Logic-based Benders decomposition with a partial assignment acceleration technique for avionics scheduling

Emil Karlsson[a,b], Elina Rönnberg[a]

[a] *Department of Mathematics, Linköping University, SE-581 83 Linköping, Sweden*
[b] *Saab AB, SE-581 88 Linköping, Sweden*

Abstract

Pre-runtime scheduling of large-scale electronic systems, as those in modern aircraft, can be computationally challenging. In this paper, we study a distributed integrated modular avionic system of practical relevance where the scheduling includes to assign communication messages to time slots and to sequence tasks on modules. For this problem, the challenge is the huge number of tasks to be scheduled and the intricate interaction between the communication and task scheduling. We present a method based on Logic-Based Benders Decomposition (LBBD) to solve this problem. In a master problem, formulated as a mixed-integer program, communication messages are assigned to time slots and in a subproblem, formulated as a constraint program, tasks are scheduled.

Our LBBD scheme is accelerated by using cut strengthening, a subproblem relaxation, and components for preprocessing and initialisation of the scheme. The cut strengthening is of the kind that systematically investigates subsets of variables to include in the cut by solving additional subproblems. To further enhance the efficiency of the scheme, we propose a new strategy that extends cut strengthening to also try to construct feasible solutions by using information from solving the additional subproblems. Our computational evaluation is based on publicly available instances with up to 2530 messages and 54,731 tasks. It shows that our method outperforms previous methods for the problem with respect to both solution times and RAM usage. A further evaluation of the different components in the method shows that our new acceleration strategy is important to achieve these results.

Keywords— Logic-based Benders decomposition, acceleration technique, cut strengthening, mixed-integer programming, constraint programming, avionics scheduling

1. Introduction

A crucial aspect to consider when designing an aircraft is that the avionic system, the electronics in the aircraft, can never be allowed to fail in a way that has consequences for the operation of the aircraft. Over the years, the aircraft industry has moved in the direction of defining an increased amount of functionality through software implementations, and this is a development that is expected to continue. Modern systems are designed to be iteratively developed, flexible, modular and ready for functionality that is not yet known. This also means that the same hardware can be used for many different functions and that the appropriate hardware resources need to be allocated to the specific software functions to ensure safe operation of the aircraft.

The design principle Integrated Modular Avionics (IMA) is one contributor to meet the expectations on future safety-critical electronic systems (Gaska et al., 2015; Wang and Niu, 2018). This has been evolved through large research initiatives such as Scarlett that included industrial partners such as Airbus and Saab. In this architecture, there is one layer that hosts the application modules that run the actual software. This layer is physically separated from the layer that hosts communication modules and these merely act as an infrastructure that supports the communication between the application modules. To pass data between the communication modules, a Time-Triggered

Email address: elina.ronnberg@liu.se (Elina Rönnberg)

Ethernet (TTEthernet) can be used (Gaska et al., 2015; Wang and Niu, 2018). TTEthernet can handle both time-triggered communication and rate-constrained communication (e.g. AFDX, used by both Airbus and Boeing) as well as best-effort traffic (Robati et al., 2017).

In more general terms, a system designed according to this principle can be referred to as a distributed modular electronic system that communicates via TTEthernet. An important characteristic of such design is the need to schedule when different functions are allowed to use the hardware resources. Because the system is complex and safety-critical, computationally cheap on-line scheduling algorithms, such as those used in e.g. PCs, are not a viable option (Xu and Parnas, 2000). The scheduling must instead be made off-line (at compile time) and in a way that guarantees the availability of required hardware resources. A recent example of off-line scheduling of an IMA system with a TTEthernet is found in Zhou et al. (2020) and earlier references on scheduling of IMA systems are reviewed in Blikstad et al. (2018).

For large-scale systems, as those in a modern aircraft, the scheduling can pose a great computational challenge. This is both described in the engineering literature (Gaska et al., 2015; Wang and Niu, 2018) and confirmed in our interdisciplinary research collaboration between Linköping University and Saab. Within this collaboration, we combine engineering knowledge with optimisation research in the study of an advanced IMA system with a specific time-triggered Ethernet communication. In previous work, we have proposed scheduling methods, see Blikstad et al. (2018) and Karlsson et al. (2021), for this problem.

This paper will present a new and more efficient method for solving this scheduling problem. Since the technical background has been detailed in previous work (Blikstad et al., 2018; Karlsson et al., 2021), we try to omit as many engineering details as possible in the problem statement. Instead, we will focus on the structure of the problem, which will mainly be described using optimisation nomenclature. The motivation for developing more efficient solution methods for the problem is that, in practice, the scheduling needs to be done each time an engineer makes a software change during the development of the system. This is typically done on a daily basis during several years. The scheduling determines if there is room for the new function or if changes are needed, and hence it is important for the engineers to get the feedback from the scheduling as quickly as possible.

Briefly, the problem can be described as a rich multiprocessor scheduling problem that also includes a communication network. The communication is scheduled by assigning messages to time slots and once these decisions are made, the remaining problem gets a relatively pure and simple task-scheduling structure. This property is exploited in the method design by applying Logic-Based Benders Decomposition (LBBD) where communication messages are assigned to time slots in an assignment-type master problem and the subproblem is used to sequence the tasks. The master problem is formulated as a Mixed-Integer Programming (MIP) model while the subproblem is formulated as a Constraint Programming (CP) model.

LBBD, introduced in Hooker (2000) and Hooker and Ottosson (2003), is a decomposition approach that has been successfully applied to large-scale discrete optimisation problems (Hooker, 2019). In particular, excellent computational performance has been reported when it has been used in the design of exact methods that hybridise MIP and CP techniques to solve resource allocation and scheduling problems (Hooker, 2007; Lombardi and Milano, 2012; Coban and Hooker, 2013; Sun et al., 2019; Emde et al., 2020). In general, the efficiency of LBBD methods rely on acceleration techniques tailored to the problem structure (Rahmaniani et al., 2017; Hooker, 2019). The most common ones described in Hooker (2019) are to strengthen the master problem by a subproblem relaxation, to design improved cuts (e.g. analytical Benders cuts), and to perform cut strengthening. In addition, heuristic strategies can be used to further improve the computational performance.

Preliminary results from applying LBBD to our scheduling problem, together with known acceleration techniques (subproblem relaxation and cut strengthening), showed great promise and confirmed that the decomposition takes advantage of the problem structure. However, it was not sufficient to solve our largest instances. To overcome this, we have developed a new acceleration technique that extends cut strengthening with a component that tries to construct feasible solutions based on information from the cut strengthening.

Our extension can briefly be described as follows. In general, a LBBD cut specifies a set of master problem variables that cannot take their current values in a solution to the problem. Cut strengthening means that this set is reduced to obtain a smaller and hence stronger cut. One way to find a stronger cut is to perform a search that systematically evaluates different subsets of variables. In our extended strategy, we include an additional step in such search; this step is applied each time a subset of variables has been evaluated and what is done depends on the outcome of this evaluation. Either, if the current subset is confirmed to yield a cut, we form a restriction of the original problem with respect to the variables not included in the current subset. If this restricted problem is feasible, its solution is also a feasible solution to the original problem. Or, if the current subset does not yield a cut, a restriction is instead constructed with respect to the variables included in this subset. Again, if a feasible solution to this restricted problem is found, we have found a feasible solution to the original problem.

Note that in any search for stronger cuts by systematically investigating subsets of variables, the subproblem needs to be relaxed with respect to the variables that are currently not included in the subset. In most descriptions of cut-strengthening algorithms, this aspect is not much highlighted. We believe that this is mainly due to the fact that the formulations of these relaxations often follow naturally from the problem formulation, such as in Cambazard et al. (2004); Hooker (2007); Coban and Hooker (2013); Lam et al. (2020); Karlsson and Rönnberg (2021). However, in order to make our extension, a more rigorous framework is needed. For this purpose, we introduce a partial assignment subproblem. Based on this, both standard cut strengthening and our extension to also search for feasible solutions can be described. A key component in this derivation is to make well-chosen relaxations and restrictions of this partial assignment subproblem. The proposed extension can be integrated with different cut-strengthening algorithms. We have chosen to use the one based on Depth-First Binary Search (DFBS) since it yields irreducible feasibility cuts and since it has been shown to perform well for strengthening of feasibility cuts for several types of problems (Karlsson and Rönnberg, 2021).

The acceleration technique that we introduce in this paper is developed to efficiently solve our avionics scheduling problem. However, to make the generic aspects of this work easily accessible for continued research, the paper is structured with this in mind. Especially, we believe that our generic way to present how cut strengthening can be derived from a partial assignment subproblem, and also how it can be generalised into the acceleration technique that we propose, is of value for continued work. To the best of our knowledge, we are the first to integrate cut strengthening in LBBD and search for feasible solutions in such systematic way.

To evaluate our LBBD method, we use the set of publicly available avionics scheduling instances[1] introduced in Karlsson et al. (2021). This set includes 120 instances with up to 2530 messages and 54,731 tasks to schedule. First our new method is compared to two problem-tailored methods from previous work, an exact constraint generation procedure introduced in Blikstad et al. (2018) and extended in Karlsson et al. (2021), and a matheuristic method introduced in Karlsson et al. (2021). Our results show that the new method outperforms the previous ones both with respect to solution time and RAM usage. Thereafter, we perform additional experiments to illustrate the impact that the acceleration strategies and other components of our method have on the computational performance. This evaluation confirms that our extension of cut strengthening to also search for feasible solutions has a significant impact on the computational performance.

This paper is structured as follows. Section 2 gives a literature review that includes acceleration strategies for LBBD based on cut strengthening and heuristically finding feasible solutions, respectively, and LBBD schemes for scheduling of hard real-time systems. Section 3 first describes – in a general context – how a search based on investigating subsets of variables to strengthen a cut can be extended into a partial assignment acceleration technique. Then, the specific integration with DFBS is described and we highlight how DFBS can be seen a special case of our acceleration technique. Section 4 describes how to apply LBBD, including known acceleration techniques, and our partial assignment acceleration technique to our avionics scheduling problem. The computational evaluations of our method are presented in Section 5. Section 6 gives concluding remarks and discusses future research directions.

2. Literature review

Since our acceleration technique integrates cut strengthening and the search for feasible solutions, we describe related works in these respective areas. Our new technique is applied to scheduling of an avionic system, which is a hard real-time system. To schedule real-time systems often involves making some allocation decision integrated with scheduling of tasks, which is a structure that lends itself well for LBBD. Therefore we also present related works on applying LBBD to schedule hard real-time systems.

2.1. Logic-based Benders decomposition and cut strengthening

It is well established that acceleration techniques are necessary to obtain good computational performance for classical Benders schemes (Rahmaniani et al., 2017). One of the techniques highlighted in Rahmaniani et al. (2017) is cut strengthening, which also has been successfully used to improve the computational performance of LBBD schemes (Hooker, 2019; Karlsson and Rönnberg, 2021). A common way to strengthen cuts in LBBD is to evaluate subsets of the set of variables that are included in an original cut. Each evaluation is made by solving a subproblem, and by systematically solving such subproblems, a strengthened cut can be found. Cut-strengthening algorithms that rely on solving additional subproblems have been used to improve the computational performance of LBBD schemes, and

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

problem-specific examples can be found in Hooker (2007), Benini et al. (2008), Coban and Hooker (2013), Riedler and Raidl (2020), Lam et al. (2020), and Lindh et al. (2022).

The standard subproblem in a LBBD scheme is formulated based on a complete master problem solution. A feasibility cut is found if this subproblem is infeasible and then this original cut will include all master problem variables. To evaluate a subset of the variables included in an original cut, a subproblem must be formulated with respect to this subset of master problem decisions only. To be able to conclude that a subset of the decisions will yield a feasibility cut, the remaining decisions must be handled in an exact way or in a way that gives a relaxation of the subproblem. In the literature that we have reviewed, this aspect is not much highlighted and instead the motivations for how to formulate the subproblems and how to handle evaluations of subsets of decisions have been problem specific. As part of our description in Section 3, we provide a more formal and problem-agnostic framework for how to formulate such relaxations.

Most common in previous work is that the problem formulations and the decompositions are such that each master problem decision corresponds to adding a demand for resources that need to be considered in a subproblem – and by omitting such assignment, a relaxation is directly obtained. In Hooker (2007), the master problem assigns tasks to facilities, and in the subproblem for each facility, the assigned tasks must be scheduled. In Coban and Hooker (2013), the master problem assigns tasks to segments of time, and in the subproblem for each segment of time, the tasks must be scheduled. In Riedler and Raidl (2020), the master problem assigns requests to vehicles, and in the subproblem for vehicle, a tour is constructed. Similarly, in Lindh et al. (2022), the master problem assigns tasks to work shifts and machines to the tasks, and in the subproblem, tasks are scheduled within the work shifts while respecting the machine assignments. For these applications, removing an assignment corresponds to not schedule a task or serve a request, hence creating a relaxation. A similar structure is also observed for the problems studied in Lam et al. (2020), where the hybrid branch-and-check-type LBBD solver Nutmeg (Lam et al., 2020) is applied to facility allocation and scheduling, plant location, vehicle routing with location congestion, and satellite scheduling.

In Benini et al. (2008), there is a slightly different approach to formulating the relaxation of the subproblem. Their LBBD scheme is of multi-stage type with a master problem that assigns tasks to cores, and then there are two subproblems. In the first subproblem, memory to be used for communication between tasks are assigned to communication channels given a task-to-core assignment from a master problem solution. In the second subproblem, they search for a task schedule that is feasible given the memory-to-communication channel assignment. Both subproblems can supply feasibility cuts if infeasible and the relaxations used when strengthening these cuts are not formulated by simply removing the tasks. The reason for this is that both subproblems contain tasks whose length is determined by the task-to-core and the memory-to-communication channel assignments. In the first subproblem, a relaxation is formed by giving tasks their minimum length if there is no task-to-core assignment that determines their length. To obtain a relaxation in the second subproblem, the memory requirements are ignored if there is no memory-to-communication channel. Both the type of relaxation used in Benini et al. (2008) and the relaxations described for Hooker (2007), Coban and Hooker (2013), Riedler and Raidl (2020), Lam et al. (2020), and Lindh et al. (2022) above fit well within the framework to be introduced in Section 3.

When comparing cut-strengthening algorithms that involve solving additional subproblems, there is a distinction between algorithms that generate cuts that are irreducible (locally minimal) with respect to the subproblem that is solved, and those that do not. A unified description of the three algorithms greedy, deletion filter, and DFBS is presented in Karlsson and Rönnberg (2021) along with an evaluation of their impact on the computational efficiency when used for strengthening of feasibility cuts. The algorithms deletion filter and DFBS give irreducible cuts, while greedy is an algorithm without guarantees. In general, to guarantee that the obtained cut is irreducible usually comes at the cost of solving additional subproblems. In the comparison made in Karlsson and Rönnberg (2021), the deletion filter and DFBS gave the best computational performance in the evaluation made with over 2000 instances from three different problem formulations. This result indicates that it is worth spending time on solving additional subproblems to find irreducible cuts as it appears to contribute to the general progress of the LBBD scheme.

There is a strong connection between the strategies used to systematically search for the variables to include in a strengthened cut and algorithms used for finding an Irreducible Infeasible Subset of constraints (IIS) for an optimisation problem. For example, the deletion filter algorithm was first introduced to find an IIS for linear programs (Chinneck and Dravnieks, 1991), and DFBS has been applied to find an IIS both in the context of CPs (Junker, 2001, 2004) and in the context of general mathematical programs (Atlihan and Schrage, 2008).

Examples of algorithms where additional subproblems are solved to evaluate subsets of variables to find stronger cuts are the following. In Hooker (2007) and Coban and Hooker (2013), a greedy strategy is used to determine which subsets of variables to evaluate and this strategy includes sorting of the variables to exploit problem structure. The cut-strengthening algorithm in Lam et al. (2020) is an application of the deletion filter for finding an IIS (Parker and Ryan, 1996). The cut-strengthening algorithms in Riedler and Raidl (2020) rely on finding several IISs for each

infeasible subproblem and they present different strategies for using this information. Their strategy for finding an ISS includes a deletion filter search. In Benini et al. (2008), the authors present a greedy algorithm but they also briefly mention and implement an extended cut-strengthening algorithm that finds irreducible cuts. This extension includes embedding their greedy algorithm within an algorithm for finding an IIS for a CP model, as described in Junker (2001). The computational results indicated that the cut-strengthening algorithm that finds irreducible feasibility cuts performs better than their greedy algorithm.

There are also strategies for strengthening of LBBD cuts that are not based on systematically solving additional subproblems. In Cambazard et al. (2004), a conflict refiner called QuickXplain (Junker, 2001, 2004) is used to produce a minimal conflict set from an infeasible CP subproblem. The minimal conflict set is then used to construct a possibly strengthened feasibility cut. In Kim and Hooker (2002), the authors briefly mention the possibility to find a minimal capacity cut to improve the convergence of the LBBD scheme for a fixed-charge network flow problem. Another strategy is to theoretically analyse a cut and extend it into a so-called analytical cut that is viable for a larger set of master problem solutions (Hooker, 2019). Examples of how to formulate such cuts are found in Hooker (2007) and Coban and Hooker (2013). In the context of combinatorial Benders decomposition, introduced in Codato and Fischetti (2006), they construct a feasibility cut from an IIS that is found using an algorithm for LPs introduced in Parker and Ryan (1996).

2.2. Heuristic search for feasible solutions within a logic-based Benders scheme

The use of heuristics to accelerate classical Benders schemes is a well-established concept (Rahmaniani et al., 2017; Raidl, 2015). It does, however, not appear to be an extensive literature on how to use heuristics to accelerate LBBD schemes, but we provide here some examples of how heuristics have been used for finding feasible solutions. In Raidl et al. (2021), a LBBD scheme for a bi-level capacitated vehicle routing problem is introduced. There, both the master problem and the subproblem are solved by variable neighbourhood search. Excellent computational results were reported for this heuristic approach, but a drawback was that the heuristically obtained cuts can remove solutions that should be feasible. To overcome this, a verification and correction procedure was introduced in Raidl et al. (2020), and this was used to find and correct cuts that are invalid. In Riedler and Raidl (2020), LBBD is applied to a dial-a-ride problem and, inspired by the results in Raidl et al. (2021), they made an attempt to solve the subproblem heuristically. However, in this case, the validation procedure required to ensure that an optimal solution is found cancelled out the gains from applying a heuristic. In Bajestani and Beck (2011) and Emeretlis et al. (2016), a LBBD scheme is introduced for an aircraft repair shop scheduling problem and a multi-core task-allocation-and-scheduling problem, respectively. In both papers, a heuristic is applied to find a feasible solution to the original problem and this is used to put a bound on the objective value of the master problem.

This review does not include classical Benders schemes, but there is one approach that we would like to highlight since it inspired how we propose to extend cut-strengthening procedures with a component that tries to construct feasible solutions. In Saharidis and Ierapetritou (2010), they introduce an approach where, if the subproblem was proven to be infeasible, an auxiliary problem was solved to find a maximal feasible subset of constraints (somewhat complementary to an IIS) from the infeasible subproblem constraints. From this maximal feasible subset of constraints, a bound on the optimal objective value of the original problem is derived and used to form a cut. Our strategy is similar since we use information from the cut strengthening to find a part of the master problem solution that might be part of a feasible solution, and then we try to construct a feasible solution from that.

2.3. Logic-based Benders decomposition and hard real-time system scheduling

Below we give a survey of LBBD schemes that have been developed to schedule hard real-time systems. In He et al. (2008), LBBD is applied to a problem that includes both the assignment and scheduling of tasks in a distributed system where the processors communicate by a Time-Triggered Protocol (TTP). In the master problem, tasks are assigned to processors, and in the subproblem there is a search for a feasible schedule for the tasks and messages together with a valid bus access configuration. The master problem is solved using a Satisfiability Modulo Theories (SMT) solver while a CP solver is applied to the subproblem. This method solves instances with up to 10 processors and 50 tasks.

A TTP is also used for communication between the processors in the distributed embedded system addressed in Zhang et al. (2018). There, they introduce a LBBD scheme for a problem that contains both the assignment and scheduling of tasks. In the master problem, tasks are assigned to processors, while the subproblem is used to search for a valid bus access configuration. The master problem is solved using an SMT solver while the subproblem is solved via a geometric programming approach. Their LBBD method solves instances with up to 8 processors and 30 tasks. In the hard real-time system studied in Cambazard et al. (2004), the processors communicate via a token-ring.

Their scheduling problem contains both the assignment and scheduling of tasks and they use a LBBD method that includes the cut-strengthening algorithm described in Section 2.1. Their master problem assigns tasks to processors and a feasible task schedule is searched for in the subproblem. Both the master problem and the subproblem are solved using a CP solver and their method is used to solve instances with up to 7 processors and 40 tasks.

A heterogeneous multi-core platform is studied in Emeretlis et al. (2016) where a LBBD scheme is developed for an assignment and scheduling problem. In their master problem, tasks are assigned to cores and the subproblem is used to find a feasible task schedule. The master problem is solved using a MIP solver while a CP solver is applied to the subproblem. Their LBBD method solves instances with up to 4 heterogenous cores and 129 tasks. A different multi-core architecture, the Cell BE processor, is addressed in Benini et al. (2008). As described in Section 2.1, a multi-stage LBBD scheme is introduced for this problem where the master problem is solved by a MIP solver and both subproblems are solved by a CP solver. Their LBBD method solves instances with up to 6 cores and 29 tasks.

There are some differences between the avionics problem that we address and the problem formulations that are reviewed above. A structural difference is that we do not assign tasks to processors as this is determined by the engineers that design the system. Instead, the assignment aspect of our problem is to decide in which slots the messages are to be sent. Also, our problem is a pure feasibility problem while the others, except Cambazard et al. (2004), have an objective. The largest instance we have found in previous work included 10 processors (or cores) and 129 tasks. Our instances are magnitudes larger with 21 modules (processors), 2530 messages, 3968 communication slots, and 54,731 tasks.

3. A partial assignment acceleration technique

The acceleration technique to be described extends the DFBS algorithm for strengthening of feasibility cuts in LBBD (e.g. Karlsson and Rönnberg (2021)) to include a search for feasible solutions. The key components in this extension do however not depend on the integration with DFBS, and therefore we describe the generic aspects independently to make it easier to include the extension also in other search algorithms.

A problem formulation and a basic LBBD scheme is provided in Section 3.1, and this description builds on the notation used in Karlsson and Rönnberg (2021). The derivation of our acceleration technique is based on the formulation of a partial assignment subproblem, to be introduced in Section 3.2. This problem is however merely of theoretical interest since it is too computationally expensive to be used as a subproblem in an acceleration technique. Instead, the method relies on solving computationally cheaper relaxations and restrictions of this subproblem and Section 3.3 introduces a framework for how to formulate these. The integration with DFBS, detailed in Section 3.4, provides a systematic way to solve these relaxations and restrictions to efficiently generate strong cuts and possibly also feasible solutions to the problem.

3.1. Logic-based Benders decomposition

Given a problem on the form

$$
\begin{aligned}
[\mathrm{P}] \quad \min \ & f(x), \\
\text{s.t.} \ & C(x), \\
& C(y), \\
& C(x,y), \\
& x_i \in \{0,1\}, \quad i \in I, \\
& y \in D_y,
\end{aligned}
$$

the decomposition is such that the binary variables $x = (x_i)_{i \in I}$ are the master problem variables and the variables y, that belong to the domain D_y, are the subproblem variables. In line with having a LBBD structure, there are constraints $C(x)$ that include only on the master problem variables as well as constraints $C(y)$ that include only the subproblem variables. Furthermore, there are constraints $C(x, y)$ that include both x and y and thereby connect the master problem decisions with the subproblem decisions. The objective function, $f(x)$, depends only on the master problem variables so that the subproblem becomes a feasibility problem. The notation $v(\cdot)$ will be used to refer to the optimal objective value of an optimisation problem, e.g. the optimal value of problem P is denoted by $v\,(\mathrm{P})$.

Below, we describe a general LBBD scheme for problem P. In iteration k of this scheme, the master problem is

$$\begin{aligned}[\text{MP}^k] \quad \min\ & f(x), \\ \text{s.t.}\ & C(x), \\ & B^k(x), \\ & \text{[Subproblem relaxation]}, \\ & x_i \in \{0,1\}, \quad i \in I,\end{aligned}$$

where $B^k(x)$ denotes the set of already generated feasibility cuts. The first iteration is $k = 1$ with $B^1(x) = \emptyset$. It is common to strengthen the master problem by a subproblem relaxation component, and this can include both additional variables and constraints. In our models, this is indicated by the notation [Subproblem relaxation].

Let an optimal solution to master problem MP^k be denoted by $\bar{x}^k$. In iteration k, the subproblem is formed by making the assignment $x = \bar{x}^k$ in problem P, which yields

$$\begin{aligned}[\text{SP}^k] \quad \min\ & 0, \\ \text{s.t.}\ & C(y), \\ & C(\bar{x}^k, y), \\ & y \in D_y.\end{aligned}$$

If subproblem SP^k has a feasible solution $\bar{y}^k$, then the solution $(\bar{x}^k, \bar{y}^k)$ is optimal in problem P. If subproblem SP^k is infeasible, then the feasibility cut

$$[b^k(I)] \qquad \sum_{i \in I : \bar{x}_i^k = 0} x_i + \sum_{i \in I : \bar{x}_i^k = 1} (1 - x_i) \geq 1, \tag{1}$$

cuts off the current master problem solution. In a LBBD scheme, this cut can be added to the set of feasibility cuts $B^k(x)$ to form the next set of feasibility cuts $B^{k+1}(x)$. The iterations continue until a feasible solution to a subproblem is found or the master problem is proved infeasible. Adding these types of cuts is sufficient for convergence of the scheme, but might be computationally inefficient.

Above, the definition of a cut $b^k(I)$ has one term for $i \in I : \bar{x}_i^k = 0$ and one term for $i \in I : \bar{x}_i^k = 1$. It is however common that the constraints $C(x, y)$ are such that only the decisions $\bar{x}_i^k = 1$ imply a restriction on the y-variables (e.g. scheduling problems where the decision indicates if a job is assigned to a machine or not). In such cases it is sufficient to only consider a cut $b^k(\bar{I}^k)$, where $\bar{I}^k = \{i \in I : \bar{x}_i^k = 1\}$, and, in the remainder of the derivation presented below, replace each occurrence of I with $\bar{I}^k$. In Section 4, $b^k(\bar{I}^k)$ will be denoted by $b(\bar{I}^k)$ for short.

If it is known that a subproblem will be infeasible by making only a partial assignment $\bar{x}_i^k$, $i \in L \subset I$, a cut $b^k(I)$ can be replaced by a stronger feasibility cut $b^k(L)$ in iteration k and the LBBD scheme will still converge. This is the fundamental property that allows for acceleration techniques based on cut strengthening.

3.2. Irreducible feasibility cuts and the partial assignment subproblem

This section introduces the partial assignment subproblem that is a key component in the derivation of the acceleration technique. Since the strengths of the generated cuts are of importance, we also introduce the notion of a cut being irreducible with respect to such partial assignment subproblem.

The partial assignment subproblem is obtained by, in iteration k of the LBBD scheme, making the partial assignment $x_i = \bar{x}_i^k$, $i \in L$, in problem P. It is formulated as

$$\begin{aligned}[\text{PAS}^k(L)] \quad \min\ & f(x), \\ \text{s.t.}\ & x_i = \bar{x}_i^k, \quad i \in L, \\ & C(x), \\ & C(y), \\ & C(x, y), \\ & x_i \in \{0,1\}, \quad i \in I, \\ & y \in D_y,\end{aligned}$$

and if this problem is infeasible, the notation $v\left(\text{PAS}^k(L)\right) = \infty$ is used for its objective value. In iteration k, the partial assignment subproblem $\text{PAS}^k(I)$ coincides with subproblem SP^k, and $\text{PAS}^k(\emptyset)$ coincides with problem P. For other partial assignments, the difference between SP^k and $\text{PAS}^k(L)$ is that the latter includes the master problem variables x_i, $i \in I \setminus L$. This makes the partial assignment subproblem a hybrid between the master problem and subproblem, and as such it is not attractive from a computational point of view. However, the purpose of introducing $\text{PAS}^k(L)$ is not to solve it directly and instead we will derive computationally tractable relaxations and restrictions from it. Before that, we establish some properties of the partial assignment subproblem.

The reason for introducing a hybrid between the master problem and the subproblem is that it can be solved with two different purposes in mind, either

(i) to determine if the partial assignment $x_i = x_i^k, i \in L$, is infeasible, or

(ii) to complete the partial assignment $x_i = x_i^k, i \in L$, into a feasible solution for problem P.

If infeasibility is established in case (i), the feasibility cut $b^k(L)$ can be added to the master problem. In case (ii), any feasible solution to $\text{PAS}^k(L)$ will be a feasible solution to problem P. Further, in the case when x^k is feasible also in SP^k, an optimal solution to $\text{PAS}^k(L)$ is optimal in problem P.

The following result is of use when comparing two partial assignments.

Proposition 1. *Let* $\text{PAS}^k(L)$, $L \subseteq I$, *be a partial assignment subproblem in iteration* k *of an LBBD scheme. For* $L_1 \subseteq L_2 \subseteq I$ *it holds that*

$$v\left(\text{PAS}^k(L_1)\right) \leq v\left(\text{PAS}^k(L_2)\right). \tag{2}$$

Proof. The result follows immediately since the same objective function is used in $\text{PAS}^k(L_1)$ and $\text{PAS}^k(L_2)$ and since each feasible solution to $\text{PAS}^k(L_1)$ is feasible also in $\text{PAS}^k(L_2)$. □

From this relation follows two domination criteria. First, the knowledge that a partial assignment subproblem is infeasible for a subset $L_1 \subseteq I$ can be used to deduce that the partial assignment subproblems for all subsets $L_2 \supset L_1$, where $L_2 \subseteq I$, are infeasible. Second, if a partial assignment for a subset $L_2 \subseteq I$ has been completed into a feasible solution to P, it is known that the partial assignment subproblem is feasible for all subsets $L_1 \subset L_2$.

To be able to refer to a cut being the strongest possible, in a local sense, with respect to this partial assignment subproblem, we make the following definition.

Definition 1. *Let* $\text{PAS}^k(L)$, $L \subseteq I$, *be a partial assignment subproblem in iteration* k *of a LBBD scheme. A feasibility cut* $b^k(L_1)$, $L_1 \subseteq I$ *is called* irreducible *with respect to a partial assignment subproblem if* $\text{PAS}^k(L_1)$ *is infeasible and* $\text{PAS}^k(L_2)$ *is feasible for all* $L_2 \subset L_1$.

Later this definition will be extended to the type of relaxation of the partial assignment subproblem that will be introduced. The definition made then will coincide with what is usually referred to as an irreducible feasibility cut in the context of cut-strengthening algorithms, e.g. Karlsson and Rönnberg (2021).

Typically, a LBBD scheme is derived to exploit problem structure in the sense that iteratively solving the master problem and the subproblem becomes more efficient than solving problem P. In such cases, it is clear that the decomposition that separates the master problem decisions and the subproblem decisions is profitable from a computational point of view. This also means that the partial assignment subproblem, which is a hybrid that contains both types of decisions, is not likely to be viable as a subproblem in an acceleration technique. The next section introduces how to form computationally tractable relaxations and restrictions of the partial assignment subproblem to be used in the acceleration technique instead of the complete partial assignment subproblem.

3.3. Subset-consistent relaxations and restrictions of the partial assignment subproblem

We begin by recalling that the partial assignment subproblem can be solved with different purposes in mind, either to show that a partial assignment is infeasible or to complete it into a feasible solution to problem P. By making a relaxation *Rel* or a restriction *Res* of the partial assignment subproblem such that the resulting problem no longer includes the master problem decisions, the resulting problem is likely to be much easier to solve. It always holds that if a relaxation lacks a feasible solution, so does the problem from which this relaxation is derived. Furthermore, any feasible solution to a restriction of a problem is feasible also in the problem from which this restriction is derived. Therefore, the cases (i) and (ii) introduced above can be extended into:

(I) A relaxation subproblem $\text{RelPAS}^k(L)$, obtained by making a relaxation *Rel* of $\text{PAS}^k(L)$, can be used to determine that a partial assignment $x_i = x_i^k, i \in L$, is infeasible in $\text{PAS}^k(L)$ and to find a feasibility cut $b^k(L)$.

(II) A restriction subproblem $\text{ResPAS}^k(L)$, obtained by making a restriction *Res* of $\text{PAS}^k(L)$, can be used to complete a partial assignment $x_i = x_i^k, i \in L$, into a feasible solution for problem P.

(III) Due to that $\text{PAS}^k(L)$ is not solved exactly, it can hold that a partial assignment $x_i = x_i^k, i \in L$ is neither infeasible in $\text{RelPAS}^k(L)$ nor possible to complete into a feasible solution to problem P by solving $\text{ResPAS}^k(L)$.

The introduction of case (III) can be considered as the price that is paid for replacing the partial assignment subproblem by a relaxation and a restriction. Further, the definitions of *Rel* and *Res* will determine both how common case (III) becomes and how computationally challenging $\text{RelPAS}^k(L)$ and $\text{ResPAS}^k(L)$ will be. Since these relaxations and restrictions are used in the acceleration technique only, the existence of case (III) will have no impact on the convergence of the LBBD scheme since the convergence depends only on finding one feasibility cut.

One aspect that contributes to the efficiency of cut-strengthening algorithms such as DFBS is the use of domination criteria that make it possible to deduce if a subproblem in the tree search is feasible or not based on the outcome from evaluations of other subproblems. For the partial assignment subproblem, such domination criteria follow from the result in Proposition 1. To make it possible to formulate the same type of domination criteria in our acceleration technique, some care is required when defining the relaxations and restrictions. We therefore introduce a certain property that is of importance in this context.

Definition 2. *Let* $\text{PAS}^k(L)$, $L \subseteq I$, *be a partial assignment subproblem in iteration* k *of a LBBD scheme.*

- *Let* $\text{RelPAS}^k(L)$ *be obtained by making a relaxation* Rel *of* $\text{PAS}^k(L)$. *The relaxation* Rel *is called* subset consistent *if* $v\left(\text{RelPAS}^k(L_1)\right) \leq v\left(\text{RelPAS}^k(L_2)\right)$ *holds for all* $L_1 \subseteq L_2 \subseteq I$.
- *Let* $\text{ResPAS}^k(L)$ *be obtained by making a restriction* Res *of* $\text{PAS}^k(L)$. *The restriction* Res *is called* subset consistent *if* $v\left(\text{ResPAS}^k(L_1)\right) \leq v\left(\text{ResPAS}^k(L_2)\right)$ *holds for all* $L_1 \subseteq L_2 \subseteq I$.

Note that in the literature, the relaxations used in cut strengthening for LBBD (e.g., Hooker (2007), Coban and Hooker (2013), Lombardi and Milano (2012), Karlsson and Rönnberg (2021)) are all subset consistent even if this has not been explicitly stated. Also, for simple problem structures, most reasonable relaxations and restrictions are subset consistent, and then this definition only removes some pathological cases. However, for our application presented in Section 4, especially the restriction needs to be carefully crafted to become subset consistent.

The following result summarises the domination criteria that follow immediately from the definitions of relaxations and restrictions, both in the general sense and according to the definition of subset consistence above.

Proposition 2. *Let* $\text{PAS}^k(L)$, $L \subseteq I$, *be a partial assignment subproblem in iteration* k *of an LBBD scheme. Let* $\text{RelPAS}^k(L)$ *be obtained by making a subset-consistent relaxation of* $\text{PAS}^k(L)$ *and let* $\text{ResPAS}^k(L)$ *be obtained by making a subset-consistent restriction of* $\text{PAS}^k(L)$.

(a) If $\text{RelPAS}^k(L_2)$, $L_2 \subseteq I$, *is feasible, then it holds for all* $L_1 \subseteq L_2$ *that* $\text{RelPAS}^k(L_1)$ *is feasible.*

(b) If $\text{RelPAS}^k(L_1)$, $L_1 \subseteq I$, *is infeasible, then it holds for all* $L_2 \supseteq L_1$, *where* $L_2 \subseteq I$, *that* $\text{RelPAS}^k(L_2)$, $\text{PAS}^k(L_2)$ *and all restrictions of* $\text{PAS}^k(L_2)$ *are infeasible.*

(c) If $\text{ResPAS}^k(L_1)$, $L_1 \subseteq I$, *is infeasible, then it holds for all* $L_2 \supseteq L_1$, *where* $L_2 \subseteq I$, *that* $\text{ResPAS}^k(L_2)$ *is infeasible.*

(d) If $\text{ResPAS}^k(L_2)$, $L_2 \subseteq I$, *is feasible, then it holds for all* $L_1 \subseteq L_2$ *that* $\text{ResPAS}^k(L_1)$, $\text{PAS}^k(L_1)$ *and all relaxations of* $\text{PAS}^k(L_1)$ *are feasible.*

If the relaxation of the partial assignment subproblem is subset consistent, the concept of a cut being irreducible with respect to the partial assignment subproblem can be extended to this relaxation as follows.

Definition 3. *Let* $\text{PAS}^k(L)$, $L \subseteq I$, *be a partial assignment subproblem in iteration* k *of an LBBD scheme and let* $\text{RelPAS}^k(L)$ *be obtained by making a subset-consistent relaxation of* $\text{PAS}^k(L)$. *A feasibility cut* $b^k(L_1)$, $L_1 \subseteq I$ *is called* irreducible *with respect to a subset-consistent relaxation of a partial assignment subproblem if* $\text{RelPAS}^k(L_1)$ *is infeasible and* $\text{RelPAS}^k(L_2)$ *is feasible for all* $L_2 \subset L_1$.

Note that a feasibility cut that is irreducible with respect to a subset-consistent relaxation $\text{RelPAS}^k(L_1)$ of a partial assignment subproblem $\text{PAS}^k(L_1)$ may not be irreducible with respect to $\text{PAS}^k(L_1)$ itself due to the relaxation made. Note also that irreducibility is a local property and that there can be more than one cut that is irreducible.

The introduction of a partial assignment subproblem and the concepts of subset-consistent relaxations and restrictions provide a foundation for extending cut-strengthening algorithms to also include a search for feasible solutions. Next section introduces how this can be done when integrated with DFBS.

3.4. Integration with depth-first binary search

The partial assignment acceleration technique is applied in iteration k of a LBBD scheme if subproblem SP^k is infeasible and a feasibility cut $B^k(L_{\text{fc}})$, $L_{\text{fc}} \subseteq I$, has been identified. To apply the technique, both a subset-consistent relaxation *Rel* and subset-consistent restriction *Res* of the partial assignment subproblem $\text{PAS}^k(L)$, $L \subseteq I$, must be known and they will be used to construct the relaxation and restriction subproblem, respectively.

We begin by outlining the search strategy in the DFBS algorithm using the notation introduced in this paper. For details we refer to the pseudo code presented in Appendix A. The output from the search is an irreducible feasibility cut defined by the set of variables $L_{\text{ifc}} \subseteq L_{\text{fc}}$. The search starts with the set L_{ifc} being empty. The set of variables that are the current candidates for being included L_{ifc} is denoted by T and initially $T = L_{\text{fc}}$. During the search, there is also an auxiliary set O that is used to keep track of variables that are neither among the current candidates in T nor among the variables that certainly not will be part of the irreducible cut under construction; initially $O = \emptyset$.

In each major iteration of the search, the goal is to identify one new variable to include in L_{ifc}. This is done by iteratively reducing the set T until only one variable remains and then this variable is added to L_{ifc}. In each such minor iteration, T it is partitioned into the the subsets $T_1 \subseteq T$ and $T_2 = T \setminus T_1$. Then, relaxation subproblem $\text{RelPAS}^k(O \cup T_1 \cup L_{\text{ifc}})$ is solved to determine how to update T and O. If $\text{RelPAS}^k(O \cup T_1 \cup L_{\text{ifc}})$ is infeasible, this gives the valuable information that T_2 can be discarded from further search and the update $T = T_1$ is made. If $\text{RelPAS}^k(O \cup T_1 \cup L_{\text{ifc}})$ is feasible, the update $T = T_2$ is made, but T_1 cannot be discarded so the update $O = O \cup T_1$ is also made. These updates conclude a minor iteration.

Each time L_{ifc} has been extended with a new variable, the problem $\text{RelPAS}^k(L_{\text{ifc}})$ is solved. If this problem is infeasible, an irreducible feasibility cut has been found and the search is terminated. If the problem is feasible, the search is continued with a new major iteration. Such major iteration is initialised with $T = O$ and $O = \emptyset$.

To extend the DFBS algorithm with our search for feasible solutions, the following changes are made. First, the output of the search is extended with a set S of feasible solutions to problem P; initially $S = \emptyset$. Second, the following step is performed in the algorithm each time a relaxation subproblem $\text{RelPAS}^k(L)$, $L \subseteq L_{\text{fc}}$, has been solved.

- If $\text{RelPAS}^k(L)$ is infeasible, solve $\text{ResPAS}^k(I \setminus L)$. If $\text{ResPAS}^k(I \setminus L)$ is feasible with extended solution $(\bar{x}^{\text{res}}, \bar{y}^{\text{res}})$ in problem P, add $(\bar{x}^{\text{res}}, \bar{y}^{\text{res}})$ to S.
- If $\text{RelPAS}^k(L)$ is feasible, solve $\text{ResPAS}^k(L)$. If $\text{ResPAS}^k(L)$ is feasible with extended solution $(\bar{x}^{\text{res}}, \bar{y}^{\text{res}})$ in problem P, add $(\bar{x}^{\text{res}}, \bar{y}^{\text{res}})$ to S.

Hence, we use the outcome from solving the relaxation subproblem to make educated guesses about which restriction subproblems that are promising to try to solve in the search for feasible solutions. The pseudo-code for our resulting partial assignment acceleration technique is presented in Algorithm 1 and there the above introduced notation for the DFBS pseudo code is used. Note that the domination criteria of Definition 2 and Proposition 2 can be checked before constructing a subproblem to determine if its feasibility can be concluded from previous results or not.

In this paper, we suggest that the DFBS algorithm is used in the acceleration technique but in principle, any cut-strengthening algorithm that systematically evaluates subsets of variables can be used. Examples of such strategies can be found in Karlsson and Rönnberg (2021).

4. Solving the avionics scheduling problem

This section introduces the avionics scheduling problem and describes the LBBD scheme implemented to solve it. A detailed problem statement has been given in previous work (Blikstad et al., 2018; Karlsson et al., 2021), and therefore we only provide a condensed version and of it here together with a detailed description of the model formulation. Our description has been adapted to the decomposition to be made and we directly present a decomposed model comprising a master problem, a subproblem, and a description of how master problem decisions are propagated to the subproblem. A compact MIP model for the same problem is presented in Karlsson et al. (2021). The decomposition is such that the master problem schedules the communication while the subproblem schedules the tasks. The intricacy of the model is within the connection between communication and task scheduling, which we here handle implicitly in the propagation of master problem decisions to the subproblem. Some illustrations of the scheduling problem and a part of a solution are found in Appendix D and can be viewed alongside reading this section.

We provide a detailed description of how to formulate a relaxation subproblem and a restriction subproblem derived from a partial assignment subproblem for the avionics scheduling problem. Further, we present a preprocessing algorithm that is used to identify cuts by detecting inconsistencies when generating a subproblem and a strategy to find a good initial solution to the master problem. The implementation of our method is made based on the description

Data: A subset-consistent relaxation *Rel* and a restriction *Res* that applied to $\text{PAS}^k(L)$, $L \subseteq I$, gives $\text{RelPAS}^k(L)$ and $\text{ResPAS}^k(L)$, respectively, for LBBD iteration k
Data: A feasibility cut $b^k(L_{\text{fc}})$ for which $\text{RelPAS}^k(L_{\text{fc}})$ is infeasible
Result: An irreducible feasibility cut $b^k(L_{\text{ifc}})$ and a set of feasible solutions S to problem P

1 $T \leftarrow L_{\text{fc}}; L_{\text{ifc}} \leftarrow O \leftarrow \emptyset, S \leftarrow \emptyset$;
2 **while** *True* **do**
3 **if** $|T| \leq 1$ **then**
4 $L_{\text{ifc}} \leftarrow L_{\text{ifc}} \cup T$;
5 **if** $\text{RelPAS}^k(L_{\text{ifc}})$ *is infeasible* **then**
6 **if** $\text{ResPAS}^k(I \setminus L_{\text{ifc}})$ *is feasible with extended solution* $(\bar{x}^{res}, \bar{y}^{res})$ *in problem* P **then**
7 $S \leftarrow S \cup (\bar{x}^{\text{res}}, \bar{y}^{\text{res}})$;
8 **end**
9 return $b^k(L_{\text{ifc}})$ and S;
10 **else**
11 **if** $\text{ResPAS}^k(L_{\text{ifc}})$ *s feasible with extended solution* $(\bar{x}^{res}, \bar{y}^{res})$ *in problem* P **then**
12 $S \leftarrow S \cup (\bar{x}^{\text{res}}, \bar{y}^{\text{res}})$;
13 **end**
14 **end**
15 $T \leftarrow O; O \leftarrow \emptyset$;
16 **if** $|T| \geq 2$ **then**
17 go to Line 3;
18 **end**
19 $T_2 \leftarrow T; T_1 \leftarrow \emptyset$
20 **else**
21 Split T into T_1 and T_2;
22 **end**
23 **if** $\text{RelPAS}^k(O \cup T_1 \cup L_{\text{ifc}})$ *is feasible* **then**
24 $O \leftarrow O \cup T_1$; $T \leftarrow T_2$;
25 **if** $\text{ResPAS}^k(T)$ *is feasible with extended solution* $(\bar{x}^{res}, \bar{y}^{res})$ *in problem* P **then**
26 $S \leftarrow S \cup (\bar{x}^{\text{res}}, \bar{y}^{\text{res}})$;
27 **end**
28 **else**
29 $T \leftarrow T_1$;
30 **if** $\text{ResPAS}^k(I \setminus (S \cup T_1 \cup L_{\text{ifc}}))$ *is feasible with extended solution* $(\bar{x}^{res}, \bar{y}^{res})$ *in problem* P **then**
31 $S \leftarrow S \cup (\bar{x}^{\text{res}}, \bar{y}^{\text{res}})$;
32 **end**
33 **end**
34 **end**

Algorithm 1: Pseudo-code of the DFBS cut-strenghtening algorithm

in Section 3 and complemented with our problem specific preprocessing and initialisation strategies. The MIP models are solved using Gurobi Optimizer (GUROBI Optimizer, 2022) and the CP models are solved using IBM ILOG CP Optimizer (Laborie et al., 2018). The commercial MIP solver Gurobi Optimizer is chosen because of its general strong performance, see e.g. the comparisons in Mittelmann (2021) on the MIPLIB2017 instances (Gleixner et al., 2021). We chose the commercial CP solver IBM ILOG CP Optimizer for solving the subproblem since it is known to be efficient for scheduling problems (Laborie et al., 2018). The constraints from IBM ILOG CP Optimizer that we use in the CP formulation are introduced in detail in Appendix B.

4.1. Problem statement

The considered avionics scheduling problem is a feasibility problem where the purpose is to find a feasible schedule of predefined length with respect to a set of constraints, or to prove that no such schedule exists. The schedule is of length P and periodic, which means it is repeated over and over again while the system is running.

Let $\mathcal{I}$ be the set of tasks and let $\mathcal{M}$ be the set of messages. Further, let $\mathcal{N}$ be the sets of communication slots. A solution to the problem – or a schedule – is defined by

- a start time, within the time interval $[0, P]$, for each task $i \in \mathcal{I}$, and
- an assignment of a communication slot $n \in \mathcal{N}$ to each message $m \in \mathcal{M}$.

Each task $i \in \mathcal{I}$ is pre-assigned to a module on which it must be scheduled without preemption for the duration of its execution requirement e_i, and without overlapping any other task. For each task $i \in \mathcal{I}$, there are sub-intervals indexed by the set $\mathcal{Q}_i$ and in a feasible schedule, task i must be assigned to one of the sub-intervals $q \in \mathcal{Q}_i$. If a task $i \in \mathcal{I}$ is assigned to sub-interval $q \in \mathcal{Q}_i$ it must start after the release time t^r_{iq} and finish before the deadline t^d_{iq}.

There are two types of modules, Application Modules (AMs) and Communication Modules (CMs). Let the set of AMs be denoted by $\mathcal{H}^{\mathrm{AM}}$ and let the set of CMs be denoted by $\mathcal{H}^{\mathrm{CM}}$. Further, let $\mathcal{H} = \mathcal{H}^{\mathrm{AM}} \cup \mathcal{H}^{\mathrm{CM}}$ and let $\mathcal{I}_h$ denote the set of tasks on module $h \in \mathcal{H}$. Typically, the AMs have very few tasks, each with a long execution requirement, while the CMs have a huge number of tasks and these tasks have a short execution requirement. Tasks on an AM are repeated with period $P/64$, which means that it is sufficient to find a schedule for an AM for the time interval $[0, P/64]$. Tasks on a CM are repeated with period P. The notation used to model the periodicity is that p_i denotes the period of a task $i \in \mathcal{I}$ and that the term task instance and an index $k \in \{1, \ldots, P/p_i\}$ is used to refer to the k^{th} of the P/p_i instances of task i in a schedule of length P.

There are various precedence relations that the tasks need to comply with, and these are modelled using the concept of dependencies. Let $\mathcal{D}$ be the set of such dependencies. A dependency $(i, i', l, l') \in \mathcal{D}$ restricts the duration between the start of an instance $l \in \{1, \ldots, P/p_i\}$ of a task $i \in \mathcal{I}$ to the start of an instance $l' \in \{1, \ldots, P/p_{i'}\}$ of another task $i' \in \mathcal{I}$ to be within the interval given by a minimal time lag $l^{\text{min-dep}}_{ii'll'}$ and a maximal time lag $l^{\text{max-dep}}_{ii'll'}$.

An overview of the system, from a hardware perspective, is given in Figure 1. As illustrated, each AM is connected to a single CM and each CM has at least one AM connected to it. A CM together with its AMs is called a system node. The CMs are connected by a Communication Network (CN) via which they can send messages to each other. A message always has one sending CM and one or more receiving CMs. Note that peripherals can be connected to a system node via the node's CM. The scheduling of the peripherals is not part of the avionics scheduling problem, but their communication with the CM is. In this paper, we only consider problem instances with one CN and thus simplify the notation for CNs. However, the method and problem definition can be extended to problems with more than one CN. When a message $m \in \mathcal{M}$ is assigned to a slot $n \in \mathcal{N}$ it claims l^{msg}_m units of the capacity available in this slot. Each slot $n \in \mathcal{N}$ has a maximum capacity of l^{slot}_n units to be claimed by the messages assigned to the slot.

To send and receive messages on the CN require the involvement of tasks on the CMs and give rise to additional constraints. In particular, to model co-allocation of messages in the same slot requires care. To describe the connection between message and task scheduling, we use the concepts message tasks and message components, and introduce constraints on these. This part of the description deviates from that in previous work and is tailored to the decomposition to be made.

For each message $m \in \mathcal{M}$, message components of task types 1 and 2 are introduced for the CM that the message is sent from and message components of task types 3 and 4 are introduced for each CM that the message is sent to. Let the set $\mathcal{TH}_m \subseteq \{1, 2, 3, 4\} \times \mathcal{H}$ index such message components for each message $m \in \mathcal{M}$. Also, for message $m \in \mathcal{M}$, let $\mathcal{I}^{\mathrm{m}}_{mth}$ include the message component of task type t on CM h, $(t, h) \in \mathcal{TH}_m$. Each message component is associated with an execution requirement e_{mth} and sub-intervals $[t^{\mathrm{r}}_{mthq}, t^{\mathrm{d}}_{mthq}]$, $q \in \mathcal{Q}_{mth}$, $(t, h) \in \mathcal{TH}_m$, $m \in \mathcal{M}$. There is also a set $\mathcal{D}^{\mathrm{mc}}$ that includes dependencies between message components and dependencies between tasks and message components.

When all messages have been assigned a slot, the message components cease to exist and all information from them are propagated to message tasks according to the description in Section 4.2.2. Then, for each slot $n \in \mathcal{N}$, there

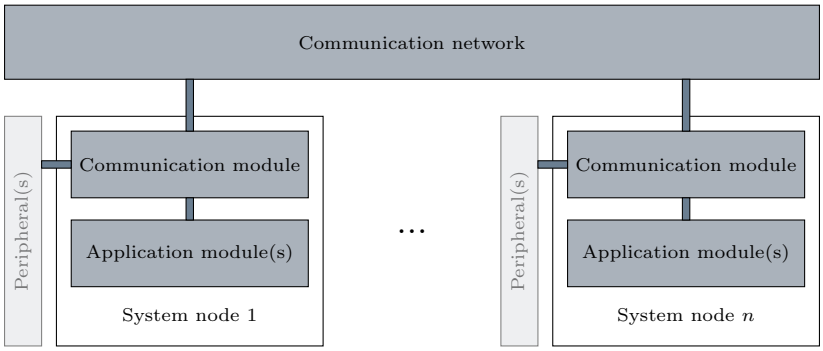

Figure 1: An overview of a system with n nodes. All nodes have the same structure and therefore their contents are only displayed for nodes 1 and n

exist message tasks of task types 1 and 2 for each CM that sends a message in this slot and there exist message tasks of task type 3 and 4 for each CM that receives at least one message in this slot. Let the set $\mathcal{TH}_n \subseteq \{1,2,3,4\} \times \mathcal{H}$ index such message tasks for each slot $n \in \mathcal{N}$. Also, for slot $n \in \mathcal{N}$, let $\mathcal{I}^{\mathrm{n}}_{nth}$ include the message tasks of task type t on CM h, $(t,h) \in \mathcal{TH}_n$.

The message tasks have some properties that are independent of the message components assigned to them. For the message tasks of task type 3, there is a slot-unique release time $t_n^{\text{queue-r}}$ and deadline $t_n^{\text{queue-d}}$, and on each CM, these types of tasks must be scheduled in the same order as the slots they are sent in. Furthermore, the start time of message tasks of task type 2 must be exactly t_n^{send}, $n \in \mathcal{N}$. The existence of a message task $i \in \mathcal{I}^{\mathrm{n}}_{nth}$ is associated with an initialisation time e^{init}_{th}, $t \in \{1,2,3,4\}$, $h \in \mathcal{H}$, that becomes part of its execution requirement. How to derive the complete execution requirements, sub-intervals and the dependencies for message tasks based on the assignment of slots to messages is detailed in Section 4.2.2. Note that when message tasks have been created based on a complete assignment of slots to messages, the problem separates over the system nodes that hence can be independently scheduled.

4.2. Mathematical model and basic logic-based Benders decomposition scheme

Generic notation for a basic LBBD scheme was introduced in Section 3. Here, we provide a mathematical model for the avionics scheduling problem that maps directly to the notation introduced for the decomposed problem in Section 3. Briefly speaking, this means that the master problem variables, used for scheduling of communication, are denoted by x and that the subproblem variables, used for the scheduling of tasks, are denoted by y. Crucial for our decomposition is that the intricate constraints that involve both the communication and the tasks scheduling are handled in the propagation of the master problem solution to the definition of the subproblem.

4.2.1. Master problem

The master problem is a MIP model used to schedule the communication. To assign slots to messages, we introduce for $(m,n) \in I$, $I = \mathcal{M} \times \mathcal{N}$, the binary variable

$$x_{mn} = \begin{cases} 1, & \text{if slot } n \text{ is assigned to message } m, \\ 0, & \text{otherwise.} \end{cases}$$

In iteration k of a LBBD scheme, the solution to the master problem is denoted by $\bar{x}^k_{mn}$, $(m,n) \in I$, and the subset $\bar{I}^k = \{(m,n) \in I : \bar{x}^k_{mn} = 1\} \subseteq I$ is used to refer to the current assignment of slots.

The constraints on $x = (x_{mn})_{(m,n)\in I}$ are

$$[C(x)] \quad \sum_{n\in\mathcal{N}} x_{mn} = 1, \quad m \in \mathcal{M}, \tag{3}$$

$$\sum_{m\in\mathcal{M}} l^{\text{msg}}_m x_{mn} \le l^{\text{slot}}_n, \quad n \in \mathcal{N}. \tag{4}$$

Constraints (3) ensure that each message is assigned exactly one slot and constraints (4) make sure that the maximum capacity of each slot is respected. Since the avionics scheduling problem is a feasibility problem, $f(x) = 0$. However, to improve the computational performance, the objective function

$$g^k(x) = \sum_{(m,n)\in \bar{I}^{k-1}} (1 - x_{mn}), \tag{5}$$

is introduced to stabilise the search. In each iteration k, this objective minimises the number of slot changes with respect to the previous assignment of slots $\bar{I}^{k-1}$. It is an adaptation of one of the objective functions used in Blikstad et al. (2018) and Karlsson et al. (2021). In the first iteration, $\bar{I}^0$ is defined by a solution to the model described in Section 4.5.

The structure of constraints (3) yield that a solution to the master problem in iteration k is uniquely defined by the set $\bar{I}^k$. For this reason, it is sufficient to force one of the variables in $\bar{I}^k$ to change its value in order to make the current solution infeasible. This can be achieved by the (not strengthened) feasibility cut

$$[b(\bar{I}^k)] \qquad \sum_{(m,n)\in \bar{I}^k} (1 - x_{mn}) \geq 1. \tag{6}$$

This cut can be replaced by a (possibly strengthened) feasibility cut $b(\bar{L}^k)$, where $\bar{L}^k \subseteq \bar{I}^k$ is a subset of variables that define a feasibility cut in the current iteration. The feasibility cuts present in the master problem in an iteration k are given by $B^k(x) = \{b(\bar{L}^{k'}) : k' = 1, \ldots, k-1\}$, $\bar{L}^k \subseteq \bar{I}^k$.

To strengthen the master problem, we use a [Subproblem relaxation] that introduces start times of tasks also in the master problem. The constraints include both some aspects that connect the communication and task scheduling and some parts of the task scheduling. However, a majority of the tasks are not prevented from overlapping, since this would be computationally expensive. This part of the master problem model is rather detailed and not central for the main contributions of this paper, and for this reason it is presented in Appendix C.

Using the definitions above, the master problem for the avionics scheduling problem in iteration k of the LBBD scheme is

$$\begin{aligned}
[\text{A-MP}^k] \quad \min\ & g^k(x), \\
\text{s.t.}\ & C(x), \\
& B^k(x), \\
& \text{[Subproblem relaxation]}, \\
& x_{mn} \in \{0,1\}, \quad (m,n) \in I.
\end{aligned}$$

4.2.2. Message tasks

When a solution to a master problem is known and it is decided in which slots to send the messages, the message tasks can be constructed from the message components. The construction of the message tasks is an important component in formulating the subproblem since it propagates information from the master problem solution to the subproblem. A master problem solution from iteration k is here only represented by the set of indices $\bar{I}^k$ since it is sufficient to specify which slot assignments that have been made.

Let the set $\bar{\mathcal{M}}_n(\bar{I}^k) = \{m \in \mathcal{M} : (m,n) \in \bar{I}^k\}$ include the messages that are assigned to slot $n \in \mathcal{N}$. For slot $n \in \mathcal{N}$, a message task of task type t on CM h, $(t,h) \in \mathcal{TH}_n$, exists if the set $\cup_{m\in\bar{\mathcal{M}}_n(\bar{I}^k)} \mathcal{I}^{\text{m}}_{mth}$ is non-empty. Let $\bar{\mathcal{TH}}_n(\bar{I}^k)$ index the message tasks for slot $n \in \mathcal{N}$ for which this is the case. Using this notation, a message task for slot $n \in \mathcal{N}$ of task type t on CM h, $(t,h) \in \bar{\mathcal{TH}}_n(\bar{I}^k)$ is constructed from the message components given by the set $\cup_{m\in\bar{\mathcal{M}}_n(\bar{I}^k)} \mathcal{I}^{\text{m}}_{mth}$ and associated with the execution requirement

$$e_{nth} = e^{\text{init}}_{th} + \sum_{m\in\bar{\mathcal{M}}_n(\bar{I}^k)} e_{mth}. \tag{7}$$

The sub-intervals for message tasks, denoted by $[t^{\text{r}}_{nthq}, t^{\text{d}}_{nthq}]$, $q \in \mathcal{Q}_{nth}$, $(t,h) \in \bar{\mathcal{TH}}_n(\bar{I}^k)$, $n \in \mathcal{N}$, are implicitly defined by

$$\cup_{q\in\mathcal{Q}_{n1h}} [t^{\text{r}}_{n1hq}, t^{\text{d}}_{n1hq}] = \cap_{m\in\bar{\mathcal{M}}_n(\bar{I}^k)} \cup_{q\in\mathcal{Q}_{m1h}} [t^{\text{r}}_{m1hq}, t^{\text{d}}_{m1hq}], \tag{8}$$

$$\cup_{q\in\mathcal{Q}_{n2h}} [t^{\text{r}}_{n2hq}, t^{\text{d}}_{n2hq}] = [t^{\text{send}}_n, t^{\text{send}}_n + e^{\text{init}}_{2h}], \tag{9}$$

$$\cup_{q\in\mathcal{Q}_{n3h}} [t^{\text{r}}_{n3hq}, t^{\text{d}}_{n3hq}] = [t^{\text{queue-r}}_n, t^{\text{queue-d}}_n] \cap \left(\cap_{m\in\bar{\mathcal{M}}_n(\bar{I}^k)} \cup_{q\in\mathcal{Q}_{m3h}} [t^{\text{r}}_{m3hq}, t^{\text{d}}_{m3hq}] \right), \tag{10}$$

$$\cup_{q\in\mathcal{Q}_{n4h}} [t^{\text{r}}_{n4hq}, t^{\text{d}}_{n4hq}] = \cap_{m\in\bar{\mathcal{M}}_n(\bar{I}^k)} \cup_{q\in\mathcal{Q}_{m4h}} [t^{\text{r}}_{m4hq}, t^{\text{d}}_{m4hq}]. \tag{11}$$

To simplify notation in the models to be presented, we introduce a mapping from each tuple (n,t,h), $(t,h) \in \bar{\mathcal{TH}}_n(\bar{I}^k)$, $n \in \mathcal{N}$, to a single task index and let the set $\mathcal{I}^{\text{mt}}(\bar{I}^k)$ include all corresponding message tasks. By this construction, the message tasks can be referred to in the same manner as tasks in $\mathcal{I}$. The quantities e_{nth}, t^{r}_{nthq}, t^{d}_{nthq} and $q \in \mathcal{Q}_{nth}$ introduced for message tasks indexed by (n,t,h), $(t,h) \in \bar{\mathcal{TH}}_n(\bar{I}^k)$, $n \in \mathcal{N}$, will in the following also be denoted by e_i, t^{r}_{iq}, t^{d}_{iq} and $q \in \mathcal{Q}_{ih}$ for $i \in \mathcal{I}^{\text{mt}}(\bar{I}^k)$. Also, we introduce the notation p_i for the period of task $i \in \mathcal{I}^{\text{mt}}(\bar{I}^k)$. The set $\mathcal{I}^{\text{mt}}(\bar{I}^k)$ can be partitioned with respect to the CMs and the notation $\mathcal{I}^{\text{mt}}_h(\bar{I}^k)$ is used to refer to the set of message tasks on CM $h \in \mathcal{H}^{\text{CM}}$.

To propagate information from the dependencies that involve message components, given by the set $\mathcal{D}^{\text{mc}}$, to dependencies that instead involve the message tasks, new dependencies are created where each message component is replaced by the message task it is included in. Denote this new set of dependencies by $\mathcal{D}^{\text{mt}}(\bar{I}^k)$ and note that if $\bar{I}^k$ represents a complete master problem solution, then all dependencies in $\mathcal{D}^{\text{mc}}$ will be included in $\mathcal{D}^{\text{mt}}(\bar{I}^k)$.

In addition to the dependencies described above, the set $\mathcal{D}^{\text{mt}}(\bar{I}^k)$ also includes the following dependencies that are directly defined for the message tasks. Given an order of the slots in $\mathcal{N}$, the message tasks of task type 3 on a CM $h \in \mathcal{H}^{\text{CM}}$, $(3,h) \in \bar{\mathcal{TH}}_n(\bar{I}^k)$, must be scheduled in the same order. This is enforced by a dependency for each pair of message tasks $(3,h) \in \bar{\mathcal{TH}}_n(\bar{I}^k)$ and $(3,h) \in \bar{\mathcal{TH}}_{n'}(\bar{I}^k)$ such that slot $n \in \mathcal{N}$ directly precedes slot $n' \in \mathcal{N}$ in this slot order. Each such dependency is with respect to the first instance of the respective tasks and they all have the minimum time lag 0 and the maximum time lag $P/64$.

When all information from the master problem solution has been propagated to the message tasks, we apply a customised preprocessing component that further reduce the set of feasible solutions before solving the subproblem. The preprocessing component propagates requirements associated with the created message tasks, their sub-intervals, execution requirements, and dependencies, while performing consistency checks. The checks can sometimes conclude in advance that the subproblem is infeasible, but if this is not the case, the next step after the preprocessing component is to create the subproblem.

4.2.3. Subproblem

Given a master problem solution and information propagated to the message tasks, the subproblem is a CP model used to schedule the tasks. The start time of task $i \in \mathcal{I} \cup \mathcal{I}^{\text{mt}}(\bar{I}^k)$ is represented by the interval variable

$$y_i = \text{start time of task } i. \tag{12}$$

The domain D_y of the variables $y = (y_i)_{i \in \mathcal{I} \cup \mathcal{I}^{\text{mt}}(\bar{I}^k)}$ is defined by the constraints

$$\textsc{ForbidStart}(y_i, Q_i(t)), \quad i \in \mathcal{I} \cup \mathcal{I}^{\text{mt}}(\bar{I}^k), \tag{13}$$

where $Q_i(t)$ is a step function that, for each time point t, takes the value 1 if $t \in \cup_{q \in \mathcal{Q}_i} [t^{\text{r}}_{iq}, t^{\text{d}}_{iq}]$ and 0 otherwise.

The schedule is periodic with main period P, and from a system perspective there is no start and end of the schedule. This is reflected in the definitions of dependencies that specify precedence relations between tasks in a cyclic manner. However, since the schedule is created for the interval $[0,P]$, it is, for some tasks, possible to determine the order within this interval. This is done by considering the sub-intervals of the tasks and the time lags of the dependency. Let the subsets $\mathcal{D}^{\text{s}} \subseteq \mathcal{D}$ and $\mathcal{D}^{\text{mt-s}}(\bar{I}^k) \subseteq \mathcal{D}^{\text{mt}}(\bar{I}^k)$ include all dependencies whose task instances have a known order within the interval $[0,P]$. If for dependency $(i,i',l,l') \in \mathcal{D}^{\text{s}}$, instance l' of task i' always need to start before instance l of task i within the interval $[0,P]$, the dependency is remodeled as a dependency (i',i,l',l) with $l^{\text{min-dep}}_{i'il'l} = P - l^{\text{max-dep}}_{iill'}$ and $l^{\text{max-dep}}_{i'il'l} = P - l^{\text{min-dep}}_{ii'll'}$. The analogous reformulation is made also for dependencies in $\mathcal{D}^{\text{mt-s}}(\bar{I}^k)$. Further, let the subsets $\mathcal{D}^{\text{u}} = \mathcal{D} \setminus \mathcal{D}^{\text{s}}$ and $\mathcal{D}^{\text{mt-u}}(\bar{I}^k) = \mathcal{D}^{\text{mt}}(\bar{I}^k) \setminus \mathcal{D}^{\text{mt-s}}(\bar{I}^k)$ include the dependencies for which the order within the interval $[0,P]$ is unknown. For a dependency $(i,i',l,l') \in \mathcal{D}^{\text{u}} \cup \mathcal{D}^{\text{mt-u}}(\bar{I}^k)$, let the binary variable $y^{\text{dep}}_{ii'll'}$ indicate if instance l of task i starts before instance l' of task i' in the interval $[0,P]$. This variable is included in the subproblem so that both options can be considered when solving the problem.

The constraints on x and y are

$$[C(\bar{x}^k, y)\ \&\ C(y)] \quad \textsc{Disjunctive}((y_i | i \in \mathcal{I}_h), (e_i | i \in \mathcal{I}_h)), \quad h \in \mathcal{H}^{\text{AM}}, \tag{14}$$

$$\textsc{Disjunctive}((y_i | i \in \mathcal{I}_h \cup \mathcal{I}^{\text{mt}}_h(\bar{I}^k)), (e_i | i \in \mathcal{I}_h \cup \mathcal{I}^{\text{mt}}_h(\bar{I}^k))), \quad h \in \mathcal{H}^{\text{CM}}, \tag{15}$$

$$\textsc{StartBeforeStart}(y_i, y_{i'}, l^{\text{min-dep}}_{ii'll'} - l'p_{i'} + lp_i), \quad (i,i',l,l') \in \mathcal{D}^{\text{s}} \cup \mathcal{D}^{\text{mt-s}}(\bar{I}^k), \tag{16}$$

$$\textsc{StartBeforeStart}(y_{i'}, y_i, P - l^{\text{max-dep}}_{ii'll'} + l'p_{i'} - lp_i), \quad (i,i',l,l') \in \mathcal{D}^{\text{s}} \cup \mathcal{D}^{\text{mt-s}}(\bar{I}^k), \tag{17}$$

$$l^{\text{min-dep}}_{ii'll'} \leq y_{i'} - y_i + l'p_{i'} - lp_i + Py^{\text{dep}}_{ii'll'} \leq l^{\text{max-dep}}_{ii'll'}, \quad (i,i',l,l') \in \mathcal{D}^{\text{u}} \cup \mathcal{D}^{\text{mt-u}}(\bar{I}^k). \tag{18}$$

Constraints (14) and constraints (15) ensure that all tasks on an AM and on a CM, respectively, are scheduled without overlap. For dependencies that involve tasks in the same period only, constraints (16)–(17) make sure that the precedence relations and time lags are respected. The fulfilment of the remaining dependencies is enforced by constraints (18).

Using the definitions above, the subproblem for the avionics scheduling problem in iteration k of the LBBD scheme is

$$\begin{aligned} [\text{A-SP}^k] \quad & \min\ 0, \\ & \text{s.t. } C(\bar{x}^k, y)\ \&\ C(y), \\ & \quad y \in D_y, \end{aligned}$$

and it can be solved independently for each system node.

4.3. Partial assignments and subproblems for the acceleration technique

This section introduces the relaxation subproblem and the restriction subproblem to be used in the acceleration technique when solving the avionics scheduling problem. These are derived as a subset-consistent relaxation and a subset-consistent restriction, respectively, of a partial assignment subproblem. A partial assignment is defined by the subset $L \subseteq \bar{I}^k$ of the assignments made in the master problem solution in iteration k of the LBBD scheme. A partial assignment subproblem, as defined by $\text{PAS}^k(L)$, is obtained by using the constraints and domains introduced above for the avionics scheduling problem together with the restriction $x_{mn} = \bar{x}^k_{mn}, (m,n) \in L$.

For both the relaxation subproblem and the restriction subproblem, message tasks will be created according to the description in Section 4.2.2, but for a partial assignment only. Let $\tilde{L} \subseteq \bar{I}^k$ denote an assignment that will define the message tasks to be constructed. For the relaxation, message tasks are created for $\tilde{L} = L$, while for the restriction, message tasks are created for an extended set $\tilde{L} = \hat{L} = \{(m,n) \in \bar{I}^k : (m',n) \in L \text{ for some } m' \in \mathcal{M}\}$. This extended set includes all assignments for a slot that has at least one assignment in L. In addition to that, the main difference between the relaxation and the restriction is how the remaining assignments, given by the set $\bar{I}^k \setminus \tilde{L}$, are handled. Before turning to the details of the respective subproblems, we begin with a joint description of how to construct the message tasks given a set $\tilde{L}$.

Analogously to the definition for $\bar{I}^k$, we let the set $\bar{\mathcal{M}}_n(\tilde{L}) = \{m \in \mathcal{M} : (m,n) \in \tilde{L}\}$ include the messages that are assigned to slot $n \in \mathcal{N}$ in the partial assignment given by the set $\tilde{L}$. For slot $n \in \mathcal{N}$, a message task of task type t on CM h, $(t,h) \in \mathcal{TH}_n$, is created if the set $\cup_{m \in \bar{\mathcal{M}}_n(\tilde{L})} \mathcal{I}^{\text{m}}_{mth}$ is non-empty. Following the derivation in Section 4.2.2, an execution requirement e_i, sub-intervals $[t^{\text{r}}_{iq}, t^{\text{d}}_{iq}]$, $q \in \mathcal{Q}_{ih}$, and a period p_i are introduced for $i \in \mathcal{I}^{\text{mt}}(\tilde{L})$, where $\mathcal{I}^{\text{mt}}(\tilde{L})$ is the set of created message tasks. Further, the set $\mathcal{I}^{\text{mt}}_h(\tilde{L})$ includes the created message tasks on CM $h \in \mathcal{H}^{\text{CM}}$. The dependencies in the set $\mathcal{D}^{\text{mt}}(\tilde{L})$ are created from the dependencies in the set $\mathcal{D}^{\text{mc}}$ according to the description in Section 4.2.2. If there are dependencies in $\mathcal{D}^{\text{mc}}$ that involve message components that have not been included in a message task, these are left in this set and not transferred to $\mathcal{D}^{\text{mt}}(\tilde{L})$.

4.3.1. Relaxation subproblem

The relaxation of the partial assignment subproblem is made so that the resulting problem has the same structure as the LBBD subproblem. When the message tasks and dependencies for $\tilde{L} = L$ have been created according to the description above, the main component of the relaxation is to discard the restrictions on the remaining task components and their dependencies. The requirement that message tasks of task type 3 need be scheduled in the same order as their slots is here only applicable for the message tasks that have been created. For these tasks, the set $\mathcal{D}^{\text{mt}}(\tilde{L})$ is extended with dependencies as described in Section 4.2.2. Following the derivation in Section 4.2.3, we let the subsets $\mathcal{D}^{\text{s}} \subseteq \mathcal{D}$ and $\mathcal{D}^{\text{mt-s}}(\tilde{L}) \subseteq \mathcal{D}^{\text{mt}}(\tilde{L})$ include all dependencies that have a known order within the interval $[0,P]$. Dependencies for which the order within the interval $[0,P]$ is not known are included in the subsets $\mathcal{D}^{\text{u}} = \mathcal{D} \setminus \mathcal{D}^{\text{s}}$ and $\mathcal{D}^{\text{mt-u}}(\tilde{L}) = \mathcal{D}^{\text{mt}}(\tilde{L}) \setminus \mathcal{D}^{\text{mt-s}}(\tilde{L})$. Further, for dependency $(i,i',l,l') \in \mathcal{D}^{\text{u}} \cup \mathcal{D}^{\text{mt-u}}(\tilde{L})$, the binary variable $y^{\text{dep}}_{ii'll'}$ indicates if instance l of task i starts before instance l' of task i' in the interval $[0,P]$ or not.

The introduced message tasks are constructed to directly correspond to the assignments made by the constraints $x_{mn} = \bar{x}^k_{mn}, (m,n) \in L$ in $\text{PAS}^k(L)$. To make the partial assignment given by L and discard the assignments not included in L corresponds to relaxing the constraints $C(x)$ by changing the equalities in constraints (3) into $\leq$-inequalities. By construction, this relaxed version of the constraints $C(x)$ and the domain requirements for x become trivially fulfilled and therefore, both $C(x)$ and the domain for x are omitted in the relaxation subproblem. The remaining constraints $C((\bar{x}^k_{mn})_{(m,n)\in L}, y)$ & $C(y)$, together with the domain for y, give a relaxation subproblem for

the subset L in iteration k of a LBBD scheme. It is formulated as

$$
\begin{aligned}
[\text{A-RelPAS}^k(L)] \quad \min\ & 0, \\
\text{s.t.}\ & \textsc{Disjunctive}((y_i|i \in \mathcal{I}_h),(e_i|i \in \mathcal{I}_h)), \quad h \in \mathcal{H}^{\text{AM}}, \\
& \textsc{Disjunctive}((y_i|i \in \mathcal{I}_h \cup \mathcal{I}_h^{\text{mt}}(L)),(e_i|i \in \mathcal{I}_h \cup \mathcal{I}_h^{\text{mt}}(L))), \quad h \in \mathcal{H}^{\text{CM}}, \\
& \textsc{StartBeforeStart}(y_i, y_{i'}, l_{ii'll'}^{\text{min-dep}} - l'p_{i'} + lp_i), \quad (i,i',l,l') \in \mathcal{D}^{\text{s}} \cup \mathcal{D}^{\text{mt-s}}(L), \\
& \textsc{StartBeforeStart}(y_{i'}, y_i, P - l_{ii'll'}^{\text{max-dep}} + l'p_{i'} - lp_i), \quad (i,i',l,l') \in \mathcal{D}^{\text{s}} \cup \mathcal{D}^{\text{mt-s}}(L), \\
& l_{ii'll'}^{\text{min-dep}} \leq y_{i'} - y_i + l'p_{i'} - lp_i + Py_{ii'll'}^{\text{dep}} \leq l_{ii'll'}^{\text{max-dep}}, \quad (i,i',l,l') \in \mathcal{D}^{\text{u}} \cup \mathcal{D}^{\text{mt-u}}(L), \\
& \textsc{ForbidStart}(y_i, Q_i(t)), \quad i \in \mathcal{I}_h \cup \mathcal{I}_h^{\text{mt}}(L), \quad h \in \mathcal{H},
\end{aligned}
$$

and it can be solved independently for each system node.

Proposition 3. *The problem* A-RelPAS$^k(L)$ *is a subset-consistent relaxation of the partial assignment subproblem for the avionics scheduling problem.*

Proof. The relaxation is constructed so that there is a direct correspondence between the created message tasks and the restriction $x_{mn} = \bar{x}_{mn}^k, (m,n) \in L$ in PAS$^k(L)$. Further, the remaining message components are discarded in a way that only relaxes the problem. This means that, given two partial assignments L_1 and L_2 such that $L_1 \subseteq L_2 \subseteq I$ hold, all restrictions imposed by the assignments in L_1 will be imposed also in L_2 so that any solution to A-RelPAS$^k(L_2)$ will be feasible also in A-RelPAS$^k(L_1)$. Therefore $v\left(\text{A-RelPAS}^k(L_1)\right) \leq v\left(\text{A-RelPAS}^k(L_2)\right)$ holds and the relaxation is subset consistent. □

4.3.2. Restriction subproblem

The restriction of the partial assignment subproblem is made so that the resulting problem has a structure that is similar to the LBBD subproblem, but with some additional types of constraints. The restriction is defined by creating message tasks in two steps. First, message tasks and the dependencies for the extended set $\hat{L}$ (that includes all messages that were assigned to a slot where at least one of the assignments is included in L) are created according to the description for partial assignments above. The use of the extended set $\hat{L}$ yields a restriction of PAS$^k(L)$ as it means that the condition $x_{mn} = \bar{x}_{mn}^k, (m,n) \in L$ in PAS$^k(L)$ is replaced by the stronger condition $x_{mn} = \bar{x}_{mn}^k, (m,n) \in \hat{L}$. Thereafter, message tasks for the remaining assignments, given by the set $\bar{I}^k \setminus \hat{L}$, are created. The construction of this second set of message tasks follows the same structure as described above for $\tilde{L}$, but with some differences to be pointed out. Especially, when a set or parameter is defined differently than to before, this is marked with a $\hat{\cdot}$ in addition to the original parameter notation.

The set of messages that are assigned to slot $n \in \mathcal{N}$ in the partial assignment given by the set $\bar{I}^k \setminus \hat{L}$ is denoted by $\bar{\mathcal{M}}_n(\bar{I}^k \setminus \hat{L})$ and we note that $\cup_{n \in \mathcal{N}}(\bar{\mathcal{M}}_n(\hat{L}) \cup \bar{\mathcal{M}}_n(\bar{I}^k \setminus \hat{L})) = \mathcal{M}$ holds. For the assignments $\bar{I}^k \setminus \hat{L}$, the messages assigned to the same slot in the solution to the master problem will be restricted to be together in a slot, but they are allowed to change slot – as long as they do it together and without using the same slot as any other messages. This corresponds to adding the restrictions $x_{mn} = x_{m'n}$ for $m, m' \in \bar{\mathcal{M}}_n(\bar{I}^k \setminus \hat{L})$, $n \in \mathcal{N}$, and $x_{mn} + x_{m'n} \leq 1$, $m' \in \bar{\mathcal{M}}_n(\bar{I}^k \setminus \hat{L})$, $m \in \mathcal{M} \setminus \bar{\mathcal{M}}_n(\bar{I}^k \setminus \hat{L})$, $n \in \mathcal{N}$. When adding these constraints to PAS$^k(L)$, they will impose a restriction on the problem. Note, however, that an important aspect of the construction is that this restriction is weaker than that made for the assignment given by $\hat{L}$.

The construction of the message tasks for the assignment $\bar{I}^k \setminus \hat{L}$ is done to directly correspond to the weaker type of restriction introduced above. For slot $n \in \mathcal{N}$, a message task of task type t on CM h, $(t,h) \in \mathcal{TH}_n$, is created if the set $\cup_{m \in \bar{\mathcal{M}}_n(\bar{I}^k \setminus \hat{L})} \mathcal{I}_{mth}^{\text{m}}$ is non-empty. Then, the set $\hat{\mathcal{TH}}_n(\bar{I}^k \setminus \hat{L})$ is used to index the message tasks for slot $n \in \mathcal{N}$. In line with the description in Section 4.2.2, a message task for slot $n \in \mathcal{N}$ of task type t on CM h, $(t,h) \in \hat{\mathcal{TH}}_n(\bar{I}^k \setminus \hat{L})$ is constructed from the message components given by the set $\cup_{m \in \bar{\mathcal{M}}_n(\bar{I}^k \setminus \hat{L})} \mathcal{I}_{mth}^{\text{m}}$ and associated with an execution requirement.

Since message tasks indexed by $\hat{\mathcal{TH}}_n(\bar{I}^k \setminus \hat{L})$, $n \in \mathcal{N}$, are associated with messages that are allowed to change slot, their sub-intervals are defined differently than those for $\hat{L}$. For $n \in \mathcal{N}$, let $\hat{\mathcal{N}}_n$ be the set of slots that messages $m \in \bar{\mathcal{M}}_n(\bar{I}^k \setminus \hat{L})$ can be assigned to. The sub-intervals $[\hat{t}_{nthq}^{\text{r}}, \hat{t}_{nthq}^{\text{d}}]$, $q \in \mathcal{Q}_{nth}$, for message tasks indexed by

$(t,h) \in \hat{\mathcal{TH}}_n(\bar{I}^k \setminus \hat{L})$, $n \in \mathcal{N}$, are implicitly defined by

$$\cup_{q\in\mathcal{Q}_{n1h}}[\hat{t}^{\mathrm{r}}_{n1hq}, \hat{t}^{\mathrm{d}}_{n1hq}] = \cap_{m\in\bar{\mathcal{M}}_n(\bar{I}^k\setminus\hat{L})} \cup_{q\in\mathcal{Q}_{m1h}} [t^{\mathrm{r}}_{m1hq}, t^{\mathrm{d}}_{m1hq}], \tag{19}$$

$$\cup_{q\in\mathcal{Q}_{n2h}}[\hat{t}^{\mathrm{r}}_{n2hq}, \hat{t}^{\mathrm{d}}_{n2hq}] = \cup_{n'\in\hat{\mathcal{N}}_n}[t^{\mathrm{send}}_{n'}, t^{\mathrm{send}}_{n'} + e^{\mathrm{init}}_{2h}], \tag{20}$$

$$\cup_{q\in\mathcal{Q}_{n3h}}[\hat{t}^{\mathrm{r}}_{n3hq}, \hat{t}^{\mathrm{d}}_{n3hq}] = \cup_{n'\in\hat{\mathcal{N}}_n}\left([t^{\text{queue-r}}_{n'}, t^{\text{queue-d}}_{n'}]\right) \cap \left(\cap_{m\in\bar{\mathcal{M}}_n(\bar{I}^k\setminus\hat{L})} \cup_{q\in\mathcal{Q}_{m3h}} [t^{\mathrm{r}}_{m3hq}, t^{\mathrm{d}}_{m3hq}]\right), \tag{21}$$

$$\cup_{q\in\mathcal{Q}_{n4h}}[\hat{t}^{\mathrm{r}}_{n4hq}, \hat{t}^{\mathrm{d}}_{n4hq}] = \cap_{m\in\bar{\mathcal{M}}_n(\bar{I}^k\setminus\hat{L})} \cup_{q\in\mathcal{Q}_{m4h}} [t^{\mathrm{r}}_{m4hq}, t^{\mathrm{d}}_{m4hq}]. \tag{22}$$

As in Section 4.2.2, we introduce a mapping from each tuple (n,t,h), $(t,h) \in \hat{\mathcal{TH}}_n(\bar{I}^k)$, $n \in \mathcal{N}$, to a single task index and let the set $\hat{\mathcal{I}}^{\mathrm{mt}}(\bar{I}^k \setminus \hat{L})$ include all corresponding message tasks. Analogously, we define an execution requirement e_i, sub-intervals $[\hat{t}^{\mathrm{r}}_{iq}, \hat{t}^{\mathrm{d}}_{iq}]$, $q \in \mathcal{Q}_{ih}$, and a period p_i for $i \in \hat{\mathcal{I}}^{\mathrm{mt}}(\bar{I}^k \setminus \hat{L})$, and $\hat{\mathcal{I}}^{\mathrm{mt}}_h(\bar{I}^k \setminus \hat{L})$ for $h \in \mathcal{H}^{\mathrm{CM}}$. Note further that $\mathcal{I}^{\mathrm{mt}}_h(\hat{L}) \cup \hat{\mathcal{I}}^{\mathrm{mt}}(\bar{I}^k \setminus \hat{L}) = \mathcal{I}^{\mathrm{mt}}_h(\bar{I}^k)$ holds.

After the dependencies in the set $\mathcal{D}^{\mathrm{mt}}(\hat{L})$ were created, some dependencies remained in the set $\mathcal{D}^{\mathrm{mc}}$. With the second set of message tasks, the remaining dependencies can be created and added to the set $\mathcal{D}^{\mathrm{mt}}(\hat{L})$; thereby the complete set of dependencies for message tasks becomes $\mathcal{D}^{\mathrm{mt}}(\hat{L} \cup (\bar{I}^k \setminus \hat{L})) = \mathcal{D}^{\mathrm{mt}}(\bar{I}^k)$. As in Section 4.2.3, the subsets $\mathcal{D}^{\mathrm{s}} \subseteq \mathcal{D}$, $\mathcal{D}^{\text{mt-s}}(\bar{I}^k) \subseteq \mathcal{D}^{\mathrm{mt}}(\bar{I}^k)$, $\mathcal{D}^{\mathrm{u}} = \mathcal{D} \setminus \mathcal{D}^{\mathrm{s}}$, and $\mathcal{D}^{\text{mt-u}}(\bar{I}^k) = \mathcal{D}^{\mathrm{mt}}(\bar{I}^k) \setminus \mathcal{D}^{\text{mt-s}}(\bar{I}^k)$ are introduced, and so is the binary variable $y^{\mathrm{dep}}_{ii'll'}$, $(i,i',l,l') \in \mathcal{D}^{\mathrm{u}} \cup \mathcal{D}^{\text{mt-u}}(\bar{I}^k)$.

The requirement that message tasks of task type 3 on a CM h, $(3,h) \in \mathcal{TH}_n(\bar{I}^k)$, must be scheduled in the same order as their slots was in the previous models handled by dependencies. In the restriction subproblem, this is instead enforced in an equivalent manner by introducing and constraining two sequence variables per CM. Here, the notations $t = 2$ and $t = 3$ will be used for short to refer to tasks of type 2 and 3, respectively, and we introduce the set $\mathcal{I}^{\mathrm{msg}}(\mathcal{I}^{\mathrm{mt}}_h(\bar{I}^k) : t = 3))$ that includes the message tasks of task type 2 that sends the messages received by the message tasks of task type 3 on CM $h \in \mathcal{H}^{\mathrm{CM}}$. The sequence variables are, for each CM $h \in \mathcal{H}^{\mathrm{CM}}$, $\mathcal{S}(y_i | i \in \mathcal{I}^{\mathrm{mt}}_h(\bar{I}^k) : t = 3)$ and $\mathcal{S}(y_i | i \in \mathcal{I}^{\mathrm{msg}}(\mathcal{I}^{\mathrm{mt}}_h(\bar{I}^k) : t = 3))$, using the notation introduced in Appendix B. A consistent order between the message tasks of type 2 and type 3 is then, for each CM that receives messages, enforced by a separate Disjunctive constraint for each sequence variable and a SameSequence constraint between the sequence variables.

The message tasks for the assignments $\hat{L}$ and $\bar{I}^k \setminus \hat{L}$, respectively, are constructed to directly correspond to the assignments made by the constraints $x_{mn} = \bar{x}^k_{mn}, (m,n) \in \hat{L}$, and $x_{mn} = x_{m'n}$, $m, m' \in \bar{\mathcal{M}}_n(\bar{I}^k \setminus \hat{L})$, $n \in \mathcal{N}$. In addition, the constraints $x_{mn} + x_{m'n} \leq 1$, $m' \in \bar{\mathcal{M}}_n(\bar{I}^k \setminus \hat{L})$, $m \in \mathcal{M} \setminus \bar{\mathcal{M}}_n(\bar{I}^k \setminus \hat{L})$, $n \in \mathcal{N}$ make sure that the messages that are not assigned to be in the same slot will use different slots (this is below equivalently formulated as a Disjunctive-constraint on message tasks of task type 2). To include these three constraints in $\mathrm{PAS}^k(L)$ corresponds to making a restriction of $\mathrm{PAS}^k(L)$ that at the same time makes the conditions $x_{mn} = \bar{x}^k_{mn}, (m,n) \in L$, the constraints $C(x)$ and the domain for x redundant to include in $\mathrm{PAS}^k(L)$. The remainder of this restriction of $\mathrm{PAS}^k(L)$ then only includes the constraints $C(\bar{x}^k, y)$ and $C(y)$, together with the domain for y and these yield a restriction subproblem for the subset L in iteration k of a LBBD scheme. It is formulated as

$$\begin{aligned}
[\text{A-ResPAS}^k(L)] \quad \min\ & 0, \\
\text{s.t. } & \textsc{Disjunctive}((y_i | i \in \mathcal{I}_h), (e_i | i \in \mathcal{I}_h)), \quad h \in \mathcal{H}^{\mathrm{AM}}, \\
& \textsc{Disjunctive}((y_i | i \in \mathcal{I}_h \cup \mathcal{I}^{\mathrm{mt}}_h(\bar{I}^k)), (e_i | i \in \mathcal{I}_h \cup \mathcal{I}^{\mathrm{mt}}_h(\bar{I}^k))), \quad h \in \mathcal{H}^{\mathrm{CM}}, \\
& \textsc{StartBeforeStart}(y_i, y_{i'}, l^{\text{min-dep}}_{ii'll'} - l'p_{i'} + lp_i), \quad (i,i',l,l') \in \mathcal{D}^{\mathrm{s}} \cup \mathcal{D}^{\text{mt-s}}(\bar{I}^k), \\
& \textsc{StartBeforeStart}(y_{i'}, y_i, P - l^{\text{max-dep}}_{ii'll'} + l'p_{i'} - lp_i), \quad (i,i',l,l') \in \mathcal{D}^{\mathrm{s}} \cup \mathcal{D}^{\text{mt-s}}(\bar{I}^k), \\
& l^{\text{min-dep}}_{ii'll'} \leq y_{i'} - y_i + l'p_{i'} - lp_i + Py^{\mathrm{dep}}_{ii'll'} \leq l^{\text{max-dep}}_{ii'll'}, \quad (i,i',l,l') \in \mathcal{D}^{\mathrm{u}} \cup \mathcal{D}^{\text{mt-u}}(\bar{I}^k), \\
& \textsc{Disjunctive}((y_i | i \in \mathcal{I}^{\mathrm{mt}}(\bar{I}^k) : t = 2), (e_i | i \in \mathcal{I}^{\mathrm{mt}}(\bar{I}^k) : t = 2)), \\
& \textsc{SameSequence}(\mathcal{S}(y_i | i \in \mathcal{I}^{\mathrm{msg}}(\mathcal{I}^{\mathrm{mt}}_h(\bar{I}^k) : t = 3)), \mathcal{S}(y_i | i \in \mathcal{I}^{\mathrm{mt}}_h(\bar{I}^k) : t = 3)), \quad h \in \mathcal{H}^{\mathrm{CM}}, \\
& \textsc{Disjunctive}(\mathcal{S}(y_i | i \in \mathcal{I}^{\mathrm{msg}}(\mathcal{I}^{\mathrm{mt}}_h(\bar{I}^k) : t = 3)), (e_i | i \in \mathcal{I}^{\mathrm{msg}}(\mathcal{I}^{\mathrm{mt}}_h(\bar{I}^k) : t = 3))), \quad h \in \mathcal{H}^{\mathrm{CM}}, \\
& \textsc{Disjunctive}(\mathcal{S}(y_i | i \in \mathcal{I}^{\mathrm{mt}}_h(\bar{I}^k) : t = 3), (e_i | i \in \mathcal{I}^{\mathrm{mt}}_h(\bar{I}^k) : t = 3)), \quad h \in \mathcal{H}^{\mathrm{CM}}, \\
& \textsc{ForbidStart}(y_i, \hat{Q}_i(t)), \quad i \in \mathcal{I}_h \cup \mathcal{I}^{\mathrm{mt}}_h(L), \quad h \in \mathcal{H}.
\end{aligned}$$

The sub-intervals are considered in the definition of the step function $\hat{Q}_i(t)$ that, for each time point t, takes the value 1 if $t \in \cup_{q\in\mathcal{Q}_i}[t^{\mathrm{r}}_{iq}, t^{\mathrm{d}}_{iq}]$, $i \in \mathcal{I}_h \cup \mathcal{I}^{\mathrm{mt}}_h(\hat{L})$, or if $t \in \cup_{q\in\mathcal{Q}_i}[\hat{t}^{\mathrm{r}}_{iq}, \hat{t}^{\mathrm{d}}_{iq}]$, $i \in \hat{\mathcal{I}}^{\mathrm{mt}}(\bar{I}^k \setminus \hat{L})$, and 0 otherwise.

Proposition 4. *The problem* $\text{A-ResPAS}^k(L)$ *is a subset-consistent restriction of the partial assignment subproblem for the avionics scheduling problem.*

Proof. For both types of message tasks introduced, a restriction of the partial assignment subproblem is made. For the assignment $\hat{L}$, this restriction is stronger than that made for the assignment $\bar{I}^k \setminus \hat{L}$. This means that, given two partial assignments L_1 and L_2 such that $L_1 \subseteq L_2 \subseteq I$ hold, any solution to A-ResPAS$^k(L_1)$ will be feasible also in A-ResPAS$^k(L_2)$. Therefore the relation $v\left(\text{A-ResPAS}^k(L_1)\right) \leq v\left(\text{A-ResPAS}^k(L_2)\right)$ holds and the restriction is subset consistent. □

4.4. Preprocessing cuts

We have included two greedy problem-specific strategies to find, typically strong, cuts before attempting to solve the subproblem and applying the acceleration technique. As previously described, the subproblem is constructed by propagating information from the master problem solution to the message tasks. Thereafter, we apply customised preprocessing to reduce the set of feasible solutions in the subproblem. The preprocessing component propagates information associated with the created message tasks, such as sub-intervals, execution requirements, and dependencies, while performing consistency checks. If an inconsistency is detected, a set of involved tasks and message tasks is identified. As detailed below, this information is used in two strategies to find subsets of master problem variables that either might form feasibility cuts or can take their current values in a feasible solution. By solving a relaxation subproblem it is determined if a subset yields a cut and thereafter, a restriction subproblem is solved as in the acceleration technique described in Section 3.4. If the relaxation subproblem was feasible, the same set of variables is used to define the restriction subproblem, and otherwise the complement of this subset is used. Since we are interested in finding master problem variables to include in a cut, the strategies consider the message components rather than the message tasks they are included in, as the former correspond directly to the messages.

The first strategy uses a graph to identify promising master problem variables. This graph has a node for each task and each message component. Two nodes are connected by an edge if there is a dependency between the tasks or the message components that the nodes represent, or if these tasks or message components are direct neighbours in the master problem solution (i.e. no other tasks or message component has a start time between their start times). A subset of master problem variables is constructed for each step $\delta = 0, \ldots$, below, where a master problem variable is included in a subset if at least one of its corresponding message components are identified to be of interest. For $\delta = 0$, only the message components that are directly involved in the given inconsistency are considered. Then, for the remaining steps $\delta = 1, \ldots$, all message components that can be reached by a path of at most δ edges from a node that is part of the inconsistency are considered. Thereby, the subset is extended for each step. The procedure stops when a feasibility cut or a feasible solution is found, or when all master problem variables are included in the subset.

The second strategy evaluates a single subset as follows. For each task and message task associated with the inconsistency, one or, in case of a tie, more message components of type 2 that are the closest neighbours are identified, and the corresponding master problem variables are included in the subset.

4.5. Creating an initial solution

Since our master problem has an objective that minimises the number of slot changes compared to the assignment in a previous iteration, it is beneficial to initialise our LBBD scheme with a high-quality solution. This is done by solving a subproblem that includes the message components instead of message tasks, and where the assignment of slots to messages is determined by the subintervals assigned to message components of type 2.

A particular property of the message components of type 2 in the instances we consider is that they have zero-valued execution requirements as their corresponding message tasks have fixed execution requirements. As an approximation when creating an initial solution, we assume that at most 5 messages can be co-allocated in a slot by changing the execution requirement of message components of type 2 into

$$e_{m2h} = e_{th}^{\text{init}}/5, \quad (t,h) \in \mathcal{TH}_m, m \in \mathcal{M} : t = 2. \tag{23}$$

The model is constructed in line with the other subproblems but includes message components instead of message tasks. This difference is handled as follows. Analogous to Section 4.2.2, introduce a mapping from each tuple (m,t,h), $(t,h) \in \mathcal{TH}_m$, $m \in \mathcal{M}$, to a single task index and let the set $\mathcal{I}^{\text{mc}}$ include all corresponding message components. The quantities e_{mth}, t^{r}_{mthq}, t^{d}_{mthq} and $q \in \mathcal{Q}_{mth}$ introduced for message component indexed by (m,t,h), $(t,h) \in \mathcal{TH}_m$, $m \in \mathcal{M}$, will in the following also be denoted by e_i, t^{r}_{iq}, t^{d}_{iq} and $q \in \mathcal{Q}_{ih}$ for $i \in \mathcal{I}^{\text{mc}}$. Introduce the notation p_i for the period of task $i \in \mathcal{I}^{\text{mc}}$. Also, let the notation $\mathcal{I}^{\text{mc}}_h$ refer to the set of message components on module $h \in \mathcal{H}$. The start time of task $i \in \mathcal{I} \cup \mathcal{I}^{\text{mc}}$ is represented by the interval variable y_i. Each task $i \in \mathcal{I} \cup \mathcal{I}^{\text{mc}}$ is assigned a sub-interval by a ForbidStart-constraint, using the step function $Q_i(t)$ that, for each time point t, takes the value 1 if $t \in \cup_{q \in \mathcal{Q}_i}[t^{\text{r}}_{iq}, t^{\text{d}}_{iq}]$ and 0 otherwise.

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

Table 1: The table presents the average numbers for key instance attributes for Categories A–D

Let the subset $\mathcal{D}^{\text{mc-s}} \subseteq \mathcal{D}^{\text{mc}}$ include dependencies for which the order within the interval $[0, P]$ is known. Also, let subset $\mathcal{D}^{\text{mc-u}} = \mathcal{D}^{\text{mc}} \setminus \mathcal{D}^{\text{mc-s}}$ include the dependencies for which the order within the interval $[0, P]$ is not known. For a dependency $(i, i', l, l') \in \mathcal{D}^{\text{u}} \cup \mathcal{D}^{\text{mc-u}}$, let the binary variable $y^{\text{dep}}_{ii'll'}$ indicate if instance l of task i starts before instance l' of task i' in the interval $[0, P]$ or not. The model used to create an initial solution then becomes

$$
\begin{aligned}
\text{[A-init]} \quad \min\ & 0, \\
\text{s.t.}\ & \textsc{Disjunctive}((y_i | i \in \mathcal{I}_h), (e_i | i \in \mathcal{I}_h)), \quad h \in \mathcal{H}^{\text{AM}}, \\
& \textsc{Disjunctive}((y_i | i \in \mathcal{I}_h \cup \mathcal{I}^{\text{mc}}_h), (e_i | i \in \mathcal{I}_h \cup \mathcal{I}^{\text{mc}}_h)), \quad h \in \mathcal{H}^{\text{CM}}, \\
& \textsc{StartBeforeStart}(y_i, y_{i'}, l^{\text{min-dep}}_{ii'll'} - l' p_{i'} + l p_i), \quad (i, i', l, l') \in \mathcal{D}^{\text{s}} \cup \mathcal{D}^{\text{mc-s}}, \\
& \textsc{StartBeforeStart}(y_{i'}, y_i, P - l^{\text{max-dep}}_{ii'll'} + l' p_{i'} - l p_i), \quad (i, i', l, l') \in \mathcal{D}^{\text{s}} \cup \mathcal{D}^{\text{mc-s}}, \\
& l^{\text{min-dep}}_{ii'll'} \leq y_{i'} - y_i + l' p_{i'} - l p_i + P y^{\text{dep}}_{ii'll'} \leq l^{\text{max-dep}}_{ii'll'}, \quad (i, i', l, l') \in \mathcal{D}^{\text{u}} \cup \mathcal{D}^{\text{mc-u}}, \\
& \textsc{ForbidStart}(y_i, Q_i(t)), \quad i \in \mathcal{I}_h \cup \mathcal{I}^{\text{mc}}_h, \quad h \in \mathcal{H},
\end{aligned}
$$

and it includes the same types of constraints as the subproblem and the relaxation subproblem.

5. Computational results

To evaluate the efficiency of our LBBD method, we used the publicly available instances for the avionics scheduling problem that were introduced in Karlsson et al. (2021). The instances, the computational environment, and the specific settings of the LBBD method are described in Section 5.1. In Section 5.2, the LBBD method is compared with two previously developed methods for the avionics scheduling problem. A deeper analysis of the characteristics of the partial acceleration technique and the obtained cuts is given in Section 5.3. In the final two sections, we present results from additional experiments that illustrate the impact that the partial assignment acceleration technique and the [Subproblem relaxation] component have on the computational performance.

5.1. Benchmark instances and computational environment

The computational evaluations have been performed on instance categories A–D introduced in Karlsson et al. (2021). The instances are publicly available[2] and were developed in collaboration with Saab to be industrially relevant for the development of future avionic systems. Categories A–D contain instances of increasing difficulty and the largest instance has 20 modules, 2530 messages, and 54,731 tasks. The average number of tasks, dependencies, messages, AMs, and CMs in the instance categories are given in Table 1.

The LBBD method has been implemented using Python 3.7. The MIP models are solved using Gurobi Optimizer version 9.0 and the CP models are solved using IBM ILOG CP Optimizer version 12.10. All tests are carried out on a computer with two Intel Xeon Gold 6130 Processors (16 cores, 2.1 GHz) with 96 GB RAM for the LBBD method and 384 GB RAM for the other methods.

In the LBBD method, the master problem is solved to optimality without any time-limit. The LBBD subproblem is, for each node in the avionic system, given 20 minutes, while the relaxation subproblems and the restriction subproblems are given 5 minutes per node. If our LBBD method cannot solve the LBBD subproblem within the time-limit, we report a time-out error. If a relaxation subproblem is not solved within the time-limit, the result is treated as being feasible with respect to the cut strengthening, but of course, it is not treated as such when checking domination criteria for future relaxation subproblems. If a restriction subproblem is not solved within the time-limit,

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

nothing is done, and again, this result is not used in any domination criteria. We have observed that if a LBBD subproblem or a relaxation subproblem is not solved within its time-limit, increasing the time-limit, even to hours, does not usually help. The partial assignment acceleration technique and DFBS cut strengthening are implemented such that when a subset of indices is split into two subsets, this is done by randomly dividing the set into two subsets of equal size.

One – somewhat surprising – finding during the computational study was the sometimes huge differences in solution times between very similar instances of the subproblems. As instances with such properties might be of interest for further studies and solver benchmarking, we made a separate report (Karlsson and Rönnberg, 2022) on this topic. There, we introduce multiprocessor scheduling instances[3], with a pure structure of only including multiple time windows and time lags, that reproduce these computational challenges in a more generic setting.

5.2. Computational comparison with previous approaches

To evaluate the performance of our LBBD method, we have chosen to compare it against two methods previously developed for the avionics scheduling problem. In previous work, we have also tried to solve the problem by directly applying either a MIP or a CP solver, but this is not a viable option for the instances considered in this paper. The first method in our comparison is an exact MIP-based constraint generation approach that was introduced in Blikstad et al. (2018) and then improved in Karlsson et al. (2021). In the comparison, we use the improved version which is also an exact method. The master problem of the constraint generation procedure finds an assignment of sub-intervals to tasks. A feasible schedule is then searched for in a subproblem where each task must be performed within the sub-interval it was assigned to in the master problem solution. If no feasible schedule is found, constraints are added to the master problem and the subproblem. The second method is a MIP-based matheuristic introduced in Karlsson et al. (2021). The matheuristic extends the constraint generation procedure by solving the master problem with a MIP-based adaptive large neighbourhood search. Below, the constraint generation procedure and the matheuristic are referred to as MIP-CG and MIP-matheuristic, respectively. In the evaluation, we have used the same settings for MIP-CG and MIP-matheuristic as in Karlsson et al. (2021), but with an update to Python 3.7 and Gurobi 9.0. Note that MIP-CG and MIP-matheuristic were given access to computers with 384 GB RAM since the methods were developed with this type of RAM availability in mind. In comparison, the LBBD method was given access to computers with 96 GB RAM.

In the computational evaluation, we compare the LBBD method, the MIP-CG method, and the MIP-matheuristic method for the instances in Categories A–D. The metric of practical interest is how long it takes to find a feasible solution or conclude that none exists. Therefore, we recorded – for each instance and each method – how much time that was needed to find a feasible solution (or determine infeasibility), if this was done within a time limit of 24 hours. The result is illustrated in Figure 2 that shows the number of solved instances after a given time.

For the smallest instance category, Category A, all 30 instances were solved within 2 hours by all methods. For Category B, the LBBD method performed best and solved all 30 instances within 1 hour while MIP-CG and MIP-matheuristic solved 25 and 28 instances, respectively, within 24 hours. For Category C, the LBBD method solved all 30 instances within 5 hours. This can be compared to the MIP-CG and the MIP-matheuristic methods that solved 16 and 26 instances, respectively, within 24 hours. For the most challenging category, Category D, the LBBD method solved 28 out of 30 instances within 24 hours, while MIP-CG and MIP-matheuristic solved 1 and 19 instances, respectively, within 24 hours. It should also be noted that if the MIP-CG and the MIP-matheuristic methods get 72 hours per instance, they solve 5 and 28 instances, respectively, for Category D. Across all the 120 instances, no time-out error was reported for the LBBD method.

To summarise, the LBBD method solves more instances within the time limit while relying on less RAM than the other methods, regardless of instance category. For all but the smallest instances, the solution times are also much shorter. There are a few reasons, we believe, for these results. The first reason is that the LBBD method often finds a solution in its first subproblem, without any feedback. For Categories A–D, the number of instances that are solved without any feedback for the LBBD scheme are 25, 12, 6, and 6, respectively. The second reason is that the LBBD method successfully exploit the task-scheduling structure of the avionics scheduling problem by using a subproblem that can be efficiently solved with IBM ILOG CP Optimizer although being very large. This made it possible to formulate a master problem in the LBBD method where the tasks do not need to be assigned to small sub-intervals as in MIP-CG and MIP-matheuristic and thereby the master problem became much easier to solve. Since the master problem is the computational bottleneck for MIP-CG and MIP-matheuristic, this plays a significant role.

[3]https://gitlab.liu.se/eliro15/multiprocessor_scheduling_inst

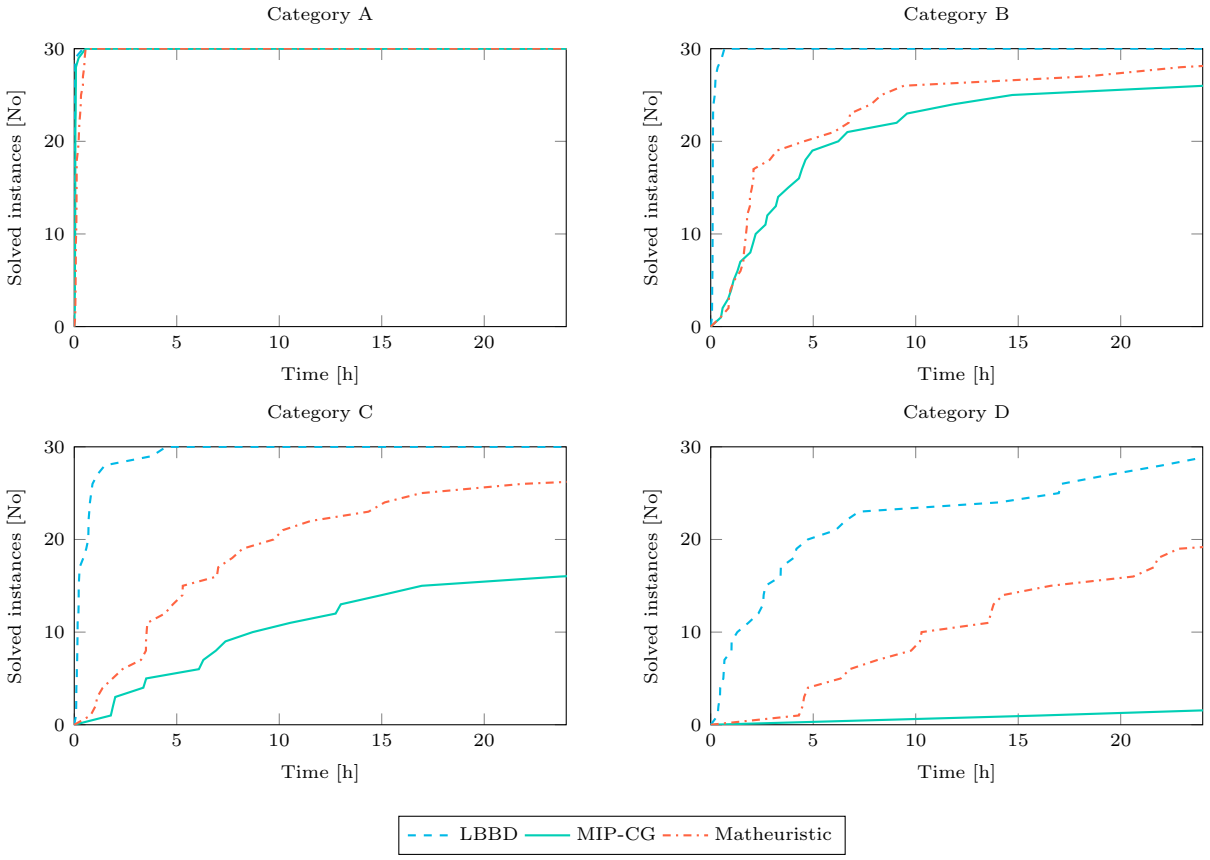

Figure 2: The number of instances of Categories A–D that are solved within a given time by the LBBD method, the MIP-CG method, and the MIP-matheuristic method

	Avg. no. cuts	Avg. cut size	Avg. no. rel. subproblems solved
Category A	0	N/A	0.2
Category B	0.1	2.8	1.6
Category C	2.0	1.5	11.8
Category D	2.0	26.3	29.4

Table 2: The table presents, for each instance category, the average number of feasibility cuts added to the master problem, the average size of these cuts, and the average number of solved relaxation subproblems. Only data from instances that were solved within 24 hours was collected.

	Subproblem	Restriction [feasible]	Restriction [infeasible]	Restriction [timed out]
Category A	26	0	3	1
Category B	24	1	4	1
Category C	15	2	11	2
Category D	7	13	5	3

Table 3: The table presents the number of instances of different instance categories that were solved in the LBBD subproblem and in the restriction subproblem. The number of solutions found in a restriction subproblem is presented for the cases that the last relaxation subproblem was feasible, infeasible, or stopped by time-out

5.3. Detailed results for the partial assignment acceleration technique

To characterise the behaviour of the partial assignment acceleration technique and the obtained cuts, we compiled data from the experiments with the LBBD method in the previous section. Table 2 contains, for each instance category, the average number of feasibility cuts added to the master problem, the average size of the added feasibility cuts, and the average number of solved relaxation subproblems. Only data from instances that were solved within 24 hours is included in these averages. Table 3 contains, for each instance category, the number of instances solved by an ordinary LBBD subproblem, by a restriction subproblem constructed after an infeasible relaxation subproblem, by a restriction subproblem constructed after a feasible relaxation subproblem, and by a restriction subproblem constructed after a time-out relaxation subproblem.

The average number of feasibility cuts added to the master problem in order to solve an instance was small for all categories. In Category A, no feasibility cuts were needed at all. For Categories B–D, the average number of feasibility cuts were 0.1, 2.0, and 2.0, respectively. Especially interesting is that the obtained feasibility cuts were very small compared to the original cut. For the solved instances in Categories B–D the average size was 2.8, 1.5, and 26.3, respectively. These figures are to be compared to the size of the original cuts for these instances, which are 447, 908, and 1923, for Categories B–D, respectively. The average size of an added feasibility cut for the instances in Category D is larger than for Category B and C, but it was primarily due to a single LBBD iteration that generated a feasibility cut of size 1339 due to time-outs in the relaxation subproblems. Without this feasibility cut, the average size of an added feasibility cut for the instances that were solved in Category D was 2.5. The last column in Table 2 shows that not that many relaxation subproblems need to be solved in order to find the strengthened cuts. For Categories A–D, the average number of relaxation subproblems were 0.2, 1.6, 11.8, and 29.4, respectively.

Table 3 shows in which of the components of the LBBD method that the solution to an instance was found. For Category A, the solutions to 26 out of 30 instances were found by solving a LBBD subproblem and only for 4 instances, the solution was found by solving a restriction subproblem. This can be compared to Category D, for which the solutions to 7 out of 28 instances were found by solving a LBBD subproblem and the solutions to 21 out of 28 instances were found by solving a restriction subproblem. For Category B, the solutions to 24 out of 30 instances were found by solving a LBBD subproblem and the solutions to 6 instances were found by solving a restriction subproblem. For Category C, the solutions to 15 out of 30 instances were found by solving a LBBD subproblem and the solutions to 15 instances were found by solving a restriction subproblem.

For the instances that were solved by a restriction subproblem, we could not discern any pattern with respect to how and when this was formulated. Rather, solutions were found by solving restriction subproblems constructed after both infeasible and infeasible relaxation subproblems. However, at least one instance per category was solved by a restriction subproblem constructed after a relaxation subproblem that timed out.

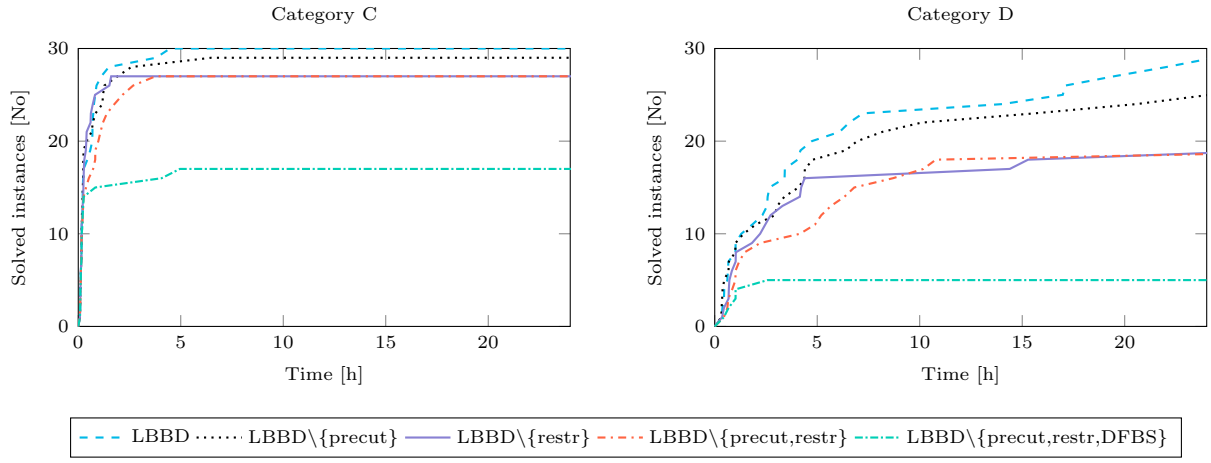

Figure 3: The number of instances of Categories C and D, respectively, that are solved within a given time when removing components of the LBBD method

5.4. Impact of the different components of the partial assignment acceleration technique

To investigate the impact that the different components of the partial assignment acceleration technique and the preprocessing cuts have on the computational performance of the LBBD method, we have performed additional experiments with and without some of these components. The different variants are denoted as follows. LBBD\{restr} is the LBBD method with the restriction subproblem removed; LBBD\{precut} is the LBBD method with the preprocessing cuts removed; LBBD\{restr,precut} is the LBBD method with both the restriction subproblem and the preprocessing cuts removed; LBBD\{restr,precut,DFBS} is the LBBD method with all types of cut strengthening removed. The pseudo-code for the partial assignment acceleration technique without the restriction subproblem, i.e. DFBS cut strengthening, is given in Appendix A. In these tests we used the instances in Categories C and D and the time-out was 24 hours. The number of solved instances for each variant after a given time is illustrated in Figure 3.

For the complete LBBD method, 30 and 28 instances of Categories C and D, respectively, were solved within 24 hours. The corresponding number of solved instances for LBBD\{restr} was 27 and 19 for Categories C and D, respectively. For LBBD\{precut}, 29 and 27 instances of Categories C and D, respectively, were solved within 24 hours. These result were better than for LBBD\{restr}, but still worse than for the complete LBBD method. For LBBD\{restr,precut}, the number of solved instances in Categories C and D was the same as LBBD\{restr}. The most drastic reduction in performance was observed when removing all types of cut strengthening. Within 24 hours, LBBD\{restr,precut,DFBS} could only solve 17 and 8 instances of Categories C and D, respectively.

The results show that all the components contribute to the computational performance of the complete LBBD method. They also show, especially for the largest instances, that our approach to extend cut strengthening with a component that tries to construct feasible solutions has a significant impact on the computational performance.

5.5. The impact of the initial solution and the subproblem relaxation component

This section gives an evaluation of the impact that our initial solution and [Subproblem relaxation] components have on the computational performance. To evaluate the former, we made changes to the initial iteration in two ways and then the experiments were made both with and without the subproblem relaxation. In the first variant, which is denoted LBBDrand with the [Subproblem relaxation] component and LBBDrand\{sr} without it, the initial solution was a random assignment of messages to slots, and then the search was stabilised with respect to this solution. In the second variant, which is denoted LBBDnofirst with the [Subproblem relaxation] component and LBBDnofirst\{sr} without it, the objective function of the master problem was omitted in the first iteration and no initial solution was used. In this case, the first master problem solution becomes the first feasible solution that Gurobi Optimizer finds for the master problem. We also tested the complete LBBD method without a subproblem relaxation, and this we denote by LBBD\{sr}. To provide start times for tasks and message components in the master problem solution for the preprocessing cuts even without a subproblem relaxation, the start time variables were included in the master problem together with constraints (C.8). The experiments were made for the instances in Categories C and D and the time-out was 24 hours. The number of solved instances after a given time are illustrated in Figure 4.

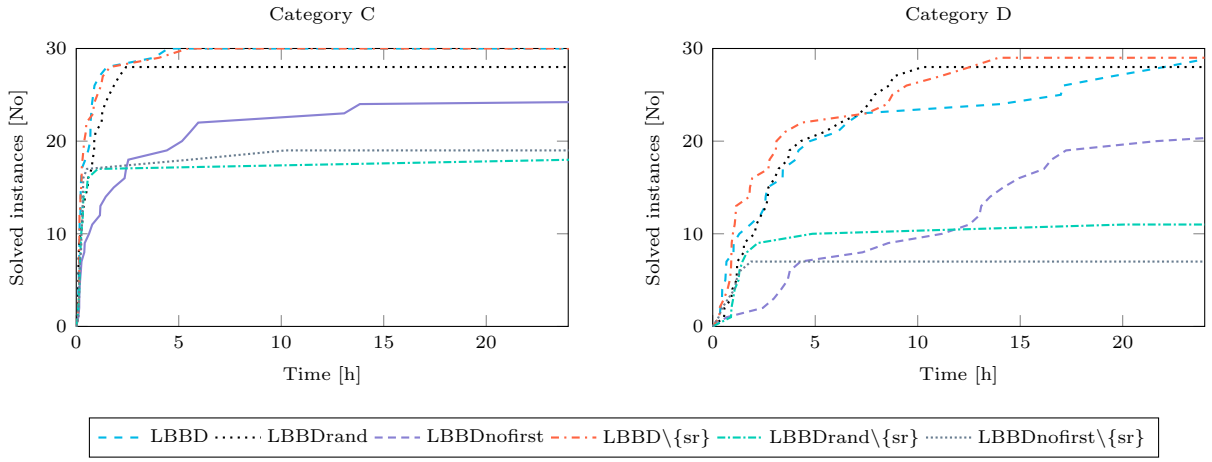

Figure 4: The number of instances of Categories C and D, respectively, that are solved within 24 hours when changing the first iteration of the LBBD method and/or removing the [Subproblem relaxation] component

The most striking result is that the computational performance for the complete LBBD method is very similar to that for LBBD\{sr} and LBBDrand. Within 24 hours, 30, 30, and 28 instances of Category C were solved, respectively, and the corresponding number of instances of Category D are 28, 29, and 28. The best results are hence obtained for the complete LBBD method without a subproblem relaxation. Important to note, however, is that if a random initial solution is used when the [Subproblem relaxation] component is omitted, then the results are drastically worse. For LBBDrand\{sr}, 17 and 11 instances of Categories C and D, respectively, were solved within 24 hours. This clearly indicates that replacing a random assignment with the assignment created from our initial solution component and including the [Subproblem relaxation] component do have an impact on the computational performance, but they appear to play a similar role and one of the additions can therefore be omitted.

The results of using the first solution found by Gurobi Optimizer are also interesting. For LBBDnofirst, 25 and 22 instances of Categories C and D, respectively, were solved within 24 hours. For LBBDnofirst\{sr}, 19 and 7 instances of Categories C and D, respectively, were solved within 24 hours. These results are not as good as when the complete LBBD method is used and it should be noted that they are also worse than if a random assignment was used to stabilise against.

6. Concluding remarks

This paper introduces a method based on LBBD for solving a large-scale and industrially relevant avionics scheduling problem. This problem can be described as a rich multiprocessor scheduling problem that also includes the scheduling of a communication network. The decomposition is such that the master problem is of assignment type and the subproblem is a feasibility problem of scheduling type. The master problem is solved by the MIP-solver Gurobi Optimizer and the subproblem is solved by the CP-solver IBM ILOG CP Optimizer. The method outperforms previous exact and matheuristic methods for this problem, both with respect to solution times and RAM usage. As seen in the computational results, our new acceleration technique is an important component to achieve this.

The new acceleration technique is obtained by extending DFBS cut strengthening of feasibility cuts for LBBD with a heuristic search for feasible solutions. Our search uses information from the cut strengthening by selecting subsets of variables that have not been chosen to be part of the strengthened cut. A restriction subproblem is formed by making a so-called subset-consistent restriction of a partial assignment subproblem where these variables are assigned the values they had in the master problem solution. If a feasible solution to the restriction subproblem is found, then this solution is feasible also in the original problem. Since our extension can have the potential to also be useful for other applications and in other acceleration techniques, we have described it in a generic way. This description includes the introduction of a partial assignment subproblem, to which a subset-consistent restriction is applied to form the restriction subproblem. In the literature, the descriptions of how to form the subproblems in a cut-strengthening algorithm are problem specific. By introducing the partial assignment subproblem, and how to make subset-consistent relaxations of it to obtain a relaxation subproblem, we also contribute to a more structured and general description of how to form such subproblems.

There are three key observations that we believe contribute to explain the efficiency of our acceleration technique when included in our LBBD scheme and applied to the avionics scheduling problem. Firstly, the non-strengthened feasibility cuts are very large compared to the irreducible feasibility cuts obtained from the DFBS cut strengthening. Secondly, the problem structure is such that it is possible to form a subset-consistent relaxation and a subset-consistent restriction that are computationally profitable. By this we mean that they carry enough relevant information and are strong enough to be meaningful to solve while they have a structure that allows them to be solved quickly. Lastly, the addressed problem is a feasibility problem and this means that any feasible solution found when solving the restriction subproblem will do. Together, these three properties make it possible for the acceleration technique to generate high-quality information and feedback in a reasonable amount of time.

A possible future research direction is to apply the acceleration technique suggested in this paper to other LBBD schemes. A good starting point can be problems that only include feasibility cuts, such as the ones in Karlsson and Rönnberg (2021) and Lam et al. (2020), and in particular problems that are decomposed to get an assignment-type master problem and a scheduling-type subproblem. A challenge when generalising the technique is to derive a computationally profitable restriction subproblem that gets a simple structure without omitting too much of the feasible space. Another interesting research direction would be to apply this type of acceleration technique in a LBBD method that includes optimality cuts. In Hooker (2007) and Coban and Hooker (2013), for example, optimality cuts are strengthened in a way that is similar to the cut-strengthening algorithm used in this paper and therefore there is potential to include our acceleration technique in their type of scheme. Finally, we note that our strategy for choosing when to apply the search for feasible solutions is rather simple, and it might be worth to further investigate if a more elaborate strategy can improve the computational performance.

Acknowledgement

The work of Emil Karlsson is supported by the Research School in Interdisciplinary Mathematics at Linköping University. The work is also partly funded by the Center for Industrial Information Technology (CENIIT), Project-ID 16.05. Computational experiments were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) at National Supercomputer Centre (NSC). This work is part of the collaboration *Efficient use of hardware resources in avionic systems* between Linköping University and Saab Aeronautics.

Declaration of competing interest

Elina Rönnberg and Emil Karlsson make no declaration of competing interest.

Data availability statement

All computational experiments have been made on data publicly available in the repository `https://gitlab.liu.se/eliro15/avionics_inst`. Based on some challenges experienced during the computational experiments, we have made a separate report (Karlsson and Rönnberg, 2022) that introduces instances of a multiprocessor scheduling problem with multiple time windows and time lags. These instances are of interest because the computational times can differ greatly between instances that are very similar, and they are publicly available in the repository `https://gitlab.liu.se/eliro15/multiprocessor_scheduling_inst`.

References

Atlihan, M.K., Schrage, L., 2008. Generalized filtering algorithms for infeasibility analysis. Comput. Oper. Res. 35, 1446–1464. URL: `https://doi.org/10.1016/j.cor.2006.08.005`.

Bajestani, M.A., Beck, J.C., 2011. Scheduling an aircraft repair shop, in: Bacchus, F., Domshlak, C., Edelkamp, S., Helmbert, M. (Eds.), Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling, AAAI Press. pp. 10–17.

Benini, L., Lombardi, M., Mantovani, M., Milano, M., Ruggiero, M., 2008. Multi-stage Benders decomposition for optimizing multicore architectures, in: Perron, L., Trick, M. (Eds.), CPAIOR 2008: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer-Verlag Berlin Heidelberg. pp. 36–50. URL: `https://doi.org/10.1007/978-3-540-68155-7_6`.

Blikstad, M., Karlsson, E., Lööw, T., Rönnberg, E., 2018. An optimisation approach for pre-runtime scheduling of tasks and communication in an integrated modular avionic system. Optim. Eng. 19, 977–1004. URL: https://doi.org/10.1007/s11081-018-9385-6.

Cambazard, H., Hladik, P.E., Déplance, A.M., Jussien, N., Trinquet, Y., 2004. Decomposition and learning for a hard real time task allocation problem, in: Wallace, M. (Ed.), Principles and Practice of Constraint Programming – CP 2004, Springer-Verlag Berlin Heidelberg. pp. 153–167. URL: https://doi.org/10.1007/978-3-540-30201-8_14.

Chinneck, J.W., Dravnieks, E.W., 1991. Locating minimal infeasible constraint sets in linear programs. ORSA J. Comput. 3, 157–168. URL: https://doi.org/10.1287/ijoc.3.2.157.

Coban, E., Hooker, J.N., 2013. Single-facility scheduling by logic-based Benders decomposition. Ann. Oper. Res. 210, 245–272. URL: https://doi.org/10.1007/s10479-011-1031-z.

Codato, G., Fischetti, M., 2006. Combinatorial Benders' cuts for mixed-integer linear programming. Oper. Res 54, 756–766. URL: https://doi.org/10.1287/opre.1060.0286.

Emde, S., Polten, L., Gendreau, M., 2020. Logic-based benders decomposition for scheduling a batching machine. Comput. Oper. Res. 113. URL: https://doi.org/10.1016/j.cor.2019.104777.

Emeretlis, A., Theodoridis, G., Alefragis, P., Voros, N., 2016. A logic-based Benders decomposition approach for mapping applications on heterogeneous multicore platforms. ACM Trans. Embed. Comput. Syst. 15, 19:1–19:28. URL: https://doi.org/10.1145/2838733.

Gaska, T., Watkin, C., Chen, Y., 2015. Integrated modular avionics – past, present, and future. IEEE Aerosp. Electron. Syst. Mag. 30, 12–23. URL: https://doi.org/10.1109/MAES.2015.150014.

Gleixner, A., Hendel, G., Gamrath, G., Achterberg, T., Bastubbe, M., Berthold, T., Christophel, P., Jarck, K., Koch, T., Linderoth, J., Lübbecke, M., Mittelmann, H.D., Ozyurt, D., Ralphs, T.K., Salvagnin, D., Shinano, Y., 2021. MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library. Math. Prog. Comp. 13, 443–490. URL: https://doi.org/10.1007/s12532-020-00194-3.

GUROBI Optimizer, 2022. URL: https://www.gurobi.com/products/gurobi-optimizer. Accessed: 17 May 2022.

He, X., Yuan, M., Gu, Z., 2008. A hierarchical framework for design space exploration and optimization of TTP-Based distributed embedded systems. IEEE Trans. Ind. Inform. 4, 237–249. URL: https://doi.org/10.1109/TII.2008.2010519.

Hooker, J.N., 2000. Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction. John Wiley & Sons. URL: https://doi.org/10.1002/9781118033036.

Hooker, J.N., 2007. Planning and scheduling by logic-based Benders decomposition. Oper. Res. 55, 588–602. URL: https://doi.org/10.1287/opre.1060.0371.

Hooker, J.N., 2019. Logic-based Benders decomposition for large-scale optimization, in: Velásquez-Bermúdez, J.M., Khakifirooz, M., Fathi, M. (Eds.), Large Scale Optimization in Supply Chains and Smart Manufacturing: Theory and Applications. Springer International Publishing, pp. 1–26. URL: https://doi.org/10.1007/978-3-030-22788-3_1.

Hooker, J.N., Ottosson, G., 2003. Logic-based Benders decomposition. Math. Program. 96, 33–60. URL: https://doi.org/10.1007/s10107-003-0375-9.

Junker, U., 2001. QuickXPlain: Conflict detection for arbitrary constraint propagation algorithms, in: IJCAI01 Workshop on Modeling and Solving Problems with Constraints (CONS-1).

Junker, U., 2004. QuickXPlain: Preferred explanations and relaxations for over-constrained problems, in: Cohn, A.G. (Ed.), AAAI'04: Proceedings of the 19th national conference on Artifical intelligence, pp. 167–172.

Karlsson, E., Rönnberg, E., 2021. Strengthening of feasibility cuts in logic-based benders decomposition, in: Stuckey, P.J. (Ed.), Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR) 2021, Springer International Publishing. pp. 45–61. URL: https://doi.org/10.1007/978-3-030-78230-6_3, doi:10.1007/978-3-030-78230-6_3.

Karlsson, E., Rönnberg, E., 2022. Instance dataset for a multiprocessor scheduling problem with multiple time windows and time lags: Similar instances with large differences in difficulty. Report.

Karlsson, E., Rönnberg, E., Stenberg, A., Uppman, H., 2021. A matheuristic approach to large-scale avionic scheduling. Ann. Oper. Res. 302, 425–459. URL: https://doi.org/10.1007/s10479-020-03608-6.

Kim, H.J., Hooker, J.N., 2002. Solving fixed-charge network flow problems with a hybrid optimization and constraint programming approach. Ann. Oper. Res. 115, 95–124. URL: https://doi.org/10.1023/A:1021145103592.

Laborie, P., Rogerie, J., Shaw, P., Vilím, P., 2018. IBM ILOG CP optimizer for scheduling. Constraints 23, 210–250. URL: https://doi.org/10.1007/s10601-018-9281-x.

Lam, E., Gange, G., Stuckey, P.J., Van Hentenryck, P., Dekker, J.J., 2020. Nutmeg: a MIP and CP hybrid solver using branch-and-check. SN Oper. Res. Forum 1, 22:1–22:27. URL: https://doi.org/10.1007/s43069-020-00023-2.

Lindh, E., Olsson, K., Rönnberg, E., 2022. Scheduling of an underground mine by combining logic-based benders decomposition and a priority-based heuristic, in: De Causmaecker, P., Özcan, E., Vanden Berghe, G. (Eds.), Accepted for publication in the Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling - PATAT 2022.

Lombardi, M., Milano, M., 2012. Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. Constraints 17, 51–85. URL: https://doi.org/10.1007/s10601-011-9115-6.

Mittelmann, H.D., 2021. The MIPLIB2017 benchmark instances. URL: http://plato.asu.edu/ftp/milp.html. Accessed: 17 May 2022.

Parker, M., Ryan, J., 1996. Finding the minimum weight IIS cover of an infeasible system of linear inequalities. Ann. Math. Artif. Intell. 17, 107–126. URL: https://doi.org/10.1007/BF02284626.

Rahmaniani, R., Crainic, T.G., Gendreau, M., Rei, W., 2017. The Benders decomposition algorithm: A literature review. Eur. J. Oper. Res. 259, 801–817. URL: https://doi.org/10.1016/j.ejor.2016.12.005.

Raidl, G.R., 2015. Decomposition based hybrid metaheuristics. Eur. J. Oper. Res 244, 66–76. URL: https://www.sciencedirect.com/science/article/pii/S0377221714009874, doi:https://doi.org/10.1016/j.ejor.2014.12.005.

Raidl, G.R., Baumhauer, T., Hu, B., 2020. Boosting an exact logic-based Benders decomposition approach by variable neighborhood search. Electron. Notes Discrete Math. 47, 149–156. URL: https://doi.org/10.1016/j.endm.2014.11.020.

Raidl, G.R., Baumhauer, T., Hu, B., 2021. Speeding up logic-based Benders' decomposition by a metaheuristic for a bi-level capacitated vehicle routing problem, in: Blesa, M.J., Blum, C., Voß, S. (Eds.), Hybrid Metaheuristics: 9th International Workshop, HM 2014 Hamburg, Germany, June 11-13, 2014 Proceedings, Springer International Publishing Switzerland. pp. 183–197. URL: https://doi.org/10.1007/978-3-319-07644-7_14.

Riedler, M., Raidl, G., 2020. Solving a selective dial-a-ride problem with logic-based Benders decomposition. Comput. Oper. Res. 96, 30–54. URL: https://doi.org/10.1016/j.cor.2018.03.008.

Robati, T., Gherbi, A., El Kouhen, A., Mullins, J., 2017. Design and simulation of distributed IMA architectures using TTEthernet. J. Ambient Intell. Humaniz. Comput. 8, 345–355. URL: https://doi.org/10.1007/s12652-017-0449-9.

Saharidis, G.K.D., Ierapetritou, M.G., 2010. Improving benders decomposition using maximum feasible subsystem (MFS) cut generation strategy. Comput. Chem. Eng. 34, 1237–1245. URL: https://doi.org/10.1016/j.compchemeng.2009.10.002.

Sun, D., Tang, L., Baldacci, R., 2019. A Benders decomposition-based framework for solving quay crane scheduling problems. Eur. J. Oper. Res 273, 504–515. URL: https://doi.org/10.1016/j.ejor.2018.08.009.

Wang, H., Niu, W., 2018. A review of key technologies of the distributed integrated modular avionics system. Int. J. Wirel. Inf. Netw. 25, 358–369. URL: https://doi.org/10.1007/s10776-018-0412-5.

Xu, J., Parnas, D.L., 2000. Priority scheduling versus pre-run-time scheduling. Real-Time Syst. 18, 7–23. URL: https://doi.org/10.1016/S1474-6670(17)44872-5.

Zhang, M., Zhang, N., Li, H., Gu, Z., 2018. A decomposition-based approach to optimization of TTP-based distributed embedded systems. J. Syst. Archit. 91, 53–61. URL: https://doi.org/10.1016/j.sysarc.2018.07.006.

Zhou, X., Xiong, H., He, F., 2020. Hybrid partition- and network-level scheduling design for distributed integrated modular avionics systems. Chin. J. of Aeronautics 33, 308–323. URL: https://doi.org/10.1016/j.cja.2019.08.027.

Appendix A. Depth-first binary search cut-strenghtening algorithm

This section gives the pseudo-code for the DFBS cut-strengthening algorithm, see Algorithm 2. The DFBS algorithm produces an irreducible feasibility cut with respect to the subset-consistent relaxation used. The difference to Algorithm 1 is that it does not include the restriction step.

Data: A subset-consistent relaxation *Rel* that applied to $\text{PAS}^k(L)$, $L \subseteq I$, gives $\text{RelPAS}^k(L)$ for LBBD iteration k
Data: A feasibility cut $b^k(L_{\text{fc}})$ for which $\text{RelPAS}^k(L_{\text{fc}})$ is infeasible
Result: An irreducible feasibility cut $b^k(L_{\text{ifc}})$ with respect to relaxation Rel

1 $T \leftarrow L_{\text{fc}}; L_{\text{ifc}} \leftarrow O \leftarrow \emptyset$;
2 **while** *True* **do**
3 **if** $|T| \leq 1$ **then**
4 $L_{\text{ifc}} \leftarrow L_{\text{ifc}} \cup T$;
5 **if** $\text{RelPAS}^k(L_{\text{ifc}})$ *is infeasible* **then**
6 return $b^k(L_{\text{ifc}})$;
7 **end**
8 $T \leftarrow O; O \leftarrow \emptyset$;
9 **if** $|T| \geq 2$ **then**
10 go to Line 3;
11 **end**
12 $T_2 \leftarrow T; T_1 \leftarrow \emptyset$
13 **else**
14 Split T into T_1 and T_2;
15 **end**
16 **if** $\text{RelPAS}^k(O \cup T_1 \cup L_{\text{ifc}})$ *is feasible* **then**
17 $O \leftarrow O \cup T_1$; $T \leftarrow T_2$;
18 **else**
19 $T \leftarrow T_1$;
20 **end**
21 **end**

Algorithm 2: Pseudo-code of the partial assignment subproblem acceleration technique

Appendix B. IBM ILOG CP Optimizer variables and constraints

To implement the CP models, we use variables and constraints from IBM ILOG CP Optimizer. This section gives a brief description of the variables and constraints that we have used and provides details of the used notation.

Appendix B.1. Interval variables and constraints

Interval variables are often used to represent decisions associated with time. An interval variable s_i is associated with a release time $r_i \in \mathbb{Z}$ and a deadline $d_i \in \mathbb{Z}$, that define its domain $[r_i, d_i]$. The decisions of an interval variable s_i are normally to select a start time $\text{start}(s_i)$ and an end time $\text{end}(s_i)$ such that $r_i \leq \text{start}(s_i) \leq \text{end}(s_i) \leq d_i$. The processing time or length of the interval variable s_i is then $\text{proc}(s_i) = \text{end}(s_i) - \text{start}(s_i)$. However, in our modelling, we only use interval variables with a fixed length and in our presentation, the processing times are added as input to the constraints where they are needed. Also, we do not use the concept of absent interval variables so this aspect is omitted from our presentation.

The ForbidStart constraint is defined for an interval variable s and a step function $F(t)$, where $t \in \mathbb{Z}$ is a time point. The constraint states that the interval variable s cannot start at a time $t \in \mathbb{Z}$ where $F(t) = 0$. The Disjunctive constraint is defined for an array of interval variables $(s_1, \ldots, s_n)$ and an array of processing times $(p_1, \ldots, p_n)$. An interval variable s_i, $i = 1, \ldots, n$, is assumed to be active in the constraint if $\text{start}(s_i) \leq t < \text{start}(s_i) + p_i$. The constraint stipulates that for each time point $t \in \mathbb{Z}$ at most one interval variable can be active. The StartBeforeStart constraint takes as input two interval variables s_1 and s_2, and an integer z. The constraint states that

$$\text{start}(s_1) + z \leq \text{start}(s_2). \tag{B.1}$$

Appendix B.2. Sequence variables and constraints

A sequence variable is defined for an array of interval variables and is used to represent an order of the interval variables. The value of a sequence variable $\mathcal{S}(s_1, \ldots, s_n)$ is a permutation of the indices of the interval variables and it represents the order of start times of the interval variables (e.g., a value of the sequence variable $\mathcal{S}(s_1, s_2, s_3)$ is the permutation $(2, 1, 3)$ if it holds that $\text{start}(s_2) \leq \text{start}(s_1) \leq \text{start}(s_3)$).

The SameSequence constraint takes as input two sequence variables $\mathcal{S}(s_1, \ldots, s_n)$ and $\mathcal{S}(\tilde{s}_1, \ldots, \tilde{s}_n)$ of the same length. The constraint states that the value (permutation) of the two sequence variables shall be identical. The Disjunctive constraint can also be applied to a sequence variable $\mathcal{S}(s_1, \ldots, s_n)$ and an array of processing times $(p_1, \ldots, p_n)$. The interval variable s_i, $i = 1, \ldots, n$, is assumed to be active in the constraint if $\text{start}(s_i) \leq t < \text{start}(s_i) + p_i$. The constraint stipulates that for each time point $t \in \mathbb{Z}$ at most one interval variable s_i, $i = 1, \ldots, n$, of the sequence variable can be active.

Appendix C. The subproblem relaxation component

This section gives a complete description of the [Subproblem relaxation] component used in the master problem in Section 4.2.1. The key decisions, besides the master problem variables, are to select start times for all tasks and message components. By ensuring that the start times are connected to the master problem variables while obeying most of the constraints in the avionics scheduling problem, the model is strengthened. The only constraints of the problem that are relaxed in the master problem are the sub-interval constraints and the constraints associated with the sequencing of the CM tasks. What to include in the subproblem relaxation component was inspired by the master problem in Karlsson et al. (2021), and although we made this formulation slightly simpler, many variables and constraints are similar.

Introduce for each $i \in \mathcal{I} \cup \mathcal{I}^{\text{mc}}$, the continuous variable

$$y_i = \text{start time of task or message component } i.$$

For each task (or message component) $i \in \mathcal{I} \cup \mathcal{I}^{\text{mc}}$, let $t_i^{\text{r}} = \min_{q \in \mathcal{Q}_i} t_{iq}^{\text{r}}$ and $t_i^{\text{d}} = \max_{q \in \mathcal{Q}_i} t_{iq}^{\text{d}}$ denote the first possible release time and last possible deadline, respectively.

Besides the master problem variables and the start times, we need auxiliary variables to support the modelling. For a dependency $(i, i', l, l') \in \mathcal{D} \cup \mathcal{D}^{\text{mc}}$, let the binary variable $y_{ii'll'}^{\text{dep}}$ indicate if instance l of task (or message component) i starts before instance l' of task (or message component) i' in a major period or not. Let $\mathcal{I}^{\text{mp}} = \{i \in \mathcal{I} \cup \mathcal{I}^{\text{mc}} : t_i^{\text{d}} - t_i^{\text{r}} > P/2\}$ denote the set of tasks that can be performed both in the beginning and end of a major period. For a task $i \in \mathcal{I}^{\text{mp}}$ and sub-interval $q \in \mathcal{Q}_i$, let the binary variable y_{iq}^{si} indicate if task i is assigned to sub-interval q or not. To further strengthen the model, we include a sequence-dependent idle time between tasks on the same AM and these are derived from dependencies with equal minimum and maximum length. For the pair of tasks $(i, i') \in \mathcal{I}_h \times \mathcal{I}_h$, $i \neq i'$, $h \in \mathcal{H}^{\text{AM}}$, the idle time is denoted by $l_{ii'}^{\text{idle}}$. Let also $\mathcal{II}_h^{\text{AM}} = \{(i, i') \in \mathcal{I}_h \times \mathcal{I}_h : i < i'\}$ denote the set of pairs of tasks on an AM $h \in \mathcal{H}^{\text{AM}}$ that needs to be ordered. For pair of tasks $(i, i') \in \mathcal{II}_h^{\text{AM}}$ on AM $h \in \mathcal{H}^{\text{AM}}$, let the binary variable $y_{i,i'}^{\text{AM}}$ indicate if task i is scheduled before task i' or not.

As part of strengthening the master problem, we ensure that message components of the same type do not overlap each other and that the order between message components of type 2 and type 3 are consistent. To accomplish this, we keep track of the order the messages are sent in a major frame. The following variables assist with these aspects. For $m',\ m \in \mathcal{M} : m < m'$, let the binary variable $w^{\mathrm{S}}_{mm'}$, be equal to 1 if message m is placed before message m' or equal to 0 if message m' is placed before message m. Also, for $m',\ m \in \mathcal{M} : m < m'$, let the binary variable $w_{mm'}$ indicate if message m is placed before message m' in a slot. Let $\mathcal{II}^{\mathrm{m}}_{mm'th} = \{(i,i') \in \mathcal{I}^{\mathrm{m}}_{mth} \times \mathcal{I}^{\mathrm{m}}_{m'th} : |\mathcal{I}^{\mathrm{m}}_{mth} \cup \mathcal{I}^{\mathrm{m}}_{mth}| \geq 2\}$ denote, if it exists, the pair of message components of type $t \in \{1,2,3,4\}$ belonging to messages $(m,m') \in \mathcal{M} \times \mathcal{M}$ on CM $h \in \mathcal{H}^{\mathrm{CM}}$. For $(i,i') \in \mathcal{II}^{\mathrm{m}}_{mm'3h}$, $m' \in \mathcal{M}$, $m \in \mathcal{M} : m < m'$, $h \in \mathcal{H}^{\mathrm{CM}}$, let the binary variable $w^{\mathrm{M}}_{ii'}$ be equal to 1 if message component i starts before message component i' and equal to 0 if message component i' starts before message component i. Further, let $n^{\min}_m$ and $n^{\max}_m$ denote the least and the greatest slot index, respectively, that is eligible for message $m \in \mathcal{M}$.

The [Subproblem relaxation] component are with these variables formulated as

$$t^{\mathrm{r}}_i \leq y_i \leq t^{\mathrm{d}}_i - e_i, \quad i \in \mathcal{I} \cup \mathcal{I}^{\mathrm{mc}}. \tag{C.1}$$

$$\sum_{q \in \mathcal{Q}_i} y^{\mathrm{si}}_{iq} = 1, \quad i \in \mathcal{I}^{\mathrm{mp}}, \tag{C.2}$$

$$\sum_{q \in \mathcal{Q}_i} t^{\mathrm{r}}_{iq} y^{\mathrm{si}}_{iq} \leq x_i \leq \sum_{q \in \mathcal{Q}_i} t^{\mathrm{d}}_{iq} y^{\mathrm{si}}_{iq} - e_i, \quad i \in \mathcal{I}^{\mathrm{mp}}, \tag{C.3}$$

$$l^{\text{min-dep}}_{ii'll'} \leq y_{i'} - y_i + l' p_{i'} - l p_i + P y^{\mathrm{dep}}_{ii'll'} \leq l^{\text{max-dep}}_{ii'll'}, \quad (i,i',l,l') \in \mathcal{D} \cup \mathcal{D}^{\mathrm{mc}}, \tag{C.4}$$

$$y_i + e_i + l^{\mathrm{idle}}_{ii'} - (t^{\mathrm{d}}_i + l^{\mathrm{idle}}_{ii'} - t^{\mathrm{r}}_{i'})(1 - y^{\mathrm{AM}}_{ii'}) \leq y_{i'}, \quad (i,i') \in \mathcal{II}^{\mathrm{AM}}_h,\ h \in \mathcal{H}^{\mathrm{AM}}, \tag{C.5}$$

$$y_{i'} + e_{i'} + l^{\mathrm{idle}}_{i'i} - (t^{\mathrm{d}}_{i'} + l^{\mathrm{idle}}_{i'i} - t^{\mathrm{r}}_i) y^{\mathrm{AM}}_{ii'} \leq y_i, \quad (i,i') \in \mathcal{II}^{\mathrm{AM}}_h,\ h \in \mathcal{H}^{\mathrm{AM}}, \tag{C.6}$$

$$\sum_{n \in \mathcal{N}} t^{\text{queue-r}}_n x_{mn} \leq y_i \leq \sum_{n \in \mathcal{N}} t^{\text{queue-d}}_n x_{mn} - e_i, \quad i \in \mathcal{I}^{\mathrm{m}}_{m3h}, m \in \mathcal{M}, h \in \mathcal{H}^{\mathrm{CM}}, \tag{C.7}$$

$$y_i = \sum_{n \in \mathcal{N}} t^{\mathrm{send}}_n x_{mn}, \quad i \in \mathcal{I}^{\mathrm{m}}_{m2h}, m \in \mathcal{M}, h \in \mathcal{H}^{\mathrm{CM}}, \tag{C.8}$$

$$\begin{aligned}\sum_{n \in \mathcal{N}} n y_{mn} + 1 - w_{mm'} - (n^{\max}_m - n^{\min}_{m'} + 1)(1 - w^{\mathrm{S}}_{mm'}) &\leq \sum_{n \in \mathcal{N}} n x_{m'n} \\ &\leq \sum_{n \in \mathcal{N}} n y_{mn} + (n^{\max}_{m'} - n^{\min}_m)(1 - w_{mm'}), \quad m' \in \mathcal{M}, m \in \mathcal{M} : m < m',\end{aligned} \tag{C.9}$$

$$\sum_{n \in \mathcal{N}} n x_{m'n} + 1 - (n^{\max}_{m'} + 1 - n^{\min}_m) w^{\mathrm{S}}_{mm'} \leq \sum_{n \in \mathcal{N}} n x_{mn}, \quad m' \in \mathcal{M}, m \in \mathcal{M} : m < m', \tag{C.10}$$

$$w_{mm'} \leq w^{\mathrm{S}}_{mm'}, \quad m' \in \mathcal{M}, m \in \mathcal{M} : m < m', \tag{C.11}$$

$$\begin{aligned}y_i + e_i - (t^{\mathrm{d}}_i - t^{\mathrm{r}}_{i'})(1 - w_{mm'}) &\leq y_{i'} \\ &\leq y_i + e_i + \sum_{i'' \in \mathcal{I}^{\mathrm{m}}_{m''th},\ m'' \in \mathcal{M} : m < m'' < m'} e_{i''} w_{mm''} + (t^{\mathrm{d}}_{i'} - t^{\mathrm{r}}_i)(1 - w_{mm'}), \\ &\qquad (i,i') \in \mathcal{II}^{\mathrm{m}}_{mm'th},\ t \in \{1,4\},\ h \in \mathcal{H}^{\mathrm{CM}}, m' \in \mathcal{M}, m \in \mathcal{M} : m < m',\end{aligned} \tag{C.12}$$

$$\begin{aligned}y_i + e_i - (t^{\mathrm{d}}_i - t^{\mathrm{r}}_{i'})(1 - w^{\mathrm{M}}_{ii'}) &\leq y_{i'} \\ &\leq y_i + e_i + \sum_{i'' \in \mathcal{I}^{\mathrm{m}}_{m''3h},\ m'' \in \mathcal{M} : m < m'' < m'} e_{i''} w_{mm''} + (t^{\mathrm{d}}_{i'} - t^{\mathrm{r}}_i)(1 - w_{mm'}), \\ &\qquad (i,i') \in \mathcal{II}^{\mathrm{m}}_{mm'3h},\ h \in \mathcal{H}^{\mathrm{CM}}, m' \in \mathcal{M}, m \in \mathcal{M} : m < m',\end{aligned} \tag{C.13}$$

$$y_{i'} + e_{i'} + e^{\mathrm{init}}_{3h}(1 - w_{mm'}) - (t^{\mathrm{d}}_{i'} + e^{\mathrm{init}}_{3h} - t^{\mathrm{r}}_i) w^{\mathrm{M}}_{ii'} \leq y_i, \quad (i,i') \in \mathcal{II}^{\mathrm{m}}_{mm'3h}, h \in \mathcal{H}^{\mathrm{CM}}, m' \in \mathcal{M}, m \in \mathcal{M} : m < m', \tag{C.14}$$

$$w^{\mathrm{S}}_{mm'} = w^{\mathrm{M}}_{ii'}, \quad (i,i') \in \mathcal{II}^{\mathrm{m}}_{mm'3h}, h \in \mathcal{H}^{\mathrm{CM}}, m' \in \mathcal{M}, m \in \mathcal{M} : m < m'. \tag{C.15}$$

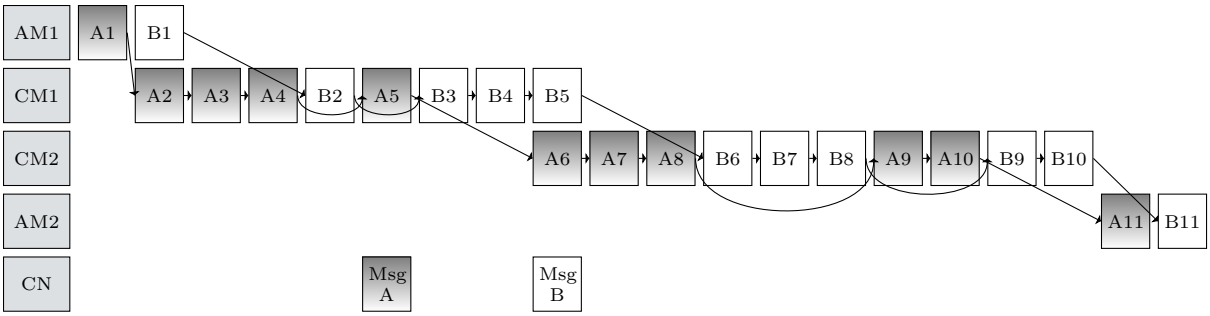

Figure D.5: An overview of the tasks, dependencies, and messages of the two communication chains A and B that send data between the same pair of AMs but in different CN slots

Appendix D. Illustrations of a problem instance and parts of a solution

To provide an illustration of the structure of our problem instances and solutions, we present an example of a so-called communication chain and provide a part of a feasible schedule. Because of the size and complexity of the problem instances, we cannot present a full instance nor a full schedule in the paper. However, further details about the instances and how they are generated can be found at `https://gitlab.liu.se/eliro15/avionics_inst` and in Karlsson et al. (2021).

A majority of the tasks in a problem instance originate from passing data between different parts of the system. A sequence of tasks that handle a specific set of data is referred to as a communication chain. These chains are designed to make the schedule comply with the system requirements dictated by communication protocols and design decisions made by the engineers. Most of these chains handle communication between software processes at different AMs, but there are also similar chains for the communication between a software process at an AM and a peripheral connected to a CM.

In essence, passing data between two software processes at different AMs via the CN requires that 11 tasks on 4 different modules are performed in a specific order, enforced by dependencies between the tasks. One of these 11 tasks is responsible for sending the CN message in a CN slot. For our illustration, we introduce the two communication chains A and B along with their tasks, messages and dependencies. Communication chain A includes task A1 on AM1, tasks A2–A5 on CM1, message A on the CN, tasks A6–A10 on CM2 and taskA11 on AM2. Similarly, communication chain B includes task B1 on AM1, tasks B2–B5 on CM1, message B on the CN, tasks B6–B10 on CM2 and finally taskB11 on AM2. In addition to the dependencies within a chain, there is also a requirement that the tasks that deques the data from the CN are scheduled in the same order as their messages were sent in.

The two communication chains A and B are illustrated in Figures D.5 and D.6. In these figures, the illustrated size of a task is not proportional to its execution requirement. In Figure D.5, the two communication chains send data in different CN slots and there is no interaction between their tasks except that they cannot overlap and data must be dequeued in the order it is sent. Since the message for communication chain A is sent before that of chain B, task A6 has to be scheduled before task B6, but except from that, there is no order enforced between tasks from different chains. In Figure D.6, the messages of the two communication chains are sent in the same slot, referred to as co-allocation of messages. This means that some pairs of their tasks must be merged and scheduled together, with a total execution requirement that is shorter than if they were not merged. In Figure D.6, co-allocation of messages A and B forces a merge of tasks A4 and B4, tasks A5 and B5, tasks A6 and B6, and tasks A7 and B7.

Our illustrations above contain only 22 tasks while, for example, in our instance set D (with the largest instances) the average number of tasks is 41,655, making them and their solutions difficult to illustrate. In Figure D.7, we instead provide 1/32 of a schedule for the rather small instance B21, and this part of the schedule contains a total of 403 tasks. In this illustration, the lengths of the tasks are proportional to their execution requirements, but the dependencies between them are not illustrated.

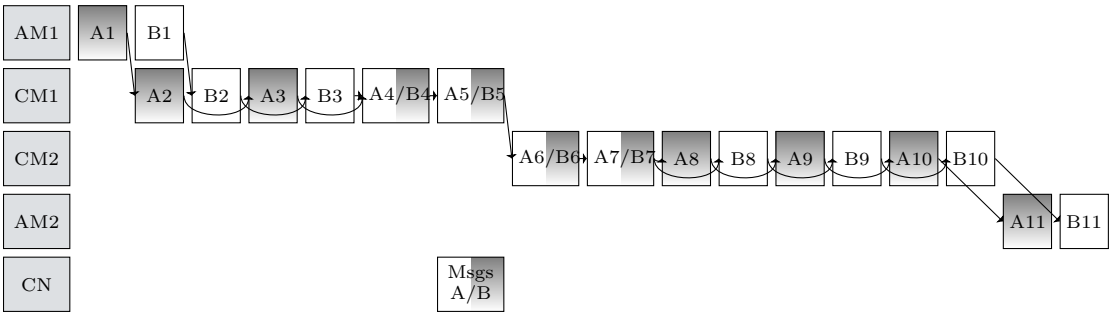

Figure D.6: An overview of the tasks, dependencies, and messages of the two communication chains A and B that send data between the same pair of AMs and in the same CN slot

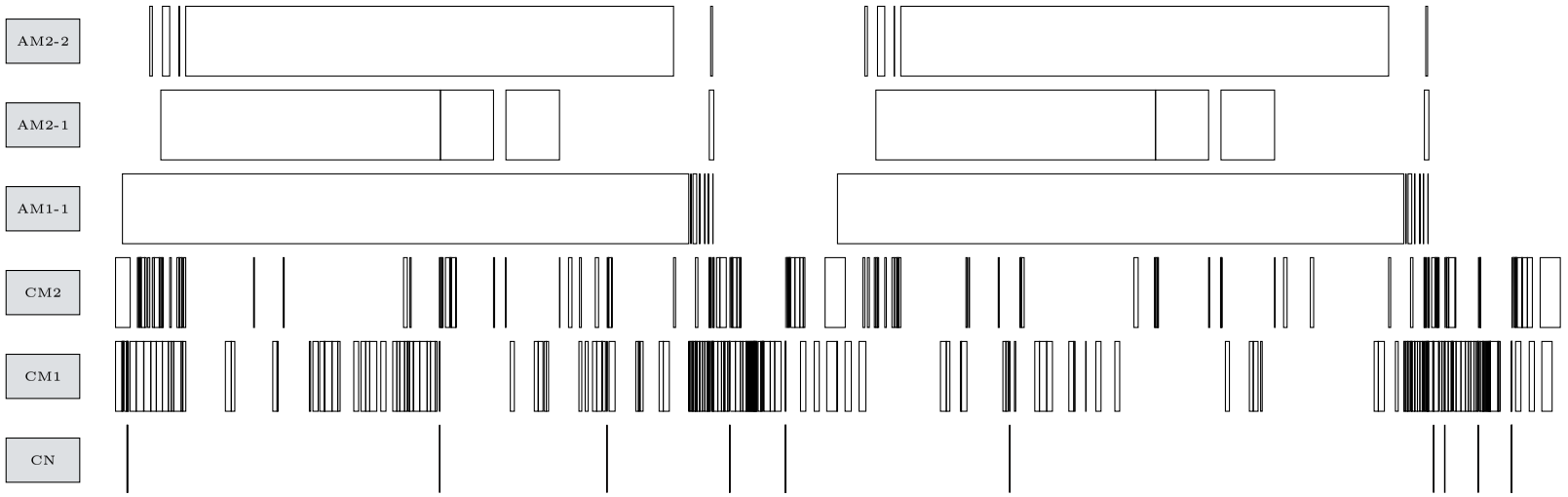

Figure D.7: An illustration of 1/32 of a feasible schedule for InstanceB21