Dynamic Task Allocation Algorithms within Intelligent Operator Support Concepts for Shore Control Centres

Tycho Brug, TNO, Soesterberg/The Netherlands, tycho.brug@tno.nl
Jasper van der Waa, TNO, Soesterberg/The Netherlands, jasper.vanderwaa@tno.nl
Valentina Maccatrozzo, TNO, Soesterberg/The Netherlands, valentina.maccatrozzo@tno.nl
Hans van den Broek, TNO, Soesterberg/The Netherlands, hans.vandenbroek@tno.nl

Abstract

This paper presents a dynamic task allocation algorithm. Its function is to dynamically allocate the supervision of tasks of many autonomous operations to suited operators. The proposed algorithm accounts for the workload capacity of an operator and a balanced work distribution over all operators. We propose an iterative deepening depth-first search algorithm to find the optimal allocation. Through a series of simulated experiments, we show how our proposed algorithm results in a versatile method that can distribute a series of complex operations to operators effectively, both in terms of computational effectiveness as well as workload distribution.

1. Introduction

This paper presents a dynamic task allocation algorithm within an Intelligent Operator Support System for Shore Control Centres in the context of the MOSES EU-project, https://moses-h2020.eu/. The MOSES project is one of three research and innovation projects within the Horizon 2020 program that contributes to more automation and autonomy in Europe's short sea logistics. This project illustrates and contributes to the rapidly growing application of Artificial Intelligent and Autonomous Systems in the maritime industry. This paper addresses one of MOSES' innovations: that of an Intelligent Operator Support System (IOSS).

The growing capability of autonomous systems and their application results in a paradigm shift from manual labour towards (remote) supervisory control. With supervisory control an operator is responsible for an autonomous operation (e.g., autonomous sailing, autonomous loading/unloading of containers, etc.). IOSS is a software component for Shore Control Centres (SCC) housing such operators. The goal of this software is to offer support to each operator by offering several vital functions such as aiding in situational awareness, assessing future risks and allocating operations to the right operator.

We envision a SCC capable of ensuring the safe and efficient completion of a large number of autonomous operations with relatively few operators. In this SCC a handful of operators should be capable of supervising several hundreds of such operations in a single shift. The assignment of the operator to the right operation can be done manually. However, this would mean strenuous, error prone, work, which additionally might be infeasible as the number of operators and operations grows. To illustrate this point further, *Van den Broek et al. (2020)* stated that "The way forward to mitigate work overload is (…) to establish an adaptive workload balancing approach among several Shore Control operators to deal with workload fluctuations in a dynamic way (p.6)". IOSS aims to offer this functionality; to dynamically allocate the required supervision task of an autonomous operation to the operator suited for that task. We refer to this function to as Dynamic Task Allocation.

The dynamic task allocation algorithm should take in account several factors that influence the workload of the individual operators and a balanced allocation of the operations over the operators. To achieve this optimization of allocation of work, a measure on quality of the allocation should be developed. Next to that, the amount of possible ways to allocate work over operators grows exponentially with more operations and operators. Calculating all possible assignments (brute-force) gets impossible very quickly. Thus, it should be considered how to optimize the task allocation among all possible options within reasonable time. For this we utilized an easily interpretable and scalable algorithm; Iterative Deepening Depth First Search (IDDFS), *Korf (1985)*. We adapted this algorithm for task allocation

so that it can dynamically re-assign operations when (a) new operations requiring supervision become available, (b) the situation changes significantly requiring an operator with a different expertise, or (c) the operator does not agree with a certain task assignment.

In this work we show how our proposed dynamic task distribution algorithm results in a versatile method that can distribute a series of complex operations to operators effectively, both in terms of computational effectiveness as well as workload distribution. We do this through a series of simulated experiments. In addition, we discuss the limitations of our algorithm as well as potential topics of future research.

Our paper describes in more detail:

- The proposed algorithm and the associated cost function that is being optimized.
- The results of several experiments to show the workings, effectiveness, stability and constraints of the algorithm.
- An in-depth discussion on the usefulness of this approach, its limitations and future research.

2. Methods

2.1. General

We base the allocation of operations to operators on the expected workload. As operators should not be overloaded with too much work or have too little work, *Neerincx (2003)*. In either case the quality of the "work" reduces. Since this work entails the supervision of autonomous operations and potential interventions, a reduced work quality poses a risk for the efficiency and safety of the operation. Below we describe how we model this workload as a function of operator expertise and operation difficulty. We also present a straightforward additive model for supervising simultaneous operations. This workload model enables the allocation of operations of various complexities while balancing the workload over all operators accounting for their expertise.

We first assume that any operation o is static in time. This means each operation has a given start time $t_{o,start}$, and a given duration d_o, which cannot be shifted in time. Each operation is composed of multiple tasks, with each a default workload $w_o(t)$, expressed as a baseline percentage of capacity for an operator. Tasks are linked to each other and sequential in a given operation. Therefore the operations are allocated, and not the single tasks. This is because switching between multiple contexts is distracting for an operator and will thus be less effective or safe. Tasks within operations might be: leaving/approaching berth, port departure/approach, sea passage and container handling. This results in that the workload per operation varies over time. For example, the first task could be "port approach" in the first half hour of the operation, requiring 50% of capacity of an operator to supervise. A second task "container handling" would take the next hour of the operation, requiring 100% capacity of an operator.

Each operation is assigned a difficulty s_o. The difficulty is compared to the expertise e_p of an operator p, to get to the actual workload $w_{o,p}(t)$ for that operation and operator at that time:

$$w_{o,p}(t) = \begin{cases} w_o(t), & e_p \geq s_o \\ w_o(t) \cdot \left(\frac{s_o}{e_p}\right), & e_p < s_o \end{cases}$$

This results in the adjusted workload $w_{o,p}(t)$ that increases linearly if operator expertise is lower than operation difficulty, and does so proportionally to the ratio between the two. It remains the default workload if the expertise is equal or higher than the difficulty of the operation. This models that a less experienced operator requires additional capacity to supervise a difficult operation. For instance, a complex operation due to environmental conditions causes a high workload for a junior operator whereas the same operation requires nothing extra from a senior operator.

We initially assume an arbitrary allocation of operations over operators. In any given allocation, each assigned operation takes some of the available capacity of an operator at that time. The required capacity is equivalent to the workload it takes for that operator to supervise that operation as described above. We assume that an operator's workload at any given time t is equal to the sum of all the workload associated to each operator that operator supervises at that time, as adjusted based on the operator's expertise and operation's difficulty. Imagine operations 1, 2 and 3 are assigned to operator 1. For that given operator 1, the workload becomes:

$$w_{p=1}(t) = w_{o=1,p=1}(t) + w_{o=2,p=1}(t) + w_{o=3,p=1}(t)$$

The mean workload of that operator then becomes:

$$\overline{w}_p = \frac{1}{N}\sum_{t=1}^{N}\sum_{o\in O} w_{o,p}(t)$$

Where N is an arbitrary finite amount of discrete time steps used in a given time window. For example N could represent an eight-hour shift and t a fifteen-minute interval during that shift. O represents the operations assign to operator p.

2.2. Cost function

To determine the "goodness" of any given allocation, we consider:

- Capacity of an operator: The full (unused) capacity of the operator is set to 100%. If $w_p(t)$ is larger than the capacity (at any point in time), the work distribution is unfeasible and should not be considered.
- Fair work distribution over all operators: For this we consider the following cost attributes:
    - Average mean workload: $\overline{w} = \frac{1}{M}\sum_{p=1}^{M}\overline{w}_p$
      Where M is the number of operators. The distribution of work where the average mean workload is lowest, is the distribution where all operations are supervised by operators with an expertise level equal to or higher than the difficulty of the operations.
    - Standard deviation of mean workload: $\sigma_w = \sqrt{\frac{1}{M}\sum_{p=1}^{M}(\overline{w}_p - \overline{w})}$.
      The standard deviation of mean workload increases when a small number of operators get assigned relatively more work than others. Optimizing towards a low σ_w will thus result in an allocation where each operator has a similar mean workload.
    - Average time in a critical workload zone: $\overline{c}_w = \frac{1}{MN}\sum_{p=1}^{M}\sum_{t=1}^{N} w_p(t) > 0.85$.
      We define the critical workload zone as a point in time where a given operator works above 85% of its capacity; $w_p(t) > 0.85$. By minimizing the average amount of time the operators work above this threshold, we aim to overcome cognitive overload for operators, *Neerincx et al. (2003).*
    - Ability for the operator to take a break: $\overline{b}_w$.
      We set $\overline{b}_w$ to be the average amount of hours that operators work for more than 4 hours straight without a break (i.e., a period of at least one hour where $w_p(t) = 0$. By optimizing $\overline{b}_w$, we penalize allocations where operators need to work for a long time without getting a break.

We want to find the work allocation where $w_p(t)$ is never higher than 100% (for all operators and all points in time) and where the cost C is lowest:

$$C = \alpha\overline{w} + \beta\sigma_w + \gamma\overline{c}_w + \delta\overline{\mathrm{b}}_\mathrm{w}$$

Here α, β, γ and δ are weights to give priority to the above stated constraints. This additive cost function takes several notions of workload into account and can easily be extended to include more.

2.2. Optimization

For cases with a low number of operators and a low number of operations, it is possible to calculate all possible allocations and find the absolute minimum (brute-force method). However, with an increasing amount of operations and operators, the amount of possible allocations grows exponentially and quickly becomes unfeasible. We therefore implemented an Iterative deepening depth-first search (IDDFS) algorithm, *Korf (1985)*. This algorithm randomly orders the operations, and then assigns each operation in turn to the best fitting operator (using the cost function above to determine which). This continues until all operations are allocated, as shown in Fig.1**Error! Reference source not found.**. This optimization algorithm does have a risk on landing on a local minimum, as the order of operations does not necessarily give the best overall fit.

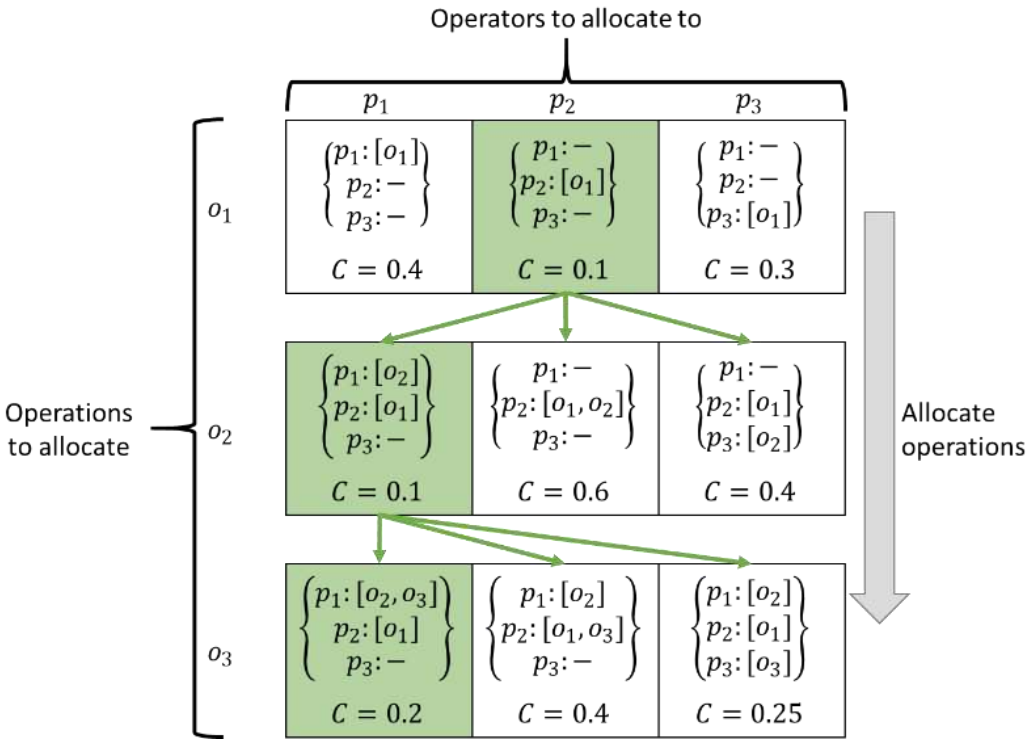

Fig.1: Illustration of IDDFS algorithm

In the example in **Error! Reference source not found.Error! Reference source not found.**, the optimization could for instance get to a lower cost if it first assigns operation 3, then 1 and finally 2, in comparison to the displayed order. Therefore, the IDDFS algorithm is run with several repetitions, each time with a different order of assigning operations. This results in an algorithm that is less prone to land in a local minimum, but still needs a significantly less calculations than the brute-force method to get to a result.

2.3. Dynamic allocation

New operations can come in and existing ones can be delayed or cancelled. Each time a potential better allocation might be possible. However, making changes to an already existing allocation is not always preferred. Operators might have already prepared for the given allocation (e.g., read documentation about a certain operation, scheduled their break, etc). Therefore it is preferred to get to a new optimal planning, while making as little changes compared to the previous planning. This is implemented by assigning an additional cost to any changes required. We denote this cost as $\varepsilon \times \Delta o$, where Δo is the

amount of operations that is shifted from one operator to another. This is multiplied by a weight ε. This is then added to the cost C' of the new allocation. By adding this additional cost, the algorithm tries to stay as close to the current plan as possible, as it will only swap operations between operators if it means a change in cost is acceptable. More formally; a change is only implemented if $|C - C'| > \varepsilon \times \Delta o$.

2.4. Implementation

The dynamic task allocation algorithm is implemented and simulated in Python 3.7, *Van Rossum and Drake (2009)*, and run using the PyCharm 2020.3.3 IDE, https://www.jetbrains.com/pycharm/. Experiments were run on a HP Elitebook 830 (running Microsoft Windows version 10) with an Intel Core i5 processor running at 1.60 GHz using 8.00 GB of RAM.

3. Results

3.1. Basic functionality

3.1.1. Scenario 1: Simple case

We start off with a simple case with six operations that need to be allocated (as depicted in Table I). hese six operations need to be allocated over three operators. These three operators have a respective expertise level e_p of 1, 2 and 3.

Table I: Simple case with 6 (non-overlapping) operations

o	$t_{o,start}$	s_o	Work profile
1	0:00	1	30% (0-1h) 60% (1-2h)
2	1:00	2	30% (0-1h) 60% (1-2h)
3	2:00	3	30% (0-1h) 60% (1-2h)
4	4:00	1	30% (0-1h) 60% (1-2h)
5	5:00	2	30% (0-1h) 60% (1-2h)
6	6:00	3	30% (0-1h) 60% (1-2h)

For the cost function, the following weights are set: $\alpha = 1$, $\beta = 1$, $\gamma = 1$, $\delta = 4$. As this simple case has a relatively low number of calculations needed to calculate all possible allocations (3^6=729 possibilities), we start off using only the brute-force method for optimization. The brute-force algorithm comes up with the following best allocation after calculating the costs for all possible allocations (C = 0.225, Fig.2): p_1: [o_1, o_4], p_2: [o_2, o_5], p_3: [o_3, o_6].

In this optimal plan:

a) None of the operators' maximum capacity is reached.
b) Each operator is assigned with the operations that suits their expertise level which minimizes the average mean workload ($\overline{w}$) over the operators.
c) The work is allocated over operators, resulting in two operations being assigned to each operator, which minimizes the standard deviation of mean workload (σ_w)
d) None of the operators work in their critical workload zone ($\overline{c}_w = 0$),
e) Each operator has enough opportunity to take breaks ($\overline{b}_w = 0$)

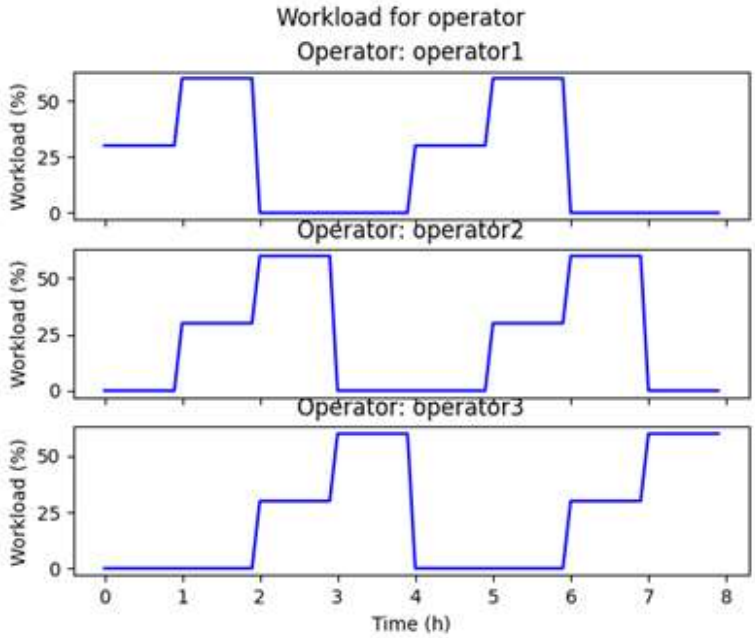

Fig.2: Simple case: workload per operator over time. Every increase (initially 30%, then 60%) signifies one allocated operation. In this specific case all operations are evenly distributed over operators and time.

3.1.2. Scenario 2: Extended simple case – difficulty increased

We now increase the difficulty of the first operation from one to two, i.e.; $s_o = 2$. The others remain at the same difficulty. The brute-force algorithm comes up with the following best allocation (C = 0.34, Fig.3): p_1: [o_4], p_2: [o_2, o_5], p_3: [o_1, o_3, o_6].

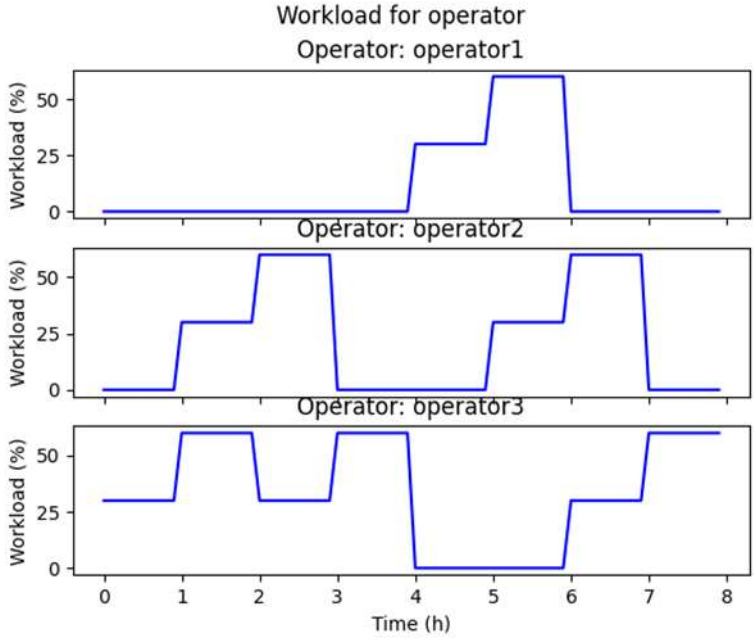

Fig.3: Simple case where operations vary in their difficulty. The more difficult operations are assigned to the third operator. This results in a reasonable workload for the third operator and the first operator having a fairly low overall workload.

The outcome might be counter-intuitive, as the third operator seems to get a substantial part of all the work. However, if the first operation would still be supervised by the first operator, the plan would

become unfeasible. As the difficulty is twice as high than the expertise of the first operator, it would take that operator to much capacity to perform ($60\% \times 2 = 120\%$). Both the second and third operator could perform the task, however, if the second operator does so, he/she would have to work in the critical workload zone (> 85%) as the first and second operations overlap partially. This would still result in a feasible plan, but with a higher cost ($C = 0.38$, an increase of 0.04).

3.1.3. Scenario 3: Extended simple case – overlapping operations

In this scenario we change the timing of the third operation to also start at $t_{o,start} = 6$ (similar to the sixth operation). The other operations remain the same as in the first scenario. The brute-force algorithm comes up with the following best allocation ($C = 0.36$, Fig.4): p_1: [o_1, o_4], p_2: [o_2, o_3], p_3: [o_5, o_6].

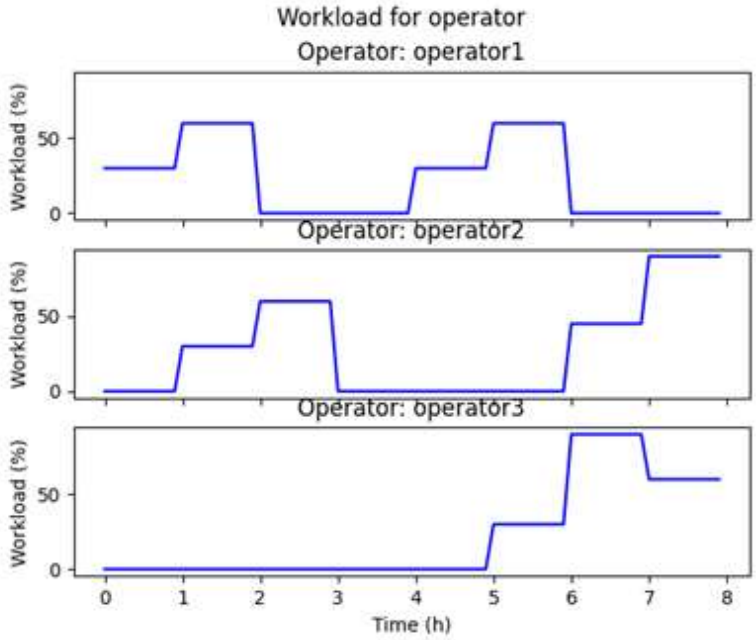

Fig.4: Simple case with overlapping operations, workload per operator over time. Due to the overlapping nature of the operations, the second operator takes on a difficult operation to prevent the third operator being overloaded.

It is unfeasible for the third operator to still supervise the third and sixth operation. They fully overlap, which requires more workload capacity then a single operator has available. The algorithm therefore finds a new optimal solution, in which the second operator takes over o_3 from the third operator, and the third operator takes up o_5 from the second operator. The plan has a higher cost than the outcome of scenario 1.

4. Scenario 4: Extended simple case – high difficulty

Next, let us increase the difficulty of all operations to $s_o = 3$. Again, the rest of the operations are kept the same as the first scenario. The brute-force algorithm comes up with the following best allocation ($C = 0.62$, Fig.5): p_1: -, p_2: [o_2, o_4], p_3: [o_1, o_3, o_5, o_6].

Due to all operations having a difficulty value of 3, the first operator is unable supervise any of the operations (as the peak would cross the capacity of this first operator by quite some margin). The second operator is able to take on operations, but suffers a penalty (1.5x) and is thus not able to handle overlap in operations. This requires the third operator to take on several additional operations.

An alternative plan would be for the second operator to take on o_6 from the third operator. This results in a slightly worse plan ($C = 0.66$). The reason for the increase in cost is twofold:

1. The average mean workload increases as this is lowest when operators supervise operations that fit their expertise level.
2. The standard deviation of mean workload increases. This is due to the fact that the difference in mean workload between the second operator and the third operator actually increases in this scenario, due to higher workload caused by the second operator when taking on a task that has a higher difficulty.

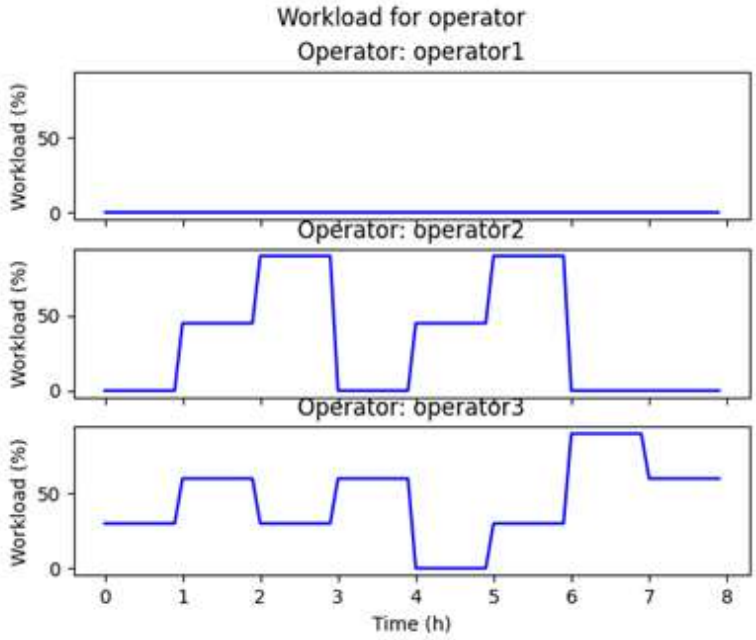

Fig.5: Simple case where all operations are difficult. Most operations are allocated to the third most experienced operator to reduce the average workload. The first operator is not able to take on any operations due to lack of experience.

An even more extreme scenario would be that the third operator takes on all operations. This would still result in a feasible plan, however with a very high cost ($C = 2.1$). Although the average mean workload decreases, this effect is overruled by the high standard deviation of mean workload and mostly because of the fact that the third operator will not be able to have a break (this is highly punished in the current cost function where $\delta = 4$).

3.1.5. Scenario 5: Extended simple case – adding an operation

As a final example, a new operation (o_7, see Table II) is added to the list. The operation has a difficulty value of 3 and a high workload. The brute-force algorithm is unable to find a feasible allocation (meaning none of all possible allocations is possible). The best, and fairly reasonable, it finds within this time is (see Fig.6); p_1: [o_1, o_4], p_2: [o_2, o_5, o_6], p_3: [o_3, o_7].

Table II: Additional operation

o	$t_{o,start}$	s_o	Work profile
7	6:00	3	50% (0-1h) 100% (1-2h)

Due to the high workload and difficulty of o_7, only the third operator can supervise this operation without crossing the capacity threshold. Additionally, the third operator is no longer able to pick up any operations that overlap o_7. This is why the second operator has to supervise o_6.

Because of the 1.5x penalty due the difficulty of o_6, and due to the overlap with o_5, the threshold for the second operator is crossed (105%). This makes even the best plan unfeasible

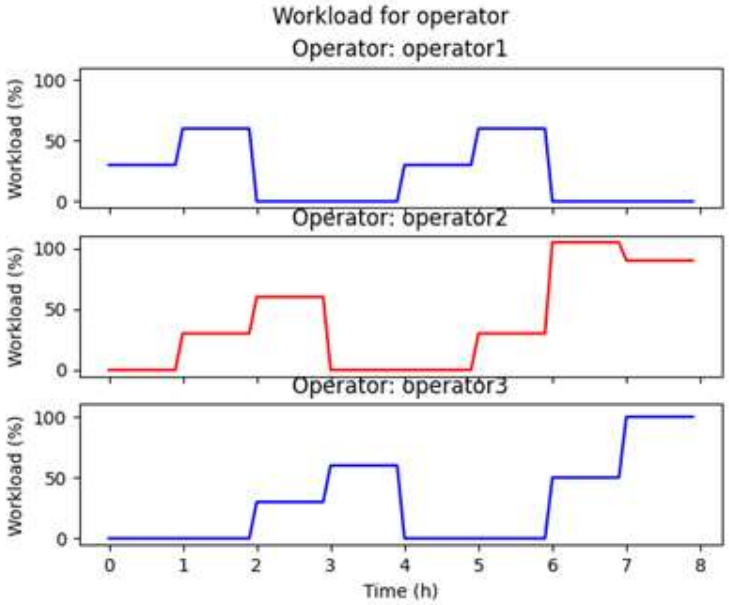

Fig.6: Simple case with an additional operation. In this scenario, no feasible allocation is possible. Even in the most optimal allocation, the second operator gets a too high workload (105%).

3.2. Moderate scenario

Next, we consider a more moderate scenario, in which three operators supervise twelve operations, Table III. The three operators have a respective expertise level e_p of 6, 3 and 2.5. Wherein the simple scenarios, it was still quite possible to get to the best allocation by hand, for this moderate scenario it already proves to be unfeasible to do so. The first step is to use the brute-force method for optimization. With three operators and twelve operations there are $3^{12} = 531.441$ possibilities. On a standard laptop (see 2.4 Implementation) this takes around 5-6 minutes to evaluate. This shows the limitation of the brute-force method. Due to the exponential nature of the number of possibilities, more operation or operators would make this type of optimization unfeasible. The brute-force algorithm finds the following optimal allocation ($C = 1.15$, Fig.7): p_1: [o_4, o_{10}, o_7, o_9], p_2: [o_{11}, o_8, o_{12}, o_2], p_3: [o_1, o_5, o_3, o_6].

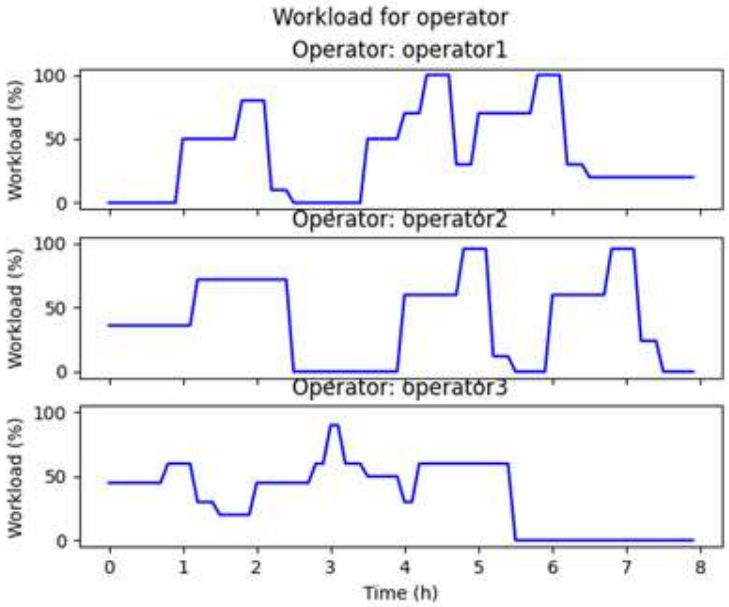

Fig.7: Optimal allocation for the moderate example scenario. In this allocation all operations are evenly distributed over operators and time.

Table III: Moderate scenario

o	$t_{o,start}$	s_o	Work profile
1	0:00	3	25% (0-0:45h) 40% (0:45-1:15h) 10% (1:15-1:30h)
2	0:00	3	30% (0 -1:15h) 60% (1:15-2:30h)
3	0:00	3	20% (0-4h)
4	1:00	3	50% (0-0:45h) 80% (0:45-1:15h) 10% (1:15-1:30h)
5	2:00	3	25% (0-0:45h) 40% (0:45-1:15h) 10% (1:15-1:30h)
6	3:00	2.5	30% (0 - 1:15h) 60% (1:15-2:30h)
7	3:30	2.5	50% (0-0:45h) 80% (0:45-1:15h) 10% (1:15-1:30h)
8	4:00	3	50% (0-0:45h) 80% (0:45-1:15h) 10% (1:15-1:30h)
9	4:00	6	20% (0-4h)
10	5:00	6	50% (0-0:45h) 80% (0:45-1:15h) 10% (1:15-1:30h)
11	6:00	3	25% (0-0:45h) 40% (0:45-1:15h) 10% (1:15-1:30h)
12	6:00	3	25% (0-0:45h) 40% (0:45-1:15h) 10% (1:15-1:30h)

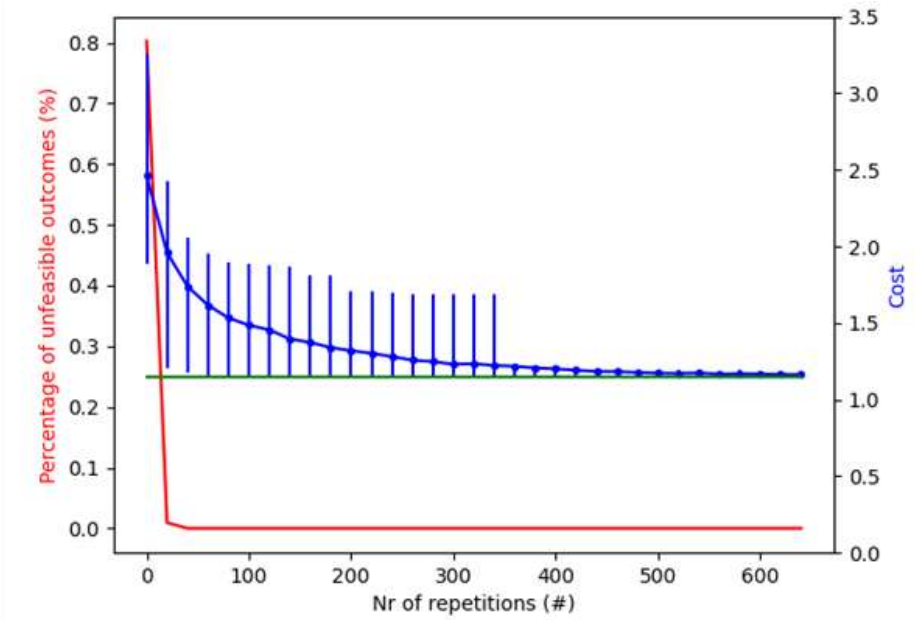

Fig.8: Cost outcome of IDDFS related to number of repetitions. Red: percentage of unfeasible outcomes related to number of repetitions. Blue: average cost of the found solution (10%-90% quartile with vertical lines). Green: global minimum (as found by brute-force method)

We can also use the IDDFS optimization algorithm to get to an approximation of the best allocation of work. For the IDDFS, the amount of repetitions is related to how close we can get to the optimal allocation at the cost of more computation time. The brute-force method will always find this optimal allocation, but only if all possible combinations are evaluated. With IDDFS we are guaranteed to find better solutions over time, whereas cutting off the brute-force method after an incomplete number of evaluations is too random to guarantee a serious outcome.

Fig.8 shows the relation between number of repetitions of the IDDFS and the average outcome (in blue). This was obtained by running the IDDFS 1500 times for each number of sampled repetitions, and analysing the results. We did so to get a reliable average on the IDDFS' performance for each sampled number of repetitions. With a low number of repetitions (< 20), the IDDFS is (in most cases) unable to find the optimal allocation, and has a high percentage of outcomes where the optimizer is not even able to find a feasible outcome at all (as indicated with the red line in Fig.8). Between 20-340 repetitions, the IDDFS method almost always finds a feasible solution, but often does not find the most optimal solution. With 340-560 repetitions, the IDDFS often finds the most optimal solution, and with >560 repetitions (badly visible in Fig.8), the algorithm is almost guaranteed to find the best possible solution.

In the moderate scenario, even with 560 repetitions, the IDDFS is still much more efficient than the brute-force algorithm. Where the brute-force algorithm requires an exponentially increasing number of cost evaluations (p^o, in this case 531.441 evaluations), the IDDFS requires $repetitions \times p \times o$, which in this specific case is $560 \times 3 \times 12 = 20.160$ evaluations. As both optimization algorithms themselves require an arbitrary amount of calculation power compared to the cost evaluations, the IDDFS with 560 repetitions is still much faster (about 10 seconds compared to 5-6 minutes on a standard laptop). Simply put, IDDFS performs in linear time ($O(n) = n$) whereas the brute-force method performs in exponential time ($O(n) = c^n$).

This shows the strength of using the IDDFS compared to the brute-force method of optimization. Both are able to find the global minimum, but the IDDFS method is able to get to the result much faster. Next to that, the IDDFS is scalable. Due to the exponential nature of the brute-force method, it becomes unusable in realistic settings. As the IDDFS performs in linear time, it scales quite well with more operators and operations in realistic future shore control centres. The disadvantage is that the amount or repetitions required to get to the global minimum reliably is case specific, which makes it a difficult parameter to determine without an analysis such as shown in Fig.8.

3.3. Complex scenario

Lastly, we look at a much more complex scenario, in which six operators supervise sixteen operations and two new operations come along after a first allocation has been computed, Table IV. We aim to illustrate the adaptive nature of our approach under complex and dynamic settings.

The six operators have an expertise level e_p of 6 (two highly experienced operators), 3 (three fairly experienced operators) and 2.5 (one least experienced operator). This complex scenario cannot be solved by a brute-force approach. This approach would take 2.8×10^{12} evaluations which would take more than 65 years to evaluate with a standard laptop.

To get a sense on how many evaluations we need for the IDDFS optimization algorithm, we can again run the IDDFS 1500 times for each number of repetition, and analyse the results, Fig.9. This shows that the IDDFS algorithm can reliably get to the likely global minimum if we use more than 350 repetitions.

The IDDFS requires in this case $350 \times 6 \times 16 = 33.600$ computations which takes 15 to 25 seconds.

One of the best solutions is ($C = 0.29$, Fig.10): p_1 ($e_p = 6$): [o_6, o_8, o_{13}], p_2 ($e_p = 6$): [o_{14}, o_7], p_3 ($e_p = 2.5$): [o_1, o_2], p_4 ($e_p = 3$): [o_4, o_3, o_{12}], p_5 ($e_p = 3$): [o_{16}, o_{11}, o_9], p_6 ($e_p = 3$): [o_5, o_{15}, o_{10}].

Table IV: Complex example. (Changes will be implemented after the initial allocations (in red))

o	$t_{o,start}$	s_o	Work profile
1	0:00	1	20% (0-4h)
2	0:00	3	20% (0-4h)
3	0:00 (3:00)	3 (6)	25% (0-0:45h) 40% (0:45-1:15h) 10% (1:15-1:30h)
4	0:00	3	25% (0-0:45h) 40% (0:45-1:15h) 10% (1:15-1:30h)
5	1:00	3	50% (0-0:45h) 80% (0:45-1:15h) 10% (1:15-1:30h)
6	2:00	3	25% (0-0:45h) 40% (0:45-1:15h) 10% (1:15-1:30h)
7	2:00	6	50% (0-0:45h) 80% (0:45-1:15h) 10% (1:15-1:30h)
8	2:00	6	25% (0-0:45h) 40% (0:45-1:15h) 10% (1:15-1:30h)
9	2:00	3	50% (0-0:45h) 80% (0:45-1:15h) 10% (1:15-1:30h)
10	3:30	3	50% (0-0:45h) 80% (0:45-1:15h) 10% (1:15-1:30h)
11	3:30	2.5	50% (0-0:45h) 80% (0:45-1:15h) 10% (1:15-1:30h)
12	4:00	6	20% (0-4h)
13	4:00	3	50% (0-0:45h) 80% (0:45-1:15h) 10% (1:15-1:30h)
14	5:00	6	50% (0-0:45h) 80% (0:45-1:15h) 10% (1:15-1:30h)
15	6:00	3	25% (0-0:45h) 40% (0:45-1:15h) 10% (1:15-1:30h)
16	6:00	3	50% (0-0:45h) 80% (0:45-1:15h) 10% (1:15-1:30h))
17	4:00	6	20% (0-4h)
18	4:30	3	25% (0-0:45h) 40% (0:45-1:15h) 10% (1:15-1:30h)

In this case, multiple optimal allocations exist. This is due to the fact multiple operators have the same expertise level and several operations are identical, making their allocations interchangeable to those operators. The IDDFS method finds an allocation for which all operators have (about) the same workload, experienced operators take on the more difficult tasks and all operators can take a break at least once every four hours.

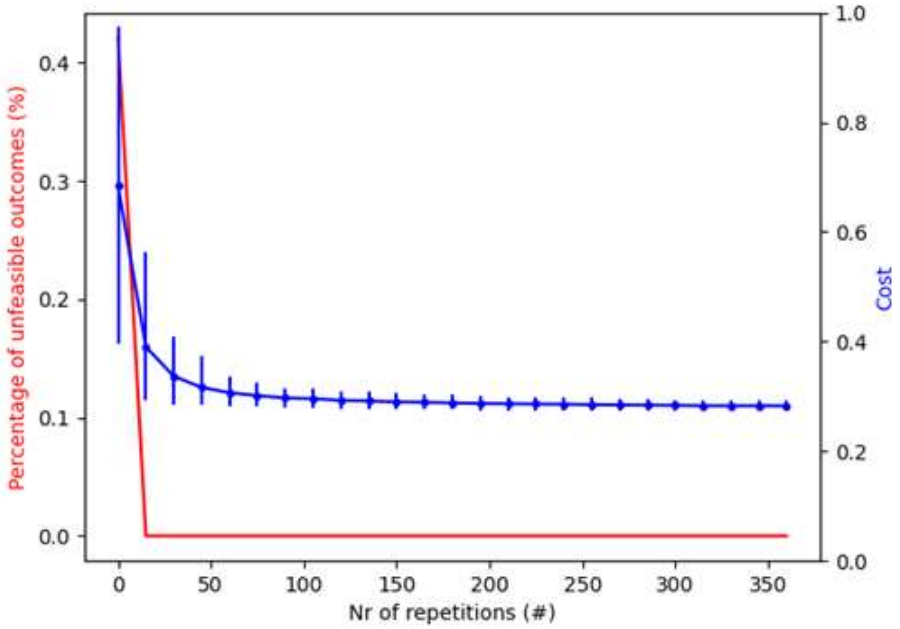

Fig.9: Cost outcome of IDDFS related to the amount of repetitions. Red: percentage of unfeasible outcomes in relation to number of repetitions. Blue: Average cost of the found solution (10%-90% quartile with vertical lines).

We now assume that during the shift, two additional operations connect to the shore control centre (o_{17} and o_{18}). Similarly, we assume a sensor malfunctions during o_3, and it is delayed until 3:00. This also causes the operation to become much more difficult as redundancy is not guaranteed any longer (**s_o:** 3→ 6).

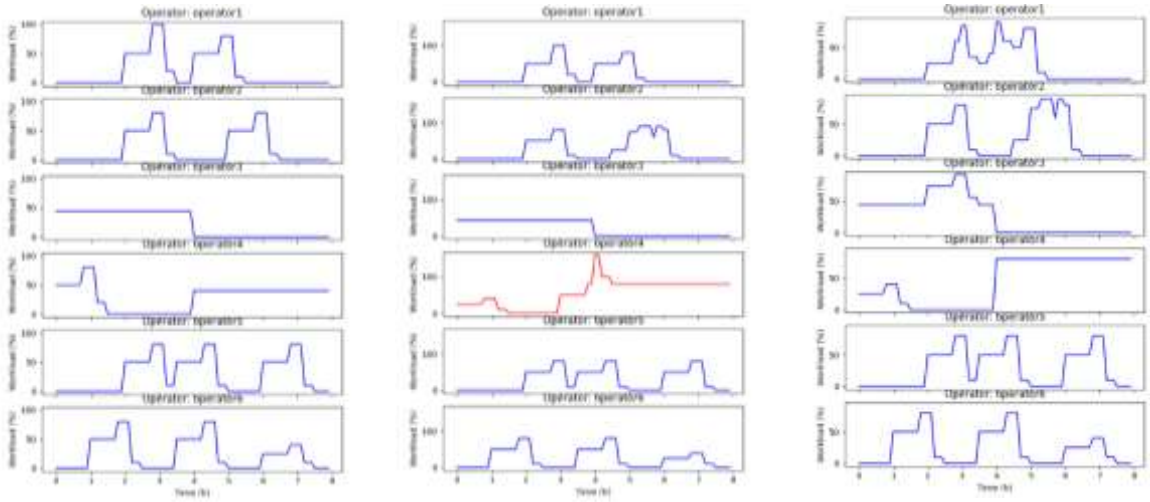

Fig.10: Workload for complex example. Left: Workload before the operations are changed. Middle: Workload after operations are naively allocated without optimization, showing overload for fourth operator. Right: After allocating the new and changed operations and re-optimizing the existing allocations.

The order of events is as follows: 1) o_{17} is added, 2) o_{18} is added, 3) the sensor malfunctions for o_3. Imagine we naively add o_{17} and o_{18} to the operators which keeps the cost as low as possible. Also, we will not re-allocate o_3 and keep it under fourth operator's supervision. No further optimisation is performed. This results in the following planning, Fig.10: p_1 ($e_p = 6$): [o_6, o_8, o_{13}], p_2 ($e_p = 6$): [o_{14}, o_7, o_{18}], p_3 ($e_p = 2.5$): [o_1, o_2], p_4 ($e_p = 3$): [o_4, o_3, o_{12}, o_{17}], p_5 ($e_p = 3$): [o_{16}, o_{11}, o_9], p_6 ($e_p = 3$): [o_5, o_{15}, o_{10}].

This planning is now unfeasible (thus: it does not get a regular score). The fourth operators gets o_{17} assigned, whereas o_{18} is assigned to the second operator. Though if time progresses and the sensor malfunctions during o_3, its difficulty spikes and so does its workload for the fourth operator. Who, with the extra assignment of o_{17}, has now more work than it can handle. This creates an unreasonable dangerous situation.

Luckily, with the IDDFS method we can not only naively allocated new operations, but we can do so while performing (minimal) changes to an already existing allocation. In other words, we can dynamically allocate new operations in an optimised way that minimizes operator workload as much as possible. In the above-described case, the method finds a more optimised allocation that requires only two reallocations; as the new operation o_{17} is allocated to the fourth operator p_4, its previously assigned operation o_3 (the one who will suffer a sensor malfunction) is allocated to the first operator p_1. Furthermore, the operation o_6 which was allocated to this first operator p_1 is now allocated to the third operator p_3. These changes create more room for the fourth operator to take on the new operation o_{17} with a minor reallocation needed to prevent other operators to become overloaded with work. Note that where in the original optimisation the method favoured the least experienced operator p_3, it now makes use of time this operator still has free to take on other work. The complete allocation is as now as follows ($C = 0.39$, see Fig. 10): p_1 ($e_p = 6$): [o_8, o_{13}, o_3], p_2 ($e_p = 6$): [o_{14}, o_7, o_{18}], p_3 ($e_p = 2.5$): [o_1, o_2, o_6], p_4 ($e_p = 3$): [o_4, o_{12}, o_{17}], p_5 ($e_p = 3$): [o_{16}, o_{11}, o_9], p_6 ($e_p = 3$): [o_5, o_{15}, o_{10}].

This plan is feasible, and has a score much closer to the initial score (increase of 0.1 in cost, opposed to an unfeasible plan before optimization). Coincidentally, the score is even close to the likely optimal solution if the proposed changes were known from the beginning ($C = 0.38$). In some cases, a significantly better plan can be found if we fully re-optimize, however, this would also mean the allocation of each operator would change completely, and as a consequence confuse them. The proposed method hence includes a penalty for each change to remedy this.

3.4. Discussion

The proposed cost function is an approximation of reality. The workload of supervising one or more operations might be hard to define as a predetermined percentage. In addition, modelling an operator's expertise or operation's complexity in a single score, can be too simplistic. However, we argue that a fully realistic model of workload is not required to achieve a dynamic allocation of autonomous operations. Ideally, a rigorous evaluation of the workload model is required, although this is hampered by the simple fact that shore control centres where operators supervise autonomous vessels is still an unachieved future. Therefore, the proposed algorithm is an initial step towards dynamic task allocation, but will need to be adapted, evaluated and optimized on more realistic scenarios.

Part of this future work is a way to elicit cost variables and their weights. Currently, the cost variables modelled given common insights from the human factors research field and their weights chosen through intuition to get logical results. A first future step thus could be to more adequately model these variables and their weights. For instance, it needs to be determined how much more (or less) important it is for an operator to have a break versus having a high peak intensity during work. Such knowledge should be elicited from human factors experts with a particular shore control centre in mind, and subsequently evaluated with the operators of such a centre.

The parameter of sampled repetitions is a difficult parameter to set. It determines the likelihood that the IDDFS method finds the optimal allocation, however how many repetitions are required is difficult to estimate beforehand. A representative example of operators and operations can be analysed, as was done in Figs.8 and 9. However, such ana analysis relies on the brute-force method to find the optimal allocation first. Which in turn requires a potential unfeasible amount of time, preventing such an analysis for realistic scenarios. Future work might aim at presenting different analysis methods or make us of unparameterized algorithmic alternatives to IDDFS.

4. Conclusion

This work proposed a way to allocate autonomous operations within the maritime domain to operators for supervision. The proposed method is intended to automate the safe and effective allocation of such operations in a shore control centre. We argued that human factor knowledge on optimal workload distribution should form the core of such an allocation. Several example scenarios with increasing complexity were presented and analysed.

The results show that the proposed dynamic task allocation algorithm can distribute operations over operators in a way that optimizes the workload following human factors knowledge. The first illustrative example scenario showed that the resulting allocations are logical and meaningful. The moderate example scenario showed that the algorithm can produce feasible solutions when manual allocation is difficult or impossible. We also shown that this scenario can be solved by a brute-force algorithm, but becomes quickly unfeasible due to the time required to find an optimal allocation. The used Iterative Deepening Depth First Search (IDDFS) algorithm however is much more scalable and can, with enough sampled repetitions, find the global optimum. However, given the nature of IDDFS as a heuristic search algorithm, we can never be assured that what seems optimal is indeed so. However, it is likely that a near-optimal allocation is still very much sufficient to enable a few operators to effectively supervise many more operations. Lastly, the complex example scenario showed that the IDDFS implementation is still robust for an increasing number of operations and operators. Next to that, it shows that the dynamic allocation makes the algorithm robust against changes that arise after any previously made allocation.

To conclude, the IDDFS algorithm combined with a cost function capturing human factors knowledge about workload, seems to be a promising technology to facilitate shore control centres where few operators supervise many autonomous operations. This work takes a first step towards automating task allocation in shore control centres. The MOSES EU project is working towards getting closer to this goal.

Acknowledgements

This research has been conducted as part of MOSES project, which has received funding from the European Union's Horizon 2020 Research and Innovation Program under Grant agreement No. 861678.

References

KORF, R.E. (1985), *Depth-first iterative-deepening: An optimal admissible tree search*, Artificial Intelligence 27(1), pp.97-109

NEERINCX, M.A. (2003), *Cognitive task load design: model, methods and examples*, Handbook of Cognitive Task Design, Ch.13, Lawrence Erlbaum Associates, pp.283-305

NEERINCX, M.A.; DOBBELSTEEN, G.J.H. VAN DEN; GROOTJEN, M.; VEENENDAAL, J. VAN (2003), *Assessing cognitive load distributions for envisioned task allocations and support functions*, 13th Int. Ship Control Systems Symposium (SCSS), Orlando

VAN DEN BROEK, J.; GRIFFIOEN, J.R.; VAN DER DRIFT, M. (2020), *Meaningful Human Control in Autonomous Shipping: An Overview*, IOP Conference Series: Materials Science and Engineering 929(1), https://doi.org/10.1088/1757-899X/929/1/012008

VAN ROSSUM, G.; DRAKE, F. L. (2009), *Python 3 Reference Manual*, CreateSpace