Optim Eng (2018) 19:977–1004
https://doi.org/10.1007/s11081-018-9385-6

RESEARCH ARTICLE

An optimisation approach for pre-runtime scheduling of tasks and communication in an integrated modular avionic system

Mathias Blikstad[1] · **Emil Karlsson**[1,2] · **Tomas Lööw**[1] · **Elina Rönnberg**[1,2]

Received: 1 June 2017 / Revised: 28 January 2018 / Accepted: 24 April 2018 /
Published online: 15 June 2018

Abstract In modern integrated modular avionic systems, applications share hardware resources on a common avionic platform. Such an architecture necessitates strict requirements on the spatial and temporal partitioning of the system to prevent fault propagation between different aircraft functions. One way to establish a temporal partitioning is through pre-runtime scheduling of the system, which involves creating a schedule for both tasks and a communication network. While avionic systems are growing more and more complex, so is the challenge of scheduling them. The scheduling of the system has an important role in the development of new avionic systems, since functionality is typically added to the system over a period of several years and a scheduling tool is used both to detect if the platform can host the new functionality and, if this is possible, to create a new schedule. For this reason an exact solution strategy for avionics scheduling is preferred over a heuristic one. In this paper we present a mathematical model for an industrially relevant avionic system and present a constraint generation procedure for the scheduling of such systems. We apply our optimisation approach to instances provided by our industrial partner. These instances are of relevance for the development of future avionic systems and contain up to 20,000 tasks to be scheduled. The computational results show that our optimisation approach can be used to create schedules for such instances within a reasonable time.

Keywords Avionic system · Scheduling · Discrete optimisation · Integer programming · Multiprocessor scheduling · Constraint generation

✉ Elina Rönnberg
elina.ronnberg@liu.se

Emil Karlsson
emil.karlsson@liu.se

1 Saab AB, 581 88 Linköping, Sweden

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

1 Introduction

For an avionic system (the electronic system in an aircraft) 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. Such systems, where the consequences are severe if a computational result is not delivered on time, are called hard real-time systems. For an introduction to real-time systems, see Kopetz (2011). The scheduling of real-time systems can refer both to on-line scheduling where the scheduling decisions are made at runtime and to pre-runtime scheduling where the schedule is created at compile time. This paper addresses a pre-runtime multiprocessor scheduling problem for an avionic system with periodic tasks, where each task is beforehand assigned to a processor.

During the last 2 decades, most of the avionics industry has gone from using federated systems to using an integrated architecture called Integrated Modular Avionics (IMA), where applications share hardware resources. This architecture necessitates strict requirements on the spatial and temporal partitioning of the system to achieve fault containment; a common standard for this partitioning is ARINC 653. For more information about IMA and ARINC 653, see Radio Technical Commission for Aeronautics (RTCA) (2005) and Airlines Electronic Engineering Committee (AEEC) (2006), respectively. Typically the IMA architecture give rise to multiprocessor scheduling-type problems that become computationally demanding for large-scale instances. The introductory sections of the PhD thesis by Al-Sheikh (2011) provide an extensive introduction to the scheduling of avionic systems.

In the process of designing an avionic system, new software functionality is developed and added to the system iteratively during a project of several years. Whenever a change is made in a software component, the scheduling tool has to provide a new schedule for the system or, if it cannot, preferably produce a proof of infeasibility. If no feasible schedule exists, either changes to the software or upgrades of the avionics platform are needed. Because of the rigid certification processes in the aircraft industry, it is extremely costly to upgrade the platform, and therefore it is important to utilise the existing platform in an efficient way and make upgrades only when necessary. The frequency of use and the importance of the outcome of the scheduling gives the scheduling tool a vital role in the process of designing an avionic system.

Most methods for the pre-runtime scheduling of large-scale real-time systems are of a heuristic nature; see for example the references in Sect. 1.2. For many types of real-time systems, a primal heuristic might be an efficient and sound way to provide a schedule. However, this does not hold for an avionic scheduling problem when the scheduling also involves determining whether or not a desired software functionality can be implemented with the existing platform. If a primal heuristic method is applied to such a problem setting and the heuristic fails to provide a solution, one does not know if this results from shortcomings of the heuristic or if no feasible solution exists. This paper contributes to the development of optimisation-based approaches for the scheduling of large and complex future avionics systems, and the research was carried out in collaboration with the Swedish defence and security company Saab.

Our scheduling problem can be considered as a multiprocessor scheduling problem with precedence relations and a communication network. In Sect. 2 this problem is presented from a mathematical modelling point of view rather than from an avionics point of view. An overview of the technical design of the system is summarised in the following section, but it is not discussed to the same extent as in the real-time system research papers referred to in Sect. 1.2.

1.1 System characteristics

There is a diversity in the type of scheduling required for different IMA systems, even if they are designed in compliance with the same ARINC standard. This section summarises the characteristics of the system considered in this paper.

The system is distributed, and at each node there is a set of processors, called modules. One of these is responsible for both the inter-node and the intra-node communication as well as the communication with external systems. The responsibility of the other modules is to run applications (software processes). The system is partitioned in the sense that all the tasks are a priori assigned to modules, and no migration is allowed.

At the communication modules, tasks that deal with the communication need to be scheduled. For the modules running applications, the software processes that share functionality are grouped into partitions by the engineers designing the system, and the tasks to be scheduled are these partitions. Processes executing in the same partition are locally scheduled by rate monotonic scheduling; see Section 28.4.3, Part IV of Leung (2004). It is within the partitions that the real-time aspect of the process scheduling is captured, with the rate monotonic scheduling assuring that all deadlines are respected. Since the assignment of processes to partitions is made independently of the pre-runtime scheduling and the scheduling within the partitions, the scheduling is referred to as being hierarchical.

The nodes communicate via a switched Ethernet which supports multicast. The Ethernet considered in this paper is such that messages are assigned to, and sent in, discrete time slots. Within a time slot, the full bandwidth can be used for the messages assigned to it. Hence, the way the communication capacity is made available to different resources within the system deviates from what is studied in previously published work.

In the well-established Avionics Full Duplex Switched Ethernet (AFDX) network (see Al-Sheikh et al. 2013) used by, for example, the Airbus A380 and Boeing Dreamliner B787, a dedicated virtual link with limited bandwidth is created for each communication flow. The communication is assured to be separated by respecting a Bandwidth Allocation Gap (BAG) and a Maximum Frame Size (MFS). For more information see Kopetz (2011) and Al-Sheikh et al. (2013); the latter suggests a strategy for the optimal design of the virtual links. For the network considered in our paper, the communication is separated by assigning the messages to time slots in which they are allowed to use the full bandwidth, facilitating very fast communication at that instant. The introduction of time slots does however make the communication capacity sharing a part of the temporal partitioning of the system, and thereby the message scheduling becomes more intricately integrated with the scheduling of partitions. For further information about the communication network used in this paper, see the patent by Danielsson et al. (2016).

The schedule is created pre-runtime and made with knowledge of the duration (worst-case execution time) and the period of all tasks to be executed as well as the precedence relations and communication messages between them. There are two types of scheduling decisions to be made; communication is scheduled by assigning messages to time slots and tasks are sequenced and assigned a start time. The solution approach we suggest in this paper is applied to instances with up to 8 application modules with 25 partitions repeated 64 times, 96 communication messages, and 7 communication modules with 19,894 tasks.

1.2 Related research

There are many papers on the scheduling of real-time systems that consider runtime scheduling, which is different to pre-runtime scheduling. The reader interested in the schedulability analysis and runtime scheduling of avionic systems is referred to, for example, Rufino et al. (2010), Davis and Burns (2011), and Easwaran et al. (2009). A comparison of runtime and pre-runtime scheduling is provided by Xu and Parnas (2000), and they suggest pre-runtime scheduling as the preferred choice for applications similar to that in this paper. They conclude in part that pre-runtime scheduling is better suited for handling complex application constraints, makes it easier to verify that all deadlines and constraints are complied with, and also improves the chances of finding a feasible schedule when the resource utilisation is high and feasibility might be a challenge. Comparing the runtime and the pre-runtime approach shows that the latter can be considered as a constructive sufficient schedulability test; see Section 10.3 of Kopetz (2011).

The pre-runtime scheduling of IMA systems is addressed in Al-Sheikh et al. (2012), who consider the scheduling of strictly periodic tasks on a multiprocessor system. Their model integrates two types of decisions, the allocation of tasks to processors (respecting both hardware capacity constraints and conditions that prevent the allocation of pairs of tasks to the same hardware) and the assignment of start times to tasks (respecting the strict periodicity conditions), with the objective of creating a schedule with maximum idle time between tasks (proportional to the processing time of the task) to provide robustness. That they assign tasks to processors adds an additional complexity compared to the problem we consider. However, in Al-Sheikh et al. (2012) it is sufficient to satisfy precedence relations between the tasks in order to ensure communication. From that point of view, their setting is significantly less complex than that in this paper. To solve the problem, they suggest a heuristic inspired by game theory principles, and they successfully apply it to instances, supplied by industrial partners, with up to 48 processors and 636 tasks.

For a problem setting of the same type, an exact integer programming method based on a bin-formulation of the problem is presented by Eisenbrand et al. (2010). They provide computational results for instances with up to 177 tasks and 16 machines and show that their approach outperforms other integer programming formulations. Another, later work studying a similar setting as in Al-Sheikh et al. (2012) is by Balashov et al. (2014), who propose a greedy heuristic for solving the problem of allocating tasks to processors and another heuristic for the scheduling of tasks. They apply their algorithm to a system with 3 nodes, 9 partitions with a total of 164 periodic tasks, and 163 communication messages.

An early work suggesting a scheduling tool for an IMA system is Lee et al. (2000), but they address significantly less complex problems than those in this paper. The same holds for the approach of Tavares et al. (2012), which generates schedules using a time Petri net model (a graph representation of concurrent processes) and develops a specially designed depth-first search over the order in which to place the tasks. Their strategy takes into account intertask relations (precedence and mutual exclusion) and overhead. They evaluate their strategy only for uni-processor systems with at most 12 tasks (real-world applications and custom-built instances), which are significantly smaller than those of interest in our context.

Another related paper on the pre-runtime scheduling of IMA systems is by Beji et al. (2014). In their system, a TTEthernet (see Kopetz 2011) is used for the communication, and the scheduling is assumed to be carried out iteratively when new tasks are added to the system. The objective of this repeated re-scheduling is to minimise the integration cost while developing the system. They apply a Satisfiability Modulo Theory (SMT) solver to create schedules, and they evaluate their approach on an application with 5 nodes and 7 partitions.

Various methods for the scheduling of distributed systems with time-triggered communication can be found in the literature, applying either heuristic methods (Theis et al. 2013; Tămaş-Selicean et al. 2012; Pop et al. 1999), a Mixed Integer Programming (MIP) solver (Zhang et al. 2014), or an SMT-solver (Beji et al. 2014; Craciunas and Oliver 2014). None of these approaches are viable in our setting, and typically the exact approaches are applicable only to relatively small instances, as seen from the examples above.

1.3 Contributions and outline of the paper

A system model, described from a mathematical modelling point of view, is given in Sect. 2. The main computational challenge of the problem stems from the huge number of tasks to be sequenced on the communication modules, and a key contribution of this paper is our constraint generation procedure to handle this. Section 3 introduces the characteristics of the problem that facilitate the design of the constraint generation procedure. Our resulting MIP formulation is presented in Sect. 4, and the constraint generation procedure and some preprocessing components are introduced in Sect. 5. Computational results to verify that our approach can be used to solve instances of practical relevance are presented in Sect. 6, followed by concluding comments in Sect. 7.

2 System model

This section introduces the concepts and notations needed to create a modelling framework for the system, which is illustrated in Fig. 1. The system executes periodically, meaning that the schedule for the system is repeated over and over again. One occurrence of such repetition is called a major frame, and its duration, denoted by P, is the least common multiple of the periods of the tasks in the system.

A schedule for a major frame is the result of two types of decisions, one assigning communication messages to time slots and the other assigning start times to tasks. The system design is such that adjacent major frames are not independent, and therefore a schedule for a major frame needs to be created such that it becomes valid for an infinite repetition of major frames.

2.1 Periodic task system

The system consists of a set of nodes, where each contains a set of modules that host tasks. The set of all tasks in the system is denoted by $\mathcal{I}$. Each task is a priori assigned to a specific module, and no migration of tasks between modules is allowed. The release time and deadline of task i is denoted by t_i^{r} and t_i^{d}, respectively, $i \in \mathcal{I}$. Between its release time and deadline, task i must be given an exclusive and non-preempted execution interval of the duration of its execution requirement, denoted by e_i, $i \in \mathcal{I}$.

A task i is executed periodically with period p_i, generating a number of instances in a major frame that have to scheduled at the same time offset by the start of the period of task i, $i \in \mathcal{I}$. Further, job k refers to all instances of a task that are scheduled at the same time offset by the start of a major frame.

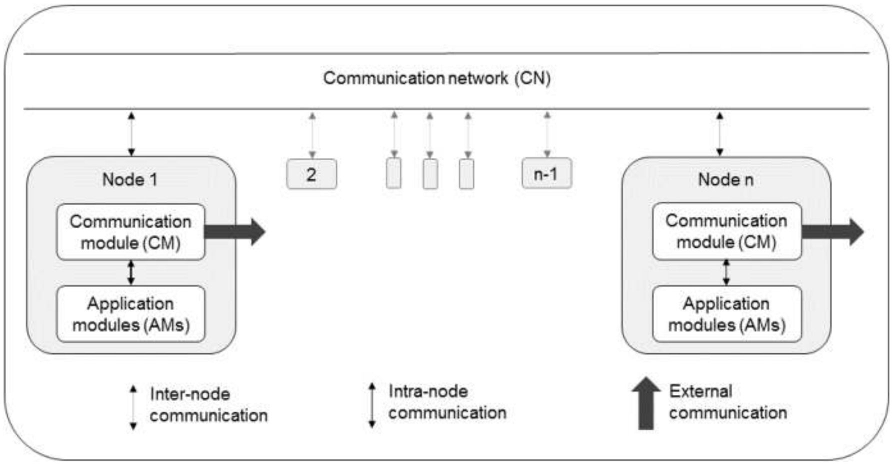
Communication network (CN)
Node 1
Communication module (CM)
Application modules (AMs)
2
n-1
Node n
Communication module (CM)
Application modules (AMs)
Inter-node communication
Intra-node communication
External communication

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

On each node there exists a single module called the communication module (CM) that handles the system external, intra-node and inter-node communication of this node. Both the system external and intra-node communication appear as tasks on a CM, while the inter-node communication is made via the communication network to be introduced in Sect. 2.3. The set of all CMs in the system is denoted by $\mathcal{H}^{\text{CM}}$. The set of tasks assigned to CM h is denoted by $\mathcal{I}_h$, $h \in \mathcal{H}^{\text{CM}}$, and each task on a CM has a period equal to the duration of a major frame, meaning that each task has only one job.

In addition to the CM, each node also has a set of application modules (AMs) that host partitions. The set of all AMs in the system is denoted by $\mathcal{H}^{\text{AM}}$. The set of tasks assigned to AM h is denoted $\mathcal{I}_h$, and these tasks have period $P/64$, meaning that each task has 64 jobs, $h \in \mathcal{H}^{\text{AM}}$. For each pair of tasks i and j there is a minimum idle time l_{ij}^{idle} between the end of task i and the start of task j, if task j follows task i on AM h, where $i, j \in \mathcal{I}_h$, $h \in \mathcal{H}^{\text{AM}}$. This idle time ensures that there is enough time to handle intra-node communication related to the two tasks. For a summary of the notation introduced in this section, see Table 1.

2.2 Precedence relations

There is a set of precedence relations called dependencies, which is denoted by $\mathcal{D}$. A dependency restricts the duration between two jobs to be within a given interval; these jobs can be executed on the same or on different modules. Formally this means that dependency d restricts the time from the start of job k_d of task i_d to the next occurring start of job l_d of task j_d to be between $l_d^{\text{min-dep}}$ and $l_d^{\text{max-dep}}$, $d \in \mathcal{D}$. Since the order of the jobs in a major frame is not known in advance, the duration between the jobs of a dependency can be measured within a major frame or from one major frame to the next.

Table 1 The parameters and sets of the entities introduced in Sect. 2.1

Entity	Parameter	Notation
System	Duration of a major frame	P
	CMs	$\mathcal{H}^{\text{CM}}$
	AMs	$\mathcal{H}^{\text{AM}}$
	Tasks	$\mathcal{I}$
	Dependencies	$\mathcal{D}$
	Chains	$\mathcal{C}$
AM h	Tasks	$\mathcal{I}_h$
	Idle time between task i and task j	l_{ij}^{idle}
CM h	Tasks	$\mathcal{I}_h$
Task i	Execution requirement	e_i
	Period	p_i
	Release time	t_i^{r}
	Deadline	t_i^{d}

Table 2 The parameters and sets of the entities introduced in Sect. 2.2

Entity	Parameter	Notation
Dependency d	Minimum length	$l_d^{\text{min-dep}}$
	Maximum length	$l_d^{\text{max-dep}}$
	From task and job	i_d, k_d
	To task and job	j_d, l_d
Chain c	Dependencies	$\mathcal{D}_c^{\text{chain}}$

There are also restrictions called chains, which are denoted by $\mathcal{C}$. A chain c specifies that a group of jobs, linked by dependencies in the set $\mathcal{D}_c^{\text{chain}}$, have to be executed in a given order from the start of one of the jobs in a major frame to the start of the same job in the following major frame, $c \in \mathcal{C}$. For a summary of the notation introduced in this section, see Table 2.

2.3 Communication network

The nodes in the system communicate through a single communication network (CN). Each node is connected to the CN through its CM. Let $\mathcal{M}$ denote the set of CN-messages that have to be transmitted through the CN. CN-message m is transmitted from a single CM to a set of receiving CMs, requiring a capacity l_m^{msg}, $m \in \mathcal{M}$. For transmission, a CN-message must be assigned to a CN-slot. Denote the set of CN-slots by $\mathcal{N}$. The total capacity requirement of the CN-messages assigned to the same CN-slot n cannot exceed the capacity l_n^{slot}, $n \in \mathcal{N}$. This paper considers the case where each CM can send only one CN-message in each CN-slot and receive only one CN-message in each CN-slot.

To transmit a CN-message on a CN, there are four types of tasks and they must be executed in a particular order. On the sending CM, there is first a task responsible for preparing the CN-message followed by a task responsible for sending the CN-message. After the CN-message has been sent there is, on each receiving CM, a task responsible for dequeuing the message followed by a task for reading the data. The set of tasks required to transmit and receive CN-message m is denoted $\mathcal{I}_m^{\text{msg}}$. If CN-message m is assigned to CN-slot n, then task i has to obey the release time t_{in}^{r} and deadline t_{in}^{d} for CN-slot n, $i \in \mathcal{I}_m^{\text{msg}}$, $n \in \mathcal{N}$, $m \in \mathcal{M}$. For a summary of the notation introduced in this section, see Table 3.

3 Sequencing approach

The computational challenge of the problem instances of practical interest stems primarily from the large number of tasks to be sequenced on the CMs. This section describes our strategy for the sequencing of CM-tasks followed by an introduction of the notation needed. Finally, we confirm the validity of our sequencing approach.

Table 3 The parameters and sets of the entities introduced in Sect. 2.3

Entity	Parameter	Notation
CN	CN-messages	$\mathcal{M}$
	CN-slots	$\mathcal{N}$
CM	CN-messages that are received on CM h	$\mathcal{M}_h^{\text{recv}}$
	CN-messages that are sent on CM h	$\mathcal{M}_h^{\text{send}}$
CN-message m	Required capacity	l_m^{msg}
	Tasks	$\mathcal{I}_m^{\text{msg}}$
CN-slot n	Capacity	l_n^{slot}
Task i	Release time in CN-slot n	t_{in}^{r}
	Deadline in CN-slot n	t_{in}^{d}

3.1 Strategy

An important characteristic of the industrially relevant avionics instances under consideration is that the CMs have a huge number of tasks, many of which are fixed. It is also known that most of the technical restrictions, like release times and deadlines of tasks, precedence relations and CN-scheduling, are not particularly tight. With this knowledge in mind, our approach to sequencing on the CMs is to:

- Create a section for each part of a major frame that is not occupied by a fixed task and require that each non-fixed task is assigned to a section.
- Create a subset for each set of non-fixed tasks that can be assigned to the same section and require that there is no overlap between tasks in this subset.

This modelling approach introduces a large number of constraints, but since each constraint has an impact only if a subset of tasks is assigned to the same section, this approach lends itself to constraint generation. This strategy for sequencing will be used in the model presented in Sect. 4 and exploited in the constraint generation procedure in Sect. 5.

3.2 Notation

For CM h, let $\mathcal{I}_h^{\text{fix}}$ denote the set of tasks that are fixed, $h \in \mathcal{H}^{\text{CM}}$. Divide the major frame of CM h into a set of sections $\mathcal{R}_h$, where only non-fixed tasks are allowed to be scheduled, $h \in \mathcal{H}$. Let l_r^{sec} denote the duration of section r and let $\mathcal{I}_r^{\text{sec}}$ denote the set of tasks that can be assigned to section r, $r \in \mathcal{R}_h$, $h \in \mathcal{H}^{\text{CM}}$. Also, let $\mathcal{R}_i^{\text{task}}$ be the set of sections where task i can be scheduled, $i \in \mathcal{I}_h^{\text{CM}} \setminus \mathcal{I}_h^{\text{fix}}$, $h \in \mathcal{H}^{\text{CM}}$. For section r let t_{ir}^{r} be the release time and t_{ir}^{d} be the deadline of task i, $i \in \mathcal{I}_r^{\text{sec}}$, $r \in \mathcal{R}$. Introduce the set $\mathcal{S}_h$ such that each subset of tasks $\mathcal{I}_s^{\text{sub}}$, $s \in \mathcal{S}_h$, $h \in \mathcal{H}^{\text{CM}}$, includes non-fixed tasks that can, for at least one section, be assigned together in the same section. For a summary of the notation introduced in this section, see Table 4.

Table 4 The parameters and sets of the entities introduced in Sect. 3.2

Entity	Parameter	Notation
CM h	Sections on CM h	$\mathcal{R}_h$
	Subsets on CM h	$\mathcal{S}_h$
Section r	Duration	l_r^{sec}
	Tasks	$\mathcal{I}_r^{\mathrm{sec}}$
Subset s	Tasks	$\mathcal{I}_s^{\mathrm{sub}}$
Task i	Release time in section r	t_{ir}^{r}
	Deadline in section r	t_{ir}^{d}
	Sections	$\mathcal{R}_i^{\mathrm{task}}$

3.3 Validity of the strategy for sequencing

In the mathematical model, to be introduced in Sect. 4, Constraints (4.1.6)–(4.1.10) assign each non-fixed task to a section and make the fixed tasks comply with their release time, while Constraints (4.1.11)–(4.1.14) require that there is no overlap between non-fixed tasks belonging to the same subset of tasks. The following proposition states the correctness of our strategy for sequencing.

Proposition 1 (Sequencing strategy) *Given a feasible scheduling instance and a solution to the mathematical model presented in Sect.* 4, *we have for each* h, $h \in \mathcal{H}^{\mathrm{CM}}$, *that no tasks in the set* $\mathcal{I}_h$ *overlap.*

Proof For each $h \in \mathcal{H}^{\mathrm{CM}}$ it is confirmed that no tasks in $\mathcal{I}_h$ overlap by showing that each ordered pair of tasks (i, j), $i, j \in \mathcal{I}_h$ is non-overlapping. Three cases of possible overlaps need to be evaluated.

- Case $i \in \mathcal{I}_h^{\mathrm{fix}}$, $j \in \mathcal{I}_h^{\mathrm{fix}}$: It follows from the feasibility of the instance that i and j cannot overlap.
- Case $i \in \mathcal{I}_h^{\mathrm{fix}}$, $j \in \mathcal{I}_h \backslash \mathcal{I}_h^{\mathrm{fix}}$: There is no overlap between task i and task j since task j is assigned to a section by Constraints (4.1.6)–(4.1.8) and task i can, according to the definition of a section, not be scheduled in a section.
- Case $i \in \mathcal{I}_h \backslash \mathcal{I}_h^{\mathrm{fix}}$, $j \in \mathcal{I}_h \backslash \mathcal{I}_h^{\mathrm{fix}}$: If task i and task j are assigned to different sections, they do not overlap since the sections are disjoint. If task i and task j are assigned to the same section, there exists a subset s, $s \in \mathcal{S}_h$, such that $\mathcal{I}_s^{\mathrm{sub}}$ contains task i and task j and they are thereby ensured not to overlap by Constraints (4.1.11)–(4.1.14). □

4 Mixed-integer programming formulation

In this section we present a MIP model for the scheduling of one major frame. The technical restrictions that can span more than one major frame will be formulated in order to propagate to the decisions made within a major frame.

4.1 Tasks and sequencing

For task i, $i \in \mathcal{I}$, introduce a continuous variable

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

The start time of a specific job k of task i becomes $x_i + kp_i$, where $0 \leq x_i + kp_i \leq P$, $i \in \mathcal{I}$. To simplify the notation we introduce two tasks, $\tilde{p}$ and $\tilde{q}$, with execution requirement 0. These will be the first and last tasks, respectively, of all the sequences.

4.1.1 AM-scheduling

For each AM, a single sequence of all its tasks is created. Let the set $\mathcal{I}_i^+$ denote all tasks that can be the immediate successor of task i and the set $\mathcal{I}_i^-$ denote all tasks that can be the immediate predecessor of task i, $i \in \mathcal{I}_h^{\text{AM}}$, $h \in \mathcal{H}^{\text{AM}}$. Introduce, for $h \in \mathcal{H}^{\text{AM}}$, $i \in \mathcal{I}_h^{\text{AM}}$, $j \in \mathcal{I}_i^+$, a binary variable

$$y_{ij} = \begin{cases} 1, & \text{if task } i \text{ is the immediate predecessor of task } j, \\ 0, & \text{otherwise.} \end{cases}$$

In order to create a sequence of the tasks on the AMs we will apply a Manne formulation (Manne 1960) to handle the idle times:

$$\sum_{j \in \mathcal{I}_i^+} y_{ij} = 1, \quad i \in \mathcal{I}_h^{\text{AM}} \setminus \{\tilde{q}\},\, h \in \mathcal{H}^{\text{AM}}, \tag{4.1.1}$$

$$\sum_{j \in \mathcal{I}_i^-} y_{ji} = 1, \quad i \in \mathcal{I}_h^{\text{AM}} \setminus \{\tilde{p}\},\, h \in \mathcal{H}^{\text{AM}}, \tag{4.1.2}$$

$$\begin{gathered} x_i + e_i + l_{ij}^{\text{idle}} \leq x_j + (t_i^{\text{d}} - t_j^{\text{r}} + l_{ij}^{\text{idle}})(1 - y_{ij}), \\ j \in \mathcal{I}_i^+,\, i \in \mathcal{I}_h^{\text{AM}},\, h \in \mathcal{H}^{\text{AM}}, \end{gathered} \tag{4.1.3}$$

$$t_i^{\text{r}} \leq x_i \leq t_i^{\text{d}} - e_i, \quad i \in \mathcal{I}_h^{\text{AM}},\, h \in \mathcal{H}^{\text{AM}}, \tag{4.1.4}$$

$$y_{ij} \in \{0, 1\}, \quad j \in \mathcal{I}_i^+,\, i \in \mathcal{I}_h^{\text{AM}},\, h \in \mathcal{H}^{\text{AM}}, \tag{4.1.5}$$

where Constraints (4.1.1) and (4.1.2) ensure that each AM-task has one successor and predecessor, respectively. Constraint (4.1.3) prevents two adjacent AM-tasks from overlapping, taking idle times into account, while Constraint (4.1.4) makes each AM-task obey its release time and deadline.

4.1.2 CM-scheduling

Each task on a CM shall be assigned to one section. Introduce, for $h \in \mathcal{H}^{\text{CM}}$, $i \in \mathcal{I}_h^{\text{CM}} \setminus \mathcal{I}_h^{\text{fix}}$, $r \in \mathcal{R}_i^{\text{task}}$, a binary variable

$$\alpha_{ir} = \begin{cases} 1, & \text{if task } i \text{ is assigned to section } r, \\ 0, & \text{otherwise.} \end{cases}$$

To handle the fixed tasks and the assignment of non-fixed tasks to sections, we introduce the constraints

$$\sum_{r \in \mathcal{R}_i^{\text{task}}} \alpha_{ir} = 1, \quad i \in \mathcal{I}_h^{\text{CM}} \backslash \mathcal{I}_h^{\text{fix}}, \, h \in \mathcal{H}^{\text{CM}}, \tag{4.1.6}$$

$$\sum_{i \in \mathcal{I}_r^{\text{sec}}} e_i \alpha_{ir} \le l_r^{\text{sec}}, \quad r \in \mathcal{R}_h, \, h \in \mathcal{H}^{\text{CM}}, \tag{4.1.7}$$

$$\sum_{r \in \mathcal{R}_i^{\text{task}}} t_{ir}^{\text{r}} \alpha_{ir} \le x_i \le \sum_{r \in \mathcal{R}_i^{\text{task}}} t_{ir}^{\text{d}} \alpha_{ir} - e_i, \quad i \in \mathcal{I}_h^{\text{CM}} \backslash \mathcal{I}_h^{\text{fix}}, \, h \in \mathcal{H}^{\text{CM}}, \tag{4.1.8}$$

$$x_i = t_i^r, \quad i \in \mathcal{I}_h^{\text{fix}}, \, h \in \mathcal{H}^{\text{CM}}, \tag{4.1.9}$$

$$\alpha_{ir} \in \{0, 1\}, \quad r \in \mathcal{R}_i^{\text{task}}, \, i \in \mathcal{I}_h^{\text{CM}} \backslash \mathcal{I}_h^{\text{fix}}, \, h \in \mathcal{H}^{\text{CM}}, \tag{4.1.10}$$

where Constraint (4.1.6) assigns each non-fixed task to one section, Constraint (4.1.7) ensures that the capacity of each section is respected, Constraint (4.1.8) makes each non-fixed task obey its release time and deadline within its section, and Constraint (4.1.9) makes the fixed tasks comply with their release time.

Further, a sequence for each subset of tasks that can be assigned to the same section is created. Let the set $\mathcal{I}_{is}^+$ denote all tasks that can be the immediate successor of task i in subset s and the set $\mathcal{I}_{is}^-$ denote all tasks that can be the immediate predecessor of task i in subset s, $i \in \mathcal{I}_s^{\text{sub}}$, $s \in \mathcal{S}_h$, $h \in \mathcal{H}^{\text{CM}}$. Introduce, for $h \in \mathcal{H}^{\text{CM}}$, $s \in \mathcal{S}_h$, $i \in \mathcal{I}_s^{\text{sub}} \backslash \{\tilde{q}\}$, $j \in \mathcal{I}_{is}^+$, a binary variable

$$y_{ijs} = \begin{cases} 1, & \text{if task } i \text{ is the immediate predecessor of task } j \text{ in subset } s, \\ 0, & \text{otherwise.} \end{cases}$$

For the sequencing of tasks, we apply to each subset of tasks a Miller–Tucker–Zemlin formulation (Miller et al. 1960):

$$\sum_{j \in \mathcal{I}_{is}^+} y_{ijs} = 1, \quad i \in \mathcal{I}_s^{\text{sub}} \backslash \{\tilde{q}\}, \, s \in \mathcal{S}_h, \, h \in \mathcal{H}^{\text{CM}}, \tag{4.1.11}$$

$$\sum_{j \in \mathcal{I}_{is}^-} y_{jis} = 1, \quad i \in \mathcal{I}_s^{\text{sub}} \backslash \{\tilde{p}\}, \, s \in \mathcal{S}_h, \, h \in \mathcal{H}^{\text{CM}}, \tag{4.1.12}$$

$$\begin{gathered} x_i + e_i \le x_j + (t_i^{\text{d}} - t_j^{\text{r}})(1 - y_{ijs}), \\ j \in \mathcal{I}_{is}^+, \, i \in \mathcal{I}_s^{\text{sub}}, \, s \in \mathcal{S}_h, \, h \in \mathcal{H}^{\text{CM}}, \end{gathered} \tag{4.1.13}$$

$$y_{ijs} \in \{0, 1\}, \quad j \in \mathcal{I}_{is}^+, \, i \in \mathcal{I}_s^{\text{sub}}, \, s \in \mathcal{S}_h, \, h \in \mathcal{H}^{\text{CM}}. \tag{4.1.14}$$

Constraints (4.1.11) and (4.1.12) ensure that each CM-task has one successor and predecessor, respectively, and for each subset Constraint (4.1.13) prevents adjacent CM-tasks from overlapping.

4.2 Precedence relations

For dependency d, between job k_d of task $i_d \in \mathcal{I}$ and the next occurence of job l_d of task $j_d \in \mathcal{I}$, introduce a continuous variable

$$u_d = \text{time from job}\, k_d \text{ of task}\, i_d \text{ to the next occurrence of job}\, l_d \text{ of task}\, j_d,$$

which will be referred to as the length of the dependency, $d \in \mathcal{D}$. Also, for $d \in \mathcal{D}$, introduce a binary variable

$$v_d = \begin{cases} 1, & \text{if job}\, k_d \text{ of task}\, i_d \text{ precedes job}\, l_d \text{ of task}\, j_d \text{ in a major frame,} \\ 0, & \text{otherwise,} \end{cases}$$

referred to as the dependency indicator variable. We ensure that the duration between the two jobs of a dependency is within its interval by introducing the constraints

$$u_d = (x_{i_d} + k_d p_{i_d}) + P v_d - (x_{j_d} + l_d p_{j_d}), \quad d \in \mathcal{D}, \tag{4.2.1}$$

$$l_d^{\text{min-dep}} \leq u_d \leq l_d^{\text{max-dep}}, \quad d \in \mathcal{D}, \tag{4.2.2}$$

$$v_d \in \{0, 1\}, \quad d \in \mathcal{D}, \tag{4.2.3}$$

where Constraint (4.2.1) defines the length of the dependency, while Constraint (4.2.2) restricts the length of a dependency to be within its minimum and maximum length.

The dependencies in a chain ensure that each pair of jobs linked by a dependency has a particular order in a major frame. Constraint (4.2.4) ensures that the cycle of these dependencies finishes within the duration of a major frame.

$$\sum_{d \in \mathcal{D}_c^{\text{chain}}} v_d = 1, \quad c \in \mathcal{C} \tag{4.2.4}$$

4.3 CN-scheduling

For each pair of CN-message m and CN-slot n, $m \in \mathcal{M}$, $n \in \mathcal{N}$, introduce a binary variable

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

The CN-scheduling is handled by introducing

$$\sum_{n\in\mathcal{N}} z_{nm} = 1, \quad m \in \mathcal{M}, \tag{4.3.1}$$

$$\sum_{m\in\mathcal{M}} l_m^{\mathrm{msg}} z_{nm} \leq l_n^{\mathrm{slot}}, \quad n \in \mathcal{N}, \tag{4.3.2}$$

$$\sum_{m\in\mathcal{M}_h^{\mathrm{send}}} z_{nm} \leq 1, \quad n \in \mathcal{N},\, h \in \mathcal{H}^{\mathrm{CM}}, \tag{4.3.3}$$

$$\sum_{m\in\mathcal{M}_h^{\mathrm{recv}}} z_{nm} \leq 1, \quad n \in \mathcal{N},\, h \in \mathcal{H}^{\mathrm{CM}}, \tag{4.3.4}$$

$$\sum_{n\in\mathcal{N}} t_{in}^{\mathrm{r}} z_{nm} \leq x_i \leq \sum_{n\in\mathcal{N}} t_{in}^{\mathrm{d}} z_{nm} - e_i, \quad i \in \mathcal{I}_m^{\mathrm{msg}},\, m \in \mathcal{M}, \tag{4.3.5}$$

$$z_{nm} \in \{0,1\}, \quad n \in \mathcal{N},\, m \in \mathcal{M}, \tag{4.3.6}$$

where Constraint (4.3.1) assigns each message to a slot and Constraint (4.3.2) makes sure that the capacity of a slot is respected. Constraint (4.3.3) and Constraint (4.3.4) ensure that at most one message can be sent or received in a slot for each CM, respectively. Further, Constraint (4.3.5) makes the tasks involved in transmitting a message respect the release times and deadlines induced by assigning their message to a specific slot.

5 Solution approach

This section presents our constraint generation procedure, the preprocessing components used, and a brief description of the scheduling tool.

5.1 Constraint generation procedure

Since the number of subsets of CM-tasks to be sequenced is typically huge, and not all of them are likely to be needed to solve the problem, the problem is initially solved without them and those needed are added iteratively in a constraint generation procedure.

This section introduces the two models, a relaxed problem and a subproblem, that are used in the constraint generation procedure. The relaxed problem is obtained by removing Constraints (4.1.11)–(4.1.14) (that require subsets of CM-tasks to be sequenced), and its purpose is to assign non-fixed CM-tasks to sections. The subproblem attempts to sequence non-fixed CM-tasks given their assignment to sections according to a solution of the relaxed problem. The solution to the subproblem is either a valid schedule or it provides information about which constraints to generate and add to both models. In that sense it acts as a separation problem. The generated constraints are referred to as generated sequences.

In practice, any feasible solution to the model introduced in Sect. 4 is a valid schedule, and therefore the model does not include an objective function. The purpose of the constraint generation procedure is to find a feasible solution to the model in Sect. 4. In the relaxed problem and the subproblem, objectives are introduced to guide the search to find such feasible solutions.

5.1.1 Relaxed problem

The aim of the relaxed problem is to assign CM-tasks to a section while respecting all other constraints. The objective functions used are described in Sect. 5.1.3.

$$\begin{aligned}
\text{min/max} \quad & \text{Objective function} \\
\text{s.t.} \quad & \text{Objective constraints} \\
& \text{AM-scheduling (Constraints (4.1.1) – (4.1.5))} \\
& \text{Generated sequences (Constraints (4.1.11) – (4.1.14))} \\
& \text{CM-assignment (Constraints (4.1.6) – (4.1.10))} \\
& \text{Precedence relations (Constraints (4.2.1) – (4.2.4))} \\
& \text{CN-scheduling (Constraints (4.3.1) – (4.3.6))}
\end{aligned}$$

In the first iteration there are no generated sequences, but in later iterations those that are identified by the subproblem are included in the model.

5.1.2 Subproblem

The subproblem is obtained from a solution of the relaxed problem as follows. For section r, let the subset of tasks that was assigned to this section be denoted by $\bar{s}_r$, $r \in \mathcal{R}_h$, $h \in \mathcal{H}^{\text{CM}}$. Restrict the release time and the deadline of task i to be t^{r}_{ir} and t^{d}_{ir}, respectively, $i \in \mathcal{I}^{\text{sub}}_{\bar{s}_r}$, $r \in \mathcal{R}_h$, $h \in \mathcal{H}^{\text{CM}}$. Introduce, for $h \in \mathcal{H}^{\text{CM}}$, $r \in \mathcal{R}_h$, $i \in \mathcal{I}^{\text{sub}}_{\bar{s}_r}$, a binary variable

$$\beta_{i\bar{s}_r} = \begin{cases} 1, & \text{if task}\, i \text{ is successfully sequenced in subset}\, \bar{s}_r, \\ 0, & \text{otherwise.} \end{cases}$$

The following formulation is referred to as β-sequences. A value of $\beta_{i\bar{s}_r} = 0$ indicates that task i cannot be guaranteed not to overlap another task in section r, $i \in \mathcal{I}^{\text{sub}}_{\bar{s}_r}$, $r \in \mathcal{R}_h$, $h \in \mathcal{H}^{\text{CM}}$.

$$\sum_{j \in \mathcal{I}^{+}_{i\bar{s}_r}} y_{ij\bar{s}_r} = \beta_{i\bar{s}_r}, \quad i \in \mathcal{I}^{\text{sub}}_{\bar{s}_r} \backslash \{\tilde{q}\},\ r \in \mathcal{R}_h,\ h \in \mathcal{H}^{\text{CM}}, \tag{5.1.1}$$

$$\sum_{j \in \mathcal{I}^{-}_{i\bar{s}_r}} y_{ji\bar{s}_r} = \beta_{i\bar{s}_r}, \quad i \in \mathcal{I}^{\text{sub}}_{\bar{s}_r} \backslash \{\tilde{p}\},\ r \in \mathcal{R}_h,\ h \in \mathcal{H}^{\text{CM}}, \tag{5.1.2}$$

$$x_i + e_i \leq x_j + (t^{\mathrm{d}}_{ir} - t^{\mathrm{r}}_{jr})(1 - y_{ij\bar{s}_r}), \quad j \in \mathcal{I}^{+}_{i\bar{s}_r},\ i \in \mathcal{I}^{\mathrm{sub}}_{\bar{s}_r},\ r \in \mathcal{R}_h,\ h \in \mathcal{H}^{\mathrm{CM}}, \tag{5.1.3}$$

$$t^{\mathrm{r}}_{ir} \leq x_i \leq t^{\mathrm{d}}_{ir} - e_i, \quad i \in \mathcal{I}^{\mathrm{sub}}_{\bar{s}_r},\ r \in \mathcal{R}_h,\ h \in \mathcal{H}^{\mathrm{CM}}, \tag{5.1.4}$$

$$x_i = t^{r}_{i}, \quad i \in \mathcal{I}^{\mathrm{fix}}_{h},\ h \in \mathcal{H}^{\mathrm{CM}}, \tag{5.1.5}$$

$$\beta_{i\bar{s}_r} \in \{0,1\}, \quad i \in \mathcal{I}^{\mathrm{sub}}_{\bar{s}_r},\ r \in \mathcal{R}_h,\ h \in \mathcal{H}^{\mathrm{CM}}, \tag{5.1.6}$$

$$y_{ij\bar{s}_r} \in \{0,1\}, \quad j \in \mathcal{I}^{+}_{i\bar{s}_r},\ i \in \mathcal{I}^{\mathrm{sub}}_{\bar{s}_r},\ r \in \mathcal{R}_h,\ h \in \mathcal{H}^{\mathrm{CM}} \tag{5.1.7}$$

The objective of the subproblem, formulated as follows, is to maximize the number of tasks that are successfully sequenced.

$$\begin{aligned} \max \quad & \sum_{h\in\mathcal{H}^{\mathrm{CM}}} \sum_{r\in\mathcal{R}_h} \sum_{i\in\mathcal{I}^{\mathrm{sub}}_{\bar{s}_r}} \beta_{i\bar{s}_r} \\ \text{s.t.} \quad & \text{AM-scheduling} \quad (\text{Constraints}\,(4.1.1) - (4.1.5)) \\ & \text{Generated sequences}\,(\text{Constraints}\,(4.1.11) - (4.1.14)) \\ & \beta - \text{sequences}\,(\text{Constraints}\,(5.1.1) - (5.1.7)) \\ & \text{Precedence relations}\,(\text{Constraints}\,(4.2.1) - (4.2.4)) \\ & \text{CN-scheduling}\,(\text{Constraints}\,(4.3.1) - (4.3.6)) \end{aligned}$$

The objective value corresponds to the number of tasks that have been successfully sequenced. If the objective value is the same as the number of β-variables, a valid schedule has been found. If it is lower, there is at least one section where at least one task has not been successfully sequenced.

Convergence of the method is ensured by adding at least one generated sequence (Constraints (4.1.11)–(4.1.14)) in each iteration as long as a valid schedule is not found. If a task has not been successfully sequenced within its subset, this implies that the sequencing constraints of this subset were not previously generated, and progress, with respect to convergence, can be obtained by adding at least one such generated sequence.

5.1.3 Objective functions of the relaxed problem

The practical behaviour of the above constraint generation procedure relies heavily on the tasks being assigned to appropriate sections in the relaxed problem. Early empirical results indicated that it would be best to use one objective in the first iteration and another in the following iterations, to stabilise the search.

In the first iteration, the section-slack-objective,

$$\begin{aligned} \max \quad & \Omega \\ \text{s.t.} \quad & \Omega \leq l^{\mathrm{sec}}_{r} - \sum_{i\in\mathcal{I}^{\mathrm{sec}}_{r}} e_i \alpha_{ir}, \quad r \in \mathcal{R}_h,\ h \in \mathcal{H}^{\mathrm{CM}}, \end{aligned} \tag{5.1.8}$$

that maximises the smallest slack in Constraint (5.1.8) is one of the objectives used. Another is the centre-task-objective,

$$\begin{aligned}
\min \quad & \sum_{i \in \mathcal{I}^\gamma} \gamma_i \\
\text{s.t.} \quad & \gamma_i \geq \frac{2M_\Delta}{1-\Delta}\left(\frac{1-\Delta}{2} - \frac{x_i - t_i^{\mathrm{r}}}{t_i^{\mathrm{d}} - e_i - t_i^{\mathrm{r}}}\right), \quad i \in \mathcal{I}^\gamma, \\
& \gamma_i \geq \frac{2M_\Delta}{1-\Delta}\left(\frac{x_i - t_i^{\mathrm{r}}}{t_i^{\mathrm{d}} - e_i - t_i^{\mathrm{r}}} - \frac{1+\Delta}{2}\right), \quad i \in \mathcal{I}^\gamma, \\
& \gamma_i \geq 0, \quad i \in \mathcal{I}^\gamma,
\end{aligned}$$

where the set $\mathcal{I}^\gamma$ contains the non-fixed CM-tasks that have $t_i^{\mathrm{r}} \leq t_i^{\mathrm{d}}$. This objective directs tasks to be placed near the middle of their task interval by minimising the value of a penalty function illustrated in Fig. 2. The parameter Δ gives the part of the task interval where there is no penalty for placing the task. The continuous variable $\gamma_i \in [0,1]$, $i \in \mathcal{I}^\gamma$, is the linearly increasing penalty for placing a task outside this interval, and M_Δ is the maximum penalty.

From the second iteration onwards, the stabilise-objective,

$$\max \sum_{h \in \mathcal{H}^{\mathrm{CM}}} \sum_{r \in \mathcal{R}_h} \sum_{i \in \mathcal{I}_{\bar{s}_r}^{\mathrm{sub}}} \alpha_{ir}, \tag{5.1.9}$$

is used to maximise the number of non-fixed CM-tasks that stay in their previously assigned section.

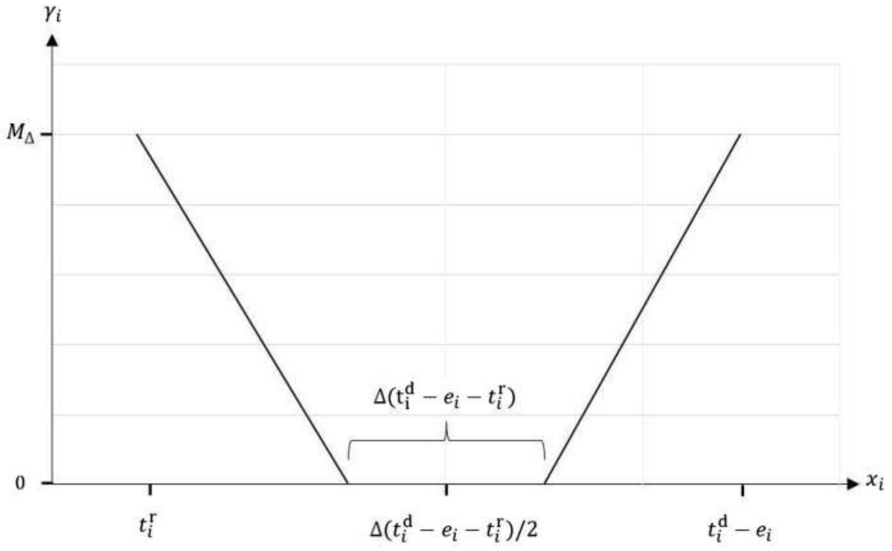

Fig. 2 An illustration of the penalty function used in the centre-task-objective

5.2 Preprocessing components

This section gives an overview of the preprocessing components implemented in the scheduling tool; for additional details see ‘‘Appendix 1’’. The purpose of Algorithm 2 is to restrict the release times and deadlines of tasks, while Algorithm 3 and Algorithm 4 are used to avoid creating variables and constraints that are redundant with respect to the data of a particular instance.

The observation behind Algorithm 2 is that, even if a task is allowed to start in the interval between its release time and deadline, dependencies with other tasks can imply that this interval is smaller. Algorithm 2 iteratively tightens the intervals of tasks to comply with the minimum and maximum length of dependencies without omitting feasible solutions.

Algorithm 3 determines the tasks that can be the immediate successor or predecessor of a task based on their release times, deadlines, and execution requirements. This translates to reducing the sets $\mathcal{I}_i^+$ and $\mathcal{I}_i^-$ for tasks on the AMs and the sets $\mathcal{I}_{is}^+$ and $\mathcal{I}_{is}^-$ for all subsets of tasks on the CMs. Algorithm 4 determines whether or not two jobs linked by a dependency can precede each other in a major frame. This translates to assigning a value to the variable v_d, $d \in \mathcal{D}$, that indicates if job k_d of task i_d precedes job l_d of task j_d in a major frame.

5.3 Overview of scheduling tool

Algorithm 1 provides an overview of the implemented scheduling tool with references to descriptions of the algorithm components and the models that have been introduced. The tool is implemented in Python Version 3.6.0, and the models are solved by Gurobi Optimizer Version 6.5.2. The choice of the objective function and parameter settings are further discussed in Sect. 6.

```
Data: A scheduling instance
Result: A valid schedule or a proof of infeasibility
Perform preprocessing according to Algorithm 2;
do
    Perform preprocessing according to Algorithms 3 and 4;
    Solve the relaxed problem;
    if Relaxed problem infeasible then
        Conclude that the problem is infeasible;
        Break;
    end
    Restrict all non-fixed CM-tasks to their assigned sections;
    Perform preprocessing according to Algorithms 2, 3 and 4;
    Solve the subproblem;
    if There are tasks that are not included in the $\beta$-sequences then
        Add at least one generated sequence to the relaxed problem and the
        subproblem;
    end
while At least one generated sequence is new;
```

Algorithm 1: Overview of the scheduling tool

5.4 Benchmark formulation

In an early stage of the project we created a benchmark formulation where, instead of our constraint generation procedure, all CM-tasks were to be sequenced by a Miller–Tucker–Zemlin formulation. The purpose of this formulation was to make comparisons for small instances and evaluate our preprocessing components.

6 Test results

This section presents our results showing that the solution strategy presented in this paper can be used to schedule avionic systems of industrial relevance. For this purpose Saab has provided three instances, named I, II, and III. Instance I corresponds to a minimum viable example of an avionic system with 2 nodes and a total of about 6500 tasks. Instance II has 5 nodes and a total of about 14,000 tasks and Instance III is the largest with 7 nodes and a total of about 20,000 tasks; see Tables 5, 6, and 7 for detailed information about each of the instances. In these tables, the numbers of dependencies, chains, and CN-messages are given for the complete system while the number of tasks is given for each module. In Instance II and Instance III there are two AMs that belong to the same CM, and this is presented by putting the number of AM-tasks for these two modules in parentheses in Tables 6 and 7, respectively.

An important characteristic of all the instances is that a large portion of the tasks at the CMs have fixed start times. For Instances I, II, and III, this portion is 65, 54, and 53%, respectively.

All the tests are carried out on a computer with two Intel Xeon E5-2640-v3 Processors (8 cores, 2.6 GHz) and 64 GB RAM. The scheduling tool is implemented to use a single core, with the exception that Gurobi is allowed to use all cores.

Early in the project we tried to solve Instance I with our benchmark formulation by using Gurobi after applying all our preprocessing components. Within a time limit of one week, no integer feasible solution was found.

Table 5 Characteristics of Instance I

Entities	Number of
CMs	2
CM-tasks	[3701, 2835]
Fixed tasks	[2816, 1404]
AMs	2
AM-tasks	[1, 1]
Dependencies	1457
CN-messages	64
Chains	998

Table 6 Characteristics of Instance II

Entities	Number of
CMs	5
CM-tasks	[5871, 2388, 2260, 1860, 1788]
Fixed tasks	[2832, 1408, 1408, 1408, 584]
AMs	6
AM-tasks	[7, 3, 3, 2, (3, 1)]
Dependencies	11779
CN-messages	96
Chains	1458

Table 7 Characteristics of Instance III

Entities	Number of
CMs	7
CM-tasks	[5871, 3867, 2388, 2260, 1860, 1860, 1788]
Fixed tasks	[2832, 1452, 1408, 1408, 1408, 1408, 584]
AMs	8
AM-tasks	[7, 4, 3, 3, 2, 2, (3, 1)]
Dependencies	15155
CN-messages	96
Chains	2002

6.1 Preprocessing effect

The purpose of our preprocessing components is to avoid creating variables that are redundant for a particular instance, and the effect of the preprocessing is summarised in Table 8.

The first step is to use Algorithm 2 to reduce the interval in which the tasks can be placed with respect to dependencies to other tasks. For Instance I, the effect is a

Table 8 Effect of preprocessing components

Measurement	Instance		
	I	II	III
Complete number of y-variables	22×10^6	52×10^6	70×10^6
Reduction of task intervals by Alg. 2	5%	18%	17%
Time for Alg. 2	27 s	103 s	123 s
Number of y-variables after Algs. 2 and 3	0.2×10^6	0.9×10^6	1.2×10^6
Reduction of v-variables by Alg. 4	47%	90%	92%

The numbers of y-variables are counted for the benchmark model (Sect. 5.4) with an earlier version of the scheduling tool, and the other results are from the scheduling tool running the constraint generation procedure

Table 9 Characterisation of subproblem

Measurement	Instance		
	I	II	III
Number of sections	2×10^3	4×10^3	6×10^3
Number of α-variables	3×10^4	11×10^4	15×10^4
Average number of β-variables in subproblem	2×10^3	6×10^3	8×10^3
Average number of y-variables in subproblem	2×10^4	5×10^4	7×10^4

reduction of about 5%. This indicates that for this instance the tasks are not particularly restricted by their release times, deadlines and dependencies with other tasks. For Instances II and III, the reduction is about 18%, and the preprocessing has practical relevance. For all instances, Algorithm 2 requires at most a few minutes of computational time.

The last row of Table 8 shows the percentage of v-variables for which Algorithm 4 detects that the value is fixed. For Instance I, 47% of the v-variables are fixed, and for Instances II and III, about 90% of the v-variables are fixed.

6.2 Solution approach evaluation

An important contribution of this paper, and the key that enables us to schedule the instances under consideration, is the reformulation of sequencing and the constraint generation procedure using the relaxed problem and the subproblem. Table 9 illustrates the impact of this decomposition in terms of the number of variables in the respective models. The first row gives the number of α-variables. An α-variable is created for a task-section pair only if a task can be scheduled in that section with respect to its release time and deadline.

In the subproblem, the β-variables are created only for tasks that are placed in a section with at least two tasks besides $\tilde{p}$ and $\tilde{q}$. For this reason, the number of β-variables can differ somewhat between iterations for the same instance. Table 9 presents the average number of β-variables over all iterations.

The outcome of the scheduling of Instances I, II, and III is summarised in Tables 10, 11, and 12, respectively. For each instance, we present the results for four choices of objective functions in the first relaxed problem. The solution times are given in seconds. The total time refers to the complete execution time for our scheduling tool, while the times for the relaxed problem and the subproblem refer to the time spent by Gurobi only. The difference between the complete execution time and the time to solve the models includes setup times and preprocessing. The most important parameter settings are:

- The time limit in the relaxed problem is 8 h.
- The relative MIP-gap in the relaxed problem is 0.10 for all runs.

Table 10 Results for Instance I

Measurement	Section-slack	Centre-task		
		$\Delta = 0.10$	$\Delta = 0.50$	$\Delta = 0.75$
Total time	164	243	467	182
Iterations	1	1	1	1
Generated sequences	–	–	–	–
Time relaxed problem	6	31	23	23
Time subproblem	1	53	283	2

Time is measured in seconds

Table 11 Results for Instance II

Measurement	Section-slack	Centre-task		
		$\Delta = 0.10$	$\Delta = 0.50$	$\Delta = 0.75$
Total time	1025	29676	2484	1623
Iterations	1	1	1	1
Generated sequences	–	–	–	–
Time relaxed problem	111	28800	946	483
Time subproblem	449	416	1081	680

Time is measured in seconds

Table 12 Results for Instance III

Measurement	Section-slack	Centre-task		
		$\Delta = 0.10$	$\Delta = 0.50$	$\Delta = 0.75$
Total time	2438	52269	7882	2210
Iterations	4	3	2	1
Generated sequences	[14, 9, 4]	[1, 1]	1	–
Time relaxed problem	[178, 33, 36, 82]	[28800, 41, 195]	[503, 34]	569
Time subproblem	[30, 38, 33, 356]	[49, 15609, 6294]	[42, 6369]	1079

Time is measured in seconds

- The time limit in the subproblem is initially 2 h, and whenever an improved integer solution is found, it is reset to 4 h.
- The relative MIP-gap in the subproblem is 0. This strict gap is required to ensure that all tasks are successfully sequenced.
- The value of M_{Δ} is 1000.
- In an iteration, at most 5 new generated sequences are added to the relaxed problem and the subproblem. If more than 5 sequences are generated, 5 of them are chosen at random.

The choices that have to be made with the most care are the objective function in the relaxed problem, the MIP-gap in the relaxed problem, and the time limit in the subproblem. The objective function and the MIP-gap in the relaxed problem are important because these choices impact in which sections the tasks are initially placed. The time limit in the subproblem is important because it has a large impact on the total running time and the quality of the feedback information in terms of subsets. If this time limit is too high, much time will be spent in the subproblem trying to include tasks in the β-sequences even if this is not possible, and if it is too low there is a risk that a solution with all the tasks included in a β-sequence might not be found even if such a solution exists.

For Instance I, a schedule is obtained within 10 min no matter which objective is used, and in all cases the section assignment made in the first relaxed problem makes it possible to include all the tasks in a β-sequence in the subproblem step. This exceptionally good outcome likely occurs because many feasible solutions exist for this instance, and the task intervals and the bounds on the dependency lengths are not very tight. This result can be contrasted with the benchmark formulation that we ran for one week without obtaining a solution.

For Instance II, the solution times range between 17 and 41 min, except when the centre-task-objective with $\Delta = 0.10$ was used. In that case the running time was 495 min. The reason for this long running time is that no solution with the required MIP-gap is obtained within the maximum running time of the first relaxed problem.

Instance III requires more than one iteration for all choices of objective function except for the centre-task objective with $\Delta = 0.75$. Also, as for Instance II, the centre-task objective for $\Delta = 0.1$ does not find a solution that meets the required MIP-gap of the first relaxed problem, which gives a significantly longer running time, 871 min. The total running times for the other objectives range between 37 and 131 min.

A conclusion that can be drawn about the choice of objective function in the first relaxed problem is that the section-slack objective and the centre-task objective with $\Delta = 0.75$ have provided the best results for the instances presented and that the performance of the solution strategy is sensitive to the choice of objective.

A general observation is that very few iterations are needed to solve each instance regardless of the choice of objective function. We believe that this is because we have chosen a suitable relaxation of the problem with respect to the characteristics of the industrially relevant instances, as described in Sect. 3.1.

7 Concluding remarks

This paper describes an avionics scheduling problem of industrial relevance and suggests a mathematical model for this problem. The exact solution strategy that we present is based on constraint generation and exploits known characteristics of our problem. Our computational results verify that we can solve industrially relevant instances that are significantly larger than those described in the literature within a reasonable time. We conclude that our approach is viable for this type of problem.

Our continued research aims to improve the components used in this strategy to further enhance the computational performance and thus solve even larger instances.

In the current model we have introduced the restriction that, for each communication module, at most one message can be sent or received in a slot. This means that the only way for us to handle the co-allocation of messages in a slot is to make this decision before the scheduling is carried out and then treat co-allocated messages as a single message. In practice, it would be highly relevant to integrate the co-allocation decisions with the scheduling. To introduce this degree of freedom makes the problem even more computationally challenging, since co-allocation of messages affects the tasks used for transmitting and receiving the messages. We leave this to future research.

The work of this paper is part of a long-term project on exact solution methods for avionics scheduling problems since they are expected to be important for future avionic development. In this paper we consider the scheduling of a system where all requirements are given, but it would also be of interest to develop decision support tools for the dimensioning of avionic systems and tools that can suggest which changes to make if no feasible solution is found by the scheduler.

Acknowledgements This work was supported by the Swedish Armed Forces, the Swedish Defence Materiel Administration and the Swedish Governmental Agency for Innovation Systems under Grant No. NFFP6-2014-00917. The work is also part of the project *Operations research methods for large scale scheduling and resource allocation problems* funded by the Center for Industrial Information Technology (CENIIT). The work of Emil Karlsson is supported by the Research School in Interdisciplinary Mathematics at Linköping University. The authors thank the reviewers for their valuable comments that helped improve the paper.

Appendix 1 Preprocessing

This section presents further details about the preprocessing components that were briefly introduced in Sect. 5.2.

Tightening of release times and deadlines of tasks

Algorithm 2 iteratively tightens the release times and deadlines of tasks linked by a dependency in order to comply with the length of the dependency. The following notation is used in the algorithm. Let L_i be the interval or union of intervals in which task $i \in \mathcal{I}$ can start. For task i_d and task j_d connected by the dependency d, $d \in \mathcal{D}$, let the function $\mathbf{L}_{i_d}(d, L_{j_d})$ return the interval or union of intervals in which task i_d can be placed with respect to dependency d and the interval or union of intervals L_{j_d}.

```
Data: Tasks and dependencies
Result: Release times and deadlines of tasks
Let $k$=0;
For $i \in \mathcal{I}$: Let $L_i^0 = L_i$;
while The interval where a task can start can be tightened do
    for Dependency $d$, $d \in \mathcal{D}$ do
        Let $L_{i_d}^{k+1} = \mathbf{L}_{i_d}(d, L_{j_d}^k) \cap L_{i_d}^k$;
        Let $L_{j_d}^{k+1} = \mathbf{L}_{j_d}(d, L_{i_d}^k) \cap L_{j_d}^k$;
        Let $k = k + 1$;
    end
end
For $h \in \mathcal{H}^{\text{AM}}$, $i \in \mathcal{I}_h$: Let $L_i = L_i^k$ and update $t_i^{\text{r}}$ and $t_i^{\text{d}}$ to define the smallest
interval where task $i$ can start with respect to $L_i$;
For $h \in \mathcal{H}^{\text{CM}}$, $r \in R_h$, $i \in \mathcal{I}_r^{\text{sec}}$: Let $L_i = L_i^k$ and update $t_{ir}^{\text{r}}$ and $t_{ir}^{\text{d}}$ to define the
smallest interval where task $i$ can start in section $r$ with respect to $L_i$;
```

Algorithm 2: Preprocessing to update the release times and deadlines of tasks with respect to their dependencies

Sequencing variables

Since the number of y-variables for each module in the worst case can grow quadratically with the number of tasks to be sequenced, it is important to not include an excessive number of variables that anyway cannot take the value one in a model. This translates to reducing the sets $\mathcal{I}_i^+$ and $\mathcal{I}_i^-$ for tasks on the AMs and the sets $\mathcal{I}_{is}^+$ and $\mathcal{I}_{is}^-$ for all subsets of tasks on the CMs. The sets are reduced by using the fact that two tasks cannot be each other's immediate predecessor and immediate successor, respectively, if another task must be executed between them. By comparing the release times and the deadlines of tasks, such cases can be identified, and this is the core idea implemented in Algorithm 3, where the tasks are assumed to be ordered with respect to ascending release time.

```
Data: Tasks and their intervals
Result: Which tasks that can follow each of the tasks in the set $\tilde{\mathcal{I}}$
/* Handle tasks that can cross a major frame                              */
for Task $i$ with $t_i^d \le t_i^r$, $i \in \tilde{\mathcal{I}}$ do
    In $\tilde{\mathcal{I}}$, replace task $i$ with
    task $i_1 : t^{\mathrm{r}}_{i_1} = 0$, $t^{\mathrm{d}}_{i_2} = t^{\mathrm{d}}_i$ and
    task $i_2 : t^{\mathrm{r}}_{i_1} = t^{\mathrm{d}}_i$, $t^{\mathrm{d}}_{i_2} = P$;
end
/* Include tasks in the sets of followers                                 */
for Task $i$, $i \in \tilde{\mathcal{I}}$ do
    Let $i'$ be such that $t^{\mathrm{d}}_{i'} = \min_{i''} t^{\mathrm{d}}_{i''}$,
    where $i'' \in \tilde{\mathcal{I}}$ such that $t^{\mathrm{r}}_{i''} \ge t^{\mathrm{d}}_i$;
    for Task $j$ such that $t_i^r \le t_j^r + e_j \le t_{i'}^d - e_{i'}$, $j \in \tilde{\mathcal{I}}$ $^{(*)}$ do
        if $t_i^r + e_i \le t_j^d - e_j$ then
            Include task $j$ in the set $\tilde{\mathcal{I}}_i^+$
        end
        if $t_j^r + e_j \le t_i^d - e_i$ then
            Include task $i$ in the set $\tilde{\mathcal{I}}_j^-$;
        end
    end
end
/* Restore tasks that can cross a major frame                             */
for Each task $i$ that were split into $i_1$ and $i_2$, $i \in \tilde{\mathcal{I}}$ do
    Where applicable, replace $i_1$ and $i_2$ by $i$;
    Let $\tilde{\mathcal{I}}_i^+ = \tilde{\mathcal{I}}_{i_1}^+ \cup \tilde{\mathcal{I}}_{i_2}^+$,
    delete $\tilde{\mathcal{I}}_{i_1}^+$ and $\tilde{\mathcal{I}}_{i_2}^+$;
end
```

Algorithm 3: Preprocessing to determine which tasks of a set $\tilde{\mathcal{I}}$ that can be the immediate successors $\tilde{\mathcal{I}}_i^+$ and the immediate predecessors $\tilde{\mathcal{I}}_i^-$ of task i, $i \in \tilde{\mathcal{I}}$. $^{(*)}$A pair of tasks i, j such that $t_i^{\mathrm{s}} = t_j^{\mathrm{s}}$ is considered only at its first occurrence

Dependency indicator variables

By an evaluation of the release times and deadlines of tasks linked by a dependency d, $d \in \mathcal{D}$, the variable v_d that indicates if job k_d of task i_d precedes job l_d of task j_d in a major frame can sometimes be a priori fixed to either 0 or 1. The computations required to determine the possible values of these variables are given in Algorithm 4.

```
Data: Tasks and dependencies
Result: Possible values of dependency indicator variables or a proof of infeasibility
for Dependency $d$, $d \in \mathcal{D}$ do
    Let:
    $t_i^{\mathrm{min}} = t_{i_d}^{\mathrm{r}} + k_d p_i$
    $t_i^{\mathrm{max}} = t_{i_d}^{\mathrm{d}} + k_d p_i - e_i$,
    $t_j^{\mathrm{min}} = t_j^{\mathrm{r}} + l_d p_j$,
    $t_j^{\mathrm{max}} = t_j^{\mathrm{d}} + l_d p_j - e_j$;
    if $t_j^{max} - t_i^{min} \geq l_d^{min\text{-}dep}$ and
    $t_j^{min} - t_i^{max} \leq l_d^{max\text{-}dep} - P$ then
        Let $v_d \in \{0, 1\}$;
    else if $t_j^{max} - t_i^{min} \geq l_d^{min\text{-}dep} - P$ and
    $t_j^{min} - t_i^{max} \leq l_d^{max\text{-}dep} - P$ then
        Let $v_d = 1$;
    else if $t_j^{max} - t_i^{min} \geq l_d^{min\text{-}dep}$ and
    $t_j^{min} - t_i^{max} \leq l_d^{max\text{-}dep}$ then
        Let $v_d = 0$;
    else
        Conclude that the problem is infeasible;
        Break;
    end
end
```

Algorithm 4: Preprocessing to determine possible values of dependency indicator variables

References

Airlines Electronic Engineering Committee (AEEC) (2006) Avionics application software standard interface, ARINC specification 653 (part 1)

Al-Sheikh A (2011) Resource allocation in hard real-time avionic systems. Scheduling and routing problems. PhD thesis, INSA de Toulouse

Al-Sheikh A, Brun O, Hladik P-E, Prabhn B (2012) Strictly periodic scheduling in IMA-based architectures. Real Time Syst 48(4):359–386

Al-Sheikh A, Brun O, Chéramy M, Hladik P-E (2013) Optimal design of virtual links in AFDX networks. Real Time Syst 49(3):308–336

Balashov VV, Balakhanov VA, Kostenko VA (2014) Scheduling of computational tasks in switched network-based IMA systems. In: Proceedings of the 1st international conference on engineering and applied sciences optimization

Beji S, Hamadou S, Gherbi A, Mullins J (2014) SMT-based cost optimization approach for the integration of avionics functions in IMA and TTEthernet architectures. In: Proceedings of the IEEE/ACM 18th international symposium on distributed simulation and real time applications, pp 165–174

Craciunas SS, Oliver RS (2014) SMT-based task- and network-level static schedule generation for time-triggered networked systems. In: Proceedings of the 22nd international conference on real-time networks and systems, pp 45–54

Danielsson T, Pettersson A, Gripsborn A, Håkegård J (2016) Ethernet for avionics. Patent, 04 2016. EP 2583419

Davis RI, Burns A (2011) A survey of hard real-time scheduling for multiprocessor systems. ACM Comput Surv 43(4):1–44

Easwaran A, Lee I, Sokolsky O, Vestal S (2009) A compositional scheduling framework for digital avionics systems. In: Proceedings of the 15th IEEE international conference on embedded and real-time computing systems and applications, pp 371–380

Eisenbrand F, Kesavan K, Mattikalli RS, Niemeier M, Nordsieck AW, Skutella M, Verschae J, Wiese A (2010) Solving an avionics real-time scheduling problem by advanced IP-methods. In: Algorithms—ESA 2010. Lecture notes in computer science, vol 6346. Springer, Berlin Heidelberg, pp 11–22

Kopetz H (2011) Real-time systems: design principles for distributed embedded applications. Real-time systems series. Springer, Berlin

Lee Y-H, Kim D, Younis M, Zhou J (2000) Scheduling tool and algorithm for integrated modular avionics systems. In: Proceedings of the digital avionics systems conference, pp 1–8

Leung JYT (ed) (2004) Handbook of scheduling: algorithms, models, and performance analysis. Chapman & Hall/CRC Computer and Information Science Series. Taylor & Francis, Milton Park, ISBN 9781135438852

Manne AS (1960) On the job-shop scheduling problem. Oper Res 8(2):219–223

Miller CE, Tucker AW, Zemlin RA (1960) Integer programming formulation of traveling salesman problems. J ACM 7(4):326–329

Pop P, Eles P, Peng Z (1999) Scheduling with optimized communication for time-triggered embedded systems. In: Proceedings of the seventh international workshop on hardware/software codesign, pp 178–182

Radio Technical Commission for Aeronautics (RTCA) (2005) Integrated modular avionics (IMA) development guidance and certification considerations, RTCA DO-297

Rufino J, Craveiro J, Verissimo P (2010) Architecting robustness and timeliness in a new generation of aerospace systems. In: Casimiro A, de Lemos R, Gacek C (eds) Architecting dependable systems VII. Lecture notes in computer science, vol 6420. Springer, Berlin, Heidelberg, pp 146–170

Tavares E, Maciel P, Sousa E, Nogueira B, Amorim L, Lira V (2012) A hierarchical pre-runtime scheduling for hard real-time systems considering fault-tolerance. In: Proceedings of the IEEE international conference on systems, man, and cybernetics, pp 1207–1212

Theis J, Fohler G, Baruah S (2013) Schedule table generation for time-triggered mixed criticality systems. In: Proceedings of the workshop on mixed criticality systems at IEEE real-time systems symposium, pp 79–84

Tămaş-Selicean D, Pop P, Steiner W (2012) Synthesis of communication schedules for TTEthernet-based mixed-criticality systems. In: Proceedings of the eighth IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis, pp 473–482

Xu J, Parnas DL (2000) Priority scheduling versus pre-run-time scheduling. Real Time Syst 18(1):7–23

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 design automation conference (ASP-DAC), 19th Asia and South Pacific, pp 119–124