Department of Mathematics

An Optimisation Approach for Pre-Runtime Scheduling of Tasks and Communication in an Integrated Modular Avionic System

Mathias Blikstad, Emil Karlsson, Tomas Lööw
and Elina Rönnberg

LiTH-MAT-R--2017/03--SE

Department of Mathematics
Linköping University
S-581 83 Linköping

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], and Elina Rönnberg[1,2]

[1]*Saab AB, SE-581 88 Linköping, Sweden*
[2]*Department of Mathematics, Linköping University, SE-581 83 Linköping, Sweden , elina.ronnberg@liu.se*

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 the avionic systems are growing more and more complex, so is the challenge of scheduling them. Scheduling of the system has an important role in the development of new avionic systems since functionality typically is 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, in case 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 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 reasonable time.

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). Scheduling of real-time systems can refer both to on-line scheduling where the scheduling

decisions are made at run time and to pre-runtime scheduling where the schedule is created at compile time. This paper addresses pre-runtime scheduling of an avionic system with periodic tasks.

During the last two decades, the majority 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. Such architecture necessitates strict requirements on the spatial and temporal partitioning of the system to achieve fault containment, and 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 parts of the PhD-thesis by Al-Sheikh (2011) provide an extensive introduction to the area of 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 fails, preferably produce a proof of infeasibility. If no feasible schedule exist either changes of the software or upgrades of the avionics platform are needed. Due to 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 approaches for pre-runtime scheduling of large-scale real-time systems are of a heuristic nature, see for example the references in Section 1.2. For many types of real-time systems, a primal heuristic might be an efficient and sound way to provide a schedule. This does however not hold for an avionic scheduling problem when the scheduling also involves the challenge to determine whether a desired software functionality can be implemented with the existing platform or not. If a primal heuristic method is applied to such problem setting and the heuristic fails to provide a solution, one does not know if this depends on shortcomings of the heuristic or if it is due to the fact that no feasible solution exists. This paper contributes in the direction of developing optimisation based approaches for scheduling of large and complex future avionics systems, and the research is 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 Section 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 Section 1.2.

1.1 System characteristics

There is a diversity in what type of scheduling that is 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 under consideration in this paper.

The system is distributed and at each node there is a set of processors, henceforth 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 tasks are beforehand assigned to modules and no migration is allowed.

At the communication modules, tasks that have to do 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, Part IV in 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 over 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 and for 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 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), of which the latter suggests a strategy for 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 aspect a part of the temporal partitioning of the system, and thereby the message scheduling becomes more intricately integrated with the scheduling of partitions. The solution approach suggested in this paper can however also be applied to systems where an AFDX is used for the communication.

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 19894 tasks.

1.2 Related research

There are many papers on scheduling of real-time systems that consider runtime scheduling, which is of a different nature than pre-runtime scheduling. The reader interested in 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 between runtime and pre-runtime scheduling is provided by Xu and Parnas (2000), and they suggest pre-runtime scheduling as the preferred choice for the type of applications similar to the one in this paper. Parts of their conclusions are 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 to achieve. Comparing the runtime and the pre-runtime approach, the latter can be considered as a constructive sufficient schedulability test, see Section 10.3 in Kopetz (2011).

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 excludes pairs of tasks to be allocated 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 under consideration in our paper. However, in Al-Sheikh et al. (2012), an AFDX network is used for communication between the partitions, and for this reason it is sufficient to respect precedence relations between tasks in order to assure communication. From that point of view, their setting is significantly less complex than in this paper. For solving the problem, they suggest a heuristic inspired by game theory principles and they successfully apply it to instances that have been supplied by industrial partners and contain up to 48 processors and 636 tasks.

For the same type of problem setting, an exact IP-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 IP-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 of 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), they do however address significantly less complex problems than what is of relevance in the context of this paper. The same hold for a later paper by Tavares et al. (2012) that presents an approach where they generate schedules by using a time Petri net model (a graph representation of concurrent processes) and 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 only evaluate their strategy

for uni-processor systems with the maximum number of tasks being 12 (real-world applications and custom-built instances), which are significantly smaller instances than what is of interest in our context.

Another related paper on 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 evaluate their approach on an application with 5 nodes and 7 partitions.

Various methods for 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 MIP-solver (Zhang et al., 2014), or a SMT-solver (Beji et al., 2014; Craciunas and Oliver, 2014). Neither of these approaches are viable in our setting and typically the exact approaches are applicable only to relatively small instances of scheduling problems, 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 Section 2. The main computational challenge of the problem stems from the huge number of tasks to be sequenced on the communication modules and Section 3 introduces our approach for exploiting known characteristics of the problem in order to design a constraint generation procedure for solving it. Our resulting MIP-formulation is presented in Section 4 and the constraint generation procedure along with some preprocessing components are introduced in Section 5. Computational results to verify that our approach can be used to solve instances of practical relevance are presented in Section 6, followed by concluding comments in Sections 7.

2 System model

This section introduces the concepts and notations needed to create a modelling framework for the system, which is illustrated in Figure 1. A schedule for a major frame is the result of two types of decisions, one is assigning communication messages to time slots and the other is assigning start times to tasks.

The system executes periodically with period P, referred to as a major frame. 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. For each node there exist a single module called the communication module (CM) and it handles the system external, intra-node and inter-node communication of this node. The set of all CMs in the system is denoted by $\mathcal{H}^{\mathrm{CM}}$. The set of tasks assigned to CM h is denoted by $\mathcal{I}_h$, $h \in \mathcal{H}^{\mathrm{CM}}$, and every task on a CM has the period of a major frame.

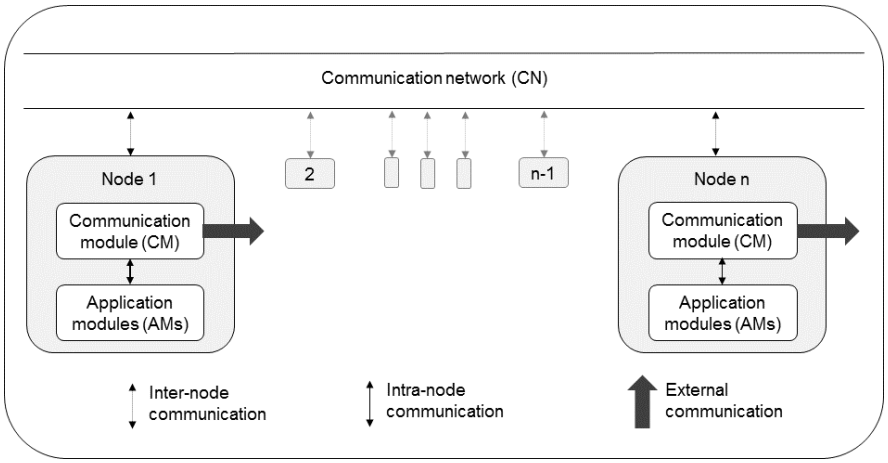

Figure 1: An overview of a system with n nodes. The structure of the content is the same for all nodes and therefore only displayed for node 1 and n

On each node there exists a set of modules, called application modules (AM), that hosts 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 every such task has a period of $P/64$, $h \in \mathcal{H}^{\text{AM}}$. For each pair of tasks i and j there is a minimum idle time l_{ij}^{idle} between the end of task i to the start of task j if task j follows task i on AM h, where $i, j \in \mathcal{I}_h$ and $h \in H^{\text{AM}}$. This idle time ensures that there is enough time to handle intra-node communication related to the two tasks.

In this system, the tasks are beforehand assigned to execute on a specific module. Task i generate an infinite succession of instances that execute once every period p_i for the duration of its execution requirement e_i, $i \in \mathcal{I}$. Task i must be given a non-preemptive execution interval of duration e_i between its release time t_i^{r} and deadline t_i^{d} and within this interval no other task is allowed to execute on the same module, $i \in \mathcal{I}$.

In order to model the problem within a single major frame, a specific instance of task i within a major frame will be called job k of task i. This means that job k of task i will be all instances $k + \frac{P}{p_i}n$ of task i where n is a non-negative integer. The set of jobs belonging to task i will be denoted $\mathcal{J}_i$, $i \in \mathcal{I}$.

2.2 Precedence relations

There is a set of precedence relations called dependencies, this set is denoted by $\mathcal{D}$. 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 $l_d^{\text{max-dep}}$, $d \in \mathcal{D}$. This means that if job k_d of task i_d occurs before job l_d of task j_d in a major frame, the distance is measured within one major frame, and if job k_d of task i_d occurs after job l_d of task j_d within a major frame, the distance is measured from one major frame into the next.

There is also a set of restrictions called chains, this set is 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 execute 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, $c \in \mathcal{C}$. Since

Table 1: The parameters and sets of the entities introduced in Section 2.1

Entity	Parameter	Notation
System	major frame	P
	CMs	$\mathcal{H}^{\mathrm{CM}}$
	AMs	$\mathcal{H}^{\mathrm{AM}}$
	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}
	jobs	$\mathcal{J}_i$

the dependencies in $\mathcal{D}_c^{\mathrm{chain}}$ create a cycle, the choice of a first job in the chain is of no significance.

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}, $m \in \mathcal{M}$. In order to transmit a CN-message it has to 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

Table 2: The parameters and sets of the entities introduced in Section 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	$\mathcal{D}_c^{\mathrm{chain}}$

Table 3: The parameters and sets of the entities introduced in Section 2.3

Entity	Parameter	Notation
CN	CN-messages	$\mathcal{M}$
	CN-slots	$\mathcal{N}$
CM	CN-messages that is received on CM h	$\mathcal{M}_h^{\text{recv}}$
	CN-messages that is sent on CM h	$\mathcal{M}_h^{\text{send}}$
CN-message m	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}

l_n^{slot}, $n \in \mathcal{N}$. This paper consider the case where each CM can only send one CN-message in each CN-slot and only receive one CN-message in each CN-slot.

To transmit a CN-message on a CN, there are four types of tasks involved and those are required to execute 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 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, $i \in \mathcal{I}_m^{\text{msg}}$, $n \in \mathcal{N}$, $m \in \mathcal{M}$.

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 presents our strategy for sequencing of CM-tasks along with the notation needed for the mathematical model.

3.1 Strategy

An important characteristic of our problem is that the CMs have a huge number of tasks and that a considerable amount of these 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 based on the following proposition.

Proposition 1 (Sequencing strategy) *To require that the tasks belonging to the union of a set of non-fixed tasks and a set of fixed tasks are sequenced is equivalent to require the following.*

Table 4: The parameters and sets of the entities introduced in Section 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	$\mathcal{I}_r^{\text{sec}}$
Subset s	tasks	$\mathcal{I}_s^{\text{sub}}$
Task i	release time in section r	t_{ir}^{r}
	deadline in section r	t_{ir}^{d}
	sections	$\mathcal{R}_i^{\text{task}}$

- *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 includes extremely many constraints, but since they are of interest only for a subset of tasks assigned to the same section, this approach lends itself to constraint generation. This strategy for sequencing will be used in the model presented in Section 4 and exploited in the constraint generation procedure in Section 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 execute, $h \in \mathcal{H}$. 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 execute, $i \in \mathcal{I}_h \setminus \mathcal{I}_h^{\text{fix}}$, $h \in \mathcal{H}^{\text{CM}}$. 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 set $\mathcal{I}_s^{\text{sub}}$, $s \in \mathcal{S}_h$, $h \in \mathcal{H}^{\text{CM}}$, includes non-fixed tasks that can, for at least one section, be assigned together in the same section.

4 Mixed-Integer Programming (MIP) 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 one major frame.

4.1 Tasks and sequencing

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

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

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

4.1.1 AM

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 for the setup times, see Manne (1960),

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

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

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

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

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

4.1.2 CM

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

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

The following constraint ensure that each non-fixed task is assigned to one section, that the capacity of each section is respected, that each non-fixed task obeys its release time and deadline within its section, and that fixed tasks comply with their release time.

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

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

$$\sum_{r \in \mathcal{R}_i^{\text{task}}} t_{ir}^{\text{r}} \alpha_{ir} \leq x_i \leq \sum_{r \in \mathcal{R}_i^{\text{task}}} t_{ir}^{\text{d}} \alpha_{ir} - e_i, \qquad i \in \mathcal{I}_h \setminus \mathcal{I}_h^{\text{fix}},\ h \in \mathcal{H}^{\text{CM}}, \quad (4.1.8)$$

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

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

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 $h \in \mathcal{H}^{\text{CM}}$, $s \in \mathcal{S}_h$, $i \in \mathcal{I}_s^{\text{sub}} \setminus \{\tilde{q}\}$, $j \in \mathcal{I}_{is}^+$, 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 each subset of tasks we apply a Miller-Tucker-Zemlin formulation, see Miller et al. (1960), for sequencing the tasks.

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

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

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

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

4.2 Precedence relations

For dependency d, introduce a variable

u_d = time between job k_d of task i_d and the next occurrence of job l_d 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

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

referred to as the dependency indicator variable. The following constraints defines the length of the dependency and restricts it to be within its minimum and maximum length.

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

$$l_d^{\text{min-dep}} \leq u_d \leq l_d^{\text{max-dep}}, \qquad d \in \mathcal{D}, \ (4.2.2)$$

$$v_d \in \{0, 1\}, \qquad d \in \mathcal{D} \ (4.2.3)$$

The dependencies in a chain ensure that each pair of jobs linked by a dependency has a particular order in a major frame. The following constraint ensure that the cycle of these dependencies finishes within the duration of a major frame.

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

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 constraints ensure that each CN-message is assigned to a slot, make sure that the capacity of a slot is respected, that at most one message can be send or received in a slot for each CM, and that tasks involved in transmitting a CN-message respect their release times and deadlines induced by assigning their message to a slot.

$$\sum_{n \in \mathcal{N}} z_{nm} = 1, \qquad m \in \mathcal{M}, \ (4.3.1)$$

$$\sum_{m \in \mathcal{M}} l_m^{\text{msg}} z_{nm} \leq l_n^{\text{slot}}, \qquad n \in \mathcal{N}, \ (4.3.2)$$

$$\sum_{m \in \mathcal{M}_h^{\text{recv}}} z_{nm} \leq 1, \qquad n \in \mathcal{N}, \ h \in \mathcal{H}^{\text{CM}}, \ (4.3.3)$$

$$\sum_{m \in \mathcal{M}_h^{\text{send}}} z_{nm} \leq 1, \qquad n \in \mathcal{N}, \ h \in \mathcal{H}^{\text{CM}}, \ (4.3.4)$$

$$\sum_{n \in \mathcal{N}} t_{in}^{\text{r}} z_{nm} \leq x_i \leq \sum_{n \in \mathcal{N}} t_{in}^{\text{d}} z_{nm} - e_i, \qquad i \in \mathcal{I}_m^{\text{msg}}, \ m \in \mathcal{M}, \ (4.3.5)$$

$$z_{nm} \in \{0, 1\}, \qquad n \in \mathcal{N}, \ m \in \mathcal{M} \ (4.3.6)$$

5 Solution approach

This section presents our constraint generation procedure, the preprocessing components used, and a summerative 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 for solving the problem, the problem is initially solved without them and the ones needed are added iteratively in a constraint generation procedure.

This section introduces the two models that are used in the constraint generation procedure. The first model is a relaxation of the full model obtained by removing the constraints used for sequencing of subsets of CM-tasks. This model is referred to as the α-model and its purpose is to assign non-fixed CM-tasks to sections. The second model is referred to as the β-model and it attempts to sequence non-fixed CM-tasks given an α-model solution. The solution to the β-model is either a valid schedule or it provides information about which constraints to generate and add to both models. Such generated constraint are referred to as generated sequences.

5.1.1 α-model

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

min/max	Objective function	
s.t.	Objective constraints	
	AM-scheduling	(Constraint 4.1.1 – 4.1.5)
	Generated sequences	(Constraint 4.1.12 – 4.1.14)
	CM-assignment	(Constraint 4.1.8 – 4.1.10)
	Precedence relations	(Constraint 4.2.2 – 4.2.4)
	CN-scheduling	(Constraint 4.3.1 – 4.3.6)

In the first iteration there are no generated sequences but in later iterations the ones that are identified by the β-model are included in the model.

5.1.2 β-model

The β-model is obtained from a solution to the α-model as follows. For section r, let the subset of tasks that was assigned to this section be denoted by $\bar{s}_r$, $r \in \mathcal{R}_h$, $h \in \mathcal{H}^{\mathrm{CM}}$. Restrict the release time and the deadline of task i to be t^{r}_{ir} and t^{d}_{ir}, respectively, $i \in \mathcal{I}^{\mathrm{sub}}_{\bar{s}_r}$, $r \in \mathcal{R}_h$, $h \in \mathcal{H}^{\mathrm{CM}}$. Introduce, for $h \in \mathcal{H}^{\mathrm{CM}}$, $r \in \mathcal{R}_h$, $i \in \mathcal{I}^{\mathrm{sub}}_{\bar{s}_r}$, 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 where a value of $\beta_{i\bar{s}_r} = 0$ indicate that task i could not be ensured not to overlap another task in section r, $i \in \mathcal{I}^{\mathrm{sub}}_{\bar{s}_r}$, $r \in \mathcal{R}_h$, $h \in \mathcal{H}^{\mathrm{CM}}$.

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

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

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

$$t^{\text{r}}_{ir} \le x_i \le t^{\text{d}}_{ir} - e_i \qquad i \in \mathcal{I}^{\text{sub}}_{\bar{s}_r},\ r \in \mathcal{R}_h,\ h \in \mathcal{H}^{\text{CM}}, \quad (5.1.4)$$

$$x_i = t^r_i \qquad i \in \mathcal{I}^{\text{fix}}_h,\ h \in \mathcal{H}^{\text{CM}}, \quad (5.1.5)$$

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

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

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

$$\max \quad \sum_{h \in \mathcal{H}^{\text{CM}}} \sum_{r \in \mathcal{R}_h} \sum_{i \in \mathcal{I}^{\text{sub}}_{\bar{s}_r}} \beta_{i\bar{s}_r}$$

s.t.

AM-scheduling	(Constraint 4.1.1 – 4.1.5)
Generated sequences	(Constraint 4.1.12 – 4.1.14)
β-sequences	(Constraint 5.1.1 – 5.1.7)
Precedence relations	(Constraint 4.2.2 – 4.2.4)
CN-scheduling	(Constraint 4.3.1 – 4.3.6)

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 the objective value is lower than that, 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 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 was 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 α-model

The practical behaviour of the above described constraint generation procedure relies heavily on the tasks being assigned to appropriate sections in the α-model. Early empirical results suggested it to be wise to use one objective in the first iteration and another in the following iterations.

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

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

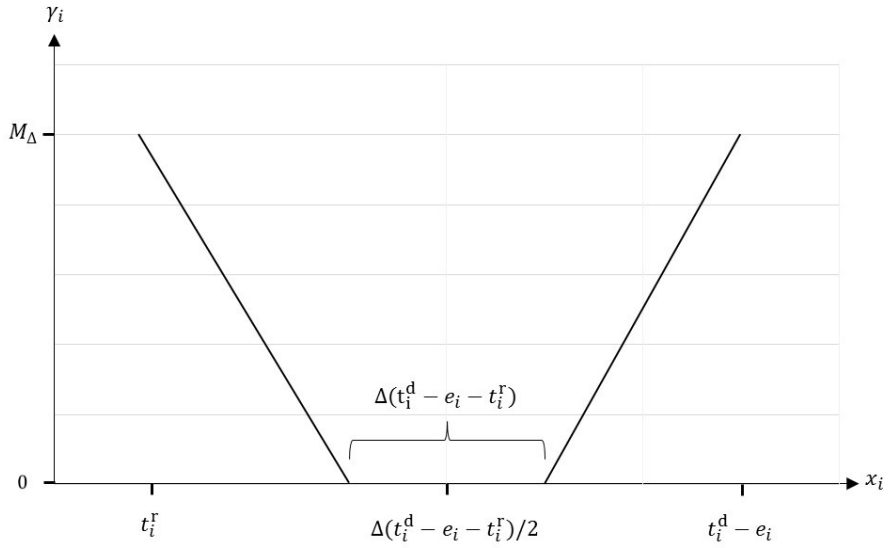

Figure 2: An illustration of the penalty function used in the center-task-objective

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

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

where the set $\mathcal{I}^\gamma$ contain 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 Figure 2. The parameter Δ gives the part of the task interval where there is no penalty for placing the task. The 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,

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

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

5.2 Pre-processing components

This section sketches the pre-processing principles that have been used to avoid creating variables and constraints that are redundant with respect to the data

of a particular instance.

5.2.1 Tightening of release times and deadlines of tasks

Originally, a task is allowed to start between its release time and deadline, but dependencies to other tasks can imply that this interval is smaller. Algorithm 1 describes how the release times and deadlines of tasks are iteratively tightened, without omitting feasible solutions.

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 $h \in \mathcal{H}^{\mathrm{AM}},\ i \in \mathcal{I}_h$: Let $L_i = L_i^k$ and update t_i^{r} and t_i^{d} to define the smallest interval where task i can start with respect to L_i;
For $h \in \mathcal{H}^{\mathrm{CM}},\ r \in R_h,\ i \in \mathcal{I}_r^{\mathrm{sec}}$: Let $L_i = L_i^k$ and update t_{ir}^{r} and t_{ir}^{d} to define the smallest interval where task i can start in section r with respect to L_i;

Algorithm 1: Preprocessing to update the release times and deadlines of tasks with respect to their dependencies

5.2.2 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. How this is done is presented in Algorithm 2, where the tasks are assumed to be ordered with respect to ascending release time.

5.2.3 Dependency indicator variables

The variable v_d, $d \in \mathcal{D}$, that indicates if job k_d of task i_d precede job l_d of task j_d in a major frame can, depending on where the involved tasks can be placed, sometimes be beforehand fixed. The computations required to determine the possible values of these variables are given in Algorithm 3.

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* $t_i^d \leq t_i^r$, $i \in \tilde{\mathcal{I}}$ **do**
  In $\tilde{\mathcal{I}}$, replace task i with
  task $i_1 : t_{i_1}^{\mathrm{r}} = 0$, $t_{i_2}^{\mathrm{d}} = t_i^{\mathrm{d}}$ and
  task $i_2 : t_{i_1}^{\mathrm{r}} = t_i^{\mathrm{d}}$, $t_{i_2}^{\mathrm{d}} = P$;
end
`/* Include tasks in the sets of followers */`
for *Task* i, $i \in \tilde{\mathcal{I}}$ **do**
  Let i' be such that $t_{i'}^{\mathrm{d}} = \min_{i''} t_{i''}^{\mathrm{d}}$,
  where $i'' \in \tilde{\mathcal{I}}$ such that $t_{i''}^{\mathrm{r}} \geq t_i^{\mathrm{d}}$;
  for *Task* j *such that* $t_i^r \leq t_j^r + e_j \leq t_{i'}^d - e_{i'}$, $j \in \tilde{\mathcal{I}}$ $^{(*)}$ **do**
    if $t_i^r + e_i \leq t_j^d - e_j$ **then**
      Include task j in the set $\tilde{\mathcal{I}}_i^+$
    end
    if $t_j^r + e_j \leq t_i^d - e_i$ **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 2: 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 only considered at its first occurrence.

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:
  $t_i^{\min} = t_{i_d}^{\mathrm{r}} + k_d p_i$
  $t_i^{\max} = t_{i_d}^{\mathrm{d}} + k_d p_i - e_i$,
  $t_j^{\min} = t_j^{\mathrm{r}} + l_d p_j$,
  $t_j^{\max} = t_j^{\mathrm{d}} + l_d p_j - e_j$;
  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 3: Preprocessing to determine possible values of dependency indicator variables.

5.3 Overview of scheduling tool

Algorithm 4 provides an overview of the implemented scheduling tool with references to descriptions of the algorithm components and the models that have been introduced. The implementation of the scheduling tool is made in Python Version 3.6.0 and the models have been solved by Gurobi Optimizer Version 7.0.2. The choice of objective function and parameter settings are further discussed in Section 6.

Data: A scheduling instance
Result: A valid schedule or a proof of infeasibility
Perform pre-processing according to Algorithm 1;
do
  Perform pre-processing according to Algorithm 3 and 2;
  Solve the α-model;
  if *α-model infeasible* **then**
    Conclude that the problem is infeasible;
    Break;
  end
  Restrict all non-fixed CM-tasks to their assigned section;
  Perform pre-processing according to Algorithm 1, 3 and 2;
  Solve the β-model;
  if *There are tasks that are not included in the β-sequences* **then**
    Add at least one new sequence formulation to the α-model and the β-model;
  end
while *At least one sequence formulation is new*;

Algorithm 4: 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 pre-processing components.

6 Test results

The purpose of this section is to present our results that verify 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 two 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, respectively. In these tables, the number of dependencies, chains and CN-messages are given for the complete system while the number of tasks are given for each module.

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

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

All test are carried out on a computer with two Intel Xeon E5-2640-v3 Processors (8 cores, 2.6 GHz) and 64 GB RAM.

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

6.1 Pre-processing effect

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

The first step is to use Algorithm 1 to reduce the interval in which the tasks can be placed with respect to dependencies to other tasks. For Instance I the effect is that about 5% of the total amount of interval is removed. This result

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

Table 8: Effect of pre-processing components. The number of y-variables are counted for the benchmark model, see Section 5.4, with an earlier version of the scheduling tool and the other results are from scheduling tool running the constraint generation procedure.

	Instances		
Measurements	I	II	III
Complete number of y-variables	$22 \cdot 10^6$	$52 \cdot 10^6$	$70 \cdot 10^6$
Reduction of task intervals by Alg. 1	5%	18%	17%
Time for Alg. 1	27s	103s	123s
Number of y-variables after Alg. 1 and 2	$0.2 \cdot 10^6$	$0.9 \cdot 10^6$	$1.2 \cdot 10^6$
Reduction of v-variables by Alg. 3	47%	90%	92%

indicate the tasks are not particularly restricted by their release times, deadlines and dependencies to other tasks for this instance. For Instances II and III about 18% of the total amount of interval is removed and the pre-processing has an effect of practical relevance. For all instances, Algorithm 1 requires at most a few minutes of computational time.

The last row of Table 8 shows for how many of the v-variables that Algorithm 3 detects that the value is fixed. In Instance I 47% of the v-variables are fixed and in Instance 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 enable us to schedule the instances under consideration, is the reformulation of sequencing and the constraint generation procedure using the α- and the β-model. 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 execute in that

Table 9: Characterisation of β-model.

	Instances		
Measurements	I	II	III
Number of sections	$2 \cdot 10^3$	$4 \cdot 10^3$	$6 \cdot 10^3$
Number of α-variables	$3 \cdot 10^4$	$11 \cdot 10^4$	$15 \cdot 10^4$
Average number of β-variables in β-model	$2 \cdot 10^3$	$6 \cdot 10^3$	$8 \cdot 10^3$
Average number of y-variables in β-model	$2 \cdot 10^4$	$5 \cdot 10^4$	$7 \cdot 10^4$

section with respect to its release time and deadline.

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

The outcome of the scheduling of Instance I, II, and III is summarised in Table 10, 11, and 12, respectively. For each instance, the results for four choices of objective functions in the first α-model step are presented. The solution times are given in seconds. The total time refers to the complete execution time for our scheduling tool, while times for the α- and the β-model refer to time spent by Gurobi only. The most important parameter settings used are:

- The time limit in the α-model is 8 h.
- The relative MIP-gap in the α-model is 0.10 for all runs.
- The time limit in the β-model is initially 2 hours and whenever an improved integer solution is found, it is reset to 4 hours.
- The relative MIP-gap in the β-model is 0. This strict gap is required to facilitate all tasks to be successfully sequenced.
- The value of M_Δ is 1000.
- In one iteration, at most 5 new generated sequences are added to the α- and the β-model, even if more than that is detected.

The choices that have to be made with most care are the objective function in the α-model, the MIP-gap in the α-model, and the time limit in the β-model. The objective function and the MIP-gap in the α-model are important because these choices have an impact on in which sections the tasks are initially placed. The time limit in the β-model 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 set too high, a lot of time will be spent in the β-model trying to include tasks into the β-sequences even if this is not possible, and if this time limit is set too low there is a risk that a solution with all tasks included in a β-sequence might not be found even if it exists.

For Instance I, a schedule is obtained within 10 minutes no matter which objective is used, and in all cases the section assignment made in the first α-model step makes it possible to include all tasks in an β-sequence in the β-model

Table 10: Results for Instance I

Measurements	Section-slack	Center-task $\Delta = 0.10$	Center-task $\Delta = 0.50$	Center-task $\Delta = 0.75$
Total time (s)	164	243	467	182
Iterations	1	1	1	1
Generated sequences	-	-	-	-
Time (s) α-model	6	31	23	23
Time (s) β-model	1	53	283	2

Table 11: Results for Instance II

Measurements	Section-slack	Center-task $\Delta = 0.10$	Center-task $\Delta = 0.50$	Center-task $\Delta = 0.75$
Total time (s)	1025	29676	2484	1623
Iterations	1	1	1	1
Generated sequences	-	-	-	-
Time (s) α-model	111	28800	946	483
Time (s) β-model	449	416	1081	680

step. This exceptionally good outcome is likely to depend on that a lot of feasible solutions exists for this instance and that the task intervals and the bounds on the dependency lengths are not very tight. This result can be compared to the benchmark formulation which we ran one week without obtaining a solution.

For Instance II the solution times ranges between 17 minutes and 41 minutes, except when for the center-task-objective with $\Delta = 0.10$ was used. In that case the running time was 495 minutes. The reason for this long running time comes from the α-model where no solution with the required MIP-gap is obtained within the maximum running time.

Instance III requires more than one iteration for all choices of objective function except for the center-task objective with $\Delta = 0.75$. Also, as for Instance II, the center-task objective for $\Delta = 0.1$ does find a solution that meets the required MIP-gap of the α-model in the first iteration which gives a significantly longer running time, 871 minutes, also for this instance. The total running time for the other objectives ranges in between 37 and 131 minutes.

A conclusion that can be drawn about the choice of objective function in the first α-model step is that the section-slack objective and the center-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.

Table 12: Results for Instance III

Measurements	Section-slack	Center-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 (s) α-model	[178, 33, 36, 82]	[28800, 41, 195]	[503, 34]	569
Time (s) β-model	[30, 38, 33, 356]	[49, 15609, 6294]	[42, 6369]	1079

7 Concluding remarks

This paper describes an avionics scheduling problem of industrial relevance and suggests a mathematical model for this problem. The non-heuristic 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, significantly larger than what is described in the literature, within reasonable time and our conclusion is that our approach is viable for this type of problem. Our continued research aims at improving the components used in this strategy to further enhance the computational performance in order to solve even larger instances.

In the current model we have introduced a restriction that, for each CM, at most one message can be sent or received in a slot. In practice there is no such restriction and co-allocation of messages is possible. However, if co-allocation is allowed, this effects the tasks used for transmitting and receiving the messages and therefore we have left for future research to include the possibility of co-allocation.

The work of this paper is part of a long term project of developing exact solution methods for avionics scheduling problems since they are expected to be important for future avionic development projects. In this paper we consider the case of scheduling a systems where all requirements are given, but it is also of interest to develop decisions 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 number 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.

References

Airlines electronic engineering committee (AEEC). Avionics application software standard interface, ARINC specification 653, (part 1). 2006.

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

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

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

V. V. Balashov, V. A. Balakhanov, and V. A. Kostenko. Scheduling of computational tasks in switched network-based IMA systems. In *Proceedings of the* 1^{st} *international conference on engineering and applied sciences optimization*, 2014.

S. Beji, S. Hamadou, A. Gherbi, and J. Mullins. 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*, pages 165–174, 2014.

S. S. Craciunas and R. S. Oliver. 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*, pages 45–54, 2014.

R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, 43(4):1–44, 2011.

A. Easwaran, I. Lee, O. Sokolsky, and S. Vestal. 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*, pages 371–380, 2009.

F. Eisenbrand, K. Kesavan, R. S. Mattikalli, M. Niemeier, A. W. Nordsieck, M. Skutella, J. Verschae, and A. Wiese. Solving an avionics real-time scheduling problem by advanced IP-methods. In *Algorithms – ESA 2010*, volume 6346 of *Lecture Notes in Computer Science*, pages 11–22. Springer Berlin Heidelberg, 2010.

H. Kopetz. *Real-time systems: Design principles for distributed embedded applications.* Real-Time Systems Series. Springer Science & Business Media, 2011.

Y.-H. Lee, D. Kim, M. Younis, and J. Zhou. Scheduling tool and algorithm for integrated modular avionics systems. In *Proceedings of the Digital Avionics Systems Conference*, pages 1–8, 2000.

J. Y. T. Leung, editor. *Handbook of scheduling: algorithms, models, and performance analysis.* Chapman & Hall/CRC Computer and Information Science Series. Taylor & Francis, 2004. ISBN 9781135438852.

A. S. Manne. On the job-shop scheduling problem. *Operations Research*, 8(2): 219–223, 1960.

C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.

P. Pop, P. Eles, and Z. Peng. Scheduling with optimized communication for time-triggered embedded systems. In *Proceedings of the Seventh International Workshop on Hardware/Software Codesign*, pages 178–182, 1999.

Radio Technical Commission for Aeronautics (RTCA). Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations, RTCA DO-297. 2005.

J. Rufino, J. Craveiro, and P. Verissimo. Architecting robustness and timeliness in a new generation of aerospace systems. In *Architecting Dependable Systems VII*, volume 6420 of *Lecture Notes in Computer Science*, pages 146–170. Springer Berlin Heidelberg, 2010.

E. Tavares, P. Maciel, E. Sousa, B. Nogueira, L. Amorim, and V. Lira. 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*, pages 1207–1212, 2012.

J. Theis, G. Fohler, and S. Baruah. Schedule table generation for time-triggered mixed criticality systems. In *Proceedings of the Workshop on Mixed Criticality Systems at IEEE Real-Time Systems Symposium*, pages 79–84, 2013.

D. Tămaş-Selicean, P. Pop, and W. Steiner. 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*, pages 473–482, 2012.

J. Xu and D. L. Parnas. Priority scheduling versus pre-run-time scheduling. *Real-Time Systems*, 18(1):7–23, 2000.

L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty. 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*, pages 119–124, 2014.