaning that the stochastic elements of maintenance scheduling are handled by **dynamic rescheduling between the actors**.

The primary contribution of this article is the development of a modular, scalable optimization component. This component leverages a proven metaheuristic to support business processes constructed from smaller mathematical models within a framework, rather than relying on a single, integrated mathematical model.

The paper is organized into four sections. Section 2 provides a detailed explanation of the weekly maintenance scheduling model, which serves as the foundation of the study. Section 3 presents the results obtained from the implemented metaheuristic, highlighting its response to simulated user interactions. Section 4 discusses the research implications and outlines potential directions for future work. While all source code for the implemented system is available at (Scipo, 2024), the instance data remains confidential and cannot be shared publicly.

1.1. The Weekly (Period) Maintenance Scheduling Model

The weekly maintenance scheduling model for the problem is a variant of the Multi-compartment Multi-knapsack Problem with capacity penalties MCMKP The notation used to describe the dynamical aspects in the model is based on the notation from the dynamic metaheuristics literature as found in (Yang et al., 2013). source on the dynamic notation. Here τ is added as a time variable on all sets, parameters, and variables that are subject to change while the metaheuristic is running. This enables precise timing on the messages that are send to the AbLNS and understand how it reacts in real-time. A company performing maintenance usually creates weekly maintenance plans for the next $p \in P(\tau)$ period. The weekly schedule is created centrally and consists of scheduling the $w \in W(\tau)$ work orders, i.e. maintenance tasks, such that all w are scheduled into a specific period p. Each work order w requires some resources $r \in R(\tau)$ to be carried out, e.g. man-power with different qualifications. Each of these resources are available in limited amounts given by $resource_{pr}(\tau)$. To correct for possible manual interventions that can make the problem infeasible a penalty ($strategic\_resource\_penalty$) is introduced. The urgency of the different maintenance work order (w) varies and is reflected in a 'tardiness' for carrying out a maintenance work order in a certain period given by $strategic\_urgency_{wp}(\tau)$. The $strategic\_urgency_{wp}(\tau)$ also captures the tardiness of the individual work orders (w), meaning that the value gained from scheduling work orders late is increasing. Urgent tasks have increasing value the further out the period p becomes. Furthermore, two sets exists which will either require work order w to be carried out in period p or not carried out in a period p. These sets are $(w, p) \in include_{wp}(\tau)$ for inclusion and $(w, p) \in exclude_{wp}(\tau)$ for exclusion.

Meta variables:

$$s \in S \tag{1}$$

$$\tau \in [0, \infty] \tag{2}$$

Minimize:

$$\begin{aligned}
&+ \sum_{w \in W(\tau)} \sum_{p \in P(\tau)} strategic_urgency_{wp}(\tau) \cdot \alpha_{wp}(\tau) \\
&+ \sum_{p \in P(\tau)} \sum_{r \in R(\tau)} strategic_resource_penalty \cdot \epsilon_{pr}(\tau) \\
&- \sum_{p \in P(\tau)} \sum_{w1 \in W(\tau)} \sum_{w2 \in W(\tau)} \quad clustering_value_{w1,w2} \cdot \alpha_{w1p}(\tau) \cdot \alpha_{w2p}(\tau)
\end{aligned} \tag{3}$$

Subject to:

$$\begin{aligned}
&\sum_{w \in W(\tau)} work_order_workload_{wr} \cdot \alpha_{wp}(\tau) \leq \ resource_{pr}(\tau) + \epsilon_{pr}(\tau) \\
&\quad \forall p \in P(\tau) \quad \forall r \in R(\tau)
\end{aligned} \tag{4}$$

$$\sum_{w \in W(\tau)} \alpha_{wp}(\tau) = 1 \quad \forall p \in P(\tau) \tag{5}$$

$$\alpha_{wp}(\tau) = 0, \quad if \quad exclude_{wp}(\tau) \ \ \forall w \in W(\tau) \quad \forall p \in P(\tau) \tag{6}$$

$$\alpha_{wp}(\tau) = 1, \quad if \quad include_{wp}(\tau) \ \ \forall w \in W(\tau) \quad \forall p \in P(\tau) \tag{7}$$

$$\alpha_{wp}(\tau) \in \{0, 1\} \quad \forall w \in W(\tau) \quad \forall p \in P(\tau) \tag{8}$$

$$\epsilon_{pr}(\tau) \in \mathbb{R}^{+} \quad \forall p \in P(\tau) \quad \forall r \in R(\tau) \tag{9}$$

The meta variables defines the broader setting that the model in implemented in. Equation (1) specicies that the model is implemented for scheduler s. Equation (2) is the time variable that binds the whole system together, by allowing us to reason about the sequence that events are happening in.

The objective function (3), minimizes the total weighted assignment of all work orders subtracted by the penalty *strategic_resource_penalty* for exceeding the resource capacity given in equation (4). The third term of the objective function handles the $clustering\_value_{w1,w2}$ which turns the model into a quadratic problem. This term optimizes the value of scheduling two work orders in the same period, if they have share similarity like close proximity, same piece/type of equipment, etc. Equation (4) ensures that all the $work_{wr}(\tau)$ for each activity in a work order, given that it has been assigned, is lower than the $resource_{pr}(\tau)$ for each p and for each resources r. $strategic\_resource\_penalty$ is the amount of exceeded capacity that is needed for the current assignment of work orders to be feasible. Equation (5) makes sure that each work order is assigned to at least a one period. Equation (7) excludes a work order from a certain period and equation (6) forces a specific work order to be included in a specific period. Constraint (8) and (9) specify the variable domain for $\alpha_{wp}(\tau)$ and $strategic\_resource\_penalty$ respectively. The effects of changing $include_{wp}(\tau)$, $exclude_{wp}(\tau)$, $resource_{pr}(\tau)$, and $strategic\_urgency_{wp}(\tau)$ in real-time will be examined in the Section 3 to determine their effects on the weekly schedules and the objective value.

2. Solution Method

2.1. Actor-based Large Neighborhood Search

A metaheuristic that is affected by the end user and requires real-time feedback inherently requires an optimization approach that is able to repair infeasible solutions while also converging quickly. The large neighborhood search (LNS) (Shaw, 1998) metaheuristic has been shown satisfy these requirements in the literature (gen, 2019). The most general form of the LNS metaheuristic is defined for static problems, meaning that the parameters that make up the problem instance is not subject to change after the algorithm has started. To make the LNS adapt to changing parameters in real-time a message system have been implemented into the existing framework. This extension is shown in algorithm 1.

2.1.1. Messages and Destroy Functions

LNS in its most basic form has one repair and one destroy function which repeatedly destroy and rebuild the solution. For the AbLNS (pseudocode shown in algorithm 1) we will generalize on this concept by including messages as destroy functions. This generalization can be seen as being somewhat similar to how the adaptive LNS (ALNS) (Pisinger and Ropke, 2007) is formulated, but where the different repair and destroy functions are also chosen from sources external to the algorithm.

Extending on the classic setup we define the following set of destroy messages, $m \in M$. Each message is received by the system at time $\tau \in \mathbb{R}$.

- $m_1^{wp}(\tau)$: Inclusion destroy message, modifies $include_{wp}(\tau)$
  - Each instance of this message destroys the solution in line 9 and updates the problem parameters in line 8 of algorithm 1. On line 11 the repair function then reschedules according to the value of $include_{wp}(\tau)$.
- $m_2^{wp}(\tau)$: Exclusion destroy message, modifies $exclude_{wp}(\tau)$
  - Each instance of this message updates the $exclude_{wp}(\tau)$ on line 8 and if the message excludes an already scheduled work order the destroy function on line 9 will update the solution as well. If the work order was already forced into the schedule by $include_{wp}(\tau)$ line 8 and line 8 will also update $include_{wp}(\tau)$.
- $m_3^{pr}(\tau)$: Capacities destroy message, modifies $resource_{pr}(\tau)$
  - Each instance of this message perturbes the value of available hours of a given resource $r \in R(\tau)$ for a given $p \in P(\tau)$, this update happens on line 8 in algorithm 1.
- $m_4^{wp}(\tau)$: Strategic urgency destroy message, modifies $strategic\_urgency_{wp}(\tau)$
  - Each instance of this message perturbes the strategic urgency of a work order, this happens on line 8 and line 9 of algorithm 1. Here it should be noted that the message changes the $strategic\_urgency_{wp}(\tau)$ $\forall p \in P(\tau)$ for a given $w \in W(\tau)$.
- $m_5^{wk}(\tau)$: Random destroy message, modifies $\alpha_{wp}(\tau)$
  - Each instance of this message modifies the main decision variable, $\alpha_{wp}(\tau)$. Removing random work order assignments from k randomly chosen $w \in W(\tau)$, where $k \in \mathbb{N}$ is a parameter of the AbLNS.

Each of these messages impacts different aspects of the weekly maintenance scheduling problem. Notably, the first four messages destroys the solution by modifying both the parameter/data space and the solution itself, while the final message serves as a random destruction function that affects only the solution.

For a correct implementation it is critical that the that destroy messages are handled one by one, and are not allowed to simply write to the solution and problem instance whenever they appear. Generalizing the destroy functions from being static structures into messages allows the solution to change in real-time to a changing parameter space. This means that the algorithm does not need to restart to handle changes in data.

Algorithm 1 Actor-based Large Neighborhood Search

1: **Input** Q = message queue
2: **Input** P = problem instance
3: **Input** X = initial schedule
4: **Input** S = shared solution
5: **repeat**
6:     $X^t = X$
7:     **while** $Q.has\_message()$ **do**
8:         $P.update(S, m_n(\tau))$
9:         $X^t.destroy(S, m_n(\tau))$
10:     **end while**
11:     $X^t.repair(S)$
12:     **if** accept (X^t, X) **then**
13:         $X.update(X^t)$
14:     **end if**
15:     **if** $c(X^t) < c(X)$ **then**
16:         $X.update(X^t)$
17:         $S.atomic\_pointer\_swap(X)$
18:     **end if**
19:     $Q.push(m_5(\tau))$
20: **until**

The basic LNS setup have here been extended with a "message queue" Q. The message queue will be read from on every iteration of the LNS's main iteration loop. Here we notice that the incoming message is able to change both the solution X (on line 9) but also the problem instance P (on line 8) itself. Due to the inherent property of LNS being able to optimize a solution that have become infeasible, LNS is well suited to handle changing parameters in the problem instance P. Another property of the message queue is that the algorithm can run indefinitely and as opposed to restarting the algorithm you pass messages containing new data and/or state. This property avoids the time consuming initial convergence as the algorithm will be found in an optimized state when the solution is perturbed. Notice that:

- The algorithm responds quickly to changes. In each iteration line 7. We call this a fine-grained response algorithm
- If an improved solution is found, it is immediately being pointer swapped on line 15
- That the optimization occurs in the repair function on line 11, which inserts work orders using a greedy algorithm.
- The atomic pointer swapping that is performed on line 17 when a new best solution is found instantaneously makes the solution available to other actors/systems. The pointer swapping is crucial as it allows for lock-free sharing of state and makes non-hierarchical model setups easier.

The AbLNS algorithm will run continuously, either optimizing the current schedule or updating the schedule with new external input. Since the effects of a message is considered in every iteration, changing conditions will immediately get optimized upon. Since a fast response is paramount, exact optimization algorithms are deemed impractical.

2.2. Atomic Pointer Swapping and Software Architecture

Atomic pointer swapping offers a novel approach for managing the best solution found by the AbLNS and LNS algorithms. Traditionally, in a metaheuristics such as Large Neighborhood Search (LNS) setup, the best solution is cloned (deep copy) before the search continues. However, atomic pointer swapping enables the best solution to be shared across system components without introducing data races. This approach is essential in multithreaded environments, as it ensures the integrity of the solution during swaps, this elaborated in (Herlihy et al., 2020). Multiprocessor programming has long been considered an art but in recent years new tools have emerged to make this type of programming much easier most notably (The Rust Programming Language). In software architecture, scaling is often achieved through vertical scaling (lengthening the dataflow) and horizontal scaling (increasing throughput) via message passing. While effective in many domains, this approach poses challenges for metaheuristics, which rely on rapid state mutations and solution improvements.

Message passing in metaheuristics can create unpredictable behaviors, especially with bidirectional communication between components and metaheuristics. The limitations of message-passing systems in metaheuristics are well-documented, particularly in the context of parallel and multi-agent metaheuristics (Talbi). These systems often suffer from excessive message generation, leading to poor scalability, explainability, extendability, and performance. Key issues include:

- The stochastic nature of metaheuristics makes message-passing behavior unpredictable.
- Using message passing often necessitates a hierarchical setup, as bidirectional message flows often results in cycles.
- The impact of additional types messages is difficult to understand and analyze.

Atomic pointer swapping addresses these challenges by eliminating state duplication, reducing system complexity, and bypassing the inefficiencies of message passing. This method provides a streamlined, modular way to integrate metaheuristics, aligning with modern software architecture principles while avoiding their typical pitfalls (Richards and Ford, 2020). It should be noted that message passing is exceptional at dealing with one-off events that have varied and sporatic nature such as interfacing with actors, external systems, or handling constraints that are hard to find the data for or are so rarely binding that you would rather handle them through manual intervention.

3. Results

To test the AbLNS algorithm, several simulations are conducted in which the data is perturbed during the algorithms execution. The data from the company is presented in Section 3.1. Then the effect of forcing work orders into specific weekly $p \in P(\tau)$ schedules is presented in Section 3.3 and excluding work orders from periods is presented in Section 3.4. The effects of reducing the period resource capacities $resource_{pr}(\tau)$ is tested in Section 3.6 and increasing r is tested in Section 3.5. Finally the effect of changing the work order values $strategic\_urgency_{wp}(\tau)$, is tested in Section 3.7.

3.1. Data Instance

The data instance used in this paper is provided by Total Energies (Total Energies, 2024) and extracted from their SAP ERP system. It pertains to a specific offshore platform and covers a two-year time horizon. The instance includes 3,487 outstanding work orders ($W(\tau)$), 16 distinct resource skill sets ($R(\tau)$) (e.g., mechanics, electricians, turbine specialists, etc.), and spans 52 bi-weekly periods ($P(\tau)$), roughly equivalent to two years.

	$\lvert W(\tau)\rvert$	$\lvert R(\tau)\rvert$	$\lvert P(\tau)\rvert$
Instance 1	3487	16	52

Table 1: Specific data instances from the case company. Here $W(\tau)$ is the set of work orders, $R(\tau)$ is the set of resources, and $P(\tau)$ is the set of weekly periods.

3.2. Value of Objective Function Parameters

The optimization problem has three terms and it could be argued that the a pareto front should be calculated on the value of the different weightings between them. To not lose sight of the main contribution of the paper the value of the $clustering\_value_{w1,w2}$ and $strategic\_resource\_penalty$ has been set so that the $strategic\_resource\_penalty$ always dominates the $strategic\_urgency_{wp}(\tau)$ and that the $strategic\_urgency_{wp}(\tau)$ always dominates the $clustering\_value_{w1,w2}$. Furthermore the $clustering\_value_{w1,w2}$ has been excluded from the results section to put more focus on the user-input interaction.

3.3. Response to Inclusion of Work Orders

The $include_{wp}(\tau)$ parameter specifies whether a work order should be scheduled into a specific period. As the parameter has the time variable τ it means that this parameter can change at any time while the metaheuristic is running. The $include_{wp}(\tau)$ parameter constrains the model in equation 6. Table 2 shows the responses that the model will be subject to while it is running for different timepoints τ.

	$\tau_1 = 60$	$\tau_2 = 120$	$\tau_3 = 180$	$\tau_4 = 240$	$\tau_5 = 300$
$\Delta include_{wp}(\tau)$	50	50	50	50	50

Table 2: The including work orders perturbations that the AbLNS will be affected by. Perturbations occur at 60 second time intervals affecting 50 randomly chosen work orders included into random periods.

Figure 2 shows the effects of changing the $include_{wp}(\tau)$ parameter in real-time. The model quickly converges and when the system is pertubed by an input response the objective value 3 shows a small spike and then quickly converges to a new solution.

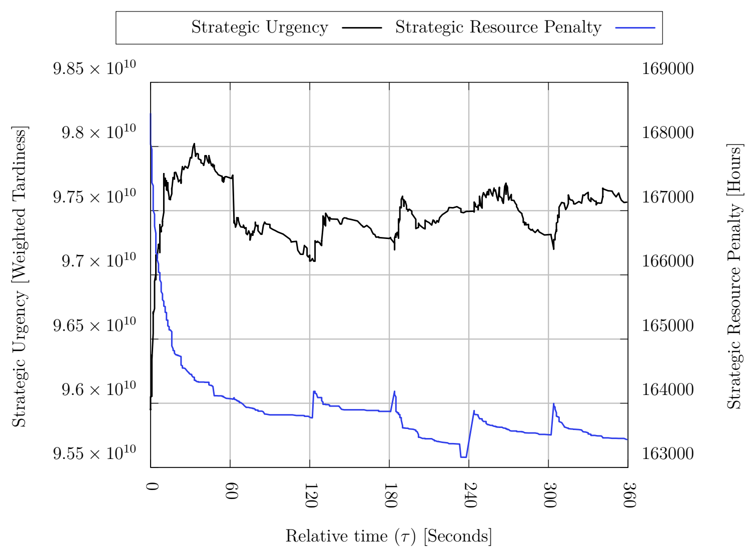

Fig 2: At each 60 second time interval the the solution is perturbed by forcing work orders into specific periods. Here the strategic urgency fluctuatues as the dominating term of the strategic resource penalty is minimized.

Figure 2 show 5 perturbations with the first at $\tau = 60s$ where the objective value slightly increases in response to the inclusion, the objective value increases due to the inclusion either causing the capacity to be exceeded or the inclusion resulting in a selected $strategic\_urgency_{wp}(\tau)$ that has a higher value. The remaining 4 perturbations all show the same pattern, an increase in the strategic urgency and resource penalty followed by a subsequent convergence.

3.4. Response to Exclusion

The response to exclusion is associated with the $exclude_{wp}(\tau)$ parameter and is found in equation 7. Here specific work orders ($w \in W(\tau)$) are being excluded from specific periods ($p \in P(\tau)$). The perturbations that the AbLNS will be affected by are shown in table 3 with the setup being very similar to the one in table 2. The main distinction being that the perturbation affects 500 instead of 50 work orders, the higher number of affected work orders is chosen as many exclusions of do not affect the assignment of a work order.

	$\tau_1 = 60$	$\tau_2 = 120$	$\tau_3 = 180$	$\tau_4 = 240$	$\tau_5 = 300$
$\Delta exclude_{wp}(\tau)$	500	500	500	500	500

Table 3: The exclusion of work orders perturbations from specific periods on the weekly schedule. Perturbations occur at 60 second time intervals affecting 500 work orders each time.

Figure 3 show a substantial spike in the strategic urgency and a substantial decrease in the strategic resource penalty after each perturbation as given by table 3.

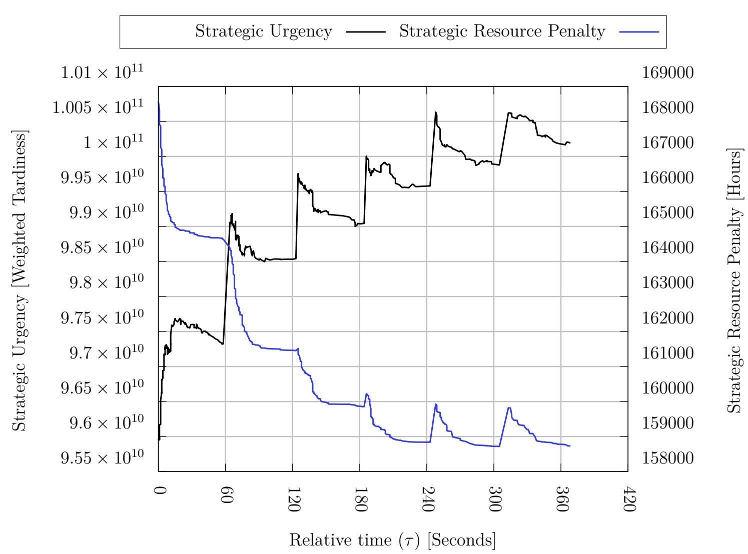

Fig 3: At each 60 second interval the strategic urgency experiences a large spike as urgent work order are being forced to be scheduled further out. The strategic resource penalty in lowered as $include_{wp}(\tau)$ parameters and corresponding constraints are removed

From figure 2 and figure 3 it is clear that the AbLNS method can handle dynamic entries of work orders. The next section will discuss the effects of dynamically changing the resource capacities $resource_{pr}(\tau)$.

3.5. Response to Additional Weekly Capacity

Table 4 details the perturbations that the AbLNS will be subject to during its 360 second execution. Perturbing the $resource_{pr}(\tau)$ affects the solution considerably more than perturbing $exclude_{wp}(\tau)$ and $include_{wp}(\tau)$ and therefore 100 second intervals are specified instead of 60 second intervals.

	$\tau_1 = 0$	$\tau_2 = 100$	$\tau_3 = 200$
$\Delta\lvert P(\tau)\rvert$	52	52	52
$\Delta\lvert R(\tau)\rvert$	16	16	16
$\lvert\lvert resource_{pr}(\tau)\rvert\rvert$ (hours)	61816	111268	173083

Table 4: The resource perturbations that the AbLNS will be affected by measured in hours. Here all $p \in P(\tau)$ and $r \in R(\tau)$ are affected

Figure 4 shows the effects of progressively increasing available resources. The AbLNS starts with an initial load which is then increased at $\tau = 100$ seconds causing the objective value to decrease as $\epsilon_{pr}(\tau)$ in equation 4 can achieve a lower value. At $\tau = 200$ the resources are increased to their final value.

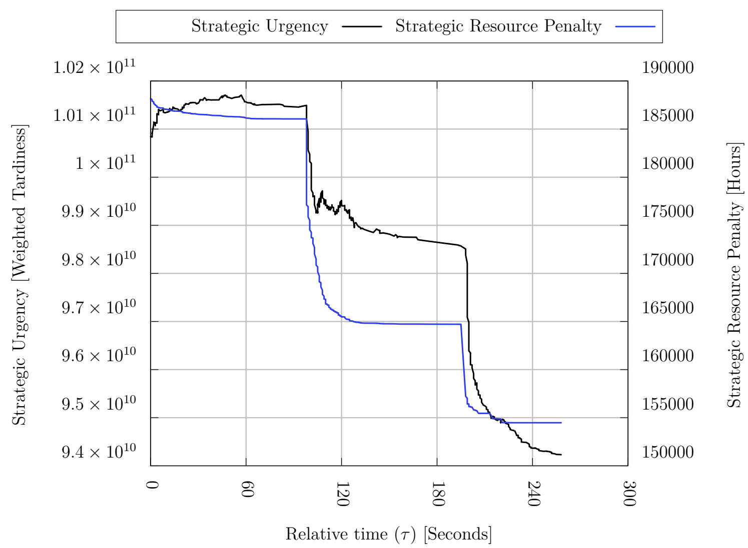

Fig 4: Starting from an initial load of 61816 hours. The resources are increased causing a drop in the objective value and the AbLNS then optimizes around the perturbation

3.6. Response to Reduced Weekly Capacity

Table 5 details the perturbations that the AbLNS will affected by. Starting from an initial amount of available resource, the resources are progressively decreased.

	$\tau_1 = 0$	$\tau_2 = 100$	$\tau_3 = 200$
$\Delta\vert P(\tau)\vert$	52	52	52
$\Delta\vert R(\tau)\vert$	16	16	16
$\vert resource_{pr}(\tau)\vert$ (hours)	173,083	111,268	61,816

Table 5: The resource perturbations that the AbLNS will be affected by measured in hours. Here all $p \in P(\tau)$ and $r \in R(\tau)$ are affected

Figure 5 shows the effects of perturbing the AbLNS by starting from an initial load and then progressively reducing capacity.

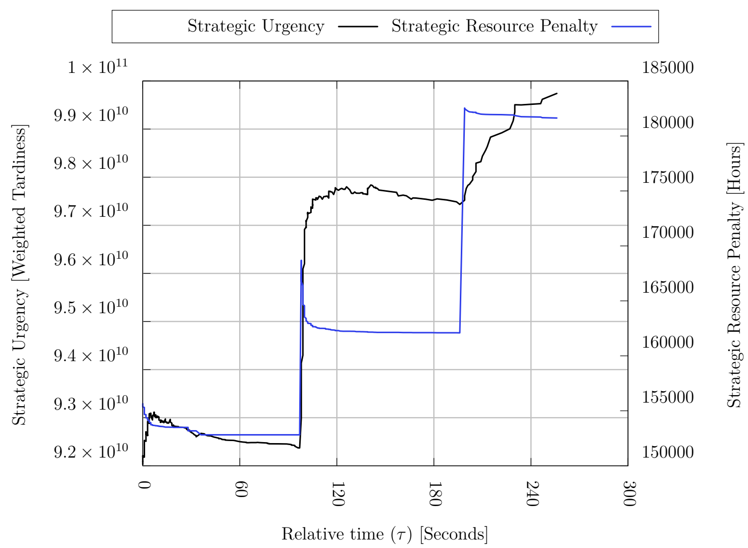

Fig 5: Starting from an initial load of 173083 total hours the AbLNS is progressively affected by decreasing levels of resources. The AbLNS is pertubed by two decreases in resources causing an initial spike in the objective value followed by the AbLNS optimizing around the perturbation

3.7. Response to Changes in Work Order Values

The final parameter that will be changed is the work order urgency parameter $strategic\_urgency_{wp}(\tau)$. Table 6 details the perturbations that the AbLNS will by affected by. On each iteration 100 work orders are having their values changed by the amount shown in the 4th row of table 6.

	$\tau_1 = 60$	$\tau_2 = 120$	$\tau_3 = 180$	$\tau_4 = 240$	$\tau_5 = 300$
$\Delta\vert W(\tau_n) \triangle W(\tau_{n-1})\vert$	100	100	100	100	100
$\Delta\vert P(\tau_n) \triangle P(\tau_{n-1})\vert$	52	52	52	52	52
$\vert\vert strategic\_urgency_{wp}(\tau_n) - strategic\_urgency_{wp}(\tau_{n-1})\vert\vert$	$3.75 \cdot 10^7$	$3.75 \cdot 10^7$	$3.75 \cdot 10^7$	$3.75 \cdot 10^7$	$3.75 \cdot 10^7$

Table 6: Perturbations that the $strategic\_urgency_{wp}(\tau)$ will be affected by

Figure 6 shows the effects of perturbing the AbLNS by changing the $strategic\_urgency_{wp}(\tau)$ parameter in the objective function 3 which specifies the value of assigning a work order to a specific period.

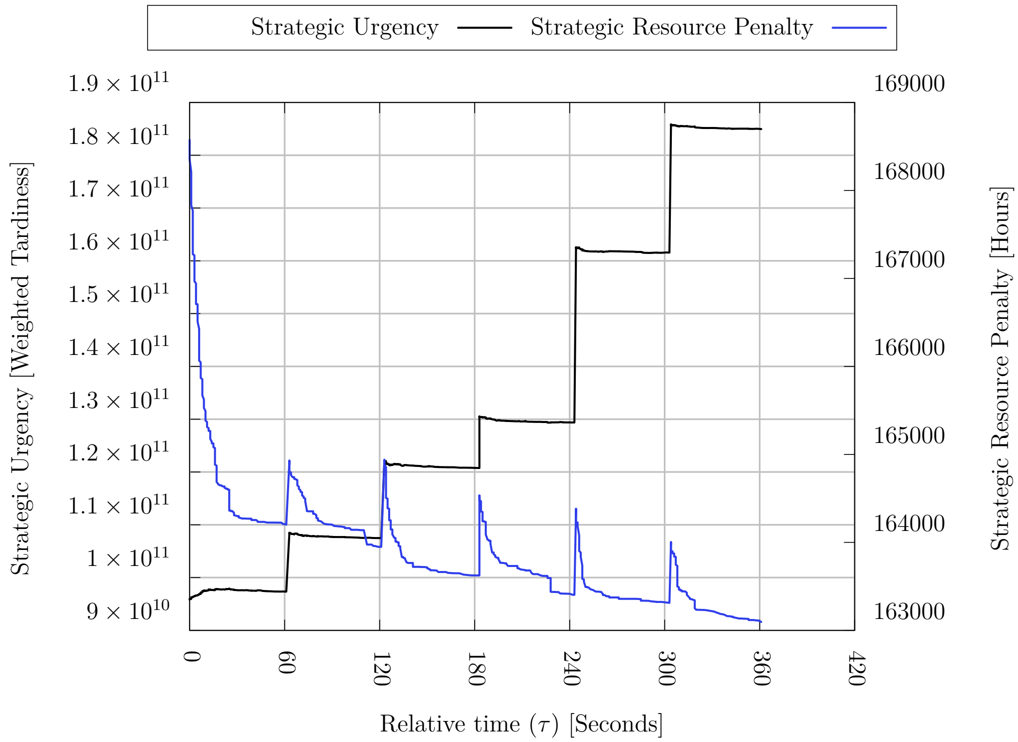

Fig 6: The effects of perturbing the AbLNS by dynamically changing the $strategic\_urgency_{wp}(\tau)$ in the objective function. The objective value is increasing in response to the higher cost associated with a late scheduling, and a higher initial resource consumption after which it optimizes around the perturbation

4. Discussion

The maintenance scheduling process effectively but not always efficiently models and determines good solutions to a complex scheduling problem by relying on multiple actors. Through the use of actors the scheduling process handles uncertainty that is difficult to reason about in a single mathematical model. These uncertainties are solved through coordination in time (modelled as τ). Each type of actor in the process acts according to a model each with different levels of aggregation and properties where each actor has a solid understanding of his own model. In the discussion interesting aspects of this approach has been divided into three sections: Section 4.1 actors and integration; Section 4.2 continuous optimization allows asynchronous optimization; and Section 4.3 future research.

4.1. Actors & Integration

Often in operation research the failure to reliably solve operational problems in industry are not due to the problems being computationally intractable (Gendreau and Potvin, 2019). It is usually a practical problem of connecting data streams so that the solution approach continually receives dynamic data to handle changes and then sends the resulting solutions to the relevant actors (stakeholders), ideally through a relevant interface (Meignan et al., 2015a). The actor-based approach proposed in this paper makes integration easier by naturally encapsulating a model with a reliable interface.

4.2. Continuous Optimization

With actor-based metaheuristics, the optimization loop can run indefinitely, optimizing based on the latest available information. This may seem like a detail as you could argue that you should only ever optimize the schedule when there is an explicit need for it, but consider the case when you start adding more than two actors to a scheduling system, then there arises a need to coordinate people temporally as each will have to run their optimizing process one after another. This is depicted in figure 4.2 where the output of one model is used as the input to the next one, leading to the hierarchical model setup.

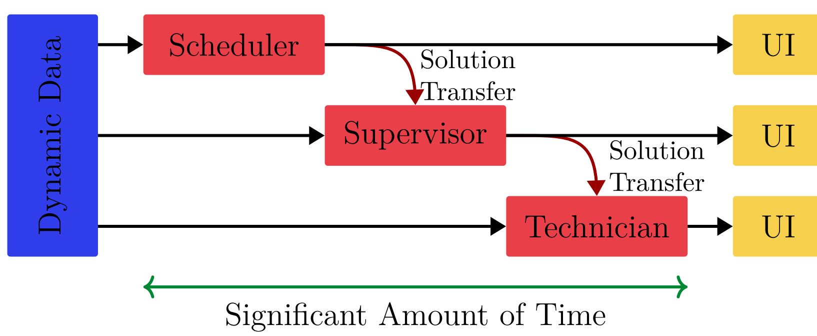

Fig 7: Effects of using hierarchical models setups in human-guided search metaheuristics. Due to the dependent nature of each metaheuristic it becomes crucial that the running of the metaheuristics are well coordinated between the actors.

In practice there are multiple problems with using a hierarchical setup. Usually the biggest one is that the information and knowledge needed to execute a feasible schedule is usually found in the lower levels of the hierarchicy. The operational setting, where the technicians are working, is usually so complex that it not feasible to centralize the knowledge that is required to create and execute a schedule. Figure 8 shows the kind of non-hierarchical setup that an actor-based approach allows for.

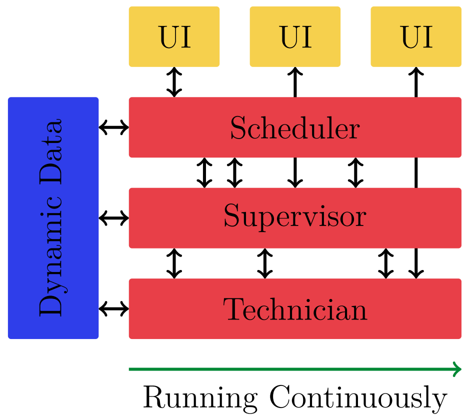

Fig 8: Asynchronous model setup where each metaheuristic runs in perpetuity. In this setup there is no need to coordinate actors to run the metaheuristics. Each actor in the scheduling process will always have the solutions of the other actors available to him to guide his own search.

When the optimization approach optimize continuously it enables tight integration between the different model implementations. Instead of running models to completion you simply handle changes in model parameters, model solutions, user inputs, and in the dynamic data source as they occur opposed to restarting the metaheuristics.

4.3. Future Research

The next step in this direction will be to model the remaining stakeholders as their own AbLNS metaheuristics, and then make them communicate together through atomic pointer swaps and message passing. This enables modular concurrency at each layer and ensures real-time synchronization across multiple optimization levels. Making each metaheuristic expose solutions to each other in real-time providing each other with high quality parameters. Figure 9 shows one such possible setup.

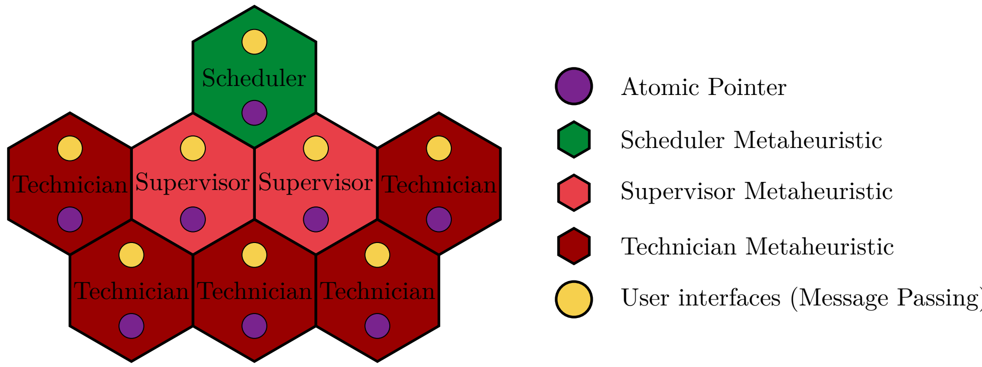

Fig 9: A software architecture tailored to metaheuristics (green and red) where solutions are shared between AbLNS algorithms in real-time through the use of atomic pointer swaps (purple) in line 17 of algorithm 1. The messages $m_n(\tau)$ where $n \in 1, 2, 3, 4$ are inputted through userinterfaces (yellow) in line 7 of algorithm 1

Scaling metaheuristics is very difficult as metaheuristics are usually highly coupled software components and this high coupling is exactly what enable them to optimize a large system of equally coupled and nested decisions. The main contribution of future research will be to provide a shared memory software pattern that can be used to scale metaheuristics given metaheuristics unique architectual requirements.

5. Conclusion

Many current planning problems that industry faces are combinatorial by nature, and many combinatorial problems have to be solved continuously to make operations run efficiently. For operation research (OR) to be helpful in this process, the solution methods should be a minimally invasive in the workflow of the working stakeholders. The AbLNS solution approach detailed in this paper aligns closely with two known problems in operation research: the lack of integration and the issues of coordination in multi-actor processes (Pinedo, 2022a), (Pinedo, 2022b). For these reasons we argue that the "standard" Operations Research approach of first collecting data, then creating a model and optimizing it, and then finally providing the solution to the planners in the company workflow, is not a scalable approach in many situations.

We have here demonstrated that the AbLNS approach works in a practical maintenance scheduling setting at Total Energies. We believe that this approach of combining a number of smaller planning/optimization problems with different actors/stakeholders responsible for their part of the overall solution is the future way of integrating Operation Research techniques in practice.

Modern industrial companies also have the available IT-infrastructure to support and connect model/metaheuristics together with the relevant actors/stakeholders in a way that was difficult just 10 years ago.

The fundamental problem with the existing paradigm is that optimizing across actors/stakeholders is very difficult, leading the literature to prefer integrated models instead of decomposing the model by each of the subprocesses that make up a business process such as maintenance scheduling. This paper argues that this is mainly due to an dated understanding of the software architecture that is available today in industry, but not acknowledged by broader the Operations Research and Metaheuristic communities (Talbi, 2009), (Gendreau and Potvin, 2019), (Meignan et al., 2015a).

References

, 2019. Handbook of Metaheuristics, Springer International Publishing, Cham. volume 272 of *International Series in Operations Research & Management Science*. chapter 4, pp. 99–127. doi:10.1007/978-3-319-91086-4.

Barthélemy, J.P., Bisdorff, R., Coppin, G., 2002. Human centered processes and decision support systems. European Journal of Operational Research 136, 233–252.

Ben-Tal, A., Nemirovski, A., El Ghaoui, L., 2009. Robust optimization .

Birge, J.R., Louveaux, F., 2011. Introduction to stochastic programming. Springer Science & Business Media.

Deming, W.E., 1982. Out of the Crisis. https://mitpress.mit.edu/9780262541152/out-of-the-crisis/.

Ehrgott, M., Gandibleux, X., 2002. Multiple criteria optimization: state of the art annotated bibliographic surveys .

Garey, M.R., Johnson, D.S., 1979. Computers and intractability. volume 174. freeman San Francisco.

Gendreau, M., Potvin, J.Y. (Eds.), 2019. Handbook of Metaheuristics. volume 272 of *International Series in Operations Research & Management Science*. Springer International Publishing, Cham. doi:10.1007/978-3-319-91086-4.

Herlihy, M., Shavit, N., Luchangco, V., Spear, M., 2020. The art of multiprocessor programming. Newnes.

Meignan, D., Knust, S., Frayret, J.M., Pesant, G., Gaud, N., 2015a. A Review and Taxonomy of Interactive Optimization Methods in Operations Research. ACM Transactions on Interactive Intelligent Systems 5, 1–43. doi:10.1145/2808234.

Meignan, D., Knust, S., Frayret, J.M., Pesant, G., Gaud, N., 2015b. A Review and Taxonomy of Interactive Optimization Methods in Operations Research. ACM Transactions on Interactive Intelligent Systems 5, 1–43. URL: https://dl.acm.org/doi/10.1145/2808234, doi:10.1145/2808234.

Palmer, R.D., 2019a. Maintenance Planning and Scheduling Handbook, 4th Edition . 4th edition ed., McGraw Hill.

Palmer, R.D., 2019b. Maintenance Planning and Scheduling Handbook, 4th Edition , 4th edition ed.. McGraw Hill. chapter 6, pp. 255–288.

Pareto, V., 1897. The new theories of economics. The Journal of Political Economy 5, 485–502.

Pinedo, M.L., 2022a. Scheduling: Theory, Algorithms, and Systems, Springer International Publishing, Cham. chapter 17, pp. 467–493. doi:10.1007/978-3-031-05921-6.

Pinedo, M.L., 2022b. Scheduling: Theory, Algorithms, and Systems, Springer International Publishing, Cham. chapter 18, pp. 494–518. doi:10.1007/978-3-031-05921-6.

Pisinger, D., Ropke, S., 2007. A general heuristic for vehicle routing problems. Computers & operations research 34, 2403–2435.

do Prado Marques, F., Arenales, M.N., 2007. The constrained compartmentalised knapsack problem. Computers & operations research 34, 2109–2129.

Richards, M., Ford, N., 2020. Fundamentals of Software Architecture: An Engineering Approach. "O'Reilly Media, Inc.". Google-Books-ID: xa7MDwAAQBAJ.

Scipo, 2024. Ordinator api. `https://github.com/scipo-code/ordinator.api`. Version 0.2.1.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems, in: International conference on principles and practice of constraint programming, Springer. pp. 417–431.

Talbi, E.G., . Metaheuristics: from design to implementation, John Wiley & Sons. chapter 6, pp. 460–534. OCLC: ocn230183356.

Talbi, E.G., 2009. Metaheuristics: From Design to Implementation. John Wiley & Sons, Hoboken, N.J.

The Rust Programming Language, . The rust programming language. URL: `https://www.rust-lang.org/`.

Total Energies, 2024. `https://totalenergies.com`.

Yang, S., Jiang, Y., Nguyen, T.T., 2013. Metaheuristics for dynamic combinatorial optimization problems. IMA Journal of Management Mathematics 24, 451–480. doi:`10.1093/imaman/dps021`.

Zadeh, L.A., 1988. Fuzzy logic. Computer 21, 83–93.