Annals of Operations Research (2021) 302:425–459
https://doi.org/10.1007/s10479-020-03608-6

S.I.: PATAT 2018

A matheuristic approach to large-scale avionic scheduling

Emil Karlsson[1,2] · Elina Rönnberg[1,2] · Andreas Stenberg[2] · Hannes Uppman[2]

Published online: 14 May 2020

Abstract
Pre-runtime scheduling of avionic systems is used to ensure that the systems provide the desired functionality at the correct time. This paper considers scheduling of an integrated modular avionic system which from a more general perspective can be seen as a multiprocessor scheduling problem that includes a communication network. The addressed system is practically relevant and the computational evaluations are made on large-scale instances developed together with the industrial partner Saab. A subset of the instances is made publicly available. Our contribution is a matheuristic for solving these large-scale instances and it is obtained by improving the model formulations used in a previously suggested constraint generation procedure and by including an adaptive large neighbourhood search to extend it into a matheuristic. Characteristics of our adaptive large neighbourhood search are that it is made over both discrete and continuous variables and that it needs to balance the search for feasibility and profitable objective value. The repair operation is to apply a mixed-integer programming solver on a model where most of the constraints are treated as soft and a violation of them is instead penalised in the objective function. The largest solved instance, with respect to the number of tasks, has 54,731 tasks and 2530 communication messages.

Keywords Multiprocessor scheduling · Avionic system · Matheuristic · Adaptive large neighbourhood search · Integer programming · Scheduling

1 Introduction

Modern aircraft host a huge amount of electronics such as sensors that gather information, units where the information is processed, actuators that control the aircraft, and equipment that presents information to the pilot. Electronics in an aircraft is called avionics and due to the real-time requirements of avionic systems, it is not sufficient that the logical result of a computation is correct, it is also crucial that the result is produced at the correct time. This punctuality can be ensured by having a schedule for all activities in the system. Scheduling of real-time systems can refer either to on-line scheduling where the scheduling decisions are

✉ Elina Rönnberg
elina.ronnberg@liu.se

1 Department of Mathematics, Linköping University, 581 83 Linköping, Sweden

2 Saab AB, 581 88 Linköping, Sweden

made at runtime or, as in this work, to pre-runtime (off-line) scheduling where the schedule is created at compile time.

This work addresses the scheduling of an industrially relevant Integrated Modular Avionic (IMA) system, that was introduced in Blikstad et al. (2018). This problem can be considered as a multiprocessor scheduling problem, where the tasks have multiple time windows and precedence relations between them, combined with the scheduling of a communication network. In the avionics context, a processor is referred to as a module and we consider the case when all tasks are beforehand assigned to a module. The communication network is scheduled by assigning communication messages to time slots. To send a communication message involves the execution of certain tasks on the involved modules and for this reason, the task and communication scheduling are closely integrated. Compared to the problem formulation used in Blikstad et al. (2018), we here also include the possibility of co-allocation of messages in a time slot as introduced in Rönnberg (2018). There is no objective function and the purpose of the scheduling is to find a feasible solution or to conclude that none exists.

The future industrial need is a capability to solve very large-scale instances of avionics scheduling problems of this kind and the contributions of this paper are extensions of previous work that facilitate this. In the previous works Blikstad et al. (2018), Karlsson and Rönnberg (2018), and Rönnberg (2018), the largest instance solved has 15 modules and about 20,000 tasks (about 50% of these are fixed tasks and 6000 of the tasks are placed on the most occupied module).

A first contribution of the paper is an exact solution approach based on a constraint generation procedure which is obtained by combining and improving the models and methods presented in Blikstad et al. (2018), Karlsson and Rönnberg (2018), Rönnberg (2018), and Boberg (2017). The constraint generation procedure alternates between solving a relaxation of the problem and using its solution to restrict the solution space to define a subproblem. If this subproblem has a feasible solution, this is a solution to the original problem, and otherwise new constraints are added to the relaxed problem and the procedure is repeated. Since it is important that the solution to the relaxed problem has a good chance to yield a feasible solution to the subproblem, the relaxed problem has an objective function with the purpose to contribute to this.

The results for the exact constraint generation procedure show that such decomposition approach is efficient for the problem structure, but in order to solve even larger instances, a second contribution of this paper is to extend it into a matheuristic method (see e.g. Maniezzo et al. (2010) and Archetti and Speranza (2014)) by combining it with an Adaptive Large Neighbourhood Search (ALNS) [see Ropke and Pisinger (2006) and Pisinger and Ropke (2007)] for solving the relaxed problem. To find a feasible solution to the relaxed problem is for larger instances a true challenge and our ALNS is therefore carefully designed with respect to searching for feasibility while taking the value of the objective function into account. This is obtained by the following two key components. Firstly, since the repair method needs to handle both discrete and continuous variables, several kinds of constraints, and an objective function, we have chosen to repair the solution by using a mixed integer programming (MIP)-solver. This gives the possibility to benefit from the very efficient heuristics implemented in commercial solvers, as initially suggested in Muller et al. (2012). Secondly, the adaptive mechanism of our ALNS is used to handle the balance between the objective value and feasibility with respect to various constraints in a strategic oscillation fashion.

Our computational results are provided for two categories of industrially relevant instances. The first category, which is not publicly available, orginates from Blikstad et al. (2018) and it is included to provide a comparison to our previous work. The second cate-

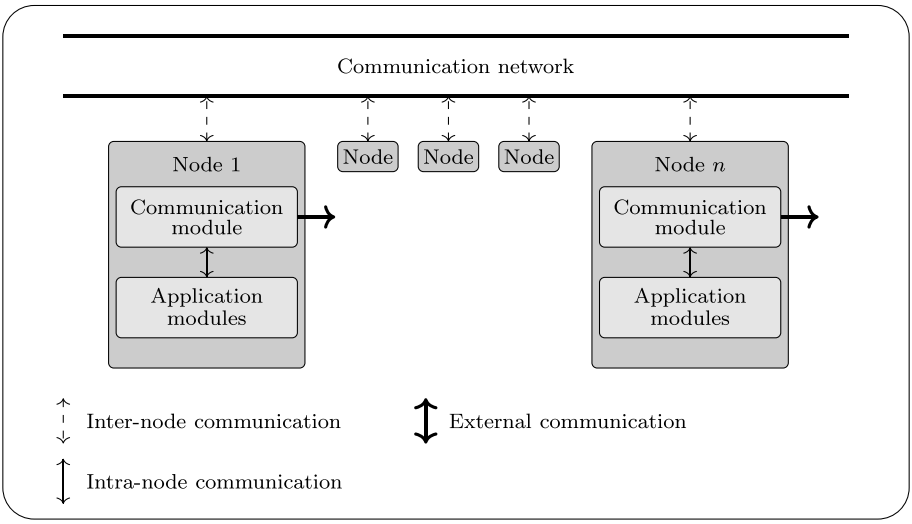

Fig. 1 An overview of a system with n nodes. The structure of the content is the same for all nodes and therefore only displayed for node 1 and n

gory is new and derived to be industrially relevant but generic enough to be made publicly available.

This section continues with a brief presentation of the industrial background of the problem and a review of research that include avionics scheduling and MIP. Thereafter follows a description of related work on matheuristics and ALNS, the latter with a particular emphasis on strategies that use MIP-models to repair the solution. The section ends with an outline of the paper and a summary of its contributions.

1.1 Problem statement

The addressed avionic system scheduling problem was introduced in Blikstad et al. (2018), where a detailed technical background of the problem statement was given. The first description was extended in Rönnberg (2018) to include further details with respect to the communication scheduling. Here, we give the problem statement without describing its complete technical background and refer the interested reader to Blikstad et al. (2018) and Rönnberg (2018) for these.

The considered avionic system, illustrated in Fig. 1, contains a set of nodes with modules (processors) that hosts tasks. Each node contains a single communication module (CM), and one or more application modules (AMs). A CM handles the external, inter-node, and intra-node communication of the node and an AM runs the software applications. These applications are grouped and assigned to partitions that are the entities treated as tasks to be scheduled. The nodes communicate by sending messages on a communication network (CN) from a single CM to a set of receiving CMs and the communication protocol is such that messages are sent in discrete time slots. The considered system executes periodically, meaning that a schedule, called a major frame, is repeated infinitely.

The tasks in the system execute periodically and all tasks on the CMs have the same period as the system. On the AMs, the period of a task can be a divisor of that of a major frame, and

then there are multiple instances of a task in a major frame. Each task must be performed on a particular module for the duration of its execution requirement without overlapping any other task. For each task, there is a set of sub-intervals and each task must be assigned to one of these. Between each ordered pair of tasks on an AM, there is an additional requirement that there should be a minimum idle time between them.

There are precedence relations between tasks and these are referred to as dependencies. A dependency restricts the duration between the start of an instance of a task to the start of an instance of another task to be within the interval given by a minimal and a maximal time lag.

To send and receive a message on a CN, there are four types of tasks that are performed in a particular order on the involved CMs. On the sending CM, there is first a task that prepares the message and then a task that sends the message. On each receiving CM, there is first a task that dequeues the message and then a task that reads the data. Each message sent through the CN must be assigned to a time slot, and thereby claim some of the capacity available in this slot. Each slot has a maximum capacity that cannot be exceeded by the combined requirement of the messages assigned to the slot. If two or more messages are sent in the same slot, their tasks of the same type must be performed in immediate succession and with a reduced execution requirement compared to if they were performed separately. Furthermore, the assignment of a message to a slot imposes slot-unique release times and deadlines on the tasks involved.

Further details are presented in Sect. 2, along with the mathematical model; there, and throughout the paper, the model is divided into four components:

- AM-scheduling
- CM-scheduling
- Precedence relations
- CN-scheduling

AM-scheduling and CM-scheduling refers to the scheduling of tasks on the AMs and CMs respectively, while Precedence relations refers to the dependencies between tasks and CN-scheduling refers to the communication message scheduling and the additional requirements on the tasks involved in sending and receiving messages.

The problem statement does not include an objective function since all feasible solutions to the problem are considered equally good. Instead, the challenge is to determine whether a feasible solution exists or not, and if there is a feasible solution, provide a schedule. A schedule for a major frame is defined by two types of decisions, the assignment of communication messages to time slots and the assignment of start times to tasks.

1.2 Related work on avionic scheduling

The solution approach in this paper utilises MIP techniques to solve a pre-runtime scheduling problem that occurs in the development of IMA systems. Therefore, we restrict the related work on avionic scheduling to contributions that apply MIP approaches to pre-runtime scheduling of IMA systems. We start by characterising the previous work with respect to the decisions to be made while scheduling and how they are constrained. This is followed by a description of solution approaches. For a technically oriented overview of scheduling in IMA systems, we refer to Blikstad et al. (2018).

In our problem setting, each task is limited to execute on a specific module and this gives a reduced complexity compared to when each task must be assigned to a module with respect to different side constraints, as in Hao et al. (2018) and Eisenbrand et al. (2010). In Hao

et al. (2018), the assignment is done with respect to memory and security constraints, while in Eisenbrand et al. (2010) this is done with respect to memory requirements, the bandwidth of the network linking together the modules, and co-habitation constraints (a subset of tasks must execute on the same or a specific module).

In an IMA system, software applications typically must communicate with each other within certain time limits. This often occurs as precedence constraints between tasks or instances of tasks forcing them to execute within a given duration of one another. In our problem setting, we have precedence constraints between particular instances of tasks, while in the problem setting of Hao et al. (2018), Zhang et al. (2014), Craciunas and Oliver (2016), and Lhachemi et al. (2016), they have precedence constraints between tasks. When precedence constraints are defined between tasks, the constraint forces the instances of the involved tasks to execute within a given duration of another. In Craciunas and Oliver (2016) and Lhachemi et al. (2016), the number of instances of the tasks involved in a precedence constraint are assumed to be the same, while in Hao et al. (2018) and Zhang et al. (2014) any number of instances of the tasks in a precedence constraint is allowed. In Hao et al. (2018), they also handle constraints related to if tasks execute on the same module or not.

An Ethernet network is often used for communication between tasks on different modules. Depending on the modelling of this network, the level of interaction between communication and task scheduling can differ. In our model, a specific time-triggered Ethernet network, described in Blikstad et al. (2018), is used where the messages sent on the network must obey sequencing constraints. The messages sent on the network are associated with tasks on the modules and this increases the complexity of the scheduling problem. In Zhang et al. (2014) and Craciunas and Oliver (2016), who both also address time-triggered Ethernet networks, there are sequencing constraints between messages sent on the network that are linked to tasks on the modules. However, in Eisenbrand et al. (2010), an Avionics Full-Duplex Switched Ethernet network is studied, and there the network communication appears as an extra knapsack constraint on the assignment of tasks to modules, where each module has a maximum bandwidth limit.

There exist various solution approaches to tackle the scheduling problems in IMA systems using MIP techniques. In Hao et al. (2018), Lhachemi et al. (2016), and Zhang et al. (2014), the authors state MIP models and solve problems with up to 5, 10 and, 270 tasks, respectively. In Eisenbrand et al. (2010) the authors present three different MIP formulations for their problem and compare their performance to a heuristic for instances with up to 177 tasks. An exact incremental solution approach utilising the problem structure, as well as a solution approach for solving the entire problem at once, are presented in Craciunas and Oliver (2016). Their performance is evaluated on instances with up to 6912 tasks, using both a satisfiability modulo theory solver and a MIP solver.

In our problem, the model for the communication between modules is very detailed, resulting in more tasks than described in existing literature concerning scheduling of IMA systems. We will present results for instances with up to 54,731 tasks, which is to be compared to that the maximum number of tasks that we have found in the existing literature is 6912.

1.3 Related work on matheuristics and ALNS

In the proceedings of the first workshop on matheuristics in 2008, Caserta and Voß (2010) described matheuristics as *exploiting mathematical programming techniques in (meta)heuristic frameworks*. In the following years, matheuristics received an increased attention from the combinatorial optimisation community, and a number of contributions on

matheuristics were published and summarised in three surveys. The first survey, by Doerner and Schmid (2010), presented matheuristic techniques applied to rich vehicle routing problems. The second survey, by Ball (2011), focused on the different application areas in which matheuristics had been used. In the third survey the topic was again matheuristic techniques applied to routing problems, this time by Archetti and Speranza (2014). The topic was relevant to survey once more thanks to the number of contributions in these years.

Since then, matheuristic techniques have been applied to an even wider range of applications. Two recent examples being Guido et al. (2018), who uses MIP models to explore large neighbourhoods in a metaheuristic search for an offline patient-to-bed assignment problem, and Framinan and Perez-Gonzalez (2018), who use a succession of MIP models in an approximation algorithm for an order scheduling problem. The matheuristic to be presented in this paper is based on an ALNS that uses restricted MIP models to explore large neighbourhoods efficiently.

The ALNS framework was introduced in the papers Ropke and Pisinger (2006) and Pisinger and Ropke (2007) as an extension of the large neighbourhood search (LNS) by Shaw (1998). LNS is a metaheuristic strategy that iteratively tries to improve a solution by exploring large neighbourhoods. In an LNS, the move of an iteration is defined by two operators, a destroy operator that creates a neighborhood by destroying the current solution and a repair operator that reconstruct a solution given a neighborhood. Different destroy and repair operators are typically defined, and a pair of these is selected at random in an iteration. The ALNS framework extends LNS by employing a master level metaheuristic (or local search) to guide the neighbourhood search, together with a mechanism to adapt the algorithm based on its performance. In Ropke and Pisinger (2006) and Pisinger and Ropke (2007), simulated annealing was used as the master level metaheuristic.

In Ropke and Pisinger (2006), the ALNS framework was applied to the pickup and delivery problem with time windows and in Pisinger and Ropke (2007), it was used to solve a general routing problem. Since then, the ALNS framework has been successfully applied to a number of different optimization problems; see for example time-tabling in Kiefer et al. (2017), optimisation of yard assignment in an automotive transshipment terminal in Cordeau et al. (2011), scheduling twin yard cranes in Gharehgozli et al. (2015), and multi-mode resource-constrained project scheduling in Gerhards et al. (2017).

In the original ALNS framework, the authors suggest using fast heuristics to repair the destroyed solution. However, in Muller et al. (2012), the authors instead use MIP models as repair operators. Their motivation for this is that the heuristics in commercial MIP solvers have become efficient enough to be used instead of fast problem-specific heuristics. This is particularly useful for complex problems where it is not straightforward to implement efficient heuristics. In Muller et al. (2012), the authors employ a strategy where the destroy operator releases a set of variables from their current solution while keeping the rest fixed. This creates a restricted MIP-model that defines the neighbourhood to be searched. The solver then acts as the repair operator and tries to find a new solution to the restricted MIP model that improves the objective compared to the solution that was destroyed. Since the solution of the restricted MIP model of an iteration is a feasible solution for the restricted MIP of the next iteration, Muller et al. (2012) use steepest descent as the master level heuristic to guide the neighbourhood search. As a mechanism to improve diversity, Muller et al. (2012) restart the search with the next best solution if no improvement is found in a predetermined number of iterations.

Since then, MIP models have been further used in ALNS. Two publications that utilise restricted MIP models as repair operators, similar to Muller et al. (2012), are Mancini (2016) that solve a vehicle routing problem, and Belo-Filho et al. (2015) that solve an integrated

production and distribution problem. Both their algorithms use the solution of the previous iteration as a feasible starting solution in the next iteration, and then only accept a new solution if it improves the objective. Neither of them employ a restart mechanism.

In Gerhards et al. (2017), the authors implement MIP models as repair operators in an ALNS to a solve a multi-mode resource scheduling problem. After each iteration, they perform a problem specific heuristic, called Serial Schedule Generation Scheme, in an attempt to improve the solution created by the repair operator. A difference in Gerhards et al. (2017), compared to Muller et al. (2012), is that they use different objectives in the repair operators, which might result in a solution with a worse global objective than the solution of the previous iteration. Regardless of the new value of the global objective, the authors always keep their new solution, and therefore their master level metaheuristic can be interpreted as a variable neighbourhood descent method.

The work of Pereira et al. (2015) uses repair operators that hybridise the use of a constructive heuristic and a MIP model within their ALNS search developed for the probabilistic maximal covering location-allocation problem. In an iteration, the authors first destroy and repair the location decision by a pair of destroy and repair heuristics, and secondly they recreate a complete solution to the original problem using a MIP model.

1.4 Outline and contribution

The motivation for the solution method to be presented in this paper is the industrial need for a scheduling tool that can be used for large-scale avionic scheduling instances of the kind addressed in this paper. To this end, Sect. 2 first presents an exact solution approach based on a constraint generation procedure which is the outcome of combining models and methods presented in Blikstad et al. (2018), Karlsson and Rönnberg (2018), Rönnberg (2018), and Boberg (2017) with some further improvements which yield a significant increase in computational performance.

To further enhance the capability to solve very large-scale problems, Sect. 3 presents how the constraint generation procedure is extended into a matheuristic. The matheuristic uses an ALNS to solve the relaxation of the problem used in the constraint generation procedure, and the search explores the neighbourhoods by solving MIP models. To achieve feasibility in this relaxed problem is difficult, and we therefore designed the ALNS to balance the focus between the objective value and feasibility in a way inspired by strategic oscillation, see Glover (1977).

Section 4 contains computational results for two categories of industrially relevant instances. The first category, which is not publicly available, is included since it is important for our project. Some of these instances were introduced in Blikstad et al. (2018) and enable comparision to our previous work. The second category is new and derived to be industrially relevant but expressed in generic enough terms to be made publicly available.[1] As illustrated by the computational results for these instances, both the improvements of the exact approach, as introduced in Sect. 2, and the design and use of an ALNS, introduced in Sect. 3, successfully contributes to solving very large-scale instances that could not previously be solved. The paper is ended by concluding remarks in Sect. 5.

[1] https://gitlab.liu.se/eliro15/avionics_inst/tree/master.

2 Exact solution approach

A mathematical model and an exact solution approach based on constraint generation were presented in Blikstad et al. (2018) and the exact procedure to be described in this section is based on the same mechanisms as the original one. Therefore, we begin this section with a short review of the original procedure, followed by a summary of the enhancements made, and then the details of the exact approach proposed in this paper are presented.

2.1 The constraint generation procedure from previous work

Already in Blikstad et al. (2018), the mathematical model was divided into the four components Precedence relations, AM-scheduling, CN-scheduling, and CM-scheduling. It was then observed that for problem instances of practical relevance

- the computational challenge is due to the large number of tasks to be sequenced on the CMs,
- a considerable amount of the tasks on the CMs are fixed, and
- the technical restrictions, like release times and deadlines of tasks, precedence relations and CN-scheduling, are not particularly tight.

Together, these characteristics make it profitable to make the following model formulation, which is suitable for constraint generation. This formulation divides the CM-scheduling component into relaxed CM-scheduling and CM-sequencing. For the relaxed CM-scheduling, a so-called section is defined for each part of a major frame that is not occupied by a fixed task, and it is required that each non-fixed task is assigned to a section, while complying with all requirements related to CM-scheduling, except preventing non-fixed CM-tasks from overlapping. In a complete model for the problem, CM-sequencing refers to that, for each set of non-fixed tasks that can be assigned to the same section, require that the tasks cannot overlap. This complete model, of course, includes an extreme number of variables and constraints, but by design, this formulations lends itself to constraint generation.

The constraint generation procedure begins with solving a relaxed problem where the mathematical model includes the components Precedence relations, AM-scheduling, CN-scheduling, and relaxed CM-scheduling. In a solution to the relaxed problem, each task is assigned to a section. By restricting the solution space of the complete model to comply with a solution to the relaxed problem, only a very small subset of CM-sequencing constraints are relevant to include in the model. Hence, a subproblem is defined by solving the complete model under the restriction imposed by the solution to the relaxed problem. In the subproblem, there is no guarantee that CM-sequencing is possible, therefore, the CM-sequencing constraints are treated as soft constraints and the objective of the subproblem is to minimize the number of overlapping CM-tasks.

If the subproblem finds a solution where there are no overlaps, a solution to the complete model has been found. Otherwise, it can be detected which of the CM-sequencing constraints are violated, and these can be added to the relaxed problem, which is then re-solved. This process can be repeated until an overlap-free solution to the subproblem is found. Convergence of the method follows from that there is a finite number of constraints to be added to the relaxed problem, and that in each iteration, at least one new constraint is added. In practice, the success of the method relies on that only very few of the constraints need to be generated.

2.2 The constraint generation procedure proposed in this work

The constraint generation procedure used in this work differs from previous work as follows. Firstly, in Blikstad et al. (2018), a restriction of the industrial problem was studied, since the possibility of co-allocating CN-messages was omitted. To allow co-allocation of messages is important from a practical point of view, but it makes the problem computationally more challenging because of the additional decisions to be made. In this paper, co-allocation is included as outlined in Rönnberg (2018), but with some improvements of the mathematical modelling.

Secondly, the relaxed CM-scheduling component has been improved by, instead of assigning the CM-tasks to a section (that is, part of a major frame that is not occupied by a fixed task) assign each CM-task to a smaller interval where the task must be placed, as outlined in Karlsson and Rönnberg (2018). This more fine-grained decision is made possible because after pre-processing, the interval between the release time and deadline of a task is divided into several sub-intervals, and it is known that in a feasible solution, a task must be placed within one of these sub-intervals. Thirdly, in the Bachelor thesis Boberg (2017), a computational study was performed to investigate which mathematical formulation of the CM-sequencing component performs best. The results of this study led to a modification of the CM-sequencing constraints compared to Blikstad et al. (2018).

By combining these enhancements, together with some further improvements of details, the resulting mathematical model is as to be presented. The model is divided into a Relaxed problem and a Subproblem, as illustrated in Fig. 2. The presentation of the model is structured such that it begins with introducing parts of the formulation that are common for AM- and CM-scheduling and it continues to present each of the model components according to the section references in Fig. 2. A consequence of the model changes is that the constraint called chains in Blikstad et al. (2018) has become redundant and it is therefore removed in this work. An overview of the solution procedure is presented after the model components.

2.3 Common for AM- and CM-scheduling

Let the set $\mathcal{I}$ index all tasks in the system and let P denote the length of the major frame. Task i has an execution requirement e_i and must execute without preemption within the interval between its release time t_i^{r} and deadline t_i^{d}, $i \in \mathcal{I}$. If $t_i^{\mathrm{r}} = t_i^{\mathrm{d}} - e_i$ holds, task i is referred to as fixed. The period of task i is denoted by p_i, $i \in \mathcal{I}$, with p_i being a divisor of P so that there are P/p_i instances of task i in a major frame. For $i \in \mathcal{I}$, introduce the continuous variable

$$x_i = \text{start time of task } i.$$

Let $\mathcal{H}^{\mathrm{CM}}$ and $\mathcal{H}^{\mathrm{AM}}$ denote the set of CMs and the set of AMs, respectively. Let the set $\mathcal{I}_h$ include the indices of the tasks assigned to module $h \in \mathcal{H}^{\mathrm{CM}} \cup \mathcal{H}^{\mathrm{AM}}$. For the industrial problem under consideration, all tasks on the CMs have a period of length P and all tasks on the AMs have a period of length $P/64$.

The interval between the release time and deadline of a task is, due to technical limitations and as a consequence of pre-processing, divided into sub-intervals. In a feasible solution, a task is assigned to one of these intervals. Let the set $\mathcal{Q}_i$ include the indices of the sub-intervals of task i, and let the release time and deadline of sub-interval q for task i be denoted by t_{iq}^{r} and t_{iq}^{d}, respectively, $q \in \mathcal{Q}_i$, $i \in \mathcal{I}$. Note that $t_i^{\mathrm{r}} = \min_{q \in \mathcal{Q}_i} t_{iq}^{\mathrm{r}}$ and that $t_i^{\mathrm{d}} = \max_{q \in \mathcal{Q}_i} t_{iq}^{\mathrm{d}}$.

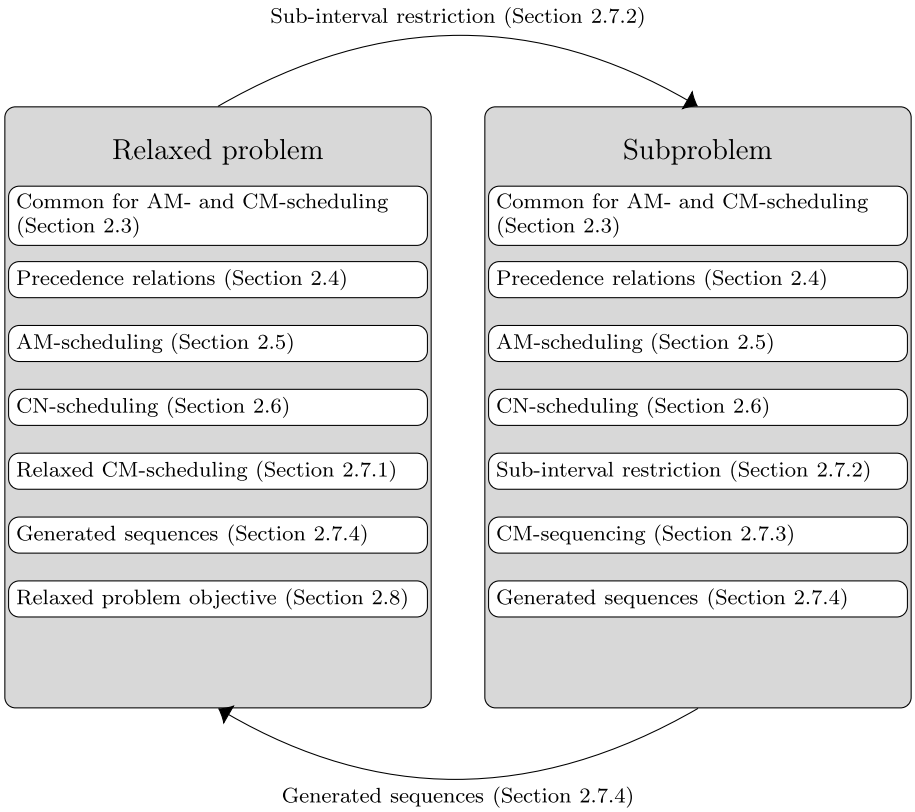

Fig. 2 Information flow and model components in the constraint generation procedure

For $i \in \mathcal{I}, q \in \mathcal{Q}_i$, introduce the binary variable

$$\alpha_{iq} = \begin{cases} 1, & \text{if task } i \text{ is assigned to sub-interval } q, \\ 0, & \text{otherwise.} \end{cases}$$

Each task is assigned to one of its sub-intervals by the constraint

$$\sum_{q \in \mathcal{Q}_i} \alpha_{iq} = 1, \quad i \in \mathcal{I}, \tag{1}$$

while the constraint

$$\sum_{q \in \mathcal{Q}_i} t^{\mathrm{r}}_{iq} \alpha_{iq} \leq x_i \leq \sum_{q \in \mathcal{Q}_i} t^{\mathrm{d}}_{iq} \alpha_{iq} - e_i, \quad i \in \mathcal{I}, \tag{2}$$

restricts the release time and deadline of the task to be that of the sub-interval it is assigned to.

2.4 Precedence relations

In this problem setting, we consider one type of precedence relation constraints called dependencies. The dependencies are indexed by quadruples where (i, j, k, l) refers to restricting

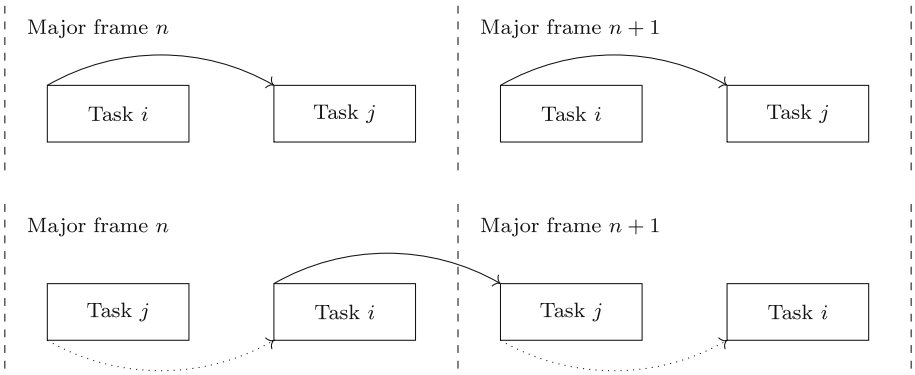

Fig. 3 Examples of the two ways in which the duration between the start of instance $k = 0$ of task i to next start of instance $l = 0$ of task j can be measured for dependency $(i, j, k, l) \in \mathcal{D}$

the duration from the start of instance k of task i to the next start of instance l of task j to be between $l^{\min}_{ijkl}$ and $l^{\max}_{ijkl}$. Let the set $\mathcal{D}$ be the set of all dependency indices. For dependency $(i, j, k, l) \in \mathcal{D}$ introduce the continuous variable

$$u_{ijkl} = \text{the length of a dependency,}$$

and the binary variable

$$y^{\mathrm{D}}_{ijkl} = \begin{cases} 1, & \text{if instance } k \text{ of task } i \text{ starts before instance } l \text{ of task } j \\ & \text{in a major frame,} \\ 0, & \text{if instance } l \text{ of task } j \text{ starts before instance } k \text{ of task } i \\ & \text{in a major frame.} \end{cases}$$

The minimum and maximum length of a dependency $(i, j, k, l) \in \mathcal{D}$ is enforced by the constraints

$$u_{ijkl} = x_j + lp_j - (x_i + kp_i) + Py^{\mathrm{D}}_{ijkl}, \tag{3}$$

$$l^{\min}_{ijkl} \leq u_{ijkl} \leq l^{\max}_{ijkl}. \tag{4}$$

Note that the length of a dependency is measured differently depending on the order of the instances within a major frame. This is illustrated in Fig. 3.

2.5 AM-scheduling

Scheduling of the AMs involves sequencing the tasks while respecting idle times between them. Let the set $\mathcal{II}^{\mathrm{O}}_h \subseteq \{(i, i') \in \mathcal{I}_h \times \mathcal{I}_h : i < i'\}$ include all pairs of task indices on AM h that with respect to their respective release times and deadlines can be placed such that they overlap or violate the idle times between them, $h \in \mathcal{H}^{\mathrm{AM}}$.

For $h \in \mathcal{H}^{\mathrm{AM}}$, $(i, i') \in \mathcal{II}^{\mathrm{O}}_h$, introduce the binary variable

$$y_{ii'} = \begin{cases} 1, & \text{if task } i \text{ starts before task } i', \\ 0, & \text{if task } i' \text{ starts before task } i. \end{cases}$$

The minimum idle time between tasks is order specific and for the pair of tasks $(i, i') \in \mathcal{II}_h^{\mathrm{O}}$, $h \in \mathcal{H}^{\mathrm{AM}}$, it is denoted by $l_{ii'}^{\mathrm{idle}}$. The AM-sequencing constraints

$$x_i + e_i + l_{ii'}^{\mathrm{idle}} - (t_i^{\mathrm{d}} + l_{ii'}^{\mathrm{idle}} - t_{i'}^{\mathrm{r}})(1 - y_{ii'}) \leq x_{i'}, \quad (i, i') \in \mathcal{II}_h^{\mathrm{O}},\ h \in \mathcal{H}^{\mathrm{AM}}, \tag{5}$$

$$x_{i'} + e_{i'} + l_{i'i}^{\mathrm{idle}} - (t_{i'}^{\mathrm{d}} + l_{i'i}^{\mathrm{idle}} - t_i^{\mathrm{r}})y_{ii'} \leq x_i, \quad (i, i') \in \mathcal{II}_h^{\mathrm{O}},\ h \in \mathcal{H}^{\mathrm{AM}}, \tag{6}$$

make the start times of the tasks comply with the order in which they are sequenced.

2.6 CN-scheduling

For the communication message scheduling, let $\mathcal{M}$ be an ordered set of CN-message indices and, further, let $\mathcal{N}$ be an ordered set of CN-slot indices, where the order is with respect to the location in a major frame. For $n \in \mathcal{N}$, $m \in \mathcal{M}$, introduce the binary variable

$$z_{nm} = \begin{cases} 1, & \text{if message } m \text{ is assigned to CN-slot } n, \\ 0, & \text{otherwise.} \end{cases}$$

Each CN-message is assigned to a CN-slot by the constraint

$$\sum_{n \in \mathcal{N}} z_{nm} = 1, \quad m \in \mathcal{M}, \tag{7}$$

while the constraint

$$\sum_{m \in \mathcal{M}} l_m^{\mathrm{msg}} z_{nm} \leq l_n^{\mathrm{slot}}, \quad n \in \mathcal{N}, \tag{8}$$

ensures that the capacity of each slot is respected, where l_n^{slot} denotes the capacity of CN-slot $n \in \mathcal{N}$ and l_m^{msg}, denotes the amount of capacity that CN-message $m \in \mathcal{M}$ claims in a slot.

2.6.1 CN-message order

To facilitate some of the modelling to follow, an order between the CN-messages needs to be established and co-allocation of CN-messages needs to be managed. Co-allocation of CN-messages refers to that two or more messages are assigned to the same CN-slot, something which implies that certain tasks are merged on the CMs. To avoid symmetries in the model with respect to co-allocation of messages, and without loss of generality, co-allocated CN-messages are forced to be placed in a slot in ascending order with respect to their CN-message indices.

Introduce, for $m',\ m \in \mathcal{M} : m < m'$, the binary variables

$$y_{mm'}^{\mathrm{S}} = \begin{cases} 1, & \text{if message } m \text{ is placed before message } m', \\ 0, & \text{if message } m' \text{ is placed before message } m, \end{cases}$$

and

$$w_{mm'} = \begin{cases} 1, & \text{if message } m \text{ is placed before message } m' \text{ in a slot,} \\ 0, & \text{otherwise.} \end{cases}$$

Further, let $n_m^{\min}$ and $n_m^{\max}$ denote the least and the greatest CN-slot index, respectively, that is eligible for message $m \in \mathcal{M}$. For $m' \in \mathcal{M}$, $m \in \mathcal{M} : m < m'$ introduce the constraints

$$\sum_{n\in\mathcal{N}} nz_{nm} + 1 - w_{mm'} - (n_m^{\max} - n_{m'}^{\min} + 1)(1 - y_{mm'}^{\mathrm{S}}) \leq \sum_{n\in\mathcal{N}} nz_{m'n}$$
$$\leq \sum_{n\in\mathcal{N}} nz_{nm} + (n_{m'}^{\max} - n_m^{\min})(1 - w_{mm'}), \tag{9}$$
$$\sum_{n\in\mathcal{N}} nz_{m'n} + 1 - (n_{m'}^{\max} + 1 - n_m^{\min})y_{mm'}^{\mathrm{S}} \leq \sum_{n\in\mathcal{N}} nz_{nm}, \tag{10}$$
$$w_{mm'} \leq y_{mm'}^{\mathrm{S}}, \tag{11}$$

to relate both to the order of the messages and co-allocation of messages to the assignment of CN-messages to CN-slots, so that the values of the involved variables comply with their definitions.

2.6.2 Time restrictions for CN-tasks

There are four types of tasks involved in communicating a CN-message, two on the sending CM and two on each of the receiving CMs. These tasks are henceforth referred to as CN-tasks and the elements of the index set $\mathcal{K} = \{1, 2, 3, 4\}$ are used to refer to which type of CN-task is considered. Let the set $\mathcal{I}_k^{\mathrm{K}}$ include all tasks of type $k \in \mathcal{K}$, and introduce the set $\mathcal{I}_m^{\mathrm{M}}$ to include the CN-tasks that are involved in communicating CN-message $m \in \mathcal{M}$.

For each CN-slot and type of CN-task, there are release times and deadlines that the corresponding CN-tasks must comply with if a CN-message is assigned to this slot. Denote the release time and deadline that task $i \in \mathcal{I}_m^{\mathrm{M}} \cap \mathcal{I}_k^{\mathrm{K}}$ must comply with if CN-message m is assigned to slot n by $t_{nk}^{\text{r-msg}}$ and $t_{nk}^{\text{d-msg}}$, respectively, $n \in \mathcal{N}$, $k \in \mathcal{K}$, $m \in \mathcal{M}$. The constraint

$$\sum_{n\in\mathcal{N}} t_{nk}^{\text{r-msg}} z_{nm} \leq x_i \leq \sum_{n\in\mathcal{N}} t_{nk}^{\text{d-msg}} z_{nm} - e_i, \quad i \in \mathcal{I}_m^{\mathrm{M}} \cap (\mathcal{I}_2^{\mathrm{K}} \cup \mathcal{I}_3^{\mathrm{K}}),\ m \in \mathcal{M}, \tag{12}$$

ensures that these times are respected.

For CN-tasks of type 2, it holds that each interval $[t_{iq}^{\mathrm{r}}, t_{iq}^{\mathrm{d}}]$, coincides with an interval $[t_{n2}^{\text{r-msg}}, t_{n2}^{\text{d-msg}}]$ for some $n \in \mathcal{N}$, and vice versa, and that $e_i = t_{iq}^{\mathrm{d}} - t_{iq}^{\mathrm{r}}$ holds, $q \in \mathcal{Q}_i$, $i \in \mathcal{I}_2^{\mathrm{K}}$. For this reason, choosing a sub-interval for a CN-task of type 2 and assigning a CN-message to a CN-slot is in fact the same decision, expressed by the constraint

$$\alpha_{iq} = z_{nm}, \quad q \in \mathcal{Q}_i :\ t_{iq}^{\mathrm{r}} = t_{n2}^{\text{r-msg}},\ i \in \mathcal{I}_2^{\mathrm{K}} \cap \mathcal{I}_m^{\mathrm{M}},\ m \in \mathcal{M},\ n \in \mathcal{N}. \tag{13}$$

In the model, these variables are, for clarity reasons, introduced separately.

2.6.3 Merging of CN-tasks

When CN-messages are co-allocated in a slot, their corresponding CN-tasks that are of the same type and on the same CM will be merged. This is possible since the execution requirement of CN-tasks constitutes of two terms, the first being initialisation and the second being a message specific part, and for tasks that are merged, this initialisation is omitted for all but the first of the merged task. For CN-task $i \in \mathcal{I}_m^{\mathrm{M}}$, $m \in \mathcal{M}$, denote initialisation by e_i^{init} and the CN-message specific part by $e_i - e_i^{\text{init}}$. For CN-tasks of type 1, 3, and 4 it hold that $e_i^{\text{init}} \approx e_i/2$ and for CN-tasks of type 2 it hold that $e_i^{\text{init}} = e_i$.

More specifically, when merging a set of CN-tasks, these tasks are placed in the order according to their CN-message index and such that, for all but the first task, the initialisation part of the execution requirement overlaps the preceding task. To handle merging in the

Relaxed problem, an order between each pair of tasks that potentially can be merged is introduced, even if the majority of the tasks are sequenced by the CM-sequencing component.

For $m' \in \mathcal{M}, m \in \mathcal{M} : m < m', h \in \mathcal{H}^{\mathrm{CM}}, k \in \mathcal{K}$, introduce the set $\mathcal{II}^{\mathrm{M}}_{khmm'} = \{(i, i') : i \in \mathcal{I}^{\mathrm{K}}_k \cap \mathcal{I}_h \cap \mathcal{I}^{\mathrm{M}}_m,\ i' \in \mathcal{I}^{\mathrm{K}}_k \cap \mathcal{I}_h \cap \mathcal{I}^{\mathrm{M}}_{m'}\}$ that contains a pair of tasks that potentially can be merged and let the set $\mathcal{II}^{\mathrm{M}} = \cup_{m' \in \mathcal{M}} \cup_{m \in \mathcal{M}: m < m'} \cup_{h \in \mathcal{H}^{\mathrm{CM}}} \cup_{k \in \mathcal{K}} \mathcal{II}^{\mathrm{M}}_{khmm'}$ include all such pairs of tasks. For $m' \in \mathcal{M}, m \in \mathcal{M} : m < m', h \in \mathcal{H}^{\mathrm{CM}}, k \in \mathcal{K}, (i, i') \in \mathcal{II}^{\mathrm{M}}_{khmm'}$, introduce the binary variable

$$y^{\mathrm{M}}_{ii'} = \begin{cases} 1, & \text{if task } i \text{ starts before task } i', \\ 0, & \text{if task } i' \text{ starts before task } i, \end{cases}$$

and note that the case $y^{\mathrm{M}}_{ii'} = 1$ includes the possibility that i and i' are merged, even if they not necessarily are. For $m' \in \mathcal{M}, m \in \mathcal{M} : m < m'$ introduce the constraints

$$\begin{aligned} & x_i + e_i - e^{\mathrm{init}}_{i'} w_{mm'} - (t^{\mathrm{d}}_i - t^{\mathrm{r}}_{i'})(1 - y^{\mathrm{M}}_{ii'}) \le x_{i'} \le x_i + e_i - e^{\mathrm{init}}_{i'} \\ & \quad + \sum_{i'' \in \mathcal{I}^{\mathrm{M}}_{m''} \cap \mathcal{I}_h \cap \mathcal{I}^{\mathrm{K}}_k,\ m'' \in \mathcal{M}: m < m'' < m'} (e_{i''} - e^{\mathrm{init}}_{i''}) w_{mm''} \\ & \quad + (t^{\mathrm{d}}_{i'} - e_{i'} - t^{\mathrm{r}}_i - e_i + e^{\mathrm{init}}_{i'})(1 - w_{mm'}), \\ & (i, i') \in \mathcal{II}^{\mathrm{M}}_{khmm'},\ k \in \mathcal{K},\ h \in \mathcal{H}^{\mathrm{CM}}, \end{aligned} \tag{14}$$

$$x_{i'} + e_{i'} - (t^{\mathrm{d}}_{i'} - t^{\mathrm{r}}_i) y^{\mathrm{M}}_{ii'} \le x_i, \quad (i, i') \in \mathcal{II}^{\mathrm{M}}_{khmm'},\ k \in \mathcal{K},\ h \in \mathcal{H}^{\mathrm{CM}}, \tag{15}$$

$$w_{mm'} \le y^{\mathrm{M}}_{ii'}, \quad (i, i') \in \mathcal{II}^{\mathrm{M}}_{khmm'},\ k \in \mathcal{K},\ h \in \mathcal{H}^{\mathrm{CM}}, \tag{16}$$

of which the first two relate the start times of the tasks to the order between the tasks, with the first one taking into account the merging. Note that constraint (14) and constraint (15) together make sure that the tasks do not overlap, except in the sense of merging. Constraint (16) forces CN-tasks of co-allocated CN-messages to be in the same order as the messages.

2.6.4 CN-task order

For a pair of CN-messages m and m', where $m, m' \in \mathcal{M}$, that are received by the same CM, the relative order between the CN-tasks of type 3 must be the same as that between message m and m'. This order is enforced by the constraint

$$y^{\mathrm{S}}_{mm'} = y^{\mathrm{M}}_{ii'}, \quad (i, i') \in \mathcal{II}^{\mathrm{M}}_{3hmm'},\ h \in \mathcal{H}^{\mathrm{CM}},\ m' \in \mathcal{M},\ m \in \mathcal{M} : m < m'. \tag{17}$$

2.7 CM-scheduling

This section describes the different aspects of CM-scheduling used in the constraint generation procedure.

2.7.1 Relaxed CM-scheduling

That the CM-scheduling is relaxed refers to that this model component does not contain anything that prevents tasks from overlapping, except in the case of tasks that potentially can be merged. What restricts how the tasks on the CMs can be placed are constraints from other model components and the assignment of sub-intervals to tasks, as introduced in Sect. 2.3. This section presents how the formulation with respect to the assignment of sub-intervals

to tasks can be strengthened to cut off solutions that cannot be feasible when the tasks are required not to overlap.

For a feasible schedule, it holds that for any interval of time, the length of this interval is at least that of the sum of the execution requirements of the tasks (or the part of the tasks) that execute within it. This property can be used to strengthen the formulation of the Relaxed problem by introducing capacity restrictions for some relevant choices of intervals. We here consider intervals that are formed by combining a sub-interval release time of a task and a sub-interval deadline of a task. Introduce for $h \in \mathcal{H}^{\mathrm{CM}}$, capacity constraints indexed by the set $\mathcal{R}_h$ and let $\mathcal{I}_r^{\mathrm{cap}}$, $r \in \mathcal{R}_h$, be the indices of a set of tasks such that for its sub-intervals, indexed by the set $\mathcal{Q}_{ir}$, it holds that $\max_{i\in\mathcal{I}_r^{\mathrm{cap}}} \max_{q\in\mathcal{Q}_{ir}} t_{iq}^{\mathrm{d}} - \min_{i\in\mathcal{I}_r^{\mathrm{cap}}} \min_{q\in\mathcal{Q}_{ir}} t_{iq}^{\mathrm{r}} < \sum_{i\in\mathcal{I}_r^{\mathrm{cap}}} e_i$. The constraint

$$\sum_{i\in\mathcal{I}_r^{\mathrm{cap}}} \sum_{q\in\mathcal{Q}_{ir}} e_i \alpha_{iq} - \sum_{(i,i')\in\mathcal{II}_{khmm'}^{\mathrm{M}}\cap\mathcal{I}_r^{\mathrm{cap}},\ k\in\mathcal{K},\ m'\in\mathcal{M},\ m\in\mathcal{M}:m<m'} e_{i'}^{\mathrm{init}} w_{mm'} \leq \max_{i\in\mathcal{I}_r^{\mathrm{cap}}} \max_{q\in\mathcal{Q}_{ir}} t_{iq}^{\mathrm{d}} - \min_{i\in\mathcal{I}_r^{\mathrm{cap}}} \min_{q\in\mathcal{Q}_{ir}} t_{iq}^{\mathrm{r}}, \quad r \in \mathcal{R}_h,\ h \in \mathcal{H}^{\mathrm{CM}}, \tag{18}$$

yields the restriction that the total execution requirement of the tasks placed within such an interval does not exceed its length,

For Constraint (18) to be valid also in the case when CN-tasks are merged, it is assumed that they reduce the capacity required by the tasks placed in the section by the initialisation length if they are merged, even if they are not placed in the section. This adjustment somewhat weakens the constraint.

2.7.2 Sub-interval restriction

In a solution to the Relaxed problem, let the notation $\bar{q}_i$, $\bar{q}_i \in \mathcal{Q}_i$, refer to the sub-interval that task i is assigned, and restrict the release time and deadline of task i to $t_{i\bar{q}_i}^{\mathrm{r}}$ and $t_{i\bar{q}_i}^{\mathrm{d}}$, respectively, $i \in \mathcal{I}_h$, $h \in \mathcal{H}^{\mathrm{CM}}$. These restrictions allow the set of tasks that are at risk to overlap to be substantially reduced and this is exploited in the Subproblem.

2.7.3 CM-sequencing

Every set of tasks that have been placed such that they are at risk to overlap on CM $h \in \mathcal{H}$ is referred to as a sequence. Let $\mathcal{S}_h$ denote the set of sequences on CM h, and note that tasks that are in different sequences cannot overlap with respect to their release times and deadlines. Let the set $\mathcal{II}_s^{\mathrm{O}}$ index all pairs of tasks in sequence $s \in \mathcal{S}_h$.

For $s \in \mathcal{S}_h$, $h \in \mathcal{H}^{\mathrm{CM}}$, introduce the binary variable

$$\beta_s = \begin{cases} 1, & \text{if there is no overlap between tasks in sequence } s, \\ 0, & \text{otherwise,} \end{cases}$$

and for $(i, i') \in \mathcal{II}_s^{\mathrm{O}}$, the binary variable

$$y_{ii'} = \begin{cases} 1, & \text{if task } i \text{ finishes before task } i' \text{ starts,} \\ 0, & \text{no requirements on the relation between task } i \text{ and task } i'. \end{cases}$$

For sequence $s \in \mathcal{S}_h$, $h \in \mathcal{H}^{\mathrm{CM}}$ introduce the CM-sequencing constraints

$$[\text{CM-seq}]_s \qquad \beta_s \leq y_{ii'} + y_{i'i} \leq 1, \quad (i, i') \in \mathcal{II}_s^{\mathrm{O}}, \tag{19}$$

$$x_i + e_i - (t_i^{\mathrm{d}} - t_{i'}^{\mathrm{r}})(1 - y_{ii'}) \leq x_{i'}, \quad (i, i') \in \mathcal{II}_s^{\mathrm{O}}, \tag{20}$$

$$x_{i'} + e_{i'} - (t_{i'}^{\mathrm{d}} - t_i^{\mathrm{r}})(1 - y_{i'i}) \leq x_i, \quad (i, i') \in \mathcal{II}_s^{\mathrm{O}}. \tag{21}$$

If $\beta_s = 1$ for sequence s, the tasks in this sequence are forced not to overlap by requiring that one task starts before the other by constraint (19), and that the start times of the tasks comply with this order by constraints (20) and (21). Otherwise if $\beta_s = 0$ for a sequence s, the tasks in this sequence are allowed to overlap.

In order to prefer to schedule tasks such that they do not overlap, we introduce the objective function of the Subproblem as

$$\max \sum_{h \in \mathcal{H}^{\mathrm{CM}}} \sum_{s \in \mathcal{S}_h} \beta_s. \tag{22}$$

If a Subproblem solution is found where none of the sequences have tasks that overlap, a valid schedule is found. Otherwise the values of the β_s variables indicate which sequences contain overlapping tasks.

2.7.4 Generated sequences

A sequence that is not free from overlaps between tasks in a solution to the Subproblem can be used for feedback to improve future solutions. If a sequence is chosen to be used as feedback, it is referred to as a generated sequence. Let the set $\mathcal{S}^{\mathrm{gen}}$ index such sequences. For each generated sequence $s \in \mathcal{S}^{\mathrm{gen}}$, the CM-sequencing constraint [CM-seq]$_s$ is combined with the requirement that $\beta_s = 1$, thereby forcing the tasks in generated sequence s not to overlap. The generated sequences constraints are added both to the Relaxed problem and to the Subproblem.

2.8 Objective functions for the relaxed problem

In the original problem formulation, any feasible solution is considered equally good. In the constraint generation procedure, however, objective functions are used to guide the search. This section introduces the two objective functions that are used in the Relaxed problem. The first objective, called the sub-interval space objective, is used in the first Relaxed problem to obtain a solution where tasks preferably are put in sub-intervals where they have a lot of space. The second objective, called the stabilise objective, is used in subsequent Relaxed problems, and it aims at assigning as many tasks as possible to the sub-interval of the previous Relaxed problem solution. To use different types of objective function in the first and subsequent Relaxed problems has turned out to be important to obtain computational efficiency.

2.8.1 Sub-interval space objective

The sub-interval space objective tries to assign tasks to sub-intervals where they have a lot of space. Each sub-interval $q \in \mathcal{Q}_i$ is given a reward m_{iq} if task $i \in \mathcal{I}_h$, $h \in \mathcal{H}^{\mathrm{CM}}$, is assigned to it. For each sub-interval $q \in \mathcal{Q}_i$ of task $i \in \mathcal{I}_h$, $h \in \mathcal{H}^{\mathrm{CM}}$, the reward is calculates as

$$m_{iq} = \min \left(\frac{t_{iq}^{\mathrm{d}} - t_{iq}^{\mathrm{r}} - p_i}{m^{\mathrm{obj}} \max_{q' \in \mathcal{Q}_i} (t_{iq'}^{\mathrm{d}} - t_{iq'}^{\mathrm{r}} - p_i)}, 1 \right), \tag{23}$$

where m^{obj}, $0 \leq m^{\text{obj}} \leq 1$, is a parameter used to set a threshold for when a sub-interval is considered to have a lot of space or not. The objective function is then be stated as

$$\max \sum_{h \in \mathcal{H}^{\text{CM}}} \sum_{i \in \mathcal{I}_h} \sum_{q \in \mathcal{Q}_i} m_{iq} \alpha_{iq}. \tag{24}$$

2.8.2 Stabilise objective

The stabilise objective maximises the number of tasks that are placed in the same sub-interval as in the solution to the previous Relaxed problem. Let $\bar{q}_i$ denote the sub-interval that task i was assigned to during the previous iteration of the Relaxed problem, $i \in \mathcal{I}$. The stabilise objective can then be stated as

$$\max \sum_{i \in \mathcal{I}} \alpha_{i\bar{q}_i}. \tag{25}$$

2.9 Implementation of the constraint generation procedure

The pseudo-code for the exact procedure can be found in Algorithm 1. Before the constraint generation procedure, we apply pre-processing components as described in Blikstad et al. (2018).

Data: A scheduling instance
Result: A valid schedule or a conclusion of infeasibility
Conduct pre-processing
while *At least one generated sequence is new* **do**
  Solve the Relaxed problem
  if *Relaxed problem is infeasible* **then**
    Conclude that the problem is infeasible
    Break
  end
  Restrict all non-fixed CM-tasks to their assigned sub-intervals
  Solve the Subproblem
  if *There are tasks that overlap* **then**
    Add at least one generated sequence to the Relaxed problem and to the Subproblem
  end
end

Algorithm 1: Overview of the exact solution approach

3 Matheuristic solution approach

The exact approach presented in the previous section is the backbone also for the matheurstic to be introduced in this section. The matheuristic components comprise of an ALNS search for solving the Relaxed problem and a constructive heuristic for creating an initial solution to be used by the ALNS. Pseudo-code for this matheuristic approach is found in Algorithm 2.

```
Data: A scheduling instance
Result: A valid schedule
Conduct pre-processing
Find an initial solution to the Repair model
while At least one generated sequence is new do
    while Solution has penalty or any of the last k iterations were successful do
        for i ← 1 to n do
            Select a destroy operator at random
            Select a repair operator based on the current solution and whether or not to diversify
            Construct a Repair model based on the destroy operator, the repair operator, and the
            previous solution
            Solve the Repair model
        end
        Determine whether or not to diversify
    end
    Restrict all non-fixed CM-tasks to their assigned sub-intervals in the current solution
    Solve the Subproblem
    if There are tasks that overlap then
        Add at least one generated sequence to the Repair problem and to the Subproblem
    end
end
```

Algorithm 2: Overview of the matheuristic

3.1 Outline of the ALNS

In the model used in the ALNS, many constraints are treated as soft and a violation of a constraint is penalised in the objective. By associating different weights with the penalty and the objective function of the Relaxed problem, it is possible to choose between focus on feasibility or objective value.

During the ALNS, a solution is defined by that each task is assigned to one of its sub-intervals and that all other variables of the relaxed problem have values obtained from solving the penalised model by a MIP-solver. A destroy operation releases some of the tasks from their sub-intervals and makes them eligible for re-assignment. This re-assignment is done in the repair operation by applying a MIP-solver to a repair model, which is the penalised model with weighted objective, where only the released tasks can be re-assigned new sub-intervals.

The master level heuristic guiding our search is based on local search with the alteration that different repair operators use different objective functions.

The destroy operators are designed to identify tasks that cause costs in the current solution, or to select tasks at random. For the destroy operators that identify tasks that cause costs, a subset of the most costly tasks is selected to be released and these are referred to as key tasks. Since the re-assignment of key tasks might be restricted by constraints involving tasks related to them, further tasks, referred to as related tasks, are also released.

Which repair operator the ALNS uses depends on whether the current solution is feasible or not with respect to the Relaxed problem. This is inspired by the concept of strategic oscillation, see Glover (1977) and Glover and Hao (2011). The selection of repair operator is done at each iteration and is part of the adaptiveness of the algorithm.

The search is divided into time epochs and each of these consists of a number of iterations. At the end of a time epoch, it is decided if the ALNS should exit or not. If the search is continued, it is further decided if a diversify repair operator shall be applied, or if the search should continue as before. This also contribute to an adaptive search behaviour.

3.2 Repair model

This section describes the MIP model, referred to as the Repair model, that is used in each iteration of the ALNS. It is a major difficulty to find a feasible solution to the Relaxed problem, and therefore the Repair model is a reformulation of the Relaxed problem where a majority of the constraints are treated as soft and violations of them are penalised in the objective function. For this purpose, we introduce the penalty variables

- $\gamma^{\text{CN-slot}}$: violation of CN-slot capacity,
- $\gamma^{\text{CN-time}}$: violation of time restriction for CN-tasks,
- $\gamma^{\text{CN-merge}}$: violation of merging of CN-tasks,
- γ^{cap}: violation of relaxed CM-scheduling,
- γ^{gen}: violation of generated sequences,

that measure the violation of feasibility of each constraint type. The total violation of feasibility of each constraint type along with the maximum violation of each constraint type are given a weight. Let γ^{infeas} denote the weighted sum of the total and maximum violation of each constraint type. Three types of constraints are always obeyed during the ALNS: the assignment of each task to a sub-interval constraints [constraints (27)–(29)], the AM-scheduling constraints (Sect. 2.5), and the Precedence relations constraints (Sect. 2.4).

In order to balance the need for feasibility of a solution with the Relaxed problem objective, the weights w^{infeas} and w^{obj} are introduced for the Relaxed problem objective and the total weighted constraint violation, respectively. The weights are decided by the repair operator and are used in objective function of the Repair model

$$\min w^{\text{infeas}}\gamma^{\text{infeas}} + w^{\text{obj}}\delta^{\text{obj}}, \tag{26}$$

where δ^{obj} denotes the Relaxed problem objective.

In each iteration of the ALNS, a set of tasks $\mathcal{I}^{\text{rel}}$ is released from their current sub-intervals and allowed to be re-assigned new sub-intervals. Each remaining task $i \in \mathcal{I} \setminus \mathcal{I}^{\text{rel}}$ is fixed to its sub-interval $\bar{q}_i$ of the previous solution. The set of tasks to be released is determined by the destroy operator and defines, together with the repair operator, the neighbourhood to be searched. In the Repair model, the common AM- and CM-scheduling constraints in Sect. 2.3 are replaced with

$$\sum_{q \in \mathcal{Q}_i} \alpha_{iq} = 1, \quad i \in \mathcal{I}, \tag{27}$$

$$\bar{\alpha}_{iq} = 1, \quad i \in \mathcal{I} \setminus \mathcal{I}^{\text{rel}}, \tag{28}$$

$$\sum_{q \in \mathcal{Q}_i} t^{\text{r}}_{iq}\alpha_{iq} \le x_i \le \sum_{q \in \mathcal{Q}_i} t^{\text{d}}_{iq}\alpha_{iq} - e_i, \quad i \in \mathcal{I}. \tag{29}$$

Note that the Repair model is free to decide the start times for the tasks within their assigned sub-intervals, see constraint (29).

3.3 Destroy operators

Given a solution to the Repair model, a destroy operator constructs the set of tasks $\mathcal{I}^{\text{rel}}$ that can be assigned to a new sub-interval in the next ALNS iteration. The ALNS utilises multiple destroy operators. In each iteration, one of the destroy operators is selected at random, with equal probabilities for all operators. The destroy operators are categorised into two types:

precedence relations operator and random operator. The details of each type are described in the following sections.

3.3.1 Precedence relations operator

The precedence relations operator is based on the idea of related removal, introduced in Shaw (1998). Given a solution to the Repair model, tasks that cause a cost in the current solution are chosen. Such tasks are referred to as key tasks. From the set of key tasks, the precedence relations operator orders related tasks in increasing distance from the key tasks. The distance between two tasks is defined as the length of the shortest sequence of dependencies through which the tasks are connected, or as infinity if no such sequence exists. The set of key tasks together with the related tasks with the shortest distance to the key tasks are then released from their current sub-interval.

3.3.2 Random operator

The random operator simply selects tasks at random to be released from their current sub-interval. In Ropke and Pisinger (2006), this operator is recommended to be included in the ALNS to improve diversification.

3.4 Repair operators

The repair operator controls the balance between the of focus on feasibility and the Relaxed problem objective by assigning values to the weights in the Repair model objective.

3.4.1 Feasibility repair

The feasibility repair focuses on achieving feasibility by setting $w^{\text{obj}} = 0$ and $w^{\text{infeas}} = 1$. The objective function of the Repair model then becomes

$$\min \gamma^{\text{infeas}}. \tag{30}$$

3.4.2 Diversify repair

The diversify repair tries to improve the solution with respect to the Relaxed problem objective. This is done by choosing a small value for the weight of the constraint violations w^{infeas} and setting $w^{\text{obj}} = 1$. The objective of the Repair model then becomes

$$\min w^{\text{infeas}} \gamma^{\text{infeas}} + \delta^{\text{obj}}. \tag{31}$$

3.4.3 Objective repair

The objective repair is used when the current solution is a feasible solution to the Relaxed problem. The repair operator tries to improve the Relaxed problem objective while maintaining feasibility with respect to the Relaxed problem. The objective function of the Repair model is

$$\min \gamma^{\text{obj}}, \tag{32}$$

with the additional constraint

$$\gamma^{\text{infeas}} = 0. \tag{33}$$

3.5 Adaptation

The adaptation part of the ALNS is done at two levels: in each iteration and at the end of each time epoch (which corresponds to a given number of iterations). The primary decisions of the adaptation are to determine if the focus should be on feasibility or the relaxed problem objective and to evaluate if the ALNS is stuck in a local optima, and if so, determine what the corresponding action should be.

With respect to local optimality, we evaluate, at the end of each time epoch, if any of the last 5 iterations were successful. If not, we act as follows:

- If the current solution is feasible: we exit the ALNS and enter the Subproblem;
- If the current solution is infeasible: we apply the diversify repair operator.

If one or more of the last 5 iterations were successful, we continue the search as usual. The decision to focus on feasibility or on the relaxed problem objective is made at each iteration by selecting different repair operators. This is done according to the following scheme:

- If the current solution is feasible: we apply the objective repair operator;
- If the current solution is infeasible: we apply the feasibility repair operator.

In the case we are at the end of a time epoch and has applied the diversify repair operator, this adaptation is omitted.

3.6 Creating an initial solution to the ALNS

An initial solution to the ALNS is created by applying a constructive heuristic that takes into account only some of the constraints. This is done by iteratively solving a MIP-model until the constraints that are kept hard during the ALNS are respected.

The MIP-model that we use in the constructive heuristic is called the Partition model, see Appendix B. The Partition model enforces a valid AM-schedule that respects all precedence relations. The objective function is to maximise a reward that is based on the duration between tasks on different AMs that communicate with each other. However, in a solution to this model, tasks are not necessarily placed within sub-intervals. As a remedy to this, constraints that assign tasks to sub-intervals, see Sect. 2.3, are added for the tasks that are not within feasible sub-intervals. This process is repeated and the MIP-model is re-solved until a solution to the Partition model is found in which all tasks are within allowed sub-intervals. This assignment of tasks to sub-intervals is used as the initial solution in the ALNS.

4 Results

In this section, we present the computational performance of both the exact approach and the matheuristic and compare these results to previous work.

Table 1 The number of tasks, dependencies, messages, fixed tasks, AMs, and CMs of the Saab instances

Name	Tasks	Dependencies	Messages	Fixed tasks	AMs	CMs
Instance I	6538	1088	64	4220	2	2
Instance II	14,186	9784	96	7640	6	5
Instance III	19,919	12,616	96	10,500	8	7
Instance IV	20,461	30,245	240	4284	2	2
Instance V	26,268	30,374	1032	4932	3	3
Instance VI	45,026	48,862	2616	11,148	10	8

4.1 Instances

The instances used for the computational evaluation were derived together with the industrial partner Saab. They are designed such that they are relevant for future avionic systems. One of the sets, called Saab instances, contains six instances with specific properties that are important for Saab, and these instances are therefore not made publicly available. The other set contains 120 public instances[2] and these have a wider range of industrially relevant properties.

4.1.1 Saab instances

The original motivation of this work was to solve the instances provided by Saab and the three smallest in this set, Instances I–III, were introduced in Blikstad et al. (2018). For these instances, the number of dependencies differ compared to what was presented in Blikstad et al. (2018) because of the modelling changes presented in this paper. Instances IV, V and VI are new and represent instances that are beyond the scope of what could be solved in previous work. The number of tasks, dependencies, messages, fixed tasks, AMs and CMs can be found in Table 1. Note that even if the total number of tasks is of the same magnitude for Instances III–V, there is a difference in how many modules the tasks are distributed on. Since Instances IV–V have fewer modules, they are more challenging to solve. Instance VI is clearly the most challenging due to its large number of tasks and messages.

4.1.2 Public instances

The public instances are divided into the four categories A–D, with 30 instances in each category. Table 2 provides, for each instance category, the mean values of the number of tasks, dependencies, messages, fixed tasks, AMs, and CMs, respectively. The relationships between the number of tasks, messages, fixed tasks, dependencies, and modules are illustrated in Fig. 4.

Category A contains instances of about the same size and difficulty as the smallest of the Saab instances, while both Category B and C contains instances designed to be of medium difficulty. Instances in Category D are constructed to be very challenging due to the large numbers of tasks and messages. Worth noting is that the public instances have been constructed in such a way that we do not know if a feasible schedule exists or not. Especially for Category D, we consider each occurrence of solving an instance to be an accomplishment

[2] https://gitlab.liu.se/eliro15/avionics_inst/tree/master.

Table 2 Mean values of key attributes of the different categories of public instances

Category	Tasks	Dependencies	Messages	Fixed tasks	AMs	CMs
Category A	4932	9516	172	1359	2.3	2.0
Category B	11,699	22,170	447	1989	2.3	2.0
Category C	20,037	37,707	908	3452	4.8	4.1
Category D	41,655	79,503	1923	6392	9.4	8.2

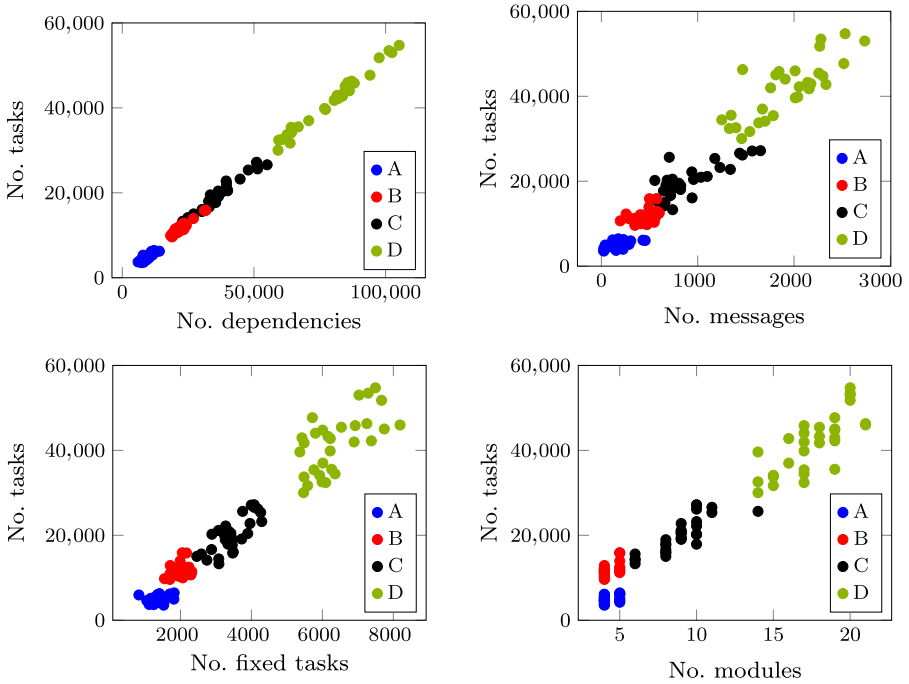

Fig. 4 Relations between the number of tasks, dependencies, messages, fixed tasks, and modules in the public instances

and together with the instance data,[3] we provide the information if an instance has ever been solved. The largest instance in this set contains 54,731 tasks and 2530 messages and this has been solved.

Instances in Category B–D have the important characteristic that they are unbalanced with respect to that certain CMs are assigned more tasks and handle more messages compared to other CMs in the system. The instances in Category A are not given this property, to keep them simpler.

[3] https://gitlab.liu.se/eliro15/avionics_inst/tree/master.

Table 3 The time-out for the scheduling tool on the different instances

Time-out	Saab instances	Public instances
6 h	Instance I–III	Category A
36 h	Instance IV–V	Category B
48 h	–	Category C
72 h	Instance VI	Category D

4.2 Test settings

We evaluate the performance of the exact approach and the matheuristic by running the scheduling tool on a number of instances, using as similar settings as possible for the exact approach and the matheuristic. Because of the importance of the Saab instances, we ran both the exact approach and the matheuristic five times, initialised with different seeds. For the public instances, the exact approach and the matheuristic were given one attempt each. The remaining parts of this section describe the computational environment and the settings of the exact approach and the matheuristic.

4.2.1 Software and hardware specifications

The scheduling tool has been implemented using Python 3.7 and the MIP models are solved using Gurobi 8.1.1. All the tests are carried out on a computer with two Intel Xeon Gold 6130 Processors (16 cores, 2.1 GHz) and 384 GB RAM. The scheduling tool is single threaded except from the calls to Gurobi, which is allowed to use all cores.

4.2.2 Common settings for the exact approach and the matheuristic

This section describes the settings that are the same for the exact approach and the matheuristic. In both solution approaches, we use the sub-interval space objective function with the parameter $m^{\text{obj}} = 0.2$ in the first Relaxed problem. For the Subproblem, there is a maximum running time of 4 h. In order to use the time of the scheduling tool efficiently, we also used a progressive set of time-outs to, in some cases, exit the Subproblem faster. These time-outs depend on the value of the current solution and the relative MIP-gap. Sometimes an early exit in the Subproblem will yield many generated sequences, and to limit the effect from this, we do not add all sequences. When not all sequences are added, we choose the ones with the least amount of tasks first, and quit when $\sum_{s \in \mathcal{S}^{\text{gen}}} |\mathcal{II}_s^{\text{O}}| > 90{,}000$. The time-limits for the entire scheduling tool are given in Table 3.

4.2.3 Settings for the exact approach

In the exact approach, the Relaxed problem will exit when a relative MIP-gap of 10% is reached in the first iteration, and when 0% is reached in the successive iterations. Since it can be difficult to fulfil these requirements, we have an additional way to exit. If a feasible solution has been found, Gurobi gets a time-limit of 30 min to find an improved solution, otherwise we exit the search. However, if an improved solution is found within this time-limit, it is reset to 30 min and the search is continued.

Table 4 An overview of the destroy operators used in the ALNS along with their settings

Operator	Key task selection strategy	No. dependency steps	No. unlocked tasks
Precedence 1	Worst	2	8000
Precedence 2	Worst	4	8000
Precedence 3	Random	2	8000
Random	–	–	8000

4.2.4 Matheuristic specific settings

The ALNS is divided into time epochs of 8 iterations. The time-limit for Gurobi is 18 min in each iteration, with a more aggressive time-out if improved solutions within certain relative MIP-gaps have been found. We consider an ALNS iteration to be successful if there is a 0.1% decrease of the weighted penalty in a repair with respect to feasibility, or if there is any improvement in the objective value in the other repair operators.

In the ALNS, we use 4 different destroy operators derived from the two types of operators described in Sect. 3.3. Each of the destroy operators unlocks 4000 tasks. We use one random-based operator and three variants of the precedence relations-based destroy operator as given in Table 4. Two of precedence relations-based destroy operators identify related tasks reachable by following two dependencies. One of them selects the key tasks as those associated with the highest cost, while the other selects the key tasks at random among tasks with costs. The last precedence related destroy operator identifies related tasks reachable by following four dependencies and selects key tasks associated with the highest cost.

4.3 Performance evaluation of the exact approach and the matheuristic

The main computational results are presented through plots illustrating the number of instances solved within a certain time. The results for the Saab instances are given in Fig. 5. For Instances I–III, both the exact approach and the matheuristic find a schedule for all five seeds. For Instances IV–VI, the matheuristic finds a schedule for all seeds while the exact approach fails in all cases. By studying the results in detail, we noted that the exact approach failed for Instance V–VI because it failed to find a feasible solution to the first Relaxed problem. For Instances I–III, the computational times are shorter for the exact approach than for the matheuristic.

To compare to previous work, we study the computational times for Instance III in Blikstad et al. (2018), where a restriction of our current problem statement was addressed, and in Rönnberg (2018), which address the same problem statement as here. In Blikstad et al. (2018), the computational times range between 0.6 and 14.5 h, and in Rönnberg (2018) it is 5.2 h. These times shall be compared to that the worst of times here is < 0.1 h. All improvements can, however, not be contributed to the model enhancements, since we use different hardware and Gurobi versions, but we still consider the outcome of the model improvements to be significant.

The results for public instances of Categories A and B are illustrated in Fig. 6. Instances in Category A pose no challenge for neither the exact approach nor the matheuristic and a solution is found for all runs within the time-limit of the scheduling tool. For Category B, the exact approach and the matheuristic solves about the same number of instances within the time-limit, but with slightly faster running times by the matheuristic.

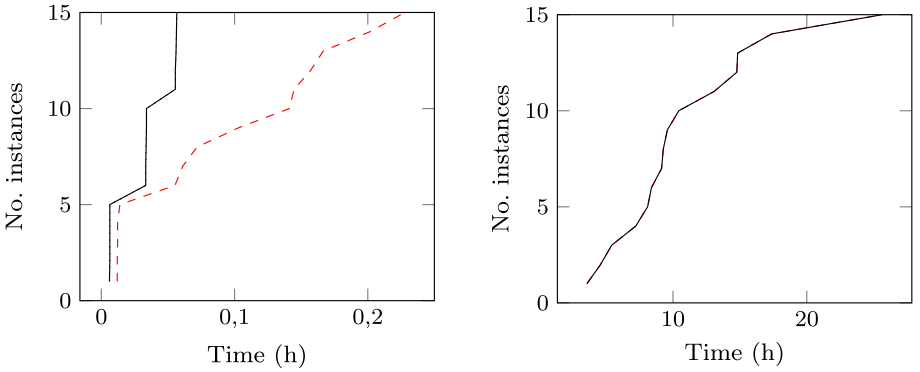

Fig. 5 The number of runs for Saab instances I–III and IV–V, respectively, that are solved within a given time by the exact approach (blue dotted line), matheuristic (red dashed line), or either of them (black solid line). Note that to the left the dotted and solid line coincides, while to the right the dashed and the solid line coincides

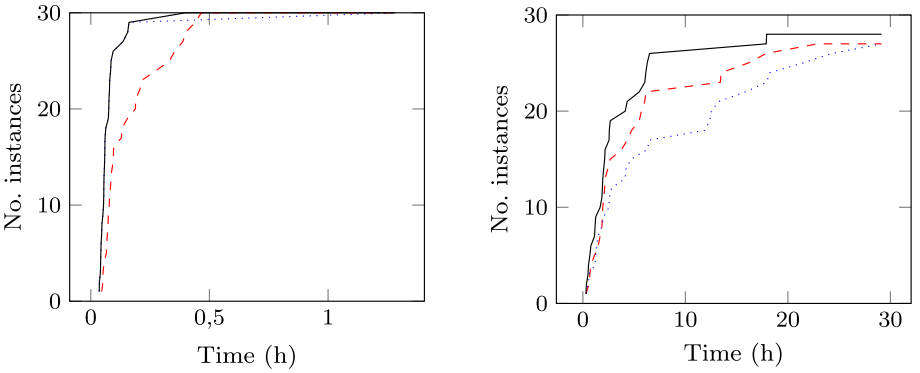

Fig. 6 The number of instances of Category A and B, respectively, that are solved within a given time by the exact approach (blue dotted line), matheuristic (red dashed line), or either of them (black solid line)

The results for instances of Categories C and D are illustrated in Fig. 7. For Category C, the matheuristic has solved 28 instances compared to the 15 by the exact approach at the final time-limit of 48 h. Among the 15 instances that were solved by the exact approach, 14 were also solved by the matheuristic.

Category D contains the most challenging instances and here we can, as for Saab instances IV–VI, see a clear benefit from the matheuristic. After about 34 h, when the exact approach solved its first instance, the matheuristic had already solved 26 instances. To further analyse the outcome for these instances, we let the scheduling tool complete its 72 h. During the remainder of the 72 h, the exact approach solved 6 additional instances and the matheuristic solved another 2. Worth noting is that all instances solved by the exact approach were also solved by the matheuristic, but with shorter computational times.

As seen in Fig. 4, there is a wide range in the number of tasks for instances in Category D. The largest instance contains 54,731 tasks and was only solved by the matheuristic. The largest instance solved by the exact approach has 36,996 tasks.

During our computational evaluations we noted that the matheuristic typically needs to generate more sequences than the exact approach in order to find a feasible schedule. Statistics

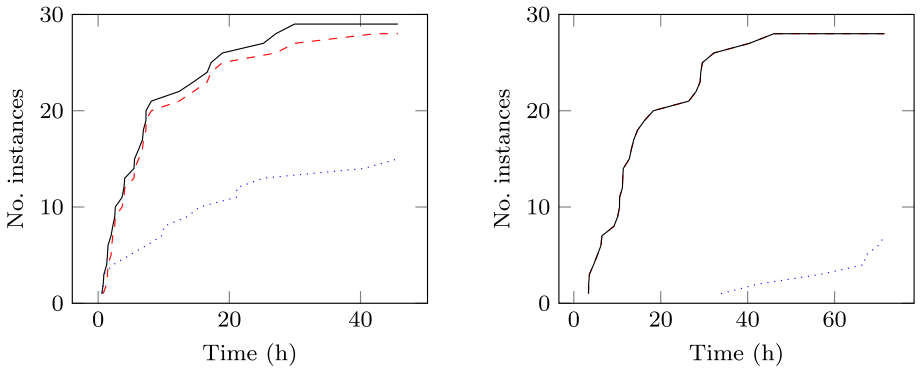

Fig. 7 The number of instances of Category C and D, respectively, that are solved within a given time by the exact approach (blue dotted line), matheuristic (red dashed line), or either of them (black solid line)

Table 5 The average number of constraint generation iterations and the number of generated sequences of the exact approach and the matheuristic on the Saab instances and the public instances

Category	No. iterations		No. generated sequences	
	Exact	Matheuristic	Exact	Matheuristic
Instance I–III	1.7	1.8	2.0	18.9
Instance IV–VI	2.0	3.7	26.0	54.7
Category A	1.4	1.8	2.2	10.9
Category B	3.0	4.1	38.8	71.3
Category C	3.2	4.2	53.3	93.2
Category D	1.9	5.5	23.6	145.4

for this is found in Table 5. We believe that the difference is due to that the exact approach produces better solutions to start the constraint generation from and that this is possible because the Relaxed problem has many solutions with the same objective value, even if the solutions are rather different in other respects. Even though the solutions obtained by Gurobi and the ALNS have similar objective values, it somehow seems that the structure of the solutions produced by Gurobi are better than those from the ALNS with respect to the constraint generation procedure. We have, however, not been able to understand the reasons for this.

4.4 Evaluation of the ALNS design

A characteristic of our ALNS is that it makes few iterations over large neighbourhoods rather than many iterations over small ones. Table 6 presents the average number of ALNS iterations made in different phases of the matheuristic. Early practical evaluations showed that we benefit from this choice and we believe that this is because of the complex constraint structure of the problem.

To study the impact of having the different destroy operators, we conducted additional tests on Category D and Saab instances IV–VI. In each test, we used only a single destroy operator and compared the result to our standard matheuristic. In all tests, all other settings were as

Table 6 The average number of ALNS iterations on the Saab instances and the public instances

Category	Total	First relaxed problem		Subsequent relaxed problems	
		Feas. sol.	Obj. val.	Feas. sol.	Obj. val.
Instance I–III	18.1	1.9	9.9	0.6	5.8
Instance IV–VI	73.1	6.1	29.1	3.7	34.2
Category A	14.7	1.3	7.3	0.8	5.4
Category B	86.8	7.3	17.6	7.2	54.7
Category C	121.2	25.2	21.3	16.6	58.2
Category D	276.7	26.5	69.8	85.5	95.0

The details of what contribute to the total number of iterations is presented for the first and subsequent Relaxed problems, respectively, and divided with respect to if they contribute to improving feasibility (feas. sol.) or objective value (obj. val.)

Table 7 The number of solved instances of Category D and the number of solved seeds for Instances IV–VI that was solved by the standard ALNS compared to the cases where only one of the four destroy operators was used

Category	Standard	Precedence 1	Precedence 2	Precedence 3	Random
Instance IV	5/5	4/5	1/5	5/5	5/5
Instance V	5/5	5/5	3/5	5/5	5/5
Instance VI	5/5	5/5	4/5	4/5	5/5
Category D	28/30	10/30	15/30	23/30	2/30

described in Sect. 4.2. Table 7 presents the number of instances or seeds that were solved when only a single destroy operator was used, together with the results from the standard matheuristic. The operator Precedence 3 performed well on Category D by solving 23 out of 30 instances. However, it failed all attempts to solve Instances IV–VI and these instances were solved for every seed by both the Random operator and the standard matheuristic. In all, we see a gain by having multiple destroy operators compared to having a single one since it enables us to catch different aspects of the problem structure and the instances that we consider.

5 Conclusions

This paper contributes by proposing approaches for solving large-scale instances of avionic scheduling problems. This is achieved by improving the model formulations used in a previously suggested constraint generation procedure and by including an ALNS to extend it into a matheuristic approach. We can now solve three practically relevant instances that were out of reach for the previous procedure. To facilitate further testing of our implemented methods, the paper also introduces a large set of publicly available avionics scheduling instances. The most challenging category contains instances with up to 54,731 tasks and 2530 messages and the largest instance among these could be solved using the matheuristic approach.

The bottleneck in the exact procedure is the difficulty to obtain feasible solutions to the Relaxed problem, and the ALNS is designed to overcome this. To balance the search for feasibility and profitable objective value of the Relaxed problem, a penalty model is used to

handle when constraints are violated. The search is made over the assignment of sub-intervals to tasks, but each such assignment corresponds to several possible solutions to this penalty model; in order to evaluate an assignment, a complete solution to the penalty model is needed. For this reason, the repair operation is to apply a MIP solver to a Repair model, which is obtained from the penalty model when only the tasks released by the destroy operation are available for re-assignment to new sub-intervals.

Each iteration of our ALNS is expensive since each neighbour, with respect to an assignment of sub-intervals to tasks, depends on that the MIP-solver also assigns values to all other variables of the Repair model. Further, we believe that because of the complex constraint structure, it is profitable to have large neighbourhoods also with respect to the number of released tasks and to compensate for these by having rather few iterations and a master level heuristic of a local search type. Also, an important aspect of the ALNS design is the alternating behaviour obtained by changing the repair operations in a systematic way.

For future work it can be of relevance to try to find smaller neighbourhoods that can be searched more efficiently and to evaluate if it is profitable to make more iterations over such neighbourhoods instead of, as in the current method, search large neighbourhoods with few iterations. A further possibility along these lines is to use heuristics for the repair operations. Another line of future research is to improve the computational efficiency with respect to the subproblem and to try to dynamically add information to strengthen the Relaxed problem without having to solve the subproblem.

Acknowledgements Open access funding provided by Linköping University. The work is part of a project funded by the Center for Industrial Information Technology (CENIIT), Project-ID 16.05. The work of Emil Karlsson is supported by the Research School in Interdisciplinary Mathematics at Linköping University.

A Repair model

The Repair model is used to define the neighbourhood in an iteration of the ALNS search. The Repair model is a reformulation of the Relaxed problem, see Sect. 2, where a majority of the constraints are treated as soft, and each task must be assigned to a sub-interval.

The weighted violation of each type of constraint

- $\gamma^{\text{CN-slot}}$: violation of CN-slot capacity,
- $\gamma^{\text{CN-time}}$: violation of time restriction for CN-tasks,
- $\gamma^{\text{CN-merge}}$: violation of merging of CN-tasks,
- γ^{cap}: violation of relaxed CM-scheduling,
- γ^{gen}: violation of generated sequences,

is defined by constraints (38)–(43). The sum of all weighted constraint violations γ^{infeas}, is defined by constraint (37).

To model the different Relaxed problem objectives, we introduce the coefficient

$$c_{iq} = \text{the reward of assigning sub-interval } q \text{ to task } i, \quad q \in \mathcal{Q}_i,\ i \in \mathcal{I}, \tag{34}$$

to define the value of the current Relaxed problem objective with constraint (36).

The Repair model is formulated as

$$\min w^{\text{infeas}}\gamma^{\text{infeas}} + w^{\text{obj}}\delta^{\text{obj}}, \tag{35}$$

subject to

[Define relaxed problem objective variable]

$$\delta^{\text{obj}} = \sum_{i\in\mathcal{I}}\sum_{q\in\mathcal{Q}_i} c_{iq}\alpha_{iq}, \tag{36}$$

[Define penalty variables]

$$\gamma^{\text{infeas}} = \gamma^{\text{cap}} + \gamma^{\text{CN-time}} + \gamma^{\text{CN-slot}} + \gamma^{\text{CN-merge}} + \gamma^{\text{gen}}, \tag{37}$$

$$\gamma^{\text{prec}} = w^{\text{M-max}}\gamma^{\text{M-max}} + w^{\text{M-min}}\gamma^{\text{M-min}} + \sum_{(i,j,k,l)\in\mathcal{D}} (w^{\max}\gamma^{\max}_{ijkl} + w^{\min}\gamma^{\min}_{ijkl}), \tag{38}$$

$$\gamma^{\text{cap}} = w^{\text{M-cap}}\gamma^{\text{M-cap}} + w^{\text{cap}}\sum_{r\in\mathcal{R}_h}\sum_{h\in\mathcal{H}^{\text{CM}}}\gamma^{\text{cap}}_r, \tag{39}$$

$$\gamma^{\text{CN-slot}} = w^{\text{M-CN-slot}}\gamma^{\text{M-CN-slot}} + w^{\text{CN-slot}}\sum_{n\in\mathcal{N}}\gamma^{\text{CN-slot}}_n, \tag{40}$$

$$\gamma^{\text{CN-time}} = w^{\text{M-r-msg}}\gamma^{\text{M-r-msg}} + w^{\text{M-d-msg}}\gamma^{\text{M-d-msg}} + \sum_{m\in\mathcal{M}}\sum_{i\in\mathcal{I}^{\text{M}}_m\cap\mathcal{I}^{\text{K}}_3} (w^{\text{r-msg}}\gamma^{\text{r-msg}}_{ik} + w^{\text{d-msg}}\gamma^{\text{d-msg}}_{ik}), \tag{41}$$

$$\gamma^{\text{gen}} = w^{\text{M-gen}}\gamma^{\text{M-gen}} + w^{\text{gen}}\sum_{s\in\mathcal{S}^{\text{gen}}}\sum_{(i,i')\in\mathcal{II}^{\text{O}}_s} (\gamma^{\text{gen}}_{ii'} + \gamma^{\text{gen}}_{i'i}), \tag{42}$$

$$\gamma^{\text{CN-merge}} = w^{\text{M-CN-m-1}}\gamma^{\text{M-CN-m-1}} + w^{\text{M-CN-m-2}}\gamma^{\text{M-CN-m-2}} + w^{\text{M-CN-m-3}}\gamma^{\text{M-CN-m-3}} + \sum_{m'\in\mathcal{M}}\sum_{m\in\mathcal{M}:m<m'}\sum_{h\in\mathcal{H}^{\text{CM}}}\sum_{k\in\mathcal{K}}\sum_{(i,i')\in\mathcal{II}^{\text{M}}_{khmm'}} (w^{\text{CN-m-1}}\gamma^{\text{CN-m-1}}_{ii'} + w^{\text{CN-m-2}}\gamma^{\text{CN-m-2}}_{ii'} + w^{\text{CN-m-3}}\gamma^{\text{CN-m-3}}_{ii'}), \tag{43}$$

[Common for AM- and CM-scheduling]

$$\sum_{q\in\mathcal{Q}_i}\alpha_{iq} = 1, \quad i\in\mathcal{I}, \tag{44}$$

$$\bar{\alpha}_{iq} = 1, \quad i\in\mathcal{I}\setminus\mathcal{I}^{\text{rel}}, \tag{45}$$

$$\sum_{q\in\mathcal{Q}_i} t^{\text{r}}_{iq}\alpha_{iq} \le x_i \le \sum_{q\in\mathcal{Q}_i} t^{\text{d}}_{iq}\alpha_{iq} - e_i, \quad i\in\mathcal{I}, \tag{46}$$

[Precedence relations]

$$u_{ijkl} = x_j + lp_j - (x_i + kp_i) + Py^{\text{D}}_{ijkl}, \quad (i,j,k,l)\in\mathcal{D}, \tag{47}$$

$$l^{\min}_{ijkl} \le u_{ijkl} \le l^{\max}_{ijkl}, \quad (i,j,k,l)\in\mathcal{D}, \tag{48}$$

[AM-scheduling]

$$x_i + e_i + l^{\text{idle}}_{ii'} - (t^{\text{d}}_i + l^{\text{idle}}_{ii'} - t^{\text{r}}_{i'})(1 - y_{ii'}) \le x_{i'}, \quad (i,i')\in\mathcal{II}^{\text{O}}_h,\ h\in\mathcal{H}^{\text{AM}}, \tag{49}$$

$$x_{i'} + e_{i'} + l^{\text{idle}}_{i'i} - (t^{\text{d}}_{i'} + l^{\text{idle}}_{i'i} - t^{\text{r}}_i)y_{ii'} \le x_i, \quad (i,i')\in\mathcal{II}^{\text{O}}_h,\ h\in\mathcal{H}^{\text{AM}}, \tag{50}$$

[CN-scheduling]

$$\sum_{n\in\mathcal{N}} z_{nm} = 1, \quad m \in \mathcal{M}, \tag{51}$$

$$\sum_{m\in\mathcal{M}} l_m^{\text{msg}} z_{nm} \le l_n^{\text{slot}} + \gamma_n^{\text{CN-slot}}, \quad n \in \mathcal{N}, \tag{52}$$

$$\gamma_n^{\text{CN-slot}} \le \sum_{m\in\mathcal{M}} l_m^{\text{msg}} - l_n^{\text{slot}}, \quad n \in \mathcal{N}, \tag{53}$$

$$\gamma_n^{\text{CN-slot}} \le \gamma_n^{\text{M-CN-slot}}, \quad n \in \mathcal{N}, \tag{54}$$

[CN-message order]

$$\sum_{n\in\mathcal{N}} n z_{nm} + 1 - w_{mm'} - (n_m^{\max} - n_{m'}^{\min} + 1)(1 - y_{mm'}^{\text{S}}) \le \sum_{n\in\mathcal{N}} n z_{m'n}$$
$$\le \sum_{n\in\mathcal{N}} n z_{nm} + (n_{m'}^{\max} - n_m^{\min})(1 - w_{mm'}), \quad m \in \mathcal{M} : m < m',\ m' \in \mathcal{M}, \tag{55}$$

$$\sum_{n\in\mathcal{N}} n z_{m'n} + 1 - (n_{m'}^{\max} + 1 - n_m^{\min}) y_{mm'}^{\text{S}} \le \sum_{n\in\mathcal{N}} n z_{nm}, \quad m \in \mathcal{M} : m < m',\ m' \in \mathcal{M}, \tag{56}$$

$$w_{mm'} \le y_{mm'}^{\text{S}}, \quad m \in \mathcal{M} : m < m',\ m' \in \mathcal{M}, \tag{57}$$

[Time restrictions for CN-tasks]

$$\sum_{n\in\mathcal{N}} t_{nk}^{\text{r-msg}} z_{nm} - \gamma_{ik}^{\text{r-msg}} \le x_i \le \sum_{n\in\mathcal{N}} t_{nk}^{\text{d-msg}} z_{nm} - e_i + \gamma_{ik}^{\text{d-msg}}, \quad i \in \mathcal{I}_m^{\text{M}} \cap \mathcal{I}_3^{\text{K}},\ m \in \mathcal{M}, \tag{58}$$

$$\gamma_{ik}^{\text{r-msg}} \le \gamma^{\text{M-r-msg}}, \quad i \in \mathcal{I}_m^{\text{M}} \cap \mathcal{I}_3^{\text{K}},\ m \in \mathcal{M}, \tag{59}$$

$$\gamma_{ik}^{\text{d-msg}} \le \gamma^{\text{M-d-msg}}, \quad i \in \mathcal{I}_m^{\text{M}} \cap \mathcal{I}_3^{\text{K}},\ m \in \mathcal{M}, \tag{60}$$

$$0 \le \gamma_{ik}^{\text{r-msg}} \le \sum_{n\in\mathcal{N}} t_{nk}^{\text{r-msg}} z_{nm}, \quad i \in \mathcal{I}_m^{\text{M}} \cap \mathcal{I}_3^{\text{K}},\ m \in \mathcal{M}, \tag{61}$$

$$0 \le \gamma_{ik}^{\text{d-msg}} \le P + e_i - \sum_{n\in\mathcal{N}} t_{nk}^{\text{d-msg}} z_{nm}, \quad i \in \mathcal{I}_m^{\text{M}} \cap \mathcal{I}_3^{\text{K}},\ m \in \mathcal{M}, \tag{62}$$

$$\alpha_{iq} = z_{nm}, \quad q \in \mathcal{Q}_i :\ t_{iq}^{\text{r}} = t_{n2}^{\text{r-msg}},\ i \in \mathcal{I}_2^{\text{K}} \cap \mathcal{I}_m^{\text{M}},\ m \in \mathcal{M},\ n \in \mathcal{N}, \tag{63}$$

[Merging of CN-tasks]

$$w_{mm'} \le y_{ii'}^{\text{M}}, \quad (i,i') \in \mathcal{II}_{khmm'}^{\text{M}},\ k \in \mathcal{K},\ h \in \mathcal{H}^{\text{CM}},\ m' \in \mathcal{M},\ m \in \mathcal{M} : m < m', \tag{64}$$

$$x_i + e_i - e_{i'}^{\text{init}} w_{mm'} - \gamma_{ii'}^{\text{CN-m-1}} - (t_i^{\text{d}} - t_{i'}^{\text{r}})(1 - y_{ii'}^{\text{M}}) \le x_{i'},$$
$$(i,i') \in \mathcal{II}_{khmm'}^{\text{M}},\ k \in \mathcal{K},\ h \in \mathcal{H}^{\text{CM}},\ m' \in \mathcal{M},\ m \in \mathcal{M} : m < m', \tag{65}$$

$$x_{i'} - \gamma_{ii'}^{\text{CN-m-2}} \le x_i + e_i - e_{i'}^{\text{init}} + \sum_{i''\in\mathcal{I}_{m'}^{\text{M}}\cap\mathcal{I}_h\cap\mathcal{I}_k^{\text{K}},\ m''\in\mathcal{M}_{kh} : m<m''<m'} (e_{i''} - e_{i''}^{\text{init}}) w_{mm''}$$
$$+ (t_{i'}^{\text{d}} - e_{i'} - t_i^{\text{r}} - e_i + e_{i'}^{\text{init}})(1 - w_{mm'}),$$
$$(i,i') \in \mathcal{II}_{khmm'}^{\text{M}},\ k \in \mathcal{K},\ h \in \mathcal{H}^{\text{CM}},\ m' \in \mathcal{M},\ m \in \mathcal{M} : m < m', \tag{66}$$
$$x_{i'} + e_{i'} - \gamma_{ii'}^{\text{CN-m-3}} - (t_{i'}^{\text{d}} - t_i^{\text{r}}) y_{ii'}^{\text{M}} \le x_i,$$

$$(i,i') \in \mathcal{II}^{\mathrm{M}}_{khmm'},\ k \in \mathcal{K},\ h \in \mathcal{H}^{\mathrm{CM}},\ m' \in \mathcal{M},\ m \in \mathcal{M} : m < m', \tag{67}$$
$$0 \le \gamma^{\text{CN-m-1}}_{ii'} \le t^{\mathrm{d}}_{i} - t^{\mathrm{r}}_{i'},$$
$$(i,i') \in \mathcal{II}^{\mathrm{M}}_{khmm'},\ k \in \mathcal{K},\ h \in \mathcal{H}^{\mathrm{CM}},\ m' \in \mathcal{M},\ m \in \mathcal{M} : m < m', \tag{68}$$
$$\gamma^{\text{CN-m-1}}_{ii'} \le \gamma^{\text{M-CN-m-1}},$$
$$(i,i') \in \mathcal{II}^{\mathrm{M}}_{khmm'},\ k \in \mathcal{K},\ h \in \mathcal{H}^{\mathrm{CM}},\ m' \in \mathcal{M},\ m \in \mathcal{M} : m < m', \tag{69}$$
$$0 \le \gamma^{\text{CN-m-2}}_{ii'} t^{\mathrm{d}}_{i'} - e_{i'} - t^{\mathrm{r}}_{i} - e_i + e^{\mathrm{init}}_{i'},$$
$$(i,i') \in \mathcal{II}^{\mathrm{M}}_{khmm'},\ k \in \mathcal{K},\ h \in \mathcal{H}^{\mathrm{CM}},\ m' \in \mathcal{M},\ m \in \mathcal{M} : m < m', \tag{70}$$
$$\gamma^{\text{CN-m-2}}_{ii'} \le \gamma^{\text{M-CN-m-2}},$$
$$(i,i') \in \mathcal{II}^{\mathrm{M}}_{khmm'},\ k \in \mathcal{K},\ h \in \mathcal{H}^{\mathrm{CM}},\ m' \in \mathcal{M},\ m \in \mathcal{M} : m < m', \tag{71}$$
$$0 \le \gamma^{\text{CN-m-3}}_{ii'} \le t^{\mathrm{d}}_{i} - t^{\mathrm{r}}_{i'},$$
$$(i,i') \in \mathcal{II}^{\mathrm{M}}_{khmm'},\ k \in \mathcal{K},\ h \in \mathcal{H}^{\mathrm{CM}},\ m' \in \mathcal{M},\ m \in \mathcal{M} : m < m', \tag{72}$$
$$\gamma^{\text{CN-m-3}}_{ii'} \le \gamma^{\text{M-CN-m-3}},$$
$$(i,i') \in \mathcal{II}^{\mathrm{M}}_{khmm'},\ k \in \mathcal{K},\ h \in \mathcal{H}^{\mathrm{CM}},\ m' \in \mathcal{M},\ m \in \mathcal{M} : m < m', \tag{73}$$

[CN-task order]

$$y^{\mathrm{S}}_{mm'} = y^{\mathrm{M}}_{ii'}, \quad (i,i') \in \mathcal{II}^{\mathrm{M}}_{3hmm'},\ h \in \mathcal{H}^{\mathrm{CM}},\ m' \in \mathcal{M},\ m \in \mathcal{M} : m < m', \tag{74}$$

[Relaxed CM-scheduling]

$$\sum_{i \in \mathcal{I}^{\mathrm{cap}}_r} \sum_{q \in \mathcal{Q}_{ir}} e_i \alpha_{iq} - \sum_{(i,i') \in \mathcal{II}^{\mathrm{M}}_{khmm'} \cap \mathcal{I}^{\mathrm{cap}}_r,\ k \in \mathcal{K},\ m' \in \mathcal{M},\ m \in \mathcal{M} : m < m'} e^{\mathrm{init}}_{i'} w_{mm'}$$
$$\le \max_{i \in \mathcal{I}^{\mathrm{cap}}_r} \max_{q \in \mathcal{Q}_{ir}} t^{\mathrm{d}}_{iq} - \min_{i \in \mathcal{I}^{\mathrm{cap}}_r} \min_{q \in \mathcal{Q}_{ir}} t^{\mathrm{r}}_{iq} + \gamma^{\mathrm{cap}}_r, \quad r \in \mathcal{R}_h,\ h \in \mathcal{H}^{\mathrm{CM}}, \tag{75}$$
$$0 \le \gamma^{\mathrm{cap}}_r \le \sum_{i \in \mathcal{I}^{\mathrm{cap}}_r} e_i - \max_{i \in \mathcal{I}^{\mathrm{cap}}_r} \max_{q \in \mathcal{Q}_{ir}} t^{\mathrm{d}}_{iq} - \min_{i \in \mathcal{I}^{\mathrm{cap}}_r} \min_{q \in \mathcal{Q}_{ir}} t^{\mathrm{r}}_{iq}, \quad r \in \mathcal{R}_h,\ h \in \mathcal{H}^{\mathrm{CM}}, \tag{76}$$
$$\gamma^{\mathrm{cap}}_r \le \gamma^{\text{M-cap}}, \quad r \in \mathcal{R}_h,\ h \in \mathcal{H}^{\mathrm{CM}}, \tag{77}$$

[Generated sequences]

$$y_{ii'} + y_{i'i} = 1, \quad (i,i') \in \mathcal{II}^{\mathrm{O}}_s,\ s \in \mathcal{S}^{\mathrm{gen}}, \tag{78}$$
$$x_i + e_i - \gamma^{\mathrm{gen}}_{ii'} - (t^{\mathrm{d}}_i - t^{\mathrm{r}}_{i'})(1 - y_{ii'}) \le x_{i'}, \quad (i,i') \in \mathcal{II}^{\mathrm{O}}_s,\ s \in \mathcal{S}^{\mathrm{gen}}, \tag{79}$$
$$x_{i'} + e_{i'} - \gamma^{\mathrm{gen}}_{i'i} - (t^{\mathrm{d}}_{i'} - t^{\mathrm{r}}_i)(1 - y_{i'i}) \le x_i, \quad (i,i') \in \mathcal{II}^{\mathrm{O}}_s,\ s \in \mathcal{S}^{\mathrm{gen}}, \tag{80}$$
$$0 \le \gamma^{\mathrm{gen}}_{ii'} \le e_i, \quad (i,i') \in \mathcal{II}^{\mathrm{O}}_s,\ s \in \mathcal{S}^{\mathrm{gen}}, \tag{81}$$
$$0 \le \gamma^{\mathrm{gen}}_{i'i} \le e_{i'}, \quad (i,i') \in \mathcal{II}^{\mathrm{O}}_s,\ s \in \mathcal{S}^{\mathrm{gen}}, \tag{82}$$
$$\gamma^{\mathrm{gen}}_{i'i} \le \gamma^{\text{M-gen}}, \quad (i,i') \in \mathcal{II}^{\mathrm{O}}_s,\ s \in \mathcal{S}^{\mathrm{gen}}, \tag{83}$$
$$\gamma^{\mathrm{gen}}_{ii'} \le \gamma^{\text{M-gen}}, \quad (i,i') \in \mathcal{II}^{\mathrm{O}}_s,\ s \in \mathcal{S}^{\mathrm{gen}}. \tag{84}$$

B Partition model

The purpose of the Partition model is to create a valid AM schedule that maximise the duration between tasks on AMs that communicate with each other. The Partition model is a simplified

model that includes the components AM-scheduling, Precedence relations, Common for AM- and CM-scheduling along with a part for modelling the objective.

To avoid a solution where tasks are placed too far from their allowed interval, a set of tasks $\mathcal{I}^{\text{assign}}$ is forced to be assigned to a sub-interval. In order to measure the duration between tasks on AMs that communicate with each other, we create an artificial dependency between each pair tasks on AMs that are connected through dependencies. Let $\mathcal{D}^{\text{part}}$ denote the set of artificial dependencies and choose the lower and upper bounds on the duration so that they never will be active. To describe the objective function, introduce for each artificial dependency $(i, j, k, l) \in \mathcal{D}^{\text{part}}$, an auxiliary variable

$$u^{\text{obj}}_{ijkl} = \text{the reward for artificial dependency } (i, j, k, l),$$

along with variable

$$u^{\text{obj-min}} = \text{the minimum reward for an artificial dependency.}$$

Let the constants l^1, l^2, l^3 and $l^{\text{obj-min}}$ be used to give different objective rewards depending on the relation between the length and lower bound of the artificial dependencies.

The Partition model is

$$\max \sum_{(i,j,k,l)\in\mathcal{D}^{\text{part}}} u^{\text{obj}}_{ijkl} + l^{\text{obj-min}} u^{\text{obj-min}}, \tag{85}$$

subject to

[Objective reward]

$$u^{\text{obj}}_{ijkl} \le u_{ijkl}, \quad (i, j, k, l) \in \mathcal{D}^{\text{part}}, \tag{86}$$

$$u^{\text{obj}}_{ijkl} \le l^1 l^{\min}_{ijkl}, \quad (i, j, k, l) \in \mathcal{D}^{\text{part}}, \tag{87}$$

$$u^{\text{obj}}_{ijkl} \le l^2 l^{\min}_{ijkl} + l^3 u_{ijkl}, \quad (i, j, k, l) \in \mathcal{D}^{\text{part}}, \tag{88}$$

$$u^{\text{obj-min}} \le u^{\text{obj}}_{ijkl}, \quad (i, j, k, l) \in \mathcal{D}^{\text{part}}, \tag{89}$$

$$0 \le u^{\text{obj}}_{ijkl}, \quad (i, j, k, l) \in \mathcal{D}^{\text{part}}, \tag{90}$$

$$0 \le u^{\text{obj-min}}, \tag{91}$$

[Common for AM- and CM-scheduling]

$$\sum_{q\in\mathcal{Q}_i} \alpha_{iq} = 1, \quad i \in \mathcal{I}^{\text{assign}}, \tag{92}$$

$$\sum_{q\in\mathcal{Q}_i} t^{\text{r}}_{iq}\alpha_{iq} \le x_i \le \sum_{q\in\mathcal{Q}_i} t^{\text{d}}_{iq}\alpha_{iq} - e_i, \quad i \in \mathcal{I}^{\text{assign}}, \tag{93}$$

$$t^{\text{r}}_i \le x_i \le t^{\text{d}}_i - e_i, \quad i \in \mathcal{I} \setminus \mathcal{I}^{\text{assign}}, \tag{94}$$

[Precedence relations]

$$u_{ijkl} \le u^{\text{part-max}}, \quad (i, j, k, l) \in \mathcal{D}^{\text{part}}, \tag{95}$$

$$u_{ijkl} = x_j + lp_j - (x_i + kp_i) + P y^{\text{D}}_{ijkl}, \quad (i, j, k, l) \in \mathcal{D} \cup \mathcal{D}^{\text{part}}, \tag{96}$$

$$l^{\min}_{ijkl} \le u_{ijkl} \le l^{\max}_{ijkl}, \quad (i, j, k, l) \in \mathcal{D} \cup \mathcal{D}^{\text{part}}, \tag{97}$$

[AM-scheduling]

$$x_i + e_i + l^{\text{idle}}_{ii'} - (t^{\text{d}}_i + l^{\text{idle}}_{ii'} - t^{\text{r}}_{i'})(1 - y_{ii'}) \le x_{i'}, \quad (i, i') \in \mathcal{II}^{\text{O}}_h,\ h \in \mathcal{H}^{\text{AM}}, \tag{98}$$

$$x_{i'} + e_{i'} + l^{\text{idle}}_{i'i} - (t^{\text{d}}_{i'} + l^{\text{idle}}_{i'i} - t^{\text{r}}_{i})y_{ii'} \leq x_i, \quad (i, i') \in \mathcal{II}^{\text{O}}_h,\ h \in \mathcal{H}^{\text{AM}}. \tag{99}$$

References

Archetti, C., & Speranza, M. G. (2014). A survey on matheuristics for routing problems. *EURO Journal on Computational Optimization*, *2*(4), 223–246.

Ball, M. O. (2011). Heuristics based on mathematical programming. *Surveys in Operations Research and Management Science*, *16*(1), 21–38.

Belo-Filho, M. A., Amorim, P., & Almada-Lobo, B. (2015). An adaptive large neighbourhood search for the operational integrated production and distribution problem of perishable products. *International Journal of Production Research*, *53*(20), 6040–6058.

Blikstad, M., Karlsson, E., Lööw, T., & Rönnberg, E. (2018). An optimisation approach for pre-runtime scheduling of tasks and communication in an integrated modular avionic system. *Optimization and Engineering*, *19*(4), 977–1004.

Boberg, J. (2017). A comparison of sequencing formulations in a constraint generation procedure for avionics scheduling. Bachelor thesis, Linköping University, ISRN: LiTH-MAT-EX–2017/18–SE.

Caserta, M., & Voß, S. (2010). Metaheuristics: Intelligent problem solving. In V. Maniezzo, T. Stützle, & S. Voß (Eds.), *Matheuristics: Hybridizing metaheuristics and mathematical programming* (pp. 1–38). USA: Springer.

Cordeau, J. F., Laporte, G., Moccia, L., & Sorrentino, G. (2011). Optimizing yard assignment in an automotive transshipment terminal. *European Journal of Operational Research*, *215*(1), 149–160.

Craciunas, S. S., & Oliver, R. S. (2016). Combined task- and network-level scheduling for distributed time-triggered systems. *Real-time Systems*, *52*(2), 161–200.

Doerner, K. F., & Schmid, V. (2010). Survey: Matheuristics for rich vehicle routing problems. In *Hybrid metaheuristics: 7th International workshop, HM 2010, Vienna, Austria, October 2010* (pp. 206–221). Berlin, Heidelberg: Springer.

Eisenbrand, F., Kesavan, K., Mattikalli, R. S., Niemeier, M., Nordsieck, A. W., Skutella, M., et al. (2010). Solving an avionics real-time scheduling problem by advanced IP-methods. In M. de Berg & U. Meyer (Eds.), *Algorithms - ESA 2010* (pp. 11–22). Berlin: Springer.

Framinan, J. M., & Perez-Gonzalez, P. (2018). Order scheduling with tardiness objective: Improved approximate solutions. *European Journal of Operational Research*, *266*(3), 840–850.

Gerhards, P., Stürck, C., & Fink, A. (2017). An adaptive large neighbourhood search as a matheuristic for the multi-mode resource-constrained project scheduling problem. *European Journal of Industrial Engineering*, *11*(6), 774–791.

Gharehgozli, A. H., Laporte, G., Yu, Y., & de Koster, R. (2015). Scheduling twin yard cranes in a container block. *Transportation Science*, *49*(3), 686–705.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decisions Sciences*, *8*(1), 156–166.

Glover, F., & Hao, J. K. (2011). The case for strategic oscillation. *Annals of Operations Research*, *183*(1), 163–173.

Guido, R., Groccia, M. C., & Conforti, D. (2018). An efficient matheuristic for offline patient-to-bed assignment problems. *European Journal of Operational Research*, *268*(2), 486–503.

Hao, Y., Mu, M., Dai, X., & Li, X. (2018). Schedulability test for IMA systems based on mixed integer linear programming formulation. *Turkish Journal of Electrical Engineering & Computer Sciences*, *26*(2), 844–855.

Karlsson, E., & Rönnberg, E. (2018). Explicit modelling of multiple intervals in a constraint generation procedure for multiprocessor scheduling. In: Kliewer, N., Ehmke, J. F., Borndörfer, R. (Eds.) *Operations research proceedings 2017*, Berlin: Springer, pp. 567–572.

Kiefer, A., Hartl, R. F., & Schnell, A. (2017). Adaptive large neighborhood search for the curriculum-based course timetabling problem. *Annals of Operations Research*, *252*(2), 255–282.

Lhachemi, H., De Azua Ortega, J. A. R., Saussie, D., & Zhu, G. (2016). Partition modeling and optimization of ARINC 653 operating systems in the context of IMA. In *2016 IEEE/AIAA 35th digital avionics systems conference (DASC)*, pp 1–8.

Mancini, S. (2016). A real-life multi depot multi period vehicle routing problem with a heterogeneous fleet: Formulation and adaptive large neighborhood search based matheuristic. *Transportation Research Part C: Emerging Technologies*, *70*, 100–112.

Maniezzo, V., Stützle, T., & Voß, S. (2010). *Matheuristics: Hybridizing metaheuristics and mathematical programming, annals of information systems* (Vol. 10). USA: Springer.

Muller, L. F., Spoorendonk, S., & Pisinger, D. (2012). A hybrid adaptive large neighborhood search heuristic for lot-sizing with setup times. *European Journal of Operational Research*, *218*(3), 614–623.

Pereira, M. A., Coelho, L. C., Lorena, L. A., & de Souza, L. C. (2015). A hybrid method for the probabilistic maximal covering location-allocation problem. *Computers & Operations Research*, *57*, 51–59.

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, *34*(8), 2403–2435.

Rönnberg, E. (2018). Co-allocation of communication messages in an integrated modular avionic system. In Kliewer, N., Ehmke, J. F., Borndörfer, R. (Eds.) *Operations research proceedings 2017*, Berlin: Springer, pp. 459–465.

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, *40*(4), 455–472.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and practice of constraint programming— CP98. CP 1998. Lecture notes in computers science, vol. 1520*, pp. 417–431.

Zhang, L., Goswami, D., Schneider, R., & Chakraborty, S. (2014). Task- and Network-level schedule co-synthesis of ethernet-based time-triggered systems. In *Proceedings of the Asia and South Pacific design automation conference, ASP-DAC*, pp. 119–124.