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-in-loop 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. This paper proposes a novel actor-based optimization framework using a large neighborhood search metaheuristic to address dynamic scheduling problems characterized by real-time data updates, external inputs influencing optimization, and interdependent decisions by multiple actors with differing objectives. The framework is tested on real-world maintenance scheduling for offshore platforms operated by Total Energies in the North Sea and demonstrates general applicability to a wide range of 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 as they are usually modelled as resource-constrained project scheduling problems, knapsack formulations, machine scheduling problems, etc. (NP-hard (Garey and Johnson, 1979)). Furthermore, for optimization to be useful in dynamic and uncertain environments where maintenance scheduling is performed it requires tight integration into existing IT infrastructure to allow the tacit knowledge of decision makers to easily influence the planning process. Often a number of different decision makers at different company levels take part in the planning process. In this way the industry usually assigns responsibility for decision-making to an individual representing only a small part of the complete 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, 2019).

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 (2019) 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 also detailed in (Palmer, 2019). The solution method is based on the large neighborhood search (LNS Shaw (1998)) metaheuristic and is in this paper describes 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 (Gendreau and Potvin, 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 in sequential pattern. 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 idea of ownership of a work order is crucial to understand the necessity of actor-based approaches. Throughout the scheduling process a work order is always owned by a specific actor and he alone is allow to modify/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 manifest themselves during planning or execution, work

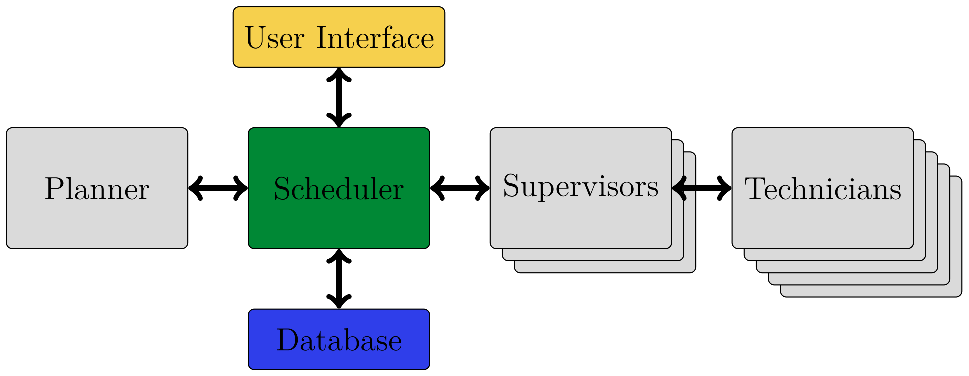

Fig 1: Simple overview of the scheduling process with its primary types of actors. The planner, the scheduler, the supervisor(s), and the technicians. The green color highlights the scheduler as it the actor in the maintenance scheduling process that is the foundation for the paper.

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 main contribution of this article is to provide a modular scalable optimization component, based on a proven metaheuristic to support a business process composed from smaller mathematical models in a framework, instead of a larger integrated mathematical model.

The paper is divided into four different sections. Section 2 explains the weekly maintenance scheduling model in detail and forms the foundation of the paper. Section 3 shows the results that come from the metaheuristic implemented, where it will be affected by the simulated user interaction. Section 4 will discuss the implications of the research and estabslish a future research direction. All source code for the implemented system can be found at (Scipo, 2024), instance data is 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). Here the 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 us to be precise in the timing on the messages that are send to the Ab-LNS 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 ($strategic\_penalty$) 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 $strategic\_value_{wp}(\tau)$. The $strategic\_value_{wp}(\tau)$ also captures the tardiness of the individual work orders (w), meaning that the value gained from scheduling work orders late is decreasing. Urgent tasks have decreasing 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:

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

Subject to:

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

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

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

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

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

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

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.

The objective function (3), maximizes the total weighted assignment of all work order subtractied by the penalty *strategic_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 putting two work orders in the same period, if they have share similarity like close proximity, same piece 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. $strategic\_penalty$ is the amount of exceeded capacity that is needed for the current assignment of work order 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 $strategic\_penalty$ respectively. The effects of changing $include_{wp}(\tau)$, $exclude_{wp}(\tau)$, $resource_{pr}(\tau)$, and $strategic\_value_{wp}(\tau)$ in real-time will be examined in the results 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 Gendreau and Potvin (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 Destructors

LNS in its most basic form has one constructor and one destructor which repeatedly destroy and rebuild the solution. For the AbLNS we will generalize on this concept by including messages as destructors. 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 constructors and destructors are also chosen from sources external to the algorithm.

Extending on the classic setup we define the following set of destructors, $m \in M$.

- m_1: Inclusion destruct message
- m_2: Exclusion destruct message
- m_3: Capacities destruct message
- m_4: Weights destruct message
- m_5: Random destruct message

Each of these messages affect different parts of the weekly maintenance scheduling problem. Notice here that the first four messages destruct the solution by changing the parameter space and the last message is a random destructor. That destroys the solution space.

For a correct implementation it is critical that the that destruct messages are handled one by one, and are not allowed to simply write to the solution and problem instance whenever they appear. Generalizing the destructors 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:     $X^t = clone(X)$
7:     while $Q.has\_message()$ do
8:         $P.update(S, m)$
9:         $X^t.destruct(S, m)$
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:    $Q.push(m)$
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 but also the problem instance P 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 itself. 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 avoid the time consuming initial convergence as the algorithm will be found in an optimal 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 pushed to the persistence on line 15

- That the optimization occurs in the repair function on line 11, which inserts operations in a greedy fashion.
- The pointer swapping that is performed on line 17 when a new best solution is found instantaneously makes the solution available to other actors. The pointer swapping is crucial as it allows non-hierarchical model setups.

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

2.2. Atomic Pointer Swapping and Software Architecture

There is an novel point in using atomic pointer swapping for when the AbLNS finds a new best solution. In a usual LNS setup the best solution is cloned and there after the search continues. But to make the solution available to other components of the larger system this best solution can be made available by the use of atomic pointer swapping. The reason that it is called atomic is that the pointers that are being swapped are being pointed to from different threads. Keeping the pointer swap atomic means that the pointed to solution can be swapped without causing dataraces. To understand the importance of this approach it is required to understand some fundamentals of software architecture.

Usually in software architecture you achieve vertical scaling (meaning making the dataflow process longer) and horizontal scaling (increasing throughput of the same dataflow process) by the use of message passing. This approach is difficult to apply when you want to scale metaheuristics as the metaheuristics function by mutating state (changing solutions) quickly and finding improvements.

Making modular components using message passing causes the system to generate a lot of messages whose behavior on the system is hard to predict especially if the message topology between metaheuristics is bidirectional.

The limitations of the are well-known with both (Talbi, 2009) in the section on parallel metaheuristics, but also from multi-agent metaheuristic literature where the results from making optimization system based on message passing between optimizing components generally do not show great results. Few of these systems have been successfully implemented in practice, with there being a general tendency that the metaheuristics flood the system with messages. The adverse effects of this is further exacerbated by three causes

- The randomized nature of metaheuristics makes massage passing highly stochastic
- If you want to avoid hierarchical setup with unidirectional message flow you very easily generate cycles
- The effects of adding additional messages to a system can be very difficult to understand

Using atomic pointer swapping as the interface between metaheuristics removes state duplication, avoids the pitfalls and complexity of message passing, and enables a way to modularize metaheuristics in a way that is not possible with general modern software architecture principles (Richards and Ford).

2.3. Atomic Pointer Swapping and Software Architecture

There is an novel point in using atomic pointer swapping for when the AbLNS finds a new best solution. In a usual LNS setup the best solution is cloned and there after the search continues. But to make the solution available to other components of the larger system this best solution can be made available by the use of atomic pointer swapping. The reason that it is called atomic is that the pointers that are being swapped are being pointed to from different threads. Keeping the pointer swap atomic means that the pointed to solution can be swapped without causing dataraces. To understand the importance of this approach it is required to understand some fundamentals of software architecture.

Usually in software architecture you achieve vertical scaling (meaning making the dataflow process longer) and horizontal scaling (increasing throughput of the same dataflow process) by the use of message passing. This approach is difficult to apply when you want to scale metaheuristics as the metaheuristics function by mutating state (changing solutions) quickly and finding improvements.

Making modular components using message passing causes the system to generate a lot of messages whose behavior on the system is hard to predict especially if the message topology between metaheuristics is bidirectional.

The limitations of the are well-known with both (Talbi, 2009) in the section on parallel metaheuristics, but also from multi-agent metaheuristic literature where the results from making optimization system based on message passing between optimizing components generally do not show great results. Few of these systems have been successfully implemented in practice, with there being a general tendency that the metaheuristics flood the system with messages. The adverse effects of this is further exacerbated by three causes

- The randomized nature of metaheuristics makes massage passing highly stochastic
- If you want to avoid hierarchical setup with unidirectional message flow you very easily generate cycles
- The effects of adding additional messages to a system can be very difficult to understand

Using atomic pointer swapping as the interface between metaheuristics removes state duplication, avoids the pitfalls and complexity of message passing, and enables a way to modularize metaheuristics in a way that is not possible with general modern software architecture principles (Richards and Ford).

2.4. Atomic Pointer Swapping and Software Architecture

Atomic pointer swapping offers a novel approach for managing the best solution found by the Actor-based Large Neighborhood Search (AbLNS) algorithm. Traditionally, in a metaheuristics such as Large Neighborhood Search (LNS) setup, the best solution is cloned before the search continues. However, atomic pointer swapping enables the best solution to be shared across system components without introducing data races. This technique is essential in multithreaded environments, as it ensures the integrity of the solution during swaps. 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. The limitations of message-passing systems in metaheuristics are well-documented, particularly in the context of parallel and multi-agent metaheuristics (Talbi, 2009). These systems often suffer from excessive message generation, leading to poor scalability and performance. Key issues include:

- The stochastic nature of metaheuristics makes message-passing behavior unpredictable.
- Avoiding hierarchical, unidirectional message flows often results in cycles.
- The impact of increased messaging is difficult to analyze and manage.

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).

3. Results

To test the AbLNS algorithm in a framework setup a number of tests, where the data are disrupted during the AbLNS algorithm is tested in this section. The data and company setup 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 3.2 and excluding them from periods is shown in Section 3.3. Afterwards, the effect of reducing the period resource capacities $resource_{pr}(\tau)$ is tested in Section 3.5 and increasing them is tested in Section 3.4. Finally the effect of changing the work order values $strategic\_value_{wp}(\tau)$, is tested in Section 3.6.

3.1. Data Instance

This data instance comes from Total Energies , where data is extracted from their SAP ERP system. The data instance used in this paper is from a specific offshore platform that for the 2 year time horizon contains 3487 outstanding work orders ($W(\tau)$), have 16 different resource skill sets ($R(\tau)$) (e.g. mechanics, electricians, turbine specialists, etc.), and finally the scheduling horizon stretches over a 52 bi-weekly periods ($P(\tau)$) meaning approximately 2 years.

	$\|W(\tau)\|$	$\|R(\tau)\|$	$\|P(\tau)\|$
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. 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 points in time 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 perturbations of including work orders into the weekly scheduling that the AbLNS is 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.

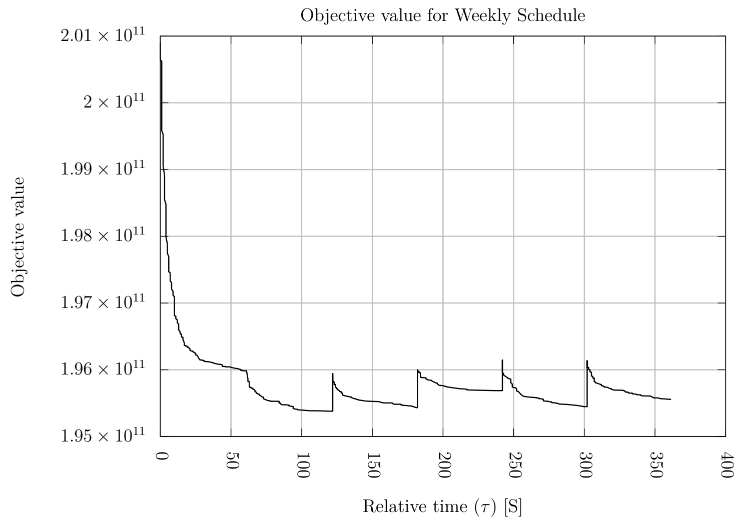

Fig 2: Effects of perturbing the solution by forcing work orders into specific periods.

Figure 2 show 5 perturbations with the first at time = 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 $strategic\_value_{wp}(\tau)$ that has a lower value. The remaining 4 perturbations all show the same pattern, an increase in the objective value followed by a subsequent convergence.

3.3. 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.

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

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

Figure 3 show a substantial spike in the objective value after a perturbation as given in table 3.

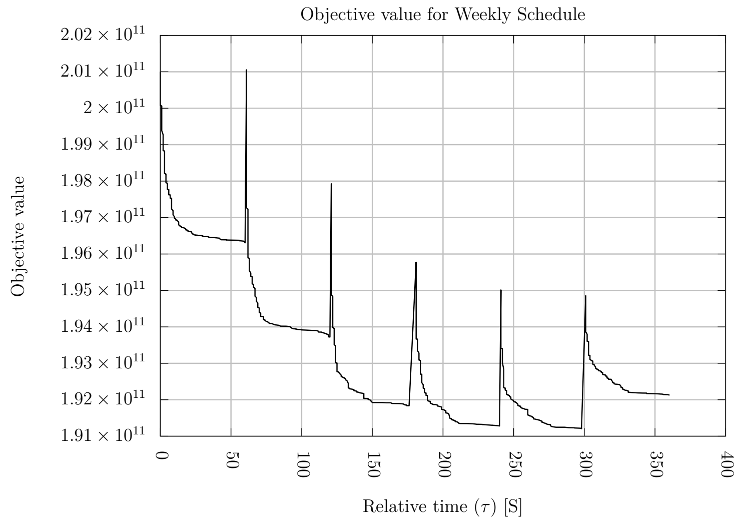

Fig 3: At each 60 second interval the objective value experiences a large spike as the exclusion of work orders make the current best assignment of work orders infeasible

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 $resource_{pr}(\tau)$.

3.4. Response to Additional Weekly Capacity

Table 4 details the perturbations that the AbLNS will be subject to during its 360 second execution. Perturbing the resources $resource_{pr}(\tau)$ 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.

	$\tau_1 = 100$	$\tau_2 = 200$	$\tau_3 = 300$
$\Delta\|P(\tau)\|$	52	52	52
$\Delta\|R(\tau)\|$	16	16	16
$\|resource_{pr}(\tau\|)$ (hours)	61816	111268	173083

Table 4:

Figure 4 shows the effects of progressively increasing available resources. The AbLNS starts with a base load which is then reduced at time = 100 seconds causing the objective value by increase as *strategic_penalty* in equation 4 absorbs the exceeded capacity. At time = 200 the resources are increased to the base load again, and at time = 300 seconds the resources are increased to their maximum value.

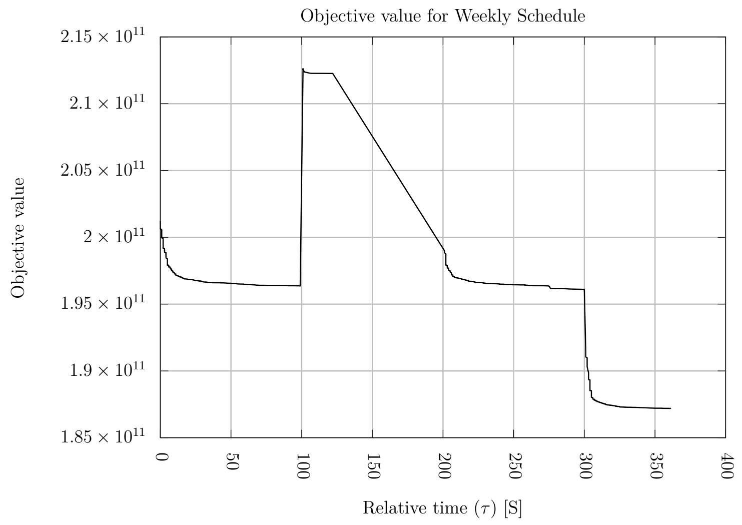

Fig 4: Starting from a base load of 111268 hours (the same amount of hours as in the second perturbation). The resources are first reduced causing a spike in the objective value. Then at each 100 second interval resources are added and the objective value decreases and the AbLNS optimize around the perturbation

3.5. Response to Reduced Weekly Capacity

Table 3.5 details the perturbations that the AbLNS will affected by. Starting from a base load of the amount of available resources are progressively decreased.

	$\tau_1 = 100$	$\tau_2 = 200$	$\tau_3 = 300$
$\Delta\|P(\tau)\|$	52	52	52
$\Delta\|R(\tau)\|$	16	16	16
$\|resource_{pr}(\tau\|)$ (hours)	173083	111268	61816

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

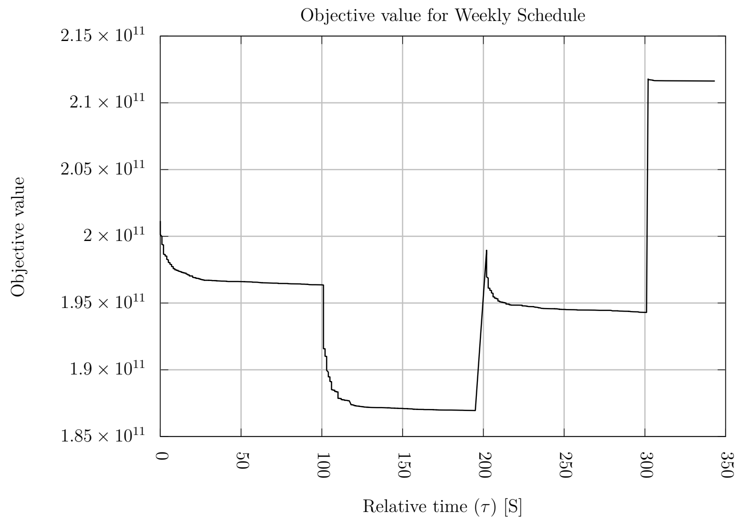

Fig 5: Starting from a baseload of 111628 hours of total hours the AbLNS is progressively affected by decreasing levels of resources. At the first 100 seconds the objective value decreases in response to the perturbation as resources are added, after which the AbLNS is pertubed by two decreases in resources causing an initial spike in the objective value followed by the AbLNS optimizing around the perturbation

3.6. Response to Changes in Work Order Values

The final parameter that will be changed is the work order value $strategic\_value_{wp}(\tau)$. Table 3.6 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 3.6.

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

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

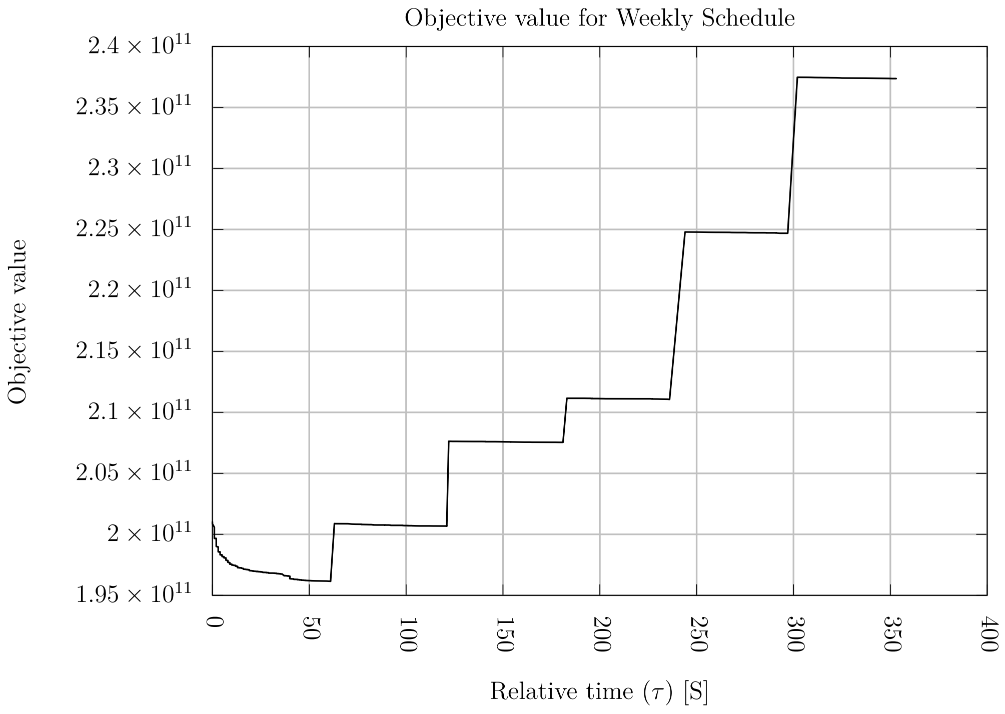

Fig 6: The effects of perturbing the AbLNS by dynamically changing the *strategic_value*$_{wp}(\tau)$ in the objective function. The objective value is increasing in response to the higher cost associated with a late scheduling, after which it optimizes around the perturbation

4. Discussion

The maintenance scheduling process effectively solves a complex scheduling problem by relying in 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. Each stakeholder in the process has his own model each with different levels of aggregation, where each actor understands how to exploit his own model. The discussion will be divided into three sections: 4.1 actors and integration; 4.2 continuous optimization allows asynchronous optimization; and 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) but a practical problem of connecting data streams so that the solution approach continually receives dynamic data to handle changes and then providing the resulting solutions to the relevant actors (stakeholders) 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.

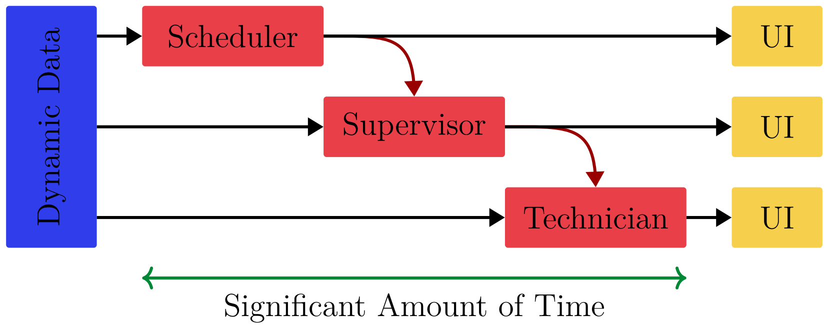

Fig 7: Effects us using hierarchical models setups in human-guided search metaheuristics. Due to the dependent nature of each metaheuristic it becomes crucial that the running of the metaheuristics are well coordinated between the metaheuristics.

In practice there are multiple problems with using a hierarchical setup. Usually the biggest one is that the information and knowledge needed to perform 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.

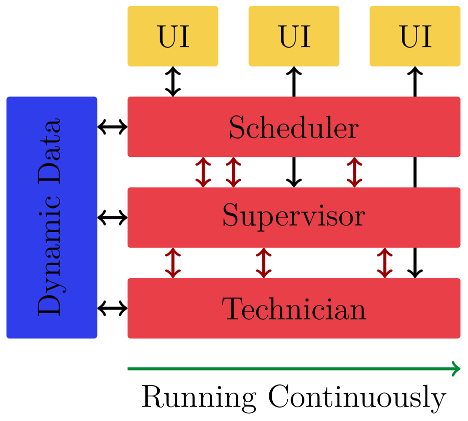

Fig 8: Asynchronous model setup where each metaheuristic runs in perpetuity. In this setup there is no need to coordinate stakeholders to run the metaheuristics. Each actor in the scheduling process will always have the solutions of the other stakeholder available to him to guide his own search.

When the optimization approache optimize continuously it becomes possible to enable 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 future research of this project is to demonstrate that the actor-based approach described here can be used to model and optimize multi-actor scheduling processes. Figure 9 shows a scheduling system architecture where each of the actors run an actor-based metaheuristic and that each metaheuristic will share its solutions with the other metaheuristics through atomic pointer swapping. Communicate with the end-user through message passing, integrate with a persistence storage through mutex locks, and the lifecycle of each of the metaheuristics will be controlled by the orchestrator through message passing.

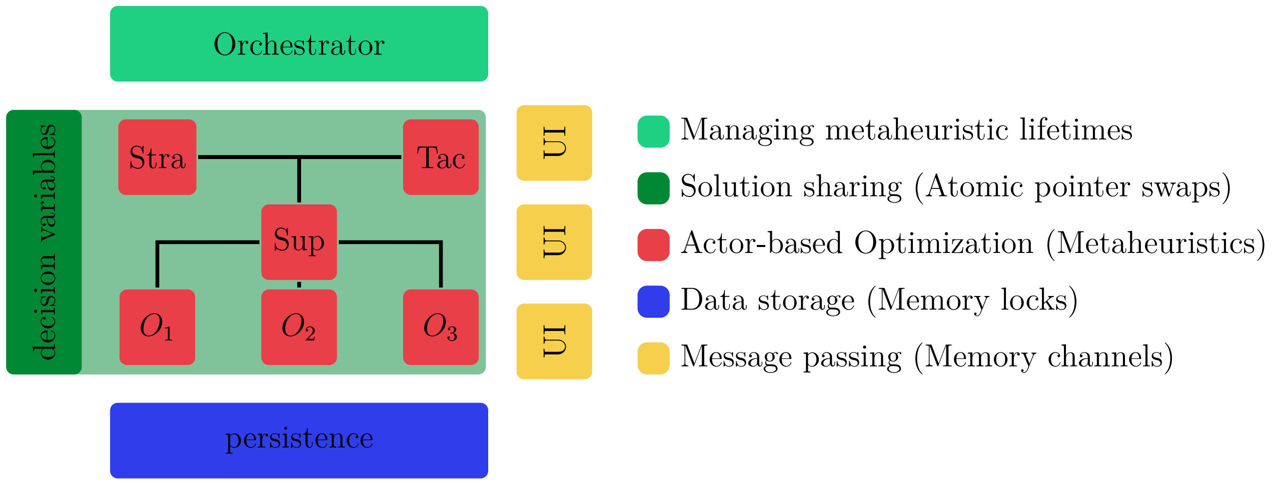
Orchestrator
decision variables
Stra
Tac
Sup
O_1
O_2
O_3
UI
UI
UI
persistence
Managing metaheuristic lifetimes
Solution sharing (Atomic pointer swaps)
Actor-based Optimization (Metaheuristics)
Data storage (Memory locks)
Message passing (Memory channels)

Fig 9: Overview of the scheduling process when modelled as actors. When LNS is encapsulated is an actor it becomes possible to optimize parts of a large process individually instead of optimizing the scheduling problem globally from a single model implementation.

The next step in this research direction will be to model the remaining stakeholders as their own AbLNS metaheuristics, and then make them communicate together. Each exposing solutions to each other in real-time providing each other with high quality parameters.

References

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., . 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.

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., 2019. Maintenance Planning and Scheduling Handbook, 4th Edition . 4th edition ed., McGraw Hill.

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

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

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.