Scalable Scheduling Algorithms for Embedded Systems with Real-Time Requirements

by

Anna Minaeva

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Control Engineering

Doctoral Thesis

Supervisors: Zdeněk Hanzálek and Benny Åkesson.
Ph.D. programme: Electrical Engineering and Information Technology
Branch of study: Control Engineering and Robotics
Submission date: February 2019

Anna Minaeva: Scalable Scheduling Algorithms for Embedded Systems with Real-Time Requirements, Ph.D. Thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Control Engineering, February 2019, Prague.

Acknowledgments

This four years long adventure was full of people who taught me invaluable lessons, and without whom this work would not exist. First of all these people are my supervisors, Zdeněk Hanzálek and Benny Åkesson, who always believed in me and supported me, sharing both science and life experience. Their wisdom and thorough approach to work and science keep inspiring me to be a better person in both personal and professional life. Thanks to Zdeněk, among other things, I have learned to approach (not only) mathematics formalism more consistently and do my best while maintaining a positive attitude. Benny is an example of endless rationality perfectly combined with friendliness and desire to share. It is impossible to overestimate his part in the quality of the articles we have written together and my writing and reasoning skills after working with him. I am deeply grateful to Benny for his patience with my mistakes and being Zen along the way to improve our articles and this thesis. My sincere gratitude goes to Přemysl Šůcha, who has introduced me to operational research and who is always here to help, support, and provide valuable advice. I am also grateful to my colleagues and friends: Pavel Piša, Michal Sojka, Rostislav Lisový, Honza Dvořák, István Módos, Antonín Novák, Libor Bukata, Roman Václavík, Přemysl Houdek, Joel Matějka, Flavio Kreiliger, Ondřej Benedikt, Marek Vlk, Aasem Ahmad, Jiří Vlasák, and others for sharing their thoughts and ideas, for having fun together and for being supportive and ready to help. Another essential component that made this work possible are people who responsibly took care of the necessary administrative work: Svatava Petrachová, Jaroslava Nováková, Petra Stehlíková, Jaroslava Matějková, Lenka Jelínková, Jana Rentková, Pavla Svítilová, Zuzana Hochmeisterová, Kateřina Sittová, Marie Duchoslavová, and others. I would also like to thank Czech Republic, Prague, Czech Technical University in Prague, Control Department of Faculty of Electrical Engineering and Czech Institute of Informatics, Robotics and Cybernetics for making this work not only possible but also highly enjoyable. I am especially thankful to Ondřej Benedikt and Mikhail Sukhotin for a very thorough proofreading of some part of the thesis and Annemette Scheltema and Kim Peterse for the wonderful cover page to this thesis.

Extending horizons by discovering other research groups, companies, and cultures is an important and nice part of a Ph.D. study. I am grateful to Dakshina Dasari, Robert Bosch GmBH for her constructive feedback and sharing her experience while publishing the joint article. I would also like to thank Samarjit Chakraborty, head of Munich Technical University Chair of Real-Time Computer Systems for a warm welcome to his group and for the opportunity to share experience and have productive discussions with himself, Debayan Roy, Licong Zhang, Nadja Peters, Michael Balszun, Alma Pröbstl, Sangyoung Park, Daniel Yunge, and others. Furthermore, although work with Petra Jelínková and Martin Kršňák from Asseco Solutions was not directly related to this thesis, it provided me with a basic view of the functioning of industrial companies and how productive and nicely people can be managed. Moreover, the HiPEAC project is a great initiative that supported me financially to visit the group of Samarjit and the summer schools on advanced computer architecture and compilation for high-performance and embedded systems.

We do not only teach our students and people we manage but also learn from them, by having conversations and sharing ideas that sometimes inspire us to realize them. Martin Hoffman, Denis Korekov, Marek Pytela, Daniel Slunečko, Richard Hladík are hardworking and gifted students that I had a pleasure to work with and to learn from them. Also, I am grateful to students of Combinatorial Optimization course at Czech Technical University in Prague that I taught as a lab teacher for being hungry for knowledge, creative, and hardworking.

Friends are people who make bad moments better and happy moments even happier. Thank you, my inspiring and supporting friends: Valeria Iushkova, Julia Vinnichenko, Marina Kretinina, Natalie Faryna, Natalie Dragunchik, Silvie Dittrich, Stanislav Baganov, Adéla Poubová, Elnaz Babayeva, Dina Deeva, Mette Scheltema, Juan Valencia, Inga Makaryan, and many others for all the funny moments together. For the half-marathon, the personal growth training, common life changes, profoundly changing conversations, time together and all the beauty and love you share and bring to my life. Without you, this way would not be as productive and enjoyable as it was. Thank you, girls from wITches group for accepting, supporting, being enthusiastic, and genuinely devoted to the thing we do and doing it with all our hearts. Thank you for all the fun together and for showing that age difference is not an obstacle for it!

To open opportunities for further growth, a person needs to learn new things constantly. Thank you, Olesea Mostipac and Anastasia Gräber, for introducing and teaching me wonderful yoga. Next, I am grateful to Instituto Cervantes in Prague and all the teachers for not only properly teaching Spanish language and culture in a very educative way, but also for spreading healthy attitude to ecology, society, and relations. Finally, thank you Olesea Lukashova, Galina Pechenkina, Nikolay Korzhik, Elena Osinceva, Valeria Iushkova, Anastasia Ladan and others for all the changes in my personal and professional life after the landmark forum, a personal growth training.

This work would not be possible without all the love and support of my family: my mom Svetlana, dad Vladislav, brother Anton, grandparents Larisa, Nikolai, Galina, and Viktor, who is here for me not only during these four years but also during my entire life. Finally, I cannot express in words how blessed I am to have such a loving and supporting husband Misha, who has experienced how annoying I can be and who is still able to see good in me. He is the best gift I have ever been given together with Nobi and Umka, our dog and cat.

I would also like to thank Eaton Corporation for financially supporting my Ph.D. study and Pavel Kučera for supervising me as a consultant from the company. This work was partially supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 688860 (HERCULES).

Anna Minaeva
Prague, February 2019

Declaration

This doctoral thesis is submitted in partial fulfillment of the requirements for the degree of doctor (Ph.D.). The work submitted in this thesis is the result of my own investigation, except where otherwise stated. I declare that I worked out this thesis independently and I quoted all used sources of information in accord with Methodical instructions about ethical principles for writing academic thesis. Moreover, I declare that it has not already been accepted for any degree and is also not being concurrently submitted for any other degree.

Anna Minaeva
Prague, February 2019

Abstract

Growing user demands result in an increasing number of applications integrated into embedded systems. These applications can have real-time requirements, which means the utility of computations is sensitive to their timing behavior. To reduce the cost, manufacturers minimize the number of platform components. As a result, applications share platform resources, which causes contention and worsens their timing behavior. Applications can be scheduled on platform resources at design time to guarantee that real-time requirements are satisfied. This scheduling problem is challenging as there are exponentially many options on how to construct a schedule that satisfies real-time requirements and optimizes system performance. During design-space exploration, the system designer needs to solve the scheduling problem many times. Therefore, the computation time of the solution approach significantly influences system development time and its cost, with the latter also depending on the system performance. Thus, a solution providing reasonable computation time and quality trade-off needs to be found. Most of the existing works either propose exact solutions that cannot solve industrial-sized instances or propose heuristic algorithms without validating its efficiency with optimal solutions.

In this thesis, we address this problem through a three-stage approach, corresponding to three problems with gradually increasing complexity and accuracy of the model. The four main contributions of this thesis are: 1) We explore and quantitatively compare three formalisms to solve the problems optimally, Integer Linear Programming (ILP), Satisfiability Modulo Theory, and Constraint Programming, and propose computation time improvements. To increase the scalability of the ILP approach, we introduce a particularly interesting optimal approach that wraps the ILP in the branch-and-price framework. 2) For each problem, we present a scalable and efficient heuristic algorithm that decomposes the problem to decrease its computation time. Each heuristic conceptually improves the heuristic from the previous stage. 3) We quantitatively and qualitatively compare the efficiency of the optimal and heuristic strategies. The results show that the heuristic algorithms can solve on average 8 times larger instances than the optimal approaches. Furthermore, the heuristic algorithms sacrifice up to 2% of solution quality on average, finding a feasible solution for more than 77% of problem instances up to 6 times faster. 4) We demonstrate the practical applicability of the proposed heuristic algorithms and optimal approaches on case studies of real systems in both the automotive and consumer electronics domains. For a case study of an Engine Management System with more than 10 000 tasks and messages, our heuristic algorithm finds a solution in less than an hour.

Keywords: embedded systems, optimization, real-time systems, resource scheduling.

Abstrakt

Rostoucí požadavky uživatelů vedou k zvyšujícímu se počtu aplikací integrovaných do vestavěných systémů. Tyto aplikace mohou mít požadavky na provoz v reálném čase, které ovlivňuji užitečnost výpočtů po deadlinu. Aby se snížili náklady, výrobci minimalizují počet součástek systému. Výsledkem je, že aplikace sdílejí jednotlivé komponenty, což způsobuje kolize a zhoršuje jejich chování v čase. Běh aplikaci může být rozvržený na komponentech systému jíž v rámci návrhu, aby bylo zaručeno splnění požadavků na běh v reálném čase. Takový rozvrhovácí problém je obtížný, protože existuje exponenciální počet možností jak sestavit rozvrh, který splňuje požadavky na provoz v reálném čase a optimalizuje výkon systému. Ve fázi optimalizace nastavení parametru systémů, návrhář řeší tento rozvrhovácí problém opakovaně. Nutný výpočetní čas pak významně ovlivňuje čas vývoje systému a náklady s tím spojené, které rovněž závisí na výkonu systému. Proto musí být nalezeno řešení poskytující přiměřený kompromis potřebného výpočetního času a výsledného výkonu. Většina stávajících prací navrhuje bud'to optimální řešení, která nedokážou vyřešit instance průmyslové velikosti, a nebo heuristické algoritmy, u kterých ale často chybí srovnání s optimálními řešení.

V této práci řešíme popsaný rozvrhovácí problém třífázovým přístupem, který odpovídá třem problémům s postupně vzrůstající složitosti a přesností modelu. Čtyři hlavní přínosy této práce jsou následující: 1) Zkoumáme a kvantitativně porovnáváme tři formalizmy pro optimální řešení problémů, Celočíselné Lineární Programování (ILP), Splnitelnosti formulí v teoriích predikátové logiky (SAT) a Programování s omezujícími podmínkami (CP) a navíc navrhujeme několik zlepšení výkonnosti modelů. Abychom zvýšili škálovatelnost přístupu ILP, navrhujeme obzvláště zajímavý optimální přístup branch-and-price, který využívá ILP model. 2) Pro každý problém předkládáme škálovatelný a výkonný heuristický algoritmus, který dekomponuje problém na menší dílčí podproblémy, čímž snižuje potřebný výpočetní čas. Přitom, každá heuristika koncepčně zlepšuje heuristiku z předchozí fáze. 3) Kvantitativně a kvalitativně porovnáváme efektivitu optimálních a heuristických strategií. Výsledky ukazují, že heuristické algoritmy mohou řešit v průměru 8krát větší instanci problému ve srovnání s optimálními přístupy. Kromě toho, heuristické algoritmy ztrácí v průměru nejvýše 2% kvality ve srovnání s optimálními přístupy, přičemž jsou schopné najít přípustné řešení pro více než 77% testovacích instanci až 6 krát rychleji. 4) Ukazujeme praktickou použitelnost navržených heuristických algoritmů a optimálních přístupů na případových studiích reálných systémů jak v oblasti automobilového průmyslu, tak i spotřební elektroniky. Pro případovou studii systému řízení motoru s více než 10 000 úloh a zpráv náš heuristický algoritmus najde řešení za méně než hodinu.

Klíčová slova: optimalizace, rozvrhování zdrojů, systémy reálního času, vestavěné systémy.

Goals and Objectives

The thesis addresses optimization algorithms for time-triggered scheduling in real-time embedded systems. The four main goals are:

1. Study the existing literature for the time-triggered scheduling of real-time embedded systems and determine its weak points in terms of terminology, problem statement, and used approaches.
2. Devise mathematical models, reflecting the most important constraints and criteria of the time-triggered scheduling of embedded real-time systems problem both in safety-critical and in non-safety-critical domains.
3. Propose heuristic and exact algorithms that use problem-specific knowledge to increase the efficiency to solve industrial-sized problems of the time-triggered scheduling of embedded real-time systems.
4. Verify the proposed algorithms on benchmark instances and compare the quality of the obtained solutions of heuristic algorithms with the solutions obtained by the exact approaches. Show applicability of the proposed approaches on realistic use-cases.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

3-LS 3-Level Scheduling . . . 106
ABS anti-lock braking system . . . 4
ADAS advanced driver-assistance system . . . 73
B&P branch-and-price . . . 13
CP Constraint Programming . . . 107
DAG Directed Acyclic Graph . . . 83
ECU Electronic Control Unit . . . 76
EMS Engine Management System . . . 111
ET Event-Triggered . . . 75
ILP Integer Linear Programming . . . 107
JC Jitter-Constrained . . . 80
JSS job shop scheduling . . . 86
LR Latency-Rate . . . 49
lcm least common multiple . . . 19
NZJ non-zero-jitter . . . 80
PMSP Periodic Maintenance Scheduling Problem . . . 19
SMT Satisfiability Modulo Theory . . . 107
TCP/LR-F TDM configuration problem/latency-rate with given frame size . . . 18
TDM Time-Division Multiplexing . . . 13
TT Time-Triggered . . . 74
WCET Worst-Case Execution Time . . . 41
ZJ zero-jitter . . . 75

Chapter 1

Introduction

Nowadays, technology develops extremely fast, which can be seen in automotive and consumer electronics domains. For automotive systems, the advanced driver-assistance system (ADAS) went all the way from lane-keeping assistance in the early 2000s to self-driving cars providing taxi services in different parts of the world. These taxis can be found, e.g., in Singapore, USA (Pittsburgh, San Francisco, Tempe, Phoenix), Russia (Skolkovo, Moscow), and Japan (Tokyo). Although these cars typically operate in limited areas with a predictable environment and lower traffic volumes, they still need to accomplish a myriad of non-trivial tasks previously carried out by a human being. The next step in the ADAS development trajectory is fully autonomous cars, safely operating in all environments, which are expected to be introduced to the mass market during the next decade [113]. For consumer electronics systems, high development speed is seen on smartphones. While the first smartphone developed 25 years ago had a minimal range of applications that mainly focused on calling and receiving faxes and emails [28], modern smartphones is a single device that we use for diverse everyday needs, including taking photos and videos, listening to the music, and using it as a navigation tool. In other words, a smartphone integrates multiple functionalities that were previously parts of different systems. Both automotive components and mobile phones are examples of embedded systems, defined as systems built for particular purposes. An embedded system is realized by one or more applications that can have real-time requirements. For real-time applications, correctness depends not only on the result but also on the time the result is obtained.

To maintain the high development speed, it is crucial to integrate multiple applications, sharing platform resources efficiently. However, sharing causes contention that needs to be resolved by resource scheduling to satisfy real-time requirements of the applications. In the rest of this section, we first present trends in embedded real-time systems scheduling. Then, the problem considered in this thesis is stated, and the overview of the solution approach is given. Next, we list key contributions, before we conclude by the outline of the thesis.

1.1 Trends in Scheduling of Embedded Real-Time Systems

This section discusses the necessary aspects of scheduling in embedded real-time systems and observed trends to create a better understanding of the problem and contributions of this thesis. We start this section by discussing applications and their characteristics, followed by an introduction to trends in real-time systems scheduling. Finally, we present the closest state-of-the-art works to provide an overview of the existing research in the domain of time-triggered scheduling of real-time systems.

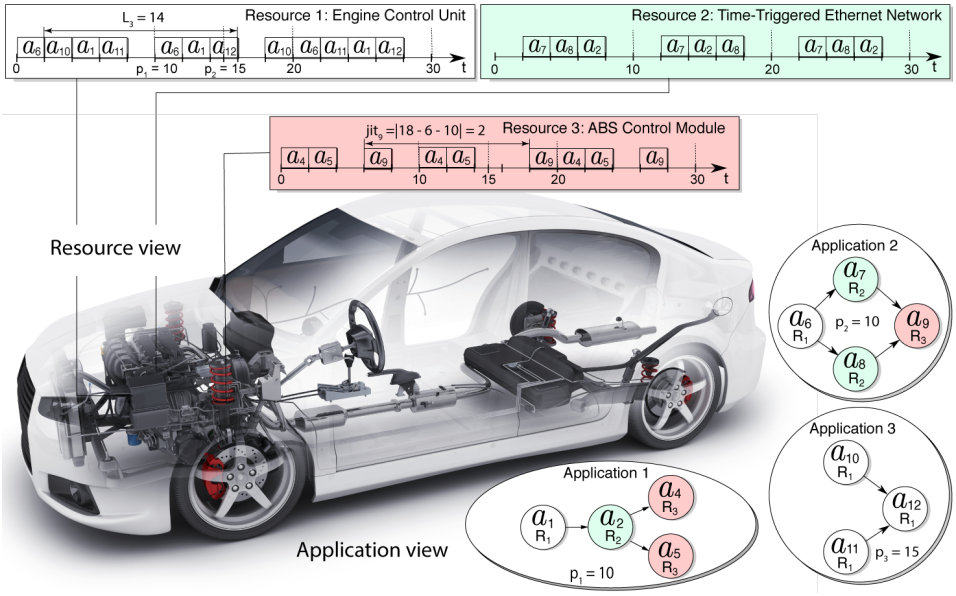


Figure 1.1: Example of an application set in a car with three periodic applications with periods $p_1 = 10$, $p_2 = 10$, and $p_3 = 15$. The applications run on an Engine Control Unit and an ABS control module and communicate via a switched time-triggered Ethernet network.

1.1.1 Applications

The functionality of an embedded system is realized with a single or multiple applications. An *application* is a program running on platform resources that performs a well-defined function for the user, such as playing video content in a smartphone or managing the engine in a car. An application comprises one or more *tasks* that may have data dependencies and may communicate by *messages*. Each task executes on a platform component, where it is assigned. Thus, tasks are statically assigned to resources and cannot migrate. Tasks can be sensing, actuation, or computation. For example, tasks in the engine management system can be sensing engine oil pressure and fuel pressure or computing and setting how much idling air and fuel to inject. Figure 1.1 shows an example of an application set with 3 periodic applications running on an Electronic Control Unit (ECU) and an anti-lock braking system (ABS) control module that communicate via a switched time-triggered Ethernet network. The applications comprise 5, 4, and 3 activities (tasks and messages), respectively. The tasks here can be sensing wheel rotation speed and sending it over the network to the motor to compute the car speed that can be used by a cruise-control application.

Based on the criticality of timing requirements, real-time applications can have *hard* or *soft real-time requirements* [23]. Applications with hard real-time requirements may be safety-critical applications, e.g., in avionics, automotive,

healthcare, and space domains. Here, a missed deadline may result in massive material losses and deaths. An example of a hard real-time application is the aforementioned cruise-control application, where the vehicle speed is read by a speed sensor, processed, and the throttle position is adjusted to maintain the required speed. This needs to be executed with a specific frequency, i.e., periodically to guarantee the safety of the system. On the other hand, for applications with soft real-time requirements, timing requirements must be respected to ensure the correct behavior of the system. However, there is more flexibility for scheduling soft-real applications, since it is feasible to fail in providing some timing requirements at the cost of system performance or perceived quality. We find soft real-time applications in the consumer electronics domain. The example is audio devices, where failure to play some samples may result in quality degradation.

1.1.2 Scheduling

There exist two basic paradigms to schedule resources in real-time systems: Event-Triggered (ET) [31] and Time-Triggered (TT) [64] scheduling. In the ET approaches, also called online or dynamic approaches, scheduling is performed at run-time and triggered by events. In contrast, the schedule is computed offline and repeated during execution in the TT approach, also called offline or static approach. The main advantage of the ET approaches over the TT approach is their flexibility in two aspects. Firstly, they are often able to adapt to the actual demand without a redesign of the system [8], which saves both time and money. Secondly, they are work-conserving in the sense that the schedule is adjusted to the actual execution time of the application, unlike the worst-case execution time used in the TT approach [63]. Considering the worst-case execution time results in a higher system utilization, increasing the system cost. However, it is often easier to provide evidence of how the system will execute with the TT approach since the schedule primarily determines this. This may result in a much shorter and less expensive process of certification for safety-critical systems [14]. Another disadvantage of the ET over TT approach is the difficulty to guarantee particular real-time requirements of applications, such as jitter or latency. For these reasons, the TT approach is commonly used for safety-critical applications.

Time-Division Multiplexing (TDM) is a resource arbiter that, similarly to the TT approach, works with a schedule of a given length, repeating it periodically. However, TDM is typically implemented on platform resources, such as Networks on Chips and memory controllers, rather than in software. Since it is implemented in hardware, it has to be simple. It hence only repeats a periodic schedule, driven by an on-chip clock. On the other hand, the TT approach requires nodes on different distributed devices to have a global notion of time, which needs to be synchronized and introduces extra cost. In this thesis, we use the TT and TDM approaches only due to the applied domain.

Scheduling Objectives and Constraints

Multiple factors can influence system performance, including left-over bandwidth for non-real-time applications and the control performance of the applications. The primary scheduling metrics of an application considered in this work are its

end-to-end latency and bandwidth, which are the time to run the application from start to end and share of the resource allocated to the application, respectively. Considering the end-to-end latency of an application as a scheduling metric results in precedence relations, caused by data dependencies between *activities*, i.e., tasks executing on processing elements and messages transmitting over the network. Furthermore, to reduce the complexity, the system designer derives various activity-level constraints, such as periods or deadlines to satisfy the latency and bandwidth requirements of the applications. At the level of single resources, these requirements make timing easier to manage. Finally, the platform can dictate some scheduling requirements of activities, such as non-preemptiveness that makes it impossible for one activity to interrupt the execution of another. For instance, messages transmitting over the TTEthernet [103] network links cannot be preempted according to the communication standard. Another source of activity-level constraints is the application domain, which may put limitations on, e.g., a jitter of periodic activities in control applications, which is the deviation from the true periodicity, to guarantee sufficient control performance.

In Figure 1.1, for Application 3 with all activities scheduled on Resource 1, the end-to-end latency in the presented schedule is 14. Note that due to the periodicity of the activities, the precedence relations can be satisfied by activity occurrences in different periods as it holds for activities a_4 and a_5 of Application 1. The earliest activity a_{10} starts at 2, and the latest activity finishes at 16 in the first period and 18 and 28, respectively, in the second period. Thus, the end-to-end latency of this application is the maximum of the end-to-end latencies in the first and the second periods, i.e., $L_3 = \max(16 - 2, 28 - 18) = 14$. Furthermore, we compute the jitter of a_9 executing on Resource 3 with period 10 as a difference of start times in the consequent periods relative to the period, i.e., $jit_9 = \max(|18 - 6 - 10|, |26 - 18 - 10|) = \max(2, 2) = 2$, since it is executed at times 6, 18, and 26 in periods 1, 2, and 3. Finally, we compute bandwidth of a_9 as its execution time divided by its period, i.e. $\rho_9 = \frac{e_9}{p_9} = \frac{1}{10} = 0.1$.

1.1.3 State-of-the-Art

This section presents an overview of the existing works in the domain of embedded real-time systems time-triggered scheduling and provides background, necessary to solve hard optimization problems. To cope with the increasing size and complexity of resource scheduling problems for embedded real-time systems, we need to carefully design the approach to deal with the resource scheduling problems. Since these problems are NP-complete, optimal approaches (such as ILP, Constraint Programming (CP), or Satisfiability Modulo Theory (SMT)) can typically solve small- to medium-sized problems only. Therefore, heuristic approaches are applied to solve complex industrial-sized problems.

Algorithms can consist of *constructive* and *generative* components. Purely constructive algorithms build the solution step-by-step and partial solutions are always feasible. In contrast, generative algorithms work with complete, but possibly infeasible schedules, gradually breaking fewer constraints and/or becoming of better quality. Purely generative algorithms pay off when it is easy to find a feasible solution, and the primary challenge lies in finding a better solution. However, in

the considered problem of embedded real-time systems scheduling, the utilization of platform resources is typically high, making it hard to find a feasible solution. Thus, the most efficient algorithms for scheduling in real-time embedded systems contain a constructive component.

Many works propose a combination of constructive and generative components for solving the problem of resource scheduling. These approaches make use of the following three techniques, often combining them: 1) decomposition of the problem into smaller sub-problems [29, 104], 2) conflict-refining [71], first making schedules with possible conflicts and incrementally fix them afterward, or 3) meta-heuristics, such as Tabu search [57], genetic algorithms [78], or simulated annealing [32]. We will proceed by discussing each of these techniques in turn.

Problem decomposition is an inevitable element of an efficient approach. Steiner in [104] decomposes the problem into subsets of activities, using an incremental backtracking approach based on an SMT solver. However, Craciunas and Oliver in [29] observed that the approach of Steiner shows good results only when the utilization of the resources is low, which is typically not the case. To address this issue, they decompose the problem into subsets of more and less difficult activities, respectively, also using an SMT solver to find the schedule for the former and a simple heuristic approach for the latter. If a valid schedule cannot be found, the set solved by SMT is enlarged. Moreover, authors in [95] apply Dantzig–Wolfe decomposition [30] to transform the solution space from individual time slots to partial schedules. A branch-and-price (B&P) algorithm [35] then goes through the search space considering only a subset of all possible partial schedules that contains the most promising schedules regarding criterion value.

The second technique, conflict refinement, is used by Lukasiewycz and Chakraborty in [71]. Here, the authors look sequentially at problems of increasing sizes using an ILP solver, preferring to find and refine smaller conflicts over larger ones, since resolving smaller conflicts is typically easier. Finally, following the third technique, the authors in [57] use Tabu search, combining it with extended list scheduling to determine the schedule for the applications.

While decomposition or conflict-refining approaches have limited scalability due to the typical usage of exact solvers, the efficiency of meta-heuristics is sensitive to many running parameters required to be set. On the other hand, purely constructive approaches do not need the usage of exact solvers, and their behavior does not depend on many parameters. Chen et al. in [26] and Kermia in [60] apply constructive approaches, proposing different schedulability tests that define the scheduling order of activities. However, the problems solved in these works do not include many important aspects of modern embedded real-time systems, such as data dependencies and end-to-end latency requirements.

The branch-and-bound method is an example of a constructive approach that searches the solution space efficiently. The method works with a solution tree, where nodes are partial solutions. In each child of a node, we set a decision variable(s) to a specific value or a set of values, both extending the partial solution and cutting the search space. Instead of searching in the entire solution space, branch-and-bound reduces it by closing nodes that are of provably poor quality. This approach is used by Syed and Fohler in [109], where the authors apply refined pruning techniques and symmetry avoidance, resulting in the scalability of the approach to industrial-sized

problems.

1.2 General Problem Statement

This thesis introduces automated methodologies to find TT schedules for applications running on shared resources with different types of real-time requirements, such as bandwidth, latency, and jitter while optimizing the performance of the system. We aim for the methodologies that find schedules for industrial-sized embedded real-time systems within a reasonable time to avoid negatively impacting design time. On the one hand, for one-off computation, it is possible to wait for an hour or slightly more for a schedule. On the other hand, design-space exploration may mean making hundreds or thousands of schedules for different platform configurations, and a reasonable time to wait for the result is maximally a couple of minutes. Next, we present an overview of the approach proposed in this thesis.

1.2.1 Overview of Approach

Our approach addresses the problem stated above with the following three stages, corresponding to three NP-hard problems to capture different aspects of the bigger problem shown in Figure 1.2a.

First, we address the *problem of scheduling a single resource shared by real-time and non-real-time applications* in the consumer electronics domain (Chapter 2). Such shared resources can be processing elements, memories, interconnects, and peripherals. We aim to satisfy the bandwidth and latency requirements of the real-time applications while minimizing their resource utilization to improve the performance of their non-real-time counterparts. Here, we consider low-level timing guarantees of hardware resources.

The *second stage* of our approach considers *coscheduling of periodic control applications*, comprising activities, including tasks such as sensing, computation, and actuation executed on processing elements, and messages transmitted over the network (Chapter 3). A message is introduced for each pair of data-dependent tasks running on different resources. Data dependencies between activities require scheduling of all resources at once to avoid unacceptably long delays caused by independent scheduling of resources. Here, we provide guarantees on periodicity, latency, and jitter, while scheduling non-preemptively. In this problem, we look more at requirements dictated by applications, although still looking on timings of tasks and messages.

Finally, the *third stage* looks at the *application and system requirements of the coscheduling problem* described previously (Chapter 4). In this step, we consider the control performance of the applications. Namely, we minimize the time that the corresponding control system takes to settle, being a function of its end-to-end latency. We believe that this three-stage approach is an appropriate way to address the problem since the complexity of the problem is handled gradually, while the problem formulation consequently captures the behavior of embedded real-time systems.

1.3 Contributions

The four key contributions of this thesis are the following:

1. We *explore and quantitatively compare different formalisms to solve the stated problems optimally*: we use ILP in all chapters, SMT in Chapters 3 and 4, and CP in Chapter 4, while applying different problem-specific computation time improvements. We formulate ILP as both time-indexed model [62] in Chapter 2 and relative-order model [20] in Chapters 3 and 4 to compare the efficiency. Moreover, to increase the scalability of the ILP approach in Chapter 2, we introduce a particularly interesting optimal approach that takes the ILP model and wraps it in the B&P framework.

2. In Chapters 2, 3 and 4, we present *scalable and efficient heuristic algorithms* for the three steps of the approach stated in Section 1.2.1. The heuristic presented in Chapter 2 is generative, while the other two chapters propose constructive heuristics.

3. We *quantitatively and qualitatively compare the efficiency of the optimal and heuristic strategies on different problems* in all three chapters. The efficiency considers the computation time of the strategy and quality of solution regarding either resource utilization or control performance of the applications.

4. We *demonstrate practical applicability of the heuristic algorithms* on case studies of real systems in both automotive and consumer electronics domains for all three problems.

An overview of the solution approaches used in each chapter is presented in Figure 1.2b, where the generative heuristic uses conflict refinement technique, while the constructive ones use problem decomposition. Moreover, an overview of the platform architecture considered in different chapters is shown in Figure 1.2c.

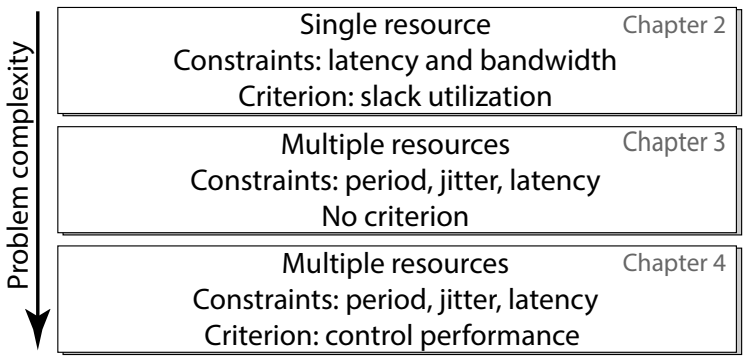
1.4 Outline

The rest of the thesis is organized as follows. Chapter 2 addresses the scheduling problem of applications that share a single resource. For this problem, an ILP model, an algorithm using the existing B&P approach, and a generative heuristic are proposed. Moreover, they are verified on synthetically generated problem instances and a case study of an HD video and graphics processing system. The proposed approaches are also compared to a state-of-the-art approach.

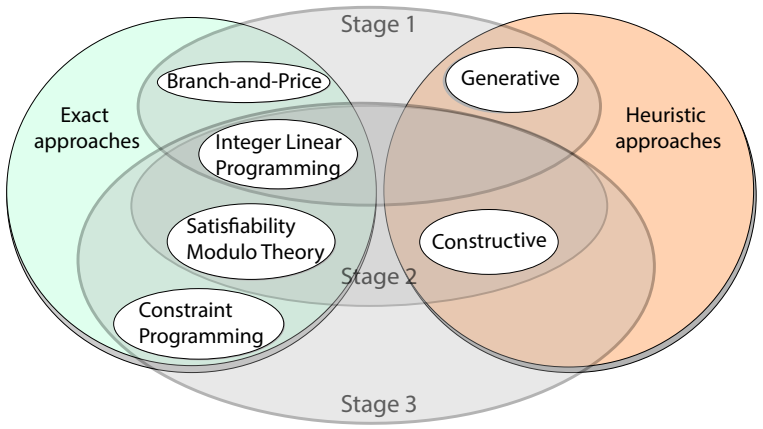
Chapter 3 proposes ILP and SMT models and a constructive heuristic approach to solve the coscheduling problem of periodic computation and communication. The efficiency of these approaches regarding computation time and maximum achievable resource utilization are compared on synthetic use-cases, provided by an industrial tool from the automotive domain [65]. Also, we demonstrate the efficiency and scalability of the approach on a case study of an Engine Management System with over 2000 tasks and 8000 messages.

Extending the coscheduling problem by an optimization criterion, assuming another platform model, generalizing end-to-end latency constraints, and fixing requirements on the jitter of activities, Chapter 4 presents ILP, CP, and SMT models to find an optimal solution as well as a constructive heuristic. The evaluation of the optimal and heuristic approaches is presented on the same problem instances as in Chapter 3, adapted to the new platform model and with realistic control performance values. Finally, the practical applicability of the presented approaches is shown on an automotive case study.

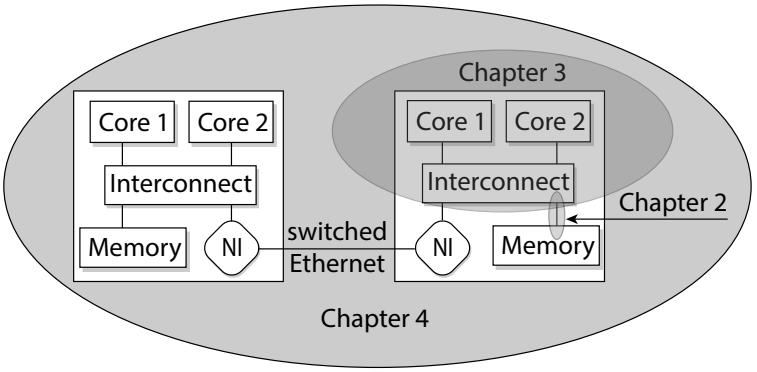
Chapters 2, 3 and 4 address different problems and each of them can be read separately as research papers. Moreover, you can find their associated nomenclatures in Appendices A, B, and C, respectively. We conclude with Chapter 5, where we summarize the achieved results, evaluate fulfillment of the goals, and discuss future work.



(a) Steps of the proposed approach



(b) Used exact and heuristic solution methods in the different chapters



(c) Platform architecture in the different chapters (NI is network interface)

Figure 1.2: Approach, solution, and architecture view of the thesis.

Chapter

Scheduling a Single Resource with Latency-Rate Abstraction

2

In this chapter, we consider the problem of scheduling of a single resource in consumer electronics systems, where a set of applications realizes the functionality of the system. The applications can have different types of requirements, as illustrated in Figure 2.1. Some of them (colored white in the figure) have *real-time requirements* and must always satisfy their deadlines, while other non-real-time applications (colored gray) only require sufficient average performance [115]. The cores and accelerators access shared resources, such as memories, interconnects and peripherals [61, 114] on behalf of the applications they execute and are referred to as resource *clients*. The contention, caused by sharing the resources is resolved by a Time-Division Multiplexing (TDM) arbiter. An important challenge with TDM arbitration in these systems is to find a schedule that assigns the time slots to the clients in a way that satisfies the *bandwidth and latency requirements* of the real-time clients while *minimizing their resource utilization* (maximizing slack capacity) to improve the performance of the non-real-time clients.

The contributions of this chapter are the following. As shown in Figure 1.2b of Chapter 1, we present an Integer Linear Programming (ILP) model to solve the TDM configuration problem optimally. Moreover, we propose another exact approach that wraps the ILP model in a branch-and-price (B&P) framework [35] to improve the scalability of the ILP model. The computation time of both the ILP and the B&P algorithm is optimized using problem-specific knowledge, including lazy constraints generation. Moreover, we present a stand-alone heuristic algorithm that can be used to solve the problem, providing a trade-off between computation time and efficiency. It is also used to reduce the computation time of the B&P algorithm. To demonstrate the scalability of the B&P approach and compare it both to the ILP model and an existing heuristic, we experimentally evaluate the approaches. We also quantify the trade-off between efficiency and computation time for the optimal and heuristic algorithms. Finally, we demonstrate the practical relevance of the approach by applying it to a case study of an HD video and graphics processing system. Also, the source code of our approach is released as open-source software and can be found in [76].

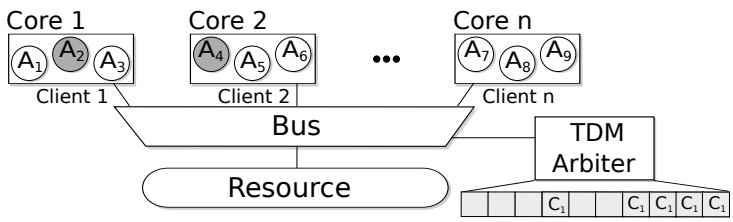


Figure 2.1: Example of a multi-core system, where n applications (A_l) with different real-time requirements are mapped to the cores. The cores act as resource clients (c_i) accessing a shared resource through an interconnect controlled by a TDM arbiter.

This chapter is organized as follows. Related work is discussed in Section 2.1. Section 2.2 proceeds by presenting background information necessary to understand the main contributions of the chapter. Then, the TDM configuration problem is formalized in Section 2.3, followed by a description of the proposed ILP formulation in Section 2.4. The B&P approach is introduced in Section 2.5 and its computation time optimizations are discussed in Section 2.6. The heuristic algorithm for slot assignment is then explained in Section 2.7. Section 2.8 presents the experimental evaluation before we summarize the chapter in Section 2.9.

2.1 Related Work

Scalability is a critical issue in system design, since design time must remain unchanged despite an exponential increase in system complexity. Most works in the area of design automation that use exact optimization techniques do not scale well enough to be able to manage the complexity of future consumer electronics systems [40, 47, 67, 72, 122], and only a few propose advanced techniques to address the complexity problem. These techniques can be classified into two major groups of approaches: 1) a decomposition of the problem into smaller sub-problems, and 2) navigating the search smartly during design-space exploration. The first approach deals with large problems by decomposing them into many smaller problems. This method is used in [69,119]. The second branch of improvements uses problem-specific information while searching the design space, which is more efficient compared to using general design-space exploration methods. Examples of this approach are shown in [91] and [71], where the authors look for a minimal reason for constraint violation and prevent this situation in the rest of the search, while using boolean satisfiability and ILP approaches, respectively.

Another way of dealing with the scalability issue is to use a heuristic approach. Some methodologies to configure TDM arbiters have been proposed in the context of off-chip and on-chip networks. An approach for synthesizing TDM schedules for TTEthernet with the goal of satisfying deadlines for time-triggered traffic, while minimizing the latency for rate-controlled traffic is proposed in [110]. Similarly, TDM scheduling for networks is also considered in [47], where a Profinet IO IRT message scheduling problem with different temporal constraints is solved, while minimizing the schedule length. The methodologies in [46, 70] consider slot assignment in contention-free TDM networks-on-chips. All of these approaches are heuristics and the efficiencies of the proposed methods have not been quantitatively compared to optimal solutions. Furthermore, the problem of scheduling networks is different from our research [77], as it considers *multiple resources* (network links) and is dependent on the problem of determining paths through the network.

The problem of TDM arbiter configuration with simplified client requirements is considered in [48], where unlike this work, the authors propose a harmonic scheduling strategy. One of the two previous solutions to the problem considered in this chapter is the configuration methodology for multi-channel memory controllers in [40]. The authors apply a commonly used heuristic for TDM slot assignment, called *continuous allocation* [37, 42, 44, 117], where slots allocated to a client appear consecutively in the schedule. The reasons for its popularity are simplicity of both

implementation and analysis and negligible computation time of the configuration algorithm. However, with growing problem sizes, this strategy results in significant over-allocation, making satisfaction of a given set of requirements difficult. This is experimentally shown in Section 2.8 when comparing to our approach.

Besides the difference in the problem formulation, this chapter advances the state-of-the-art by being the *first to apply a theoretically well-founded advanced optimization approach, called B&P [35] in the field of consumer electronics systems design.* B&P combines both of the mentioned approaches to manage complexity; it decomposes the problem into smaller sub-problems and uses more sophisticated search-space exploration methods. Although [95] applies B&P to the problem of FlexRay scheduling in the automotive domain, this chapter gives more elaborate explanation of the approach and concentrates on the computation time optimizations.

2.2 Background

This section presents relevant background information to understand the work in this chapter. First, we present the concept of latency-rate servers, which is an abstraction of the service provided to a client by a resource arbiter. We then proceed by discussing how TDM arbitration fits with this abstraction and explain how to derive its latency and rate parameters.

2.2.1 Latency-Rate Servers

Latency-rate (Latency-Rate (LR)) [107] servers is a shared resource abstraction that guarantees a client c_i sharing a resource a minimum allocated rate (bandwidth), ρ_i, after a maximum service latency (interference), Θ_i, as shown in Figure 2.2. The figure illustrates a client requesting service from a shared resource over time (upper solid red line) and the resource providing service (lower solid blue line). The LR service guarantee, the dashed line indicated as service bound in the figure, provides a lower bound on the amount of data that can be transferred to a client during any interval of time.

The LR service guarantee is conditional and only applies if the client produces enough requests to keep the server busy. This is captured by the concept of *busy periods*, which intuitively are periods in which a client requests at least as much service as it has been allocated (ρ_i) on average. This is illustrated in Figure 2.2, where the client is in a busy period when the requested service curve is above the dash-dotted reference line with slope ρ_i that we informally refer to as the *busy line.* We now have all the necessary ingredients to provide a formal definition of a LR server in Definition 1.

Definition 1 (LR server)**.** *A LR server provides guarantees on minimum provided service r_i^j to a client c_i requesting the service during a busy period with duration j. These guarantees are expressed by Equation* (2.1) *and are parametrized by service latency Θ_i and rate ρ_i. The minimum non-negative constant Θ_i satisfying Equation* (2.1) *is the service latency of the server.*

$$r_i^j \geq \max(0, \rho_i \cdot (j - \Theta_i)) \tag{2.1}$$

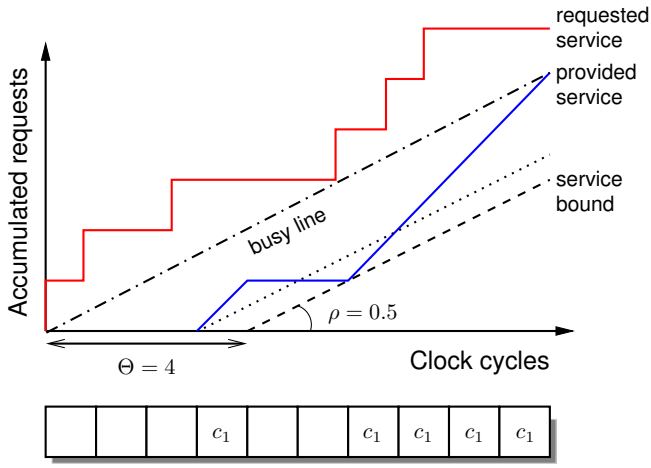


Figure 2.2: A latency-rate server and associated concepts for a client sharing a resource.

The values of Θ_i and ρ_i of each client depend on the particular choice of arbiter and how it is configured. Examples of arbiters that belong to the class of LR servers are TDM, several varieties of the Round-Robin and Fair-Queuing algorithms [107], as well as priority-based arbiters like Credit-Controlled Static-Priority [6] and Frame-based Static Priority [4]. The main benefit of the LR abstraction is that it enables performance analysis of systems with shared resources in a unified manner, irrespective of the chosen arbiter and configuration. It has been shown in [118] that the worst-case finishing time of the k^{th} request from a client c_i in a LR server can be bounded according to Equation (2.2), where sz_i^k is the size of the request in number of required slots, arr_i^k is the arrival time and fin_i^{k-1} is the worst-case finishing time of the previous request from the client. This bound called service bound is visualized for the k^{th} request in Figure 2.2. Note that this bound is slightly pessimistic, but only serves to illustrate how the LR abstraction is used to compute the finishing time of requests. For optimized bounds and a quantitative evaluation of the abstraction, refer to [100].

$$fin_i^k = \max(arr_i^k + \Theta_i, fin_i^{k-1}) + sz_i^k/\rho_i \tag{2.2}$$

Equation (2.2) forms the base for timing verification of applications at a higher level and does not make any assumptions on the applications by itself. Instead, restrictions on the application are imposed by the higher level analysis frameworks. For example, as shown in [92], it is possible to integrate Equation (2.2) into a worst-case execution time estimation tool to enable bounds on execution time of an application sharing resources to be computed. However, these tools are often limited to analyzing applications executing on a single core. This restriction does not apply to verification based on the data-flow model of computation [101], which can verify that distributed applications with data dependencies meet their real-time requirements, as demonstrated in [81]. This type of verification is particularly suitable for throughput-oriented streaming applications, such as audio and video encoders/decoders [19, 108, 117], and wireless radios [80, 117]. In this case, Equation (2.2) is integrated into the data-flow graph as a data-flow component with two

actors [118] before the analysis to capture the effects of resource sharing.

2.2.2 Time-Division Multiplexing

Having introduced LR servers as a general abstraction of shared resources, we proceed by showing how the abstraction applies to resources shared using TDM arbitration. We do this by first defining TDM arbitration and then show how the service latency and rate parameters of the corresponding LR server are derived.

A TDM arbiter operates by periodically repeating a schedule, or frame, with a fixed number of slots, H. The schedule comprises a number of slots, each corresponding to a single resource access with bounded execution time in clock cycles. Every client c_i is allocated a number of slots ϕ_i in the schedule at design time. The rate (bandwidth) allocated to a client, ρ_i, is determined purely by the number of allocated slots in the schedule and is computed according to Equation (2.3).

$$\rho_i = \phi_i / H \tag{2.3}$$

The service latency, on the other hand, depends on the *slot distribution* that determines where the allocated slots are located in the schedule.

Although it may intuitively seem like computing the service latency is just a matter of identifying the largest gap between slots allocated to the client in the schedule, this is not correct. The reason is that according to Definition 1, the rate ρ has to be *continuously* provided after the service latency Θ. The problem is illustrated in Figure 2.2, which shows a TDM schedule along with its corresponding service bound. As we can see, the service latency of $\Theta = 3$, which is the largest gap in the schedule, does not provide a conservative LR guarantee (it fails at time slot 6 in the TDM table) and the actual service latency of this schedule is $\Theta = 4$. This example shows us that the notion of service latency is more complex than it initially seems, making it harder to compute. This chapter uses a very general way to compute the service latency, Θ, of a TDM arbiter by directly using Definition 1. The rate of a given schedule is known by Equation (2.3) and the worst-case provided service to a client c_i during a busy period of any duration j (i.e. r_i^j) can be derived by analyzing the schedule (later shown in Section 2.4). For a given schedule, it is hence only a matter of finding the minimum service latency that satisfies the LR characterization in Equation (2.1).

In terms of implementation, we assume a scalable interconnect supporting TDM arbitration. A simple bus fails to scale as the number of clients are increasing, as the critical path gets longer and prevents it from synthesizing at high frequencies. Although TDM-based networks-on-chips address this scalability problem, they behave like multiple TDM resources (one per link), resulting in a different configuration problem. Instead, we consider a pipelined tree-shaped interconnect supporting a distributed implementation of TDM arbitration, such as the memory tree proposed in [41], which provides the required scalability yet behaves like a single resource.

2.3 Problem Formulation

The problem of finding a TDM slot allocation with a given frame size that satisfies the requirements of a *set of clients*, while minimizing the rate allocated to real-time clients is formulated in this section. We refer to this problem as *TDM Configuration Problem/Latency-Rate with given frame size* (TCP/LR-F).

An instance of the TDM configuration problem/latency-rate with given frame size (TCP/LR-F) problem is defined by a tuple of requirements $\langle C, \hat{\Theta}, \hat{\rho}, H \rangle$, where:

- $C = \{c_1, ..., c_n\}$ is the set of real-time clients that share a resource, where n is the number of clients.
- $\hat{\Theta} = [\hat{\Theta}_1, \hat{\Theta}_2, ..., \hat{\Theta}_n] \in \mathbb{R}^n_{\geq 0}$ and $\hat{\rho} = [\hat{\rho}_1, \hat{\rho}_2, ..., \hat{\rho}_n] \in \mathbb{R}^n_{\geq 0}$ are given *service latency* (in number of TDM slots) and *rate (bandwidth)* (required fraction of total available slots) *requirements* of the clients, respectively.
- H is a given TDM frame size, $H \in \mathbb{Z}^+$.

To satisfy the given requirements of a problem instance, we proceed by formalizing a TDM schedule and its associated parameters:

- The set $F = \{1, 2, \cdots, H\}$ denotes TDM slots.
- $D = [d_1, d_2, ..., d_H]$ is a schedule we want to find, where $d_i \in \{C \cup \varnothing\}$ indicates the client scheduled in slot i or $\varnothing$ (empty element) if the slot is not allocated.
- $\phi = \{\phi_1, \phi_2, ..., \phi_n\}$ is the number of slots allocated to each client, i.e. $\phi_i = | \{d_j\} : d_j = c_i |$.
- $\Theta = [\Theta_1, \Theta_2, ..., \Theta_n] \in \mathbb{R}^n_{\geq 0}$ and $\rho = [\rho_1, \rho_2, ..., \rho_n] \in \mathbb{R}^n_{\geq 0}$ are the *service latency* and *allocated rate*, respectively, provided by the TDM schedule.

The goal of TCP/LR-F is to find a schedule D for n clients sharing the resource such that the objective function, Φ, being the total allocated rate of all the real-time clients in C is minimized as shown in Equation (2.4), while the service latency and rate constraints (Equations (2.5) and (2.6) below) are fulfilled. This ensures that all real-time requirements are satisfied while maximizing the unallocated resource capacity available to non-real-time clients, thus maximizing their performance.

$$\textit{Minimize:} \sum_{c_i \in C} \rho_i = \Phi \tag{2.4}$$

$$\rho_i \geq \hat{\rho}_i,\ c_i \in C \tag{2.5}$$

$$\Theta_i \leq \hat{\Theta}_i,\ c_i \in C \tag{2.6}$$

Note that although the considered TCP/LR-F problem has a frame size H as a given parameter, the problem of finding the best frame size is addressed in [5], where both optimal and heuristic approaches are presented. It is also shown that the formulated problem with arbitrary frame size is NP-hard by transforming the

Periodic Maintenance Scheduling Problem (PMSP) [11] to the TCP/LR problem with arbitrary frame size. To prove NP-hardness of our problem with a given frame size using the same logic, the frame size H of the instance we transform Periodic Maintenance Scheduling Problem (PMSP) to is set to the least common multiple (lcm) of the client periods, making it a special case that is covered by the existing proof. The instance of PMSP is NP-hard with the schedule length equal to the lcm of the periods since the solution to PMSP has either this length or its integer multiples. Due to cyclicity of the schedule, if there exists any schedule for a given instance of PMSP, there always exist a schedule of length equal to lcm. Therefore, TCP/LR-F problem is NP-hard for any fixed frame size.

2.4 ILP Model

Having established that TCP/LR-F is NP-hard, we know that there exist no algorithms with polynomial complexity that solves the problem optimally unless *P=NP*. An approach to find optimal solutions based on ILP is hence justified, as there are available solvers to efficiently explore the vast solution space. In this section, we start by presenting an ILP model of our problem using only four simple constraints. After this, we present three optimizations of the model that significantly reduce the solution space and the computation time of the solver.

We now present the basic ILP formulation. The proposed model is based on the time-indexed scheduling approach [62]. This means that for each client $c_i \in C$, there are exactly H binary variables t_i^j, defined according to:

$$t_i^j = \begin{cases} 1, & \text{if slot } j \text{ is allocated to client } c_i. \\ 0, & \text{otherwise.} \end{cases}$$

The minimization criterion (2.7) is reformulated from Equation (2.4) in terms of the variables presented above. The solution space is defined by four constraints: Constraint (2.8) states that a slot can be allocated to maximally one client. Constraint (2.9) then dictates that enough slots must be allocated to a client to satisfy its rate requirement, which is computed according to Equation (2.3). The following two constraints focus on the worst-case provided service offered by the TDM schedule to a client, $\underline{r}_i^j$ ($\underline{r}_i^j \leq r_i^j$), where r_i^j corresponds to the lower solid blue line labeled 'provided service' in Figure 2.2. Constraint (2.10) states that the worst-case provided service to a client c_i during a busy period of any duration j starting in any slot k cannot be larger than the service provided by its allocated slots.

Lastly, Constraint (2.11) states that the worst-case provided service of the client, $\underline{r}_j^i$, must satisfy its LR requirements and is a straight-forward implementation of Definition 1.

$$\textit{Minimize: } \frac{\sum_{c_i \in C} \sum_{j \in F} t_i^j}{H}. \tag{2.7}$$

subject to:

$$\sum_{c_i \in C} t_i^j \leq 1, \qquad j \in F. \tag{2.8}$$

$$\sum_{j=1}^{H} t_i^j \geq H \cdot \hat{\rho}_i, \qquad c_i \in C. \tag{2.9}$$

$$\underline{r}_i^j \leq \sum_{l=k}^{(k+j) \bmod H} t_i^l, \qquad k \in F,\, c_i \in C,\, j \in F. \tag{2.10}$$

$$\underline{r}_i^j \geq \hat{\rho}_i \cdot (j - \hat{\Theta}_i), \qquad j \in F,\, c_i \in C. \tag{2.11}$$

After introducing the basic ILP model of TCP/LR-F, we proceed by discussing a few computation time optimizations. The first optimization exploits that an *increased lower bound on the number of slots allocated to a client*, $\underline{\phi}_i$ can be found by considering both its rate (first part) and service latency (second part) requirements in Equation (2.12). Unlike the rate requirement, the number of slots required to satisfy the service latency requirement depends on where the slots are allocated in the frame, which is not known beforehand. This lower bound is obtained by assuming an equidistant allocation, which results in the minimum number of slots required to be allocated to satisfy the service latency requirement.

$$\sum_{j \in F} t_i^j \geq \underline{\phi}_i = \max(\lceil \hat{\rho}_i \cdot H \rceil, \left\lceil \frac{H}{\hat{\Theta}_i + 1} \right\rceil). \tag{2.12}$$

For example, having requirements $\hat{\rho}_1 = 0.5$ and $\hat{\Theta}_1 = 3$ for client c_1 and a frame size $H = 10$, the lower bound on the number of allocated slots $\underline{\phi}_1$ is the maximum of the $\lceil 0.5 \cdot 10 \rceil = 5$ slots required to satisfy the bandwidth requirement and the $\left\lceil \frac{10}{3+1} \right\rceil = 3$ slots required to allocate each fourth slot in the TDM table to the client. Thus, the lower bound for client c_1 equals $\underline{\phi}_1 = 5$, which in this case is determined by its bandwidth requirement.

The second optimization *removes redundant constraints* generated by Constraints (2.10) and (2.11). As one can see, $H^2 \cdot n$ constraints are generated by Constraint (2.10) and $H \cdot n$ constraints by Constraint (2.11). However, it is not necessary to generate Constraints (2.10) and (2.11) for $j < \hat{\Theta}_i$, since the service bound provided by the LR guarantee is always zero in this interval by Definition 1. This is clearly seen in Figure 2.2, where the provided service curve is zero for the first four slots.

Additional constraints can be removed if more slots are required to satisfy the service latency requirements than the rate requirements, i.e. when the second term in the max-expression in Constraint (2.12) is dominant. In the remainder of this chapter, we refer to clients with this property as *latency-dominated*, as opposed to *bandwidth-dominated*, clients. As shown in [5], for latency-dominated clients Constraints (2.10) and (2.11) only need to be generated for a single point where $j = \lfloor \hat{\Theta}_i \rfloor + 1$.

The third optimization reduces the solution space by *reducing rotational symmetry*. This means that for any given TDM schedule, $H - 1$ similar schedules can be generated by rotating the given schedule and wrapping around its end. The problem is that all these schedules have the same criterion value and only one of them needs to be in the considered solution space. Constraint (2.13) addresses this problem

by adding a constraint that fixes the allocation of the first slot to the client with the smallest minimum number of required slots $\underline{\phi}_i$ (defined in Constraint (2.12)). This particular choice of client has been experimentally determined to significantly reduce the computation time of the solver.

$$t_t^1 = 1, t = argmin_{c_i \in C}\ \underline{\phi}_i. \tag{2.13}$$

2.5 Branch-and-Price Approach

The presented ILP model finds optimal solutions in reasonable time for current multi-core systems. However, despite the optimizations, it does not scale to many-core systems with 32 or more clients. To expand the range of problems that we are able to solve by the ILP model described in the previous section, a B&P approach [35] is introduced, which uses the ILP problem formulation from the previous section as a building block. B&P allows solving instances of the TCP/LR-F problem with larger number of clients, where the ILP formulation becomes too slow. The first reason for this behavior is that it does not need as many constraints in the ILP formulation for non-latency-dominated clients, where $H^2 \cdot n$ Constraints (2.10) are required in the ILP. Moreover, it reduces the number of explored symmetrical solutions and typically has a smaller branching tree. Both of these properties result in significantly reduced computation time for large problem instances compared to the previously described ILP model. The structure of this section is the following. First, background on the B&P approach is provided. Then, all the necessary problem-dependent parts of the algorithm are described. The computation time optimizations of the algorithm are given in the following section.

2.5.1 Preliminaries

Branch-and-price is an exact method to solve ILP problems, which combines column generation and branch-and-bound approaches. In order to obtain a problem formulation for the B&P approach, Dantzig-Wolfe decomposition [35] is performed on the ILP model from the previous section. At higher level, this decomposition transforms the space of binary variables t_i^j of the ILP model into the space of complete solutions for individual clients, i.e. B&P works with *complete schedules for individual clients* instead of dealing with allocation of single slots. The solutions for a single client are called *columns*.

The process of applying Dantzig-Wolfe decomposition on the ILP model results in an ILP *master model* $MM(\Omega)$ that contains a *set of all possible columns* $\Omega = \{\Omega_1, \Omega_2, \cdots, \Omega_n\}$ for each client. Columns are iteratively generated by a so called *sub-model*, here an ILP model for a single client. Then, they are combined into a complete solution for all clients by the master model. Note that each client requires its own instance of the sub-model, since they have distinct requirements.

An example column set is shown in Figure 2.3. In the considered TCP/LR-F problem, columns are complete TDM schedules for individual clients.

Here, the set of columns Ω_1 for the first client with requirements $\hat{\rho}_1, \hat{\Theta}_1$ contains two columns on the top of the figure and the set of columns Ω_2 for the second client is at the bottom. The decision variables $\omega_{i,k}$ indicate whether or not column z_k is

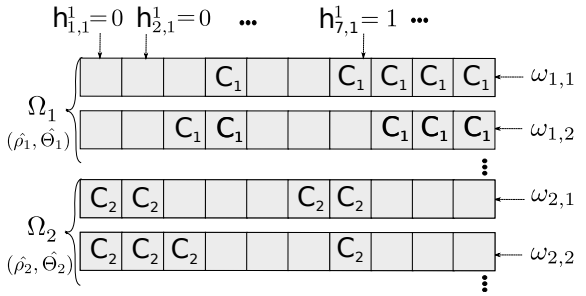


Figure 2.3: Example of a set of columns for the restricted master model.

included in the schedule for client c_i. One of the possible solutions here is to use column 2 for the first client and column 1 for the second one, i.e. $\omega_{1,1} = 0$, $\omega_{1,2} = 1$, $\omega_{2,1} = 1$, $\omega_{2,2} = 0$. Each column z_k is defined via a set of coefficients $h^j_{i,k}$. This coefficient is equal to 1 if column z_k allocates slot j to client c_i and 0 otherwise.

A drawback of using columns instead of the binary variables t^j_i of the ILP model is the large number of possible columns. However, it is sufficient to gradually generate only the most promising ones and expand the search space of solutions step-by-step. At a certain (final) moment it can be proven (see [35]) that the optimal solution is found. A master model that considers only a subset of columns $\Omega^R = \{\Omega^R_1, \Omega^R_2, \cdots, \Omega^R_n\} \subseteq \Omega$ is called *the Restricted Master Model* and is denoted as $MM(\Omega^R)$.

Thus, the idea, described above, is known as the *column generation* approach. Since column generation is only able to solve the linear relaxation of the master model, it is necessary to extend this approach with a *branch-and-bound* technique in order to get an integer solution. This combination is known in the literature as *branch-and-price*.

2.5.2 Outline of the Algorithm

The overall scheme of the B&P algorithm in 8 steps is shown in Figure 2.4. First of all, the algorithm must generate a set of initial columns Ω^R in Step 1 using a heuristic. Note that the quality of the initial columns only affects the computation time and not the optimality of the solution. Step 2 starts the process of column generation by solving the linear relaxation of the restricted master model. The output of this step is quantitative directions (dual values) that guide the column search for the sub-model. Next, a new column for some client is constructed by the sub-model (Step 3) subject to the directions obtained in the previous step by $MM(\Omega^R)$. If a new promising column for any client is found, the column is added to Ω^R and the next iteration of the column generation algorithm starts (back to Step 2). Promising column is a column that have a chance to improve the solution if chosen for a client. Otherwise, the optimal linear solution of the relaxed $MM(\Omega^R)$ is *a lower bound for the optimal integral solution* and is denoted as Φ^{LB}. If bounding takes place in Step 4, i.e. this branch is already worse or the same as the best solution known so far, the current node is closed in Step 8. In case the branch is still promising, Step 5 checks whether or not the solution obtained by column

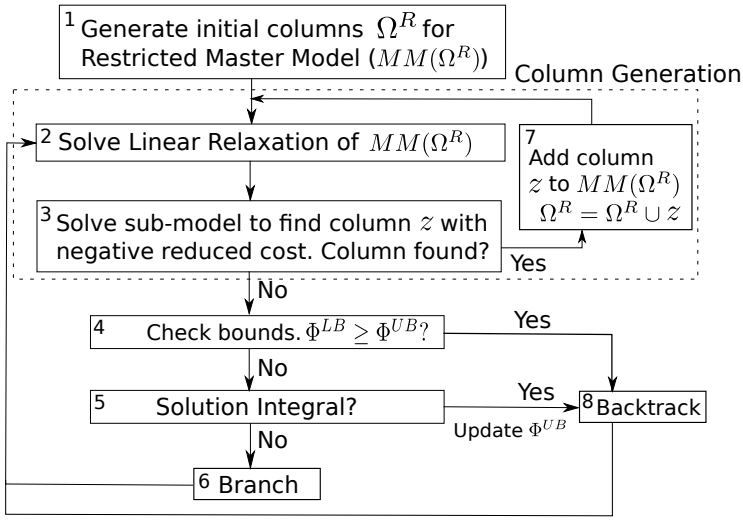


Figure 2.4: Outline of the branch-and-price algorithm.

generation is integral. In case it is, a new candidate solution to the initial integer master model is found. This solution defines a new *upper bound on the criterion*, Φ^{UB}, which is updated before the node is closed in Step 8. In case neither bounding nor the check on integrality closes the node, branching takes place.

Using the branch-and-bound technique means having a branching tree, which is in essence a set of nodes with parent-child relationships. The node is defined by a partial solution, i.e. a chain of slot assignment decisions made in the parent nodes. A decision is to impose/forbid assignment of a slot to a client. At the beginning, the first slot is fixed in the root node, as mentioned in the optimizations of the ILP model in Section 2.4. The column generation procedure (Steps 2, 3, 7) together with bounding (Step 4) and checking solution on integrality (Step 5) are launched in each node. In case the node is not closed, a child node is generated with a new decision, i.e. which slot to impose/forbid assignment to which client.

2.5.3 Master Model Formulation

The master model combines TDM tables for individual clients in order to provide a feasible solution, where each slot is allocated at most once in the schedule and requirements of all clients are satisfied. Moreover, it searches for the feasible schedule with the minimum total allocation. The master model is formulated as an integer linear programming problem, but in column generation it is solved as a linear problem, as stated earlier.

For TCP/LR-F, the Dantzig-Wolfe decomposition of the ILP model from Section 2.4 results in the following master model. The columns in Ω^R are defined via coefficients $h^j_{i,k}$, indicating whether or not a slot in a column is allocated to a client, defined according to

$$h^j_{i,k} = \begin{cases} 1, & \text{if slot } j \text{ is allocated to client } c_i \text{ in } z_k \in \Omega_i \\ 0, & \text{otherwise.} \end{cases}$$

Decision variables indicating whether client $c_i \in C$ uses column $z_k \in \Omega_i$, are defined as

$$\omega_{i,k} = \begin{cases} 1, & \text{if client } c_i \text{ uses column } z_k \in \Omega_i \\ 0, & \text{otherwise.} \end{cases}$$

It is a binary decision variable, which can be interpreted in the linear relaxed master model as the weight or "probability" of using column z_k for client c_i.

Furthermore, another set of decision variables is introduced. Variables y_j reflect the over-allocation of slot j in the final solution, i.e. $y_j = \max(0, v_j)$, where slot j is allocated by $v_j - 1$ clients. It means that any feasible solution has $y_j = 0, \forall j \in F$. These variables are introduced in order to have an initial set of columns Ω^R even in cases where a feasible solution could not be found in reasonable time in Step 1 of Figure 2.4. This is important since it is in general not possible to find Ω^R that contains a feasible solution in polynomial time.

Minimization of the total allocated rate for the restricted master model is formalized in the objective function

$$Minimize: \ \frac{\sum_{c_i \in C} \sum_{z_k \in \Omega_i^R} \phi_{i,k} \cdot \omega_{i,k}}{H} + \mathbb{M} \cdot \sum_{j \in F} y_j = \Phi^{MM}, \tag{2.14}$$

where $\phi_{i,k}$ is the total number of allocated slots to client c_i in column z_k and $\mathbb{M}$ is some sufficiently big number, in this work chosen to be $\mathbb{M} = 10$. The sum of over-allocation variables y_j are multiplied by $\mathbb{M}$ to provide a major penalty for overlapping (infeasible) schedules to ensure they are weeded out quickly.

The master model comprises only two constraints. The first one, Constraint (2.15), is meant for counting the number of times slot j is over-allocated in the schedule, which is expressed by variable y_j.

$$\sum_{c_i \in C} \sum_{z_k \in \Omega_i} h_{i,k}^j \cdot \omega_{i,k} \leq y_j + 1, \qquad j \in F. \tag{2.15}$$

Coefficients $b_{i,k}$ are introduced for the second constraint, indicating whether column z_k was constructed for client c_i:

$$b_{i,k} = \begin{cases} 1, & \text{if } z_k \in \Omega_i^R. \\ 0, & \text{otherwise.} \end{cases}$$

The second constraint, Constraint (2.16), forces the solver to choose at least one column for each client. For the linear relaxation, the sum of "probabilities" of using columns for a particular client c_i from Ω_i^R must be greater than or equal to 1. Note that this sum is always 1 in the optimal solution of the initial problem, as criterion minimization pushes it down. Considering this constraint reduces computation time, since a feasible solution can be found earlier in the branching tree of the master model, resulting in a better upper bound. Having a good upper bound earlier allows bounding more efficiently, which reduces computation time.

$$\sum_{z_k \in \Omega_i^R} b_{i,k} \cdot \omega_{i,k} \geq 1, \qquad c_i \in C. \tag{2.16}$$

2.5.4 Sub-model Formulation

The main purpose of the sub-model is to generate the columns Ω^R for the master model. Apart from that, the sub-model is used to check optimality of the restricted master model $MM(\Omega^R)$. The following condition guarantees optimality of the solution obtained by $MM(\Omega^R)$ (see [116] for the proof) and it is used for constructing the sub-model formulation:

Condition 1 (Optimality condition)**.** *A solution to a linear relaxation of $MM(\Omega^R)$ is optimal if and only if there is no column $z_k \in \Omega$ that is infeasible for the dual model to the linear relaxation of $MM(\Omega^R)$.*

The notion of duality between two LP models is described in [35]. Basically, it exchanges constraints and variables in their LP formulations. Note that the procedure of constructing a *dual master model* formulation $DMM(\Omega^R)$ to $MM(\Omega^R)$ does not contain any design decisions and follows automatically from the formulation of $MM(\Omega^R)$.

The formulation of $DMM(\Omega^R)$ is given below, where λ_j are dual variables that correspond to the set of Constraints (2.15) and σ_i are dual variables for Constraints (2.16) of the master model. $DMM(\Omega^R)$ aims to maximize Criterion (2.17) with respect to Constraints (2.18)–(2.20).

$$Maximize: -\sum_{j \in F} \lambda_j + \sum_{c_i \in C} \sigma_i. \tag{2.17}$$

subject to:

$$-\sum_{j \in F} h_{i,k}^{j} \cdot \lambda_j + b_{i,k} \cdot \sigma_i \leq \frac{\phi_{i,k}}{H}, \qquad c_i \in C, z_k \in \Omega_i, \tag{2.18}$$

$$\lambda_j \leq M', \qquad j \in F \tag{2.19}$$

$$\lambda_j \geq 0, \qquad j \in F \tag{2.20}$$

Variables λ_j and σ_i are called *shadow prices*. In terms of the TCP/LR-F problem, they can be interpreted as how much the value of Criterion (2.14) of the master model would decrease if the corresponding constraints of MM are relaxed. Particularly, λ_j indicates a potential gain in terms of criterion value if slot j is allowed to be allocated twice (in general, allowing slot j to be allocated k times, the criterion reduces by $(k-1) \cdot \lambda_j$). Meanwhile, σ_i is the price of having one column for client c_i both in terms of its allocated rate ρ_i and its slot assignment, i.e. the criterion value (2.14) would reduce by σ_i if client c_i could have no columns in the final solution.

As already stated, the sub-model generates new promising columns for an individual client. Promising are those that violate feasibility of $DMM(\Omega^R)$ according to the optimality stated in Condition 1. The negated value by which Constraints (2.18)–(2.20) are violated by a new column z_k is called the *reduced cost*. In TCP/LR-F, the reduced cost value can be interpreted as how much the cost of having certain column in a final solution (in terms of the criterion value) must be reduced before it

is included in the optimal solution. From the point of view of Condition 1, negative reduced cost means the current solution of $MM(\Omega^R)$ is not optimal. Basically, the sub-model searches for a new column, defined by Equations (2.9), (2.10) and (2.11). To find such a column with minimal reduced cost, it uses the ILP model from Section 2.4 for a single client. This is how the ILP model is used as a building block in a bigger framework.

Having minimal reduced cost does not necessarily mean that the column brings the result towards an optimal solution. However, choosing the column with the minimal reduced cost is a heuristic that behaves very well in practice [35] and eventually leads to the optimal solution due to Condition 1.

Next, the formulation of the sub-model is given. Since it is required to minimize the reduced cost, violation of Constraints (2.18)-(2.20) needs to be formulated in terms of the sub-model variables. Remember that the values λ_j and σ_i are constants from the sub-model point of view, since they are obtained from the master model after Step 2 in Figure 2.4. Thus, Constraints (2.19) and (2.20) are satisfied by default and it is sufficient to consider violation of Constraint (2.18) only.

Looking closely at Constraint (2.18), it is clear that the sub-model only needs to determine slot allocation for the given client c_i, i.e. variables $h^j_{i,k}$ are in fact the same as t^j_i from the ILP model, previously described in Section 2.4. Furthermore, $\phi_{i,k}$ is the number of allocated slots in column k, which implies $\phi_{i,k} = \phi_i = \sum_{j \in F} t^j_i$. Finally, $b_{i,k}$ is always 1 in the sub-model, since the schedule is constructed for client c_i. Thus, the sub-model for client c_i has the criterion Φ^{sub} (2.21), which is the reduced price expression.

$$Minimize : \sum_{j \in F} t^j_i \cdot \lambda_j + \frac{\sum_{j \in F} t^j_i}{H} - \sigma_i = \Phi^{sub}. \tag{2.21}$$

The constraints of the sub-model duplicate the constraints of the ILP model in Section 2.4 for $C = \{c_i\}$, i.e. considering client c_i only. Constraint (2.22) states that the bandwidth requirement must be fulfilled, Constraint (2.23) computes the points of the worst-case provided service line and Constraint (2.24) guarantees satisfying the service latency requirement for the given client according to Definition 1. Moreover, all the optimizations for the ILP model, described in Section 2.4, are used for the sub-model as well.

$$\sum_{j=1}^{H} t^j_i \geq H \cdot \hat{\rho}_i \tag{2.22}$$

$$r^j_i \leq \sum_{l=k}^{(k-j) \bmod H} t^j_i,\ k \in F,\ j \in F \tag{2.23}$$

$$r^j_i \geq \hat{\rho}_i \cdot (j - \hat{\Theta}_i),\ j \in F. \tag{2.24}$$

Thus, here the ILP model, formulated in Section 2.4, is used as a piece in a larger framework to solve large problems more efficiently due to the ILP decomposition, i.e. instead of solving a large model that considers all clients at the same time, the master model and the sub-model solve smaller sub-problems. We present a small

illustrative example of the column generation algorithm on a problem instance with 2 clients in the following subsection.

2.5.5 Illustration of Column Generation

A short illustration of the column generation algorithm described earlier in this section is shown here. The small problem instance used for this purpose considers 2 clients c_1, c_2 with $\hat{\Theta} = [3, 3]$ and $\hat{\rho} = [0.5, 0.3]$. The TDM frame size is $H = 10$. We assume initial columns with $h_{1,1}$, $h_{2,1}$ for clients c_1, c_2, respectively, according to:

$$\Omega^R = \{h_{1,1} = [0,0,1,1,0,0,0,1,1,1]\,, \tag{2.25}$$
$$h_{2,1} = [1,1,0,0,0,1,1,0,0,0]\}\,, \tag{2.26}$$

the restricted master model $MM(\Omega^R)$ is formulated as:

$$Minimize: \ \frac{5}{10} \cdot \omega_{1,1} + \frac{4}{10} \cdot \omega_{2,1} + 10 \sum_{j \in F} y_j$$

subject to :

$$\begin{aligned}
0 \cdot \omega_{1,1} + 1 \cdot \omega_{2,1} &\leq 1 + y_1 \\
0 \cdot \omega_{1,1} + 1 \cdot \omega_{2,1} &\leq 1 + y_2 \\
1 \cdot \omega_{1,1} + 0 \cdot \omega_{2,1} &\leq 1 + y_3 \\
1 \cdot \omega_{1,1} + 0 \cdot \omega_{2,1} &\leq 1 + y_4 \\
0 \cdot \omega_{1,1} + 0 \cdot \omega_{2,1} &\leq 1 + y_5 \\
0 \cdot \omega_{1,1} + 1 \cdot \omega_{2,1} &\leq 1 + y_6 \\
0 \cdot \omega_{1,1} + 1 \cdot \omega_{2,1} &\leq 1 + y_7 \\
1 \cdot \omega_{1,1} + 0 \cdot \omega_{2,1} &\leq 1 + y_8 \\
1 \cdot \omega_{1,1} + 0 \cdot \omega_{2,1} &\leq 1 + y_9 \\
1 \cdot \omega_{1,1} + 0 \cdot \omega_{2,1} &\leq 1 + y_{10} \\
1 \cdot \omega_{1,1} + 0 \cdot \omega_{2,1} &\geq 1 \\
0 \cdot \omega_{1,1} + 1 \cdot \omega_{2,1} &\geq 1.
\end{aligned}$$

The trivial solution to the relaxed $MM(\Omega^R)$ is $\omega_{1,1} = 1$, $\omega_{2,1} = 1$, $y_j = 0 \ \forall j \in F$ having objective function $\Phi = 0.9$. The corresponding dual solution is $\lambda_j = 0 \ \forall j \in F$, $\sigma_1 = -0.5$, $\sigma_2 = -0.4$.

When we formulate the sub-model from Equations (2.21)-(2.24) for client c_1 to obtain a new column, its solution is $[0,1,1,0,1,1,0,0,1,0]$. However, its reduced price $\Phi^{sub} = \sum_{j \in F} t_1^j \cdot 0 + 0.5 - 0.5 = 0$ (see Equation (2.21)) is not negative, which means that this column cannot improve the objective function of $MM(\Omega^R)$. On the other hand, the sub-model for client c_2 finds $[1,0,0,0,1,0,0,1,0,0]$ with reduced price $\sum_{j \in F} t_2^j \cdot 0 + 0.3 - 0.4 = -0.1$, which has potential to improve the objective function of $MM(\Omega^R)$.

When the new column for c_2 is added into $MM(\Omega^R)$ (see Step 7 in Figure 2.4), $MM(\Omega^R)$ is solved again in Step 2. In this case, the primal solution stays practically

the same (i.e. $\omega_{1,1} = 1$, $\omega_{2,1} = 1$, $\omega_{2,2} = 0$, $y_j = 0\ \forall j \in F$), but the dual solution changes to $\lambda = [0, 0, 0, 0, 0, 0, 0, 0.1, 0, 0]$, $\sigma_1 = -0.6$, $\sigma_2 = -0.4$. For this dual solution, the sub-model for client c_1 finds column $[0, 1, 1, 0, 1, 1, 0, 0, 1, 0]$ with reduced price $\Phi^{sub} = 0 \cdot 0.1 + 0.5 - 0.6 = -0.1$. This new column allows $MM(\Omega^R)$ to find solution $\omega_{1,1} = 0.5$, $\omega_{2,1} = 0.5$, $\omega_{2,2} = 0.5, \omega_{1,2} = 0.5$, $y_j = 0\ \forall j \in F$ with a better value of the objective function equal to $\Phi = 0.85$.

The last column $h_{2,3} = [1, 0, 0, 1, 0, 0, 1, 0, 0, 0]$ is added in the next iteration when $MM(\Omega^R)$ achieves the optimal solution of the relaxed master model, which equals to $\Phi = 0.8$. After that there is no column with negative reduced price and column generation stops with

$$\begin{aligned}
\Omega^R = \quad & \{h_{1,1} = [0, 0, 1, 1, 0, 0, 0, 1, 1, 1]\,, && (2.27)\\
& h_{2,1} = [1, 1, 0, 0, 0, 1, 1, 0, 0, 0]\,, && (2.28)\\
& h_{2,2} = [1, 0, 0, 0, 1, 0, 0, 1, 0, 0]\,, && (2.29)\\
& h_{1,2} = [0, 1, 1, 0, 1, 1, 0, 0, 1, 0]\,, && (2.30)\\
& h_{2,3} = [1, 0, 0, 1, 0, 0, 1, 0, 0, 0]\}\,. && (2.31)
\end{aligned}$$

If the solution is integer (i.e. $\omega_{1,1} = 0$, $\omega_{2,1} = 0$, $\omega_{2,2} = 0, \omega_{1,2} = 1, \omega_{2,3} = 1$, $y_j = 0\ \forall j \in \Omega$), a new better solution was found and the B&P algorithm continues by Step 8. Otherwise, the algorithm goes to Step 6 where the branching takes place.

2.5.6 Branching Strategy

The third and last main component of the B&P approach is the branch-and-bound procedure. Here, we focus on the *branching strategy* used. B&P approaches usually use the 0/1 branching scheme, i.e. some binary variable is set either to 0 or to 1 in two child nodes. First of all, the important decision is how to choose variables to branch on. Branching on master model variables ($\omega_{i,p}$) is not effective since new columns are added in each iteration and the number of decisions to make increases with added columns, which results in a large decision tree and long computation time. Moreover, it is more efficient to consider variables of the sub-model as the branching variables, since it influences several columns at once.

Generally, branching adopts a depth-first search. Note that two branching decisions must be made: which *client* to allocate to which *slot*. Two branching strategies show good results for different sizes of problems. Both strategies start by sorting clients in ascending order according to their service latency requirements to begin branching from the clients with more critical requirements.

The first branching strategy has experimentally shown good results for smaller problems (up to 16-32 clients). It fixes t_i^j in the following order, assuming the above described sorting of the clients: $t_1^2, \cdots, t_1^H, t_2^1, \cdots, t_n^H$, i.e. slots are assigned sequentially from left to right for each client. Note that the first slot is always allocated to the client with the tightest service latency requirements to reduce symmetry in the solution space (similar to the ILP from Section 2.4). This branching first makes decisions of type $t_i^j = 0$, i.e. branching goes first to the branch where slot j is forbidden to be allocated to client c_i and only then to the branch where it is fixed to be allocated. The reason for this is to obtain a feasible solution as fast as

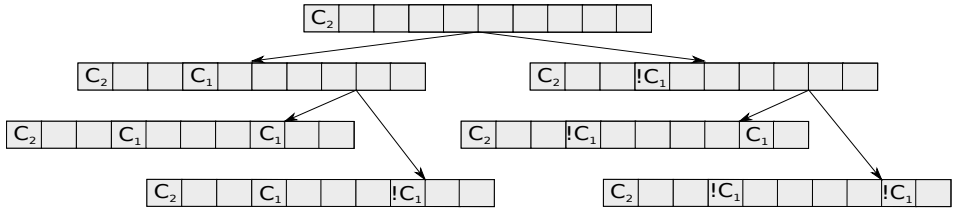


Figure 2.5: The first three layers of the example branching tree.

possible, since negative decisions are less likely to eliminate any column in Ω^R and therefore do not require time to be spent running column generation. Moreover, going through the solution space in a systematic manner reduces the number of explored symmetrical solutions.

The second branching strategy shows better results for larger use-cases (more than 16-32 clients). It begins by computing the total "probabilities" of including each slot $j \in F$ in the solution for all its columns, ω_j^{total}. For example, for the first client this means computing $\omega_j^{\text{total}} = \sum_{z_k \in \Omega_1^R : h_{1,k}^j = 1} \omega_{1,k}$. Then it chooses the non-decided slot j_c with maximum total "probability" of being in the solution, $j_c = argmax_{j \in F}\, \omega_j^{\text{total}}$ and branches on it. In case there are multiple slots with the same maximum ω_j^{total}, the leftmost slot j_c is chosen. While branching on variable $t_{j_c}^1$, it goes to the branch $t_{j_c}^1 = 1$ first. If the first client is fully decided or all the "probabilities" of undecided slots are zero, the procedure continues with the following client. This branching strategy typically leads to a feasible solution early on as slots with non-zero probability are the promising ones to allocate.

An example with the first three layers of a branching tree using the second branching strategy is shown in Figure 2.5. It assumes the root node contains the set of columns Ω^R from Figure 2.3 and the master model resulted in the variables $\omega_{1,1} = 0.2$, $\omega_{1,2} = 0.8$, $\omega_{2,1} = 0.7$, $\omega_{2,2} = 0.3$ in Step 2 of Figure 2.4. For the first client there are 4 slots $j = (4, 8, 9, 10)$ with the same $\omega_j^{\text{total}} = 1$ and neither of them is decided yet. Therefore, the branching decision in the root node is to use client $c_i = 1$ and slot $j = 4$ for branching. As a consequence, the left child has a partial solution, where $t_1^4 = 1$ and the right child has $t_1^4 = 0$ ($!c_1$). Note that column generation in the nodes of the second layer starts with a changed set of initial columns. Therefore, the branching decisions are made based on different values of $\omega_{i,j}$.

2.6 Computation Time Optimizations

The main components of the B&P approach are specified for TCP/LR-F in the previous section. However, there are plenty of opportunities for reducing the computation time of the approach. Numerous experiments and observations were done and this section presents the five most successful optimizations to reduce computation time.

First of all, starting the B&P approach from a feasible initial solution obtained by a *heuristic* implies a significant reduction of the computation time, since having a good upper bound on the solution in the beginning reduces the size of the branching

tree. If the heuristic is good, it may often be able to find the optimal solution by itself. In this case, the B&P approach only has to prove the optimality in the root node of the branching tree, considerably reducing the computation time. This is the case for our heuristic, later presented in Section 2.7.

Secondly, since the sub-model for non-latency-dominated clients requires H^2 constraints for checking service latency guarantees, *lazy constraints* [53] are used. The basic idea is to initially formulate the problem only with the most esssential constraints, omitting those that are only rarely violated. These other constraints are checked and added one-by-one to the model only if the solution violates any of them. This process is done in an efficient way. Instead of always starting from the beginning of the branching tree, it continues search from the place in the tree it last finished. We apply this technique to the sub-model, which we initially formulate assuming all clients are latency dominated, i.e. having only H Constraints (2.10) and (2.11) for $j = \lfloor \hat{\Theta}_i \rfloor + 1$. Next, for bandwidth-dominated clients, the solution is checked for satisfaction of Constraints (2.10) and (2.11) for $j > \lfloor \hat{\Theta}_i \rfloor + 1$. If a constraint is violated, it is added to the model and the solver continues its search until it is finished. The key idea behind this optimization is to exploit that although service latency is not always the largest gap between two consecutively allocated slots to the same client, it often is, and the extra constraints for bandwidth-dominated clients are hence typically not necessary.

The next optimization also concerns the branch-and-bound part. We use problem-specific information to *set the bounding condition* more effectively. The bounding condition in every node is set to be $\Phi^{LB} > \Phi^{UB} - 1/H$, where the discretization step $1/H$ is subtracted. The reason is that unless there is at least one slot less, which is exactly $1/H$ in terms of utilization, it is not an improved solution to the problem.

Column generation often suffers from a so called "tailing-off" effect, i.e. at the moment the reduced prices of sub-models are close to zero it starts to take a lot of iterations to converge to exact zeros. To deal with this issue, the fourth optimization, *Lagrangian relaxation*, is introduced. It estimates the lower bound Φ^{LB} on Φ inside the column generation loop by Equation (2.32), before the master model is solved to optimum. Thus, using Lagrangian relaxation stops the column generation process before Condition 1 holds and closes the node, saving computation time. It is applicable when either the estimated lower bound $\overline{\Phi}^{LB} \geq \Phi^{UB}$ or if the estimated lower bound $\overline{\Phi}^{LB}$ and the one given by the master model after previous iteration Φ^{MM}_{curr} discretize to the same number of allocated slots. Equation (2.32) computes the lower bound on the criterion value of the master model after each iteration inside the column generation procedure after Step 3 of Figure 2.4. The estimation is the current criterion value of the master model Φ^{MM}_{curr} plus the sum of all of the reduced prices Φ^{sub}_i of the sub-models [51]. The Lagrangian relaxation cannot lower bound it incorrectly [51], i.e. it does not break optimality.

$$\overline{\Phi}^{LB} = \Phi^{MM}_{curr} + \sum_{c_i \in C} \Phi^{sub}_i \tag{2.32}$$

Note that each time the Lagrangian relaxation is applied, all the sub-models must be solved to optimum. Hence, running Lagrangian relaxation each iteration of column generation could increase the total computation time, so it is an important

decision when to start this process. In our approach, Lagrangian relaxation is performed each n iterations, where n is the number of clients. This design decision is motivated by that when new columns are generated for all n clients, there is higher chance that the reduced prices have changed significantly and there is space for the Lagrangian relaxation to close the node.

The fifth and the final optimization of the computation time is *solution completion* by the ILP from Section 2.4. Sometimes branching goes deep enough so that it is possible to reduce computation time by launching the ILP model to find a schedule for all clients simultaneously. It is done when some percent of positive or negative decisions has been made. Our experiments have shown that for different sizes of problems these parameters should be set differently. For example, it is necessary to increase them with increasing number of clients, since larger problems could require running the ILP for a long time. Thus, nodes at this depth are solved to optimality using the ILP model and are closed afterwards. This procedure is done by taking the decisions that have already been made, fixing corresponding variables in the ILP model and solving this problem with an ILP solver.

2.7 Heuristic Approach

Although the proposed exact approaches solve TCP/LR-F optimally, it is sometimes acceptable to sacrifice the quality of the solution in order to reduce computation time. Moreover, as mentioned earlier, the B&P approach can use a heuristic solution to compute good initial columns that reduce the total computation time. Thus, the purpose of this section is to present a heuristic that solves the TCP/LR-F problem.

Heuristics of the constructive type (ones that construct a solution step-by-step) for the considered TCP/LR-F problem lack a good strategy to backtrack from low quality solutions and this is the reason we propose a *generative heuristic* that produces a complete solution at once. Although the generated solution may initially be infeasible, the heuristic gradually changes it towards a feasible one.

The proposed heuristic exploits the sub-model, previously described in Section 2.5.4. It is used in combination with the lazy constraints presented in Section 2.6 and with the optimality gap set to 5%, i.e. it is not necessarily the optimal solution that is returned by the solver, but one that is no further than 5% relative distance from the result of the linear relaxation of the problem. These improvements significantly reduce the computation time.

The heuristic constructs the schedule by iteratively running the sub-model for different clients in a cyclic manner. Remember that the sub-model aims to minimize $\Phi^{sub} = \sum_{j \in F} t_i^j \cdot \lambda_j + \frac{\sum_{j \in F} t_i^j}{H} - \sigma_i$, where the dual price coefficients λ_j control allocation of slot j. In this heuristic, λ_j is not coming from the master model, but is determined by the solutions from previous iterations. Note that in the column generation procedure, this is done by the master model in Step 2 of Figure 2.4. Here, the heuristic substitutes Steps 2 and 7 of Figure 2.4 with a procedure that assigns appropriate coefficients λ_j. Meanwhile, the dual price σ_i is omitted in the heuristic as it is a constant in a sub-model and constants play no role in minimization.

Algorithm 2.1 shows the proposed heuristic. The number of clients n, service latency $\hat{\Theta}$ and bandwidth requirements $\hat{\rho}$ are used as input. Furthermore, there

are two parameters of the heuristic, a coefficient α, which controls the speed of convergence to the final solution, and the maximum number of iterations of the sub-model N_{iter}^{max}. Each iteration includes two main steps: first coefficients λ_j are computed on Line 4 (explained later) and then the sub-model for client c_i is launched on Line 5. Note that the current solution is the schedule constructed out of the n last created columns for individual clients. The heuristic stops either when the maximum number of iterations, N_{iter}^{max}, is reached or if the current solution $t_i^{j,curr}$ is collision-free, i.e. there are no two clients that share a slot in the current solution.

Algorithm 2.1 The proposed generative heuristic

1: Inputs: $n, \hat{\Theta}, \hat{\rho}, \alpha, N_{iter}^{max}$
2: $N_{iter} = 0$, $i = 1$, $t_i^{j,curr} = 0$
3: **while** $N_{iter} < N_{iter}^{max}$ and $t_i^{j,curr}$ has collisions **do**
4: $\lambda =$ Coefficients computation(c_i, α)
5: $t_i^{j,curr} =$ SubModel($\hat{\Theta}_i, \hat{\rho}_i$, λ)
6: $N_{iter} = N_{iter} + 1$
7: $i = (i \mod n) + 1$
8: **end while**
9: Output: $t_i^{j,curr}$

The core of the heuristic is fast assignment of coefficients λ_j to each slot for a given client, such that if multiple clients allocate the same slot, some of them will change their allocation. As we are minimizing the criterion value, a higher value of the coefficient means it is less desirable for a client to allocate slot j ($\lambda_j \in [0.9, 2.5], j \in F$). The procedure of assigning coefficients λ_j for client c_i is presented in Algorithm 2.2. The algorithm considers four mutually exclusive and jointly exhaustive cases that are detailed below:

1. *Slot j is allocated to some client $c_k \neq c_i$ and not allocated to client c_i* in the current schedule $t_i^{j,curr}$. Generally, in this situation slot j should not be allocated to client c_i to avoid conflict, but in early stages of the heuristic this slot can be used if necessary, since it is not known in advance which allocation is better. The corresponding coefficient is assigned $\lambda_j = min(2,\ 1 + d_{j,i} \cdot \alpha)$, where $d_{j,i}$ is the number of times slot j was allocated in the previous iterations to any client c_k, $k \neq i$. The coefficient $d_{j,i}$ hence adds state to the iterative algorithm: the more times the slot was allocated to other clients before, the less attractive it is to allocate this slot to client c_i. Furthermore, the higher the value of coefficient α, the faster the schedule converges, since $d_{j,i} \cdot \alpha$ becomes larger. The upper bound of 2 limits the maximal penalization.

2. *Slot j is allocated to client c_i in $t_i^{j,curr}$ and there is no conflict.* The constant value of $\lambda_i = 0.9$ is chosen here, since the algorithm prefers not to change the clients allocation unless necessary. In case this value would be set too low, the algorithm will tend to allocate a new slot rather than changing the allocation of the currently assigned slots.

3. *Slot j is allocated in $t_i^{j,curr}$ to client c_i and there is a conflict.* All but one of the conflicting clients should leave the slot. However, it is not clear in advance which client should have the slot. Therefore randomness is introduced here - the coefficient in this case is selected from uniformly distributed numbers between 1 and 2.5.

4. In case j *is not allocated* in the current schedule, a value $\lambda_j = 1.0$ is assigned.

Algorithm 2.2 Coefficients computation

1: Inputs: c_i, α
2: **for all** $j \in F$ **do**
3: $d_{j,i}$ = number of times slot j was allocated in the previous iterations to any client $c_k \neq c_i$
4:
$$\lambda_j = \begin{cases} min(2, 1 + d_{j,i} \cdot \alpha), & \text{if } t_i^{j,curr} = 0,\ t_k^{j,curr} = 1, \\ & \text{for some } k \neq i. \ (1) \\ 0.9, & \text{if } t_i^{j,curr} = 1,\ t_k^{j,curr} = 0, \\ & \text{for every } k \neq i. \ (2) \\ 1 + rand() \cdot 1.5, & \text{if } t_i^{j,curr} = 1,\ t_k^{j,curr} = 1, \\ & \text{for some } k \neq i. \ (3) \\ 1, & \text{if } t_k^{j,curr} = 0 \\ & \text{for every } k = 1, ..., n. \ (4) \end{cases}$$
5: **end for**
6: Output: λ

2.8 Experimental Results

This section experimentally evaluates and compares the TDM configuration methodology based on the B&P approach and the proposed heuristic. First, the experimental setup is explained, followed by an experiment that compares the proposed B&P and heuristic approaches to the existing ILP from Section 2.4 in terms of scalability. Furthermore, it shows the trade-off between criterion value and computation time for the heuristic approach.

2.8.1 Experimental Setup

Experiments are performed using three sets of 5×200 synthetic use-cases, each comprising 8, 16, 32, 64 or 128 real-time clients on one resource. The three sets are bandwidth-dominated, latency-dominated and mixed-dominated use-cases. The concepts of latency-dominated and bandwidth-dominated clients were previously introduced in Section 2.4. A mixed-dominated client is one that requires approximately equal allocated rate to satisfy both service latency and bandwidth requirements according to the right or left part of Equation (2.12), respectively. A mixed-dominated use-case comprises only mixed-dominated clients. This type

of use-cases was not considered in [5] due to high time complexity. The reason for looking at this group of instances is that the problem is more difficult than in bandwidth-dominated use-cases, but unlike latency-dominated cases, constraints cannot be removed by the computation time optimizations in Section 2.4. The reason for evaluating these three classes is to show the impact of the requirements on the computation time of the proposed B&P approach, as well as fairly evaluate the efficiency of the heuristic. We proceed by explaining how bandwidth and service latency requirements are generated for the three sets.

Parameters for synthetic use-case generation are given in Table 2.1. Firstly, bandwidth requirements of each client in a use-case are generated. Here, β is an interval from which bandwidth requirements for each client are uniformly drawn. The use-case is accepted if the total required rate of all clients is in the range [0.8, 0.95] for bandwidth-dominated use-cases, [0.35, 0.5] for latency-dominated use-cases and [0.7, 0.9] for mixed-dominated use-cases. Otherwise, it is discarded and the generation process restarts. The interval of acceptance is lower for both mixed-dominated and latency-dominated sets to leave space for over-allocation to satisfy the tighter service latency requirements. Each time the number of clients is doubled, the range of bandwidth requirements is divided by 2. This is to make sure the total load is comparable across use-cases with different number of clients, which is required to fairly evaluate scalability.

Table 2.1: The parameters for use-case generation

Clients	*Bandwidth-dominated*		*Latency-dominated*		*Mixed-dominated*	
	β	γ	β	γ	β	γ
8	[0.06, 0.16]	[0.6, 0.9]	[0.02, 0.07]	[1.6, 3.3]	[0.06, 0.14]	[0.95, 1.4]
16	[0.03, 0.08]	[0.5, 0.75]	[0.01, 0.035]	[1.58, 3.26]	[0.03, 0.07]	[0.9, 1.3]
32	[0.015, 0.04]	[0.4, 0.6]	[0.005, 0.0175]	[1.56, 3.22]	[0.015, 0.035]	[0.85, 1.2]
64	[0.0075, 0.02]	[0.3, 0.45]	[0.0025, 0.00875]	[1.54, 3.18]	[0.0075, 0.0175]	[0.8, 1.1]
128	[0.00375, 0.01]	[0.2, 0.3]	[0.00125, 0.004375]	[1.52, 3.14]	[0.00375, 0.00875]	[0.75, 1.0]

Service latency requirements are uniformly distributed according to $\frac{1}{\gamma \cdot \hat{\rho}}$, where a larger value of γ indicates a tighter requirement. The γ values are given in Table 2.1. The reduction of service latency requirements with increasing number of clients is empirically determined to provide instances with comparable difficulty by having similar total allocated rates for the final optimal schedules. Lastly, if the total possible load due to the service latency requirements (the right part of Equation (2.12)) is outside the interval $[0.75, 0.95]$ and $[0.7, 0.9]$, for latency-dominated and mixed-dominated use-cases, respectively, new latency requirements for the use-case are generated. For all sets, generated use-cases that are found infeasible using the optimal approach are discarded and replaced to ensure a sufficient number of feasible use-cases. Finally, the frame size is set to $H = n \cdot 8$ to make sure that the number of slots available to each client is constant across the experiment.

All in all, this generation process ensures that all use-cases are feasible, have comparable difficulty, and that all clients in the three sets have the desirable dominating property. Experiments were executed on a high-performance server equipped with 2x Intel Xeon X5570 (2.93 GHz, 20 cores total) and 100 GB memory. The ILP model and ILP part of B&P and the heuristic were implemented in IBM ILOG CPLEX Optimization Studio 12.5.1 and solved with the CPLEX solver using concert technology for the latter two. The B&P and heuristic approaches were

implemented in JAVA. The source code of the ILP model, B&P approach, and the heuristic together with benchmarks is available at `https://github.com/CTU-IIG/BandP_TDM`.

2.8.2 Results

The experiments evaluate the scalability of the proposed B&P approach and the trade-off between computation time and the total rate (the criterion) allocated to 8, 16, 32, 64 and 128 real-time clients for the optimal and heuristic approaches. Moreover, it compares the proposed approaches with already existing exact (the ILP formulation from Section 2.4) and heuristic (continuous allocation) strategies. A time limit of 3 000 seconds per use-case was set in order to obtain the results of the experiments in reasonable time. Furthermore, to get higher quality solutions at expense of increased computation time, the heuristic was launched 8 times for latency-dominated and mixed-dominated use-cases to exploit the random component of the heuristic. Note that the heuristic is only run once for bandwidth-dominated use-cases, since these are easier for the heuristic to solve. The heuristic parameters were set to $\alpha = 0.1$, $N_{iter}^{max} = 250$. Besides, the used branching strategy for the use-cases with 8 and 16 clients is the first (consecutive) one mentioned in Section 2.6, while for the use-cases with 32, 64 and 128 clients the second (maximum total probability) one was selected. Furthermore, after 10, 30, 60, 80, and 95% of positively decided slot allocations and 40, 100, 120, 260 and 300% of negative decisions in terms of time slots (with 100% being H) is done for the use-cases with 8, 16, 32, 64 and 128 clients, respectively, the completion by the ILP model is launched as discussed in Section 2.6. These numbers were empirically determined to provide a reasonable trade-off between computation time and quality of the solution.

The distribution of all values later in this section is shown in the form of box plots [58], where the quartile, median and three quartiles together with outliers (plus signs) are shown. Outliers are numbers that lie outside 1.5×the interquartile range away from the top or bottom of the box that are represented by the top and the bottom whiskers, respectively. Note that outliers were also successfully solved within the time limit.

Bandwidth-dominated use-cases

Figure 2.6 shows the vertical axis in logarithmic scale of the computation time for the bandwidth-dominated use-cases for the heuristic, B&P and ILP approaches for 8, 16, 32, and 64 clients, respectively. The ILP struggles to scale to use-cases with 32 clients, as 52 out of 200 use-cases are not solved to optimality within the given time limit, resulting in a failure rate of above 25%. Therefore, the results for 64 and 128 clients are only represented by the heuristic on the left and B&P on the right. The results show that the B&P approach significantly outperforms the ILP model, scaling well to the use-cases with 32, 64 and 128 clients. More specifically, B&P requires 10% (4 minutes) and 2% (14 minutes) of the ILP computation time for 8 and 16 clients, respectively. Moreover, it solves the use-cases with 32, 64 and 128 clients in 0.5, 6 and 29 hours, respectively, resulting in less than 9 minutes computation time per use-case on average for the use-cases with 128 clients. The quality of the solution is shown in Table 2.2, where the first number is the number

of failures and the second one is the average distance (excluding the failures) to the best solution obtained by one of the two optimal approaches. Failure is defined as no feasible solution for the heuristic and no optimal solution for the exact approaches within the time limit. The results for 128 clients are not presented in the table, since they are identical to the results for 64 clients. The reason for this is that for both types of use-cases, the heuristic was able to solve all 200 use-cases to optimality and B&P only had to prove optimality of the solutions.

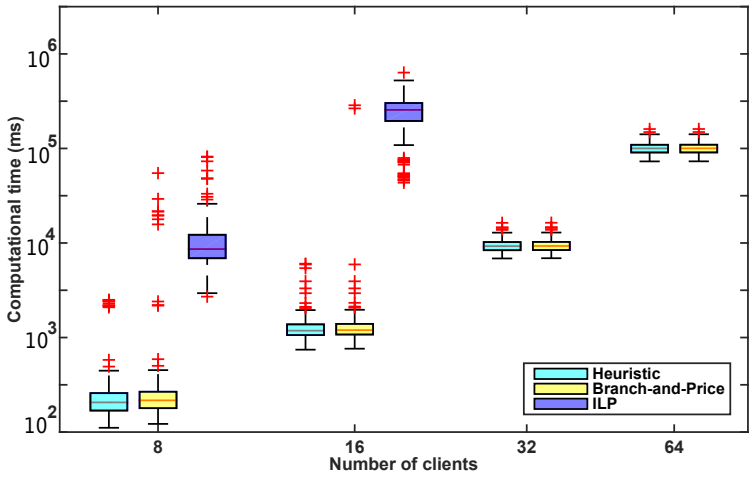


Figure 2.6: Computation time distribution for the bandwidth-dominated use-cases.

Comparing the heuristic and the B&P approach for the bandwidth-dominated use-cases, the results indicate a slight time reduction with some loss in the quality of the solution for the heuristic. The distributions of the computation time of both the heuristic and the B&P approach for 8, 16, 32, 64 and 128 clients look similar with exception of the use-cases marked as plus signs, for which the heuristic was not able to find any feasible solutions. As B&P starts with the heuristic solution, if the heuristic fails to find a feasible solution, it takes significantly longer for B&P. However, the results in Table 2.2 show that when the heuristic managed to find feasible solutions, it always found the optimal one. The second issue that seems suspicious is the visible similarity of the distribution of computation time of both the heuristic and B&P approaches, which is caused by using log_{10} scale, where the small difference becomes invisible. In reality, the total computation time of the heuristic and the B&P differ. The heuristic runs in 25%, 29%, 96%, 99% and 99% of the B&P computation time for 8, 16, 32, 64 and 128 clients, respectively. Such a similarity for 32, 64 and 128 clients is a result of the heuristic being successful in all use-cases, which could be caused by having more space for allocation without collisions when the frame size is longer.

To push the limits of the B&P approach and find the maximum number of clients it can manage, a use-case with 256 clients was generated by following the rules in Table 2.1. However, because of the memory limit of 100 GB, B&P was not able to finish the run for a single use-case. The current limits for our approach is

hence somewhere between 128 and 256 bandwidth-dominated clients in a use-case.

Table 2.2: Number of failures and average distance to the best obtained solution by either B&P or ILP approaches. (BD – bandwidth-dominated, LD – latency-dominated and MD – mixed-dominated use-cases)

Clients	*8 clients*			*16 clients*			*32 clients*			*64 clients*		
	BD	*LD*	*MD*	*BD*	*LD*	*MD*	*BD*	*LD*	*MD*	*BD*	*LD*	*MD*
ILP	0/0	0/0	1/0	0/0	1/0	1/0	52/0.0005	0/0	172/0	-	0/0	-
B&P	0/0	6/0	3/0	0/0	1/0	0/0	0/0	0/0	0/0	0/0	5/0	0/0
Heuristic	8/0	46/0.01	23/0.01	2/0	76/0.006	26/0.003	0/0	71/0.001	8/0.0001	0/0	61/0.0001	0/0

This experiment shows that for bandwidth-dominated use-cases the B&P approach significantly outperforms the ILP model for all sizes of use-cases, both in terms of computation time and quality of the obtained solution. Moreover, considering all 1 000 use-cases, the heuristic saves up to 75% of the computation time. This is done while sacrificing less than 1.5% of the use-cases that it fails to solve and giving optimal results for the other 98.5%.

Latency-dominated use-cases

The second group of use-cases that we focus on is latency-dominated use-cases. Since these use-cases are more complex than the bandwidth-dominated ones, instances with 128 clients take too long to run for all approaches and are not included in the results. The distribution of log_{10} of the computation time is shown in Figure 2.7. Here, it is clear that for smaller use-cases with 8 and 16 clients, the ILP model significantly outperforms the B&P approach. That is, the ILP runs in 2% (15 minutes) and 42% (176 minutes) of the B&P computation time for 8 and 16 clients, respectively. However, with increasing number of clients the situation becomes different. For 64 clients, B&P requires less time (total 26 hours versus the ILP with 65 hours, i.e. 2.5 times faster), but it is not able to find any feasible solution within the time limit for 5 use-cases. For the use-cases with 128 clients, running the heuristic 8 times in the beginning takes more than 50 minutes on average and in case the heuristic is not able to find a feasible solution, which happened in 14 use-cases out of 56, B&P was able to find a solution in 2 use-cases out of 14 within the given time limit.

As expected, the heuristic does not show good results on this type of use-cases with very demanding service latency requirements. Its failure rate goes up to 38% for 16 clients, while the lowest failure rate is observed for 8 clients (23%). However, the average distance from the best solution found by wither B&P or ILP does not exceed 1% for any feasible use-case. It manages to save 40%, 60% and 90% of the computation time of the least demanding optimal approach (whichever is faster, ILP or B&P) for 16, 32 and 64 clients, respectively.

The main reason the results for latency-dominated use-cases are different from their bandwidth-dominated counterparts is that latency-dominated clients require $O(H)$ instead of $O(H^2)$ constraints. Therefore, the ILP model is able to quickly find a solution for larger problems and, as a consequence, B&P starts to be faster only from 64 clients.

For the latency-dominated use-cases, we conclude that B&P shows better results

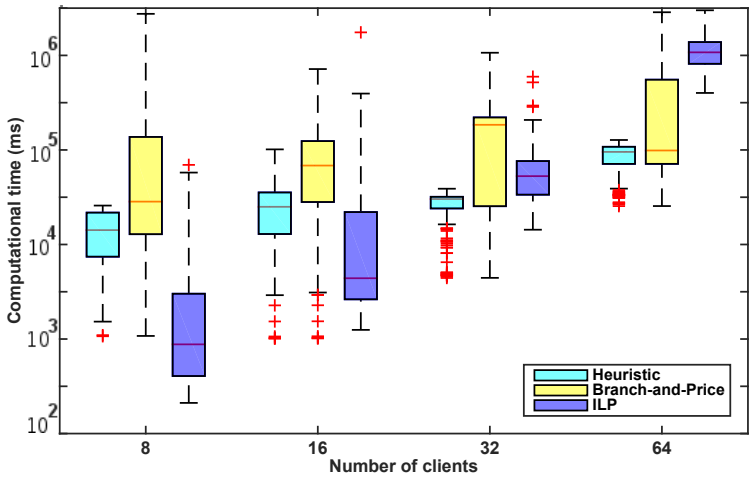


Figure 2.7: Computation time distribution for the latency-dominated use-cases.

than the ILP starting from larger problem instances with 64 clients. The heuristic saves more time than B&P, sacrificing approximately the same number of solvable use-cases with increasing size of problem instances.

Mixed-dominated use-cases

Finally, experimental results for the set of mixed-dominated use-cases is shown in Figure 2.8.

Again, the ILP model fails to scale to use-cases with 32 clients, not even a feasible solution was found within a given time limit in 172 out of 200 use-cases, which means a failure rate above 85%. Thus, the results for 32 and 64 clients are only represented by the heuristic on the left and B&P on the right. For the use-cases with 8 clients, the ILP model outperforms B&P in terms of total computation time, although B&P is slightly better in terms of median of the computation time. However, B&P is not able to prove the optimality within the time limit for 3 use-cases, while the ILP failed only once. For 16 clients, the ILP model runs in 20 hours, while the branch-and price approach needs less than 5 hours, saving 77% of the computation time. For 32 and 64 clients, B&P requires on average less than 1 and 5 minutes for one use-case, respectively, demonstrating improved scalability.

For the mixed-dominated use-cases, the heuristic fails to find a feasible solution in 23 and 26 use-cases for 8 and 16 clients, respectively, and saves 95% and 88% of the computation time of the fastest optimal approach. The heuristic shows good results, especially on the use-cases with 32 and 64 clients, where it is able to solve almost all of them in 90 minutes and 14 hours, respectively, leaving only the proof of optimality to the B&P approach. This result is caused by having more latency-dominated clients in the sets with 8 and 16 clients than in the sets of 32 and 64 clients, which makes the work for the heuristic easier. For the use-cases with 128 clients, the average time of running the heuristic 8 times is 75 minutes, which

already exceeds the given time limit.

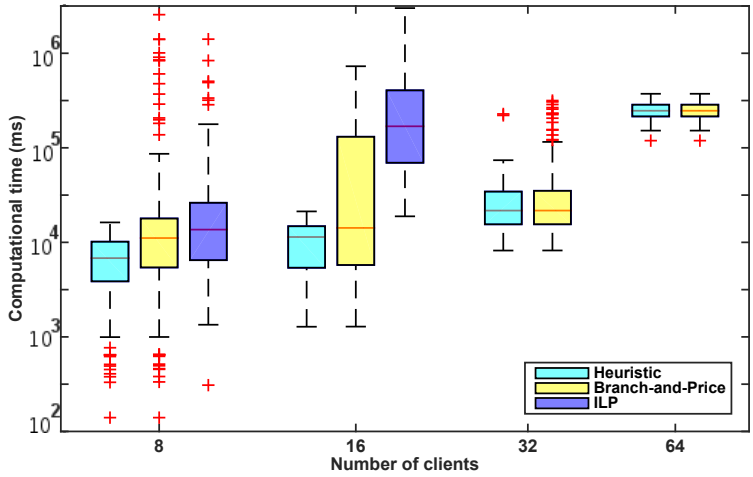


Figure 2.8: Computation time distribution for the mixed-dominated use-cases.

The results for the mixed-dominated use-cases show significant reduction of computation time of the B&P approach compared to the ILP model, starting approximately from use-cases with 16 clients, while giving optimal solutions for all synthetic use-cases. The heuristic saves 32% of computation time on average, sacrificing approximately 8% of the solvable use-cases.

Conclusion from experiments

From these experiments, we confirm the exponential complexity of the problem, although our implementation solves an instance with 64 clients and 512 slots in less than 8 minutes on average for all type of use-cases. Moreover, the ILP model is not able to solve use-cases with more than 16 clients for bandwidth-dominated and mixed-dominated use-cases and 32 clients for latency-dominated use-cases. Thus, the B&P approach is better for more complex use-cases, while ILP typically shows better results for the use-cases with smaller number of clients. More specifically, the proposed B&P approach improves scalability from 16 to 128 clients for bandwidth-dominated use-cases and from 16 to 64 clients for mixed-dominated use-cases, while latency-dominated use-cases remain unchanged at 64 clients. In total, the work in this chapter improves scalability with approximately a factor 8.

To further improve the scalability of the B&P approach, it is necessary to apply additional advanced methods on the given problem. Alternatively, it is possible to use the proposed heuristic, which enables saving up to 95% of the computation time with loss of maximally 38% of feasible use-cases, being 1% in distance from the optimal solution on average. In contrast, the commonly used continuous slot assignment algorithm [37, 42, 44, 117] failed to find *any* feasible solution in all 3 000 use-cases. This simple yet common heuristic is hence unable to cover any use-cases of reasonable complexity. Finally, the heuristic presented in this chapter is not able

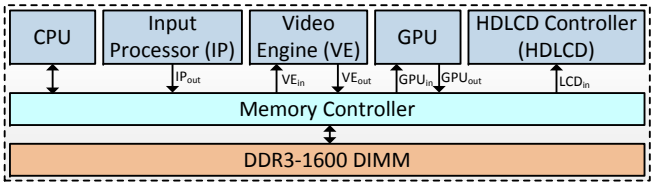


Figure 2.9: Architecture of the HD video and graphics processing system.

to find any feasible solution for some use-cases, and we believe that it is a limitation given by the usage of generative approach only.

2.8.3 Case Study

We now proceed by demonstrating the practical applicability of our proposed TDM configuration methodology by applying it to a small case study of an HD video and graphics processing system, where 7 memory clients share a 64-bit DDR3-1600 memory DIMM [55]. The considered system is illustrated in Figure 2.9.

Similarly to the multi-channel case study in [40], we derive the client requirements from a combination of the industrial systems in [102, 106] and information about the memory traffic of the decoder application from [21]. However, we assume 720p resolution instead of 1 080p and that all memory requests have a fixed size of 128 B to be able to satisfy the requirements with a single memory channel.

The Input Processor receives an H.264 encoded YUV 4:2:0 video stream with a resolution of 720 × 480, 12 bits per pixel (bpp), at a frame rate of 25 frames per second (fps) [102], and writes to memory (IP_{out}) at less than 1 MB/s. The Video Engine (VE) generates traffic by reading the compressed video and reference frames for motion compensation (VE_{in}), and writing decoder output (VE_{out}). The motion compensation requires at least 285.1 MB/s to decode the video samples at a resolution of 1280 × 720, 8 bpp, at 25 fps [21]. The bandwidth requirement to output the decoded video image is 34.6 MB/s.

The GPU is responsible for post-processing the decoded video. The bandwidth requirement depends on the complexity of the frame, but can reach a peak bandwidth of 50 MB/frame in the worst case [106]. Its memory traffic can be split into pixels read by the GPU for processing (GPU_{in}) and writing the frame rendered by the GPU (GPU_{out}). For GPU_{in}, we require a guaranteed bandwidth of 1 000 MB/s, which should be conservative given that the peak bandwidth is not required continuously. GPU_{out} must communicate the complete uncompressed 720p video frame at 32 bpp within the deadline of 40 ms (25 fps). With a burst size of 128 B, this results in a maximum response time (finishing time - arrival time) of 1388 ns per request. To provide a firm guarantee that all data from this client arrives before the deadline, we separate this into a service latency and a rate requirement according to the LR server approach. There are multiple (Θ, ρ) pairs that can satisfy a given response time requirement according to Equation (2.2), where a higher required bandwidth results in a more relaxed service latency requirement. Here, we require a bandwidth of 184.3 MB/s, twice the continuous bandwidth that is needed, to budget time for interference from other clients. According to Equation (2.2), this results in a service

latency requirement of 718 ns (574 clock cycles for an 800 MHz memory).

The HDLCD Controller (HDLCD) writes the image processed by the GPU to the screen. It is latency critical [102] and has a hard deadline to ensure that data arrives in the frame buffer before the screen is refreshed. Similarly to GPU_{out}, HDLCD requires at least 184.3 MB/s to output a frame every 40 ms. Note that each rendered frame is displayed twice by the HDLCD controller to achieve a screen refresh rate of 50 Hz with a frame rate of 25 fps. Lastly, a host CPU and its associated Direct Memory Access (DMA) controller also require memory access with a total bandwidth of 150 MB/s to perform system-dependent activities [106].

The derived requirements of the memory clients in the case study are summarized in Table 2.3. We conclude the section by explaining how to transform the requirements into the abstract units of rate and service latency (in slots) used by our approach. The rate is determined by dividing the bandwidth requirement of the client with the minimum guaranteed bandwidth provided by the memory controller. The service latency requirement in slots is computed by dividing the latency requirement in clock cycles by the Worst-Case Execution Time (WCET) of a memory request. Given a request size of 128 B and assuming the real-time memory controller in [3], the WCET of a memory request to a DDR3-1 600 is 46 clock cycles at 800 MHz and the memory guarantees a minimum bandwidth of 2 149 MB/s [43]. For simplicity, we ignore effects of refresh interference in the memory, which may increase the total memory access time over a video frame with up to 3.5% for this memory. The total required bandwidth of the clients in the case study is 1 839.3 MB/s. This corresponds to 85.6% of the guaranteed bandwidth of the memory controller, suggesting a suitably high load. In this use-case, all clients are bandwidth dominated.

Table 2.3: Client requirements in the case study

Client	*Bandwidth [MB/s]*	*Latency [cc]*	$\hat{\rho}$	$\hat{\Theta}[slots]$
IP_{out}	1.0	-	0.0005	-
VE_{in}	285.1	-	0.1326	-
VE_{out}	34.6	-	0.0161	-
GPU_{in}	1000.0	-	0.4652	-
GPU_{out}	184.3	574	0.0858	12.5
LCD_{in}	184.3	574	0.0858	12.5
CPU	150.0	-	0.0698	-
Total	1839.3		0.8558	

We apply our configuration methodologies to find the optimal TDM schedule to satisfy the client requirements, while minimizing the total allocated bandwidth. The frame size is set to 64, which ensures that the use-case is solvable and provides a reasonable trade-off between access granularity and total TDM schedule size for the number of clients in the case study. Although the size of the problem is rather small, the B&P approach results in more than 10 times reduction of the computation time compared to the ILP model. More specifically, the B&P approach requires 138 milliseconds, while the ILP model finishes in 1 395 milliseconds. The reason B&P is faster even though the case study is small is that the heuristic provides an optimal solution, as it typically does for bandwidth-dominated use-

cases, significantly reducing the computation time of the approach. From this case study, we conclude that *B&P can be advantageous not only for larger models, but also for the models of smaller sizes.*

2.9 Summary

This chapter introduces two optimal and a heuristic approach to configure single resources shared by Time-Division Multiplexing (TDM) to satisfy the bandwidth and latency requirements of real-time clients. This is done while minimizing their total allocated rate (bandwidth) to improve the average performance of non-real-time clients. The problem considered here is to assign TDM slots to the clients, i.e., to find a TDM schedule with a given length. To solve the TDM configuration problem, we propose an integer linear programming problem (ILP) and an optimal approach that takes this model and wraps it in a branch-and-price (B&P) framework to reduce its computation time and thereby improve its scalability. We further present computation time improvements for both approaches. They use problem-specific properties to remove unnecessary constraints and reduce the solution space. In addition, a stand-alone generative heuristic that quickly finds a schedule is proposed for cases where an optimal solution is not required. This is also used to provide promising initial schedules for the B&P approach.

We experimentally evaluate the scalability of the ILP and the B&P approaches and quantify the trade-off between computation time and solution quality for the proposed optimal and heuristic algorithms on three groups of synthetic use-cases, representing different groups of possible bandwidth and latency requirements of the clients. The three groups are: 1) bandwidth-dominated clients with more intense bandwidth requirement than latency requirement; 2) latency-dominated clients where the latency requirement is more demanding; and 3) mixed-dominated clients with equally demanding bandwidth and latency requirements. The results show that the computation time of all approaches depends on the type of the group, being the shortest for the use-cases with bandwidth-dominated clients and the longest for the ones with mixed-dominated clients for the same size of use-cases. This can be intuitively explained by the complexity and the number of constraints involved. Whereas for bandwidth dominated clients due to one of the computation time improvements we can remove the most demanding constraints, for mixed-dominated clients these constraints are present. The results on all use-cases also show that the heuristic provides near-optimal solutions in 86% of the use-cases with an average allocated bandwidth less than 0.26% from the optimum using less than 50% of the time of the fastest optimal approach (either ILP or B&P). For example, the average time for the heuristic to solve one problem instance with 64 clients and 512 TDM slots is less than 2 minutes. However, we believe that the efficiency of the heuristic is limited by the usage of the generative approach only and can be improved by incorporating a constructive component to the approach as well.

Finally, throughout the experiments, the B&P approach outperforms the ILP model on larger use-cases, improving the scalability by a factor of 8, or more concretely from 16 to 128 clients. On the other hand, the ILP model shows better results on some use-cases of modest size and, therefore, it is used in the heuristic to

solve sub-problems of smaller sizes. Finally, we demonstrate the practical relevance of our approach by applying it to a case study of an HD video and graphics processing system.

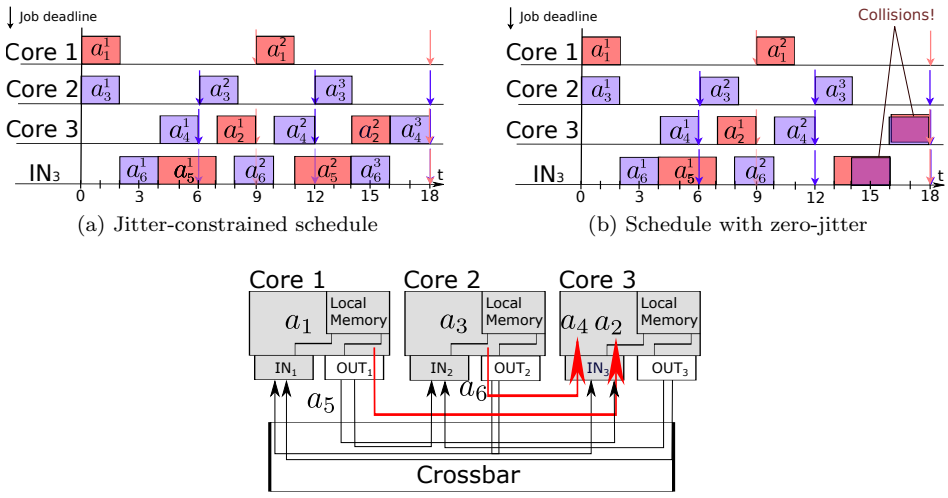
Chapter 3

Computation and Communication Coscheduling

Whereas the previous chapter focused on the scheduling of a single independent resource in consumer electronics domain, where the applications have soft real-time requirements, in this chapter, we focus on coscheduling of multiple resources and safety-critical applications with hard real-time requirements running on these resources in the automotive domain. The functionality of such applications is realized by a set of tightly coupled periodic tasks that communicate with each other either over an on-chip or off-chip interconnect, such as buses, networks, or crossbars. Additionally, these embedded applications are required to realize many end-to-end control functions within predefined time bounds, while also executing the constituent tasks in a specific order.

The considered problem is illustrated in Figure 3.1c, where tasks a_1, a_2, a_3, a_4 are mapped to Cores 1 to 3, where each core has its local memory and communicate via a crossbar switch. This architecture is inspired by Infineon AURIX TriCore [1], and it is also captured in a more general form in Figure 1.2c of Chapter 1. The crossbar switch is assumed to be a point-to-point connection that links an output port of each core with input ports of the remaining cores. Although there is *no contention on output ports* since tasks on cores are statically scheduled, *scheduling of the incoming messages on the input ports* must be done to prevent contention. Moreover, there are two chains of dependencies, indicated by thicker (red) arrows, i.e., $a_1 \rightarrow a_5 \rightarrow a_2$ and $a_3 \rightarrow a_6 \rightarrow a_4$. Note that although this example contains 6 resources to be scheduled, the only input port that must be scheduled in this case is the one of Core 3, since there are no incoming messages to other cores.

Recent works on Time-Triggered (TT) scheduling mostly consider *zero-jitter (ZJ) scheduling* [29, 45, 72, 104, 123] also called strictly periodic scheduling, where the start time of an activity is at a fixed offset (instant) in every period. If there are two consecutive periods in which the activity is scheduled at different times (relative to the period), we call it *Jitter-Constrained (JC) scheduling*. On the one hand, ZJ scheduling results in lower memory requirements, since the schedule takes less space to store and typically needs less time to find an optimal solution. On the other hand, it puts strict requirements on a schedule, causing many problem instances to be infeasible, as we later show in Section 3.5. This may lead to increasing requirements on the number of cores for the given application, thus making the system more expensive. Even though some applications or even systems, are restricted to being ZJ, e.g., some systems in the avionics domain [7], many systems in the automotive domain allow JC scheduling [38, 89]. Therefore, this chapter explores the trade-off between JC and ZJ scheduling. Although not all activities have ZJ requirements, some of them are typically sensitive to the delay between consecutive occurrences, since it greatly influences the quality of control [13, 33, 68]. Assuming *constrained jitter* instead of ZJ scheduling allows the resulting schedule to both satisfy strict jitter requirements of the jitter-critical activities and to have more freedom to schedule their non-jitter-critical counterpart.



(a) Jitter-constrained schedule

(b) Schedule with zero-jitter

(c) Problem description

Figure 3.1: Coscheduling problem description with examples of zero-jitter and jitter-constrained solution, where a_5 is a message between a_1 and a_2 and a_6 is a message between a_3 and a_4.

The example JC schedule in Figure 3.1a considers the scheduling problem illustrated in Figure 3.1c. It assumes that activities a_1, a_2 and a_5 have a required period of 9 time units, while a_3, a_4, and a_6 must be scheduled with a period of 6. The resulting JC schedule in Figure 3.1a has a hyper-period (length) of 18 time units, which is the least common multiple of both periods. Hence, activities a_1, a_2 and a_5 are scheduled 2 times and activities a_3, a_4, and a_6 are scheduled 3 times during one hyper-period, defining its number of *jobs*, i.e., activity occurrences. Note that activities a_2 and a_5 are not scheduled with zero-jitter since a_2 in the first period is scheduled at time 7, while in the second period at time 5 (+9). Similarly, a_5 is scheduled at different times in the first and second periods (4 and 2 (+9), respectively), thus with jitter equal to 2. In contrast, Figure 4.3 illustrates that using ZJ scheduling results in collisions between a_2 and a_4 on Core 3, and between a_5 and a_6 in the crossbar switch. Moreover, an exact approach (see SMT formulation in Section 4.4.1) can prove that the instance is infeasible with ZJ scheduling. Thus, if an application can tolerate some jitter in the execution of activities a_2 and a_5 without unacceptable quality degradation of control, then the system cost could be reduced, as shown in Figure 3.1a.

The three main contributions of this chapter are: 1) Two models, one Integer Linear Programming (ILP) formulation and one Satisfiability Modulo Theory (SMT) model with problem-specific improvements to reduce the complexity and the computation time of the formulations. The two models are proposed due to significantly different computation times on problem instances of low and high complexity, respectively. The formulations optimally solve the problem for smaller applications with up to 50 activities in a reasonable time. 2) A constructive heuristic approach, called 3-LS, employing a three-step approach that scales to

real-world use-cases with more than 10000 activities. 3) An experimental evaluation of the proposed solution for different jitter requirements on synthetic data sets that quantifies the computation time and resource utilization trade-off and shows that relaxing jitter constraints enables an average increase in resource utilization of up to 28% for the price of up to 10 times longer computation time. Moreover, the 3-LS heuristic is demonstrated on a case study of an engine management system, which it successfully solves in 43 minutes.

The rest of this chapter is organized as follows: the related work is discussed in Section 3.1. Section 3.2 proceeds by presenting the system model and the problem formulation. The description of the ILP and SMT formulations and their computation time improvements follow in Section 3.3. Section 3.4 introduces the proposed heuristic approach for scheduling periodic activities, and Section 3.5 proceeds by presenting the experimental evaluation before the chapter is concluded in Section 3.6.

3.1 Related Work

Even though this chapter targets the TT approach, the survey of related work would not be complete without mentioning works that consider the Event-Triggered (ET) paradigm. A broad survey of works related to periodic (hard real-time) scheduling is provided by Davis and Burns in [31]. Next, Baruah et al. [15] introduce the notion of Pfair schedules, which relates to the concept of ZJ scheduling while scheduling preemptively, i.e. where execution of an activity can be preempted by another activity. Similarly to the ZJ approach that requires the time intervals equal to execution times of activities to be scheduled equidistantly in consecutive periods as a whole, Pfair requires equidistant allocation, while scheduling by intervals of one time unit. On the non-preemptive scheduling front, Jeffay at al. [56] propose an approach to schedule periodic activities on a single resource with precedence constraints. The problem of coscheduling tasks and messages in an event-triggered manner is considered in [54, 59, 112]. However, these works do not consider jitter constraints, as done in our research [75].

The TT approach attracted the attention of many researchers over the past twenty years for solving the problem of periodic scheduling. The pinwheel scheduling problem [49] can be viewed as a relaxation of the JC scheduling concept, where each activity is required to be scheduled at least once during each predefined number of consecutive time units. If minimizing number of jobs, the solution of the pinwheel problem approaches the ZJ scheduling solution, since it tends to have an equidistant schedule for each activity. Moreover, the Periodic Maintenance Scheduling Problem [11] is identical to ZJ scheduling, as it requires jobs to be executed exactly a predefined number of time units apart.

Considering works that formulate the problem similarly, some authors deal with scheduling only one core [39, 79], while others focus only on interconnects [16, 34]. These works neglect precedence constraints and schedule each core or interconnect independently, unlike the coscheduling of cores and interconnects in this chapter. The advantage of coscheduling lies in synchronization between tasks executing on the cores and messages transmitted through an interconnect that results in high

efficiency of the system in terms of resource utilization. Steiner [104] introduces precedence dependencies between activities, while dealing with the problem of scheduling a TTEthernet network. However, Steiner assumes that all activities have identical processing times, which in our case will increase resource utilization significantly.

Some works do not put any constraints on jitter requirements, which is not realistic in the automotive domain, since there can be jitter-sensitive activities. Puffitsch et al. in [89] assume a platform with imperfect time synchronization and propose an exact constraint programming approach. Abdelzaher and Shin in [2] solve a similar problem by applying both an optimal and a heuristic branch-and-bound method. Furthermore, the authors in [86] consider the preemptive version of our problem that makes it impossible to apply their solution to the problem considered in this chapter, which assumes non-preemptive execution.

Jitter requirements are not considered in the problem formulations of [78] and [90], where the authors propose heuristic algorithms to deal with the coscheduling problem. Finally, in [33] the authors solve the considered problem with an objective to minimize the jitter of the activities using simulated annealing, while we assume a set of constraints rather than an objective. Note that these approaches with JC activities are heuristics and the efficiency of the proposed methods have not been compared to optimal solutions.

There also exist works that schedule both tasks and messages, while assuming ZJ scheduling. Lukasiewycz and Chakraborty [71] solve the coscheduling problem assuming the interconnect to be a FlexRay bus, which results in a different set of constraints. Their approach involves decomposing the initial problem and solving the smaller parts by an ILP approach to manage scalability. Besides, Lukasiewycz et al. in [72] introduce preemption into the model formulation. Moreover, Craciunas and Oliver [29] consider an Ethernet-based interconnect and solve the problem using both SMT and ILP. However, ZJ scheduling is less schedulable and requires more or faster resources, as shown in Section 3.5. In summary, *this work is different in that it is the first to consider the coscheduling problem of periodic applications with jitter requirements and solves it by a heuristic approach, whose quality is evaluated by comparing with the exact solution for smaller instances.*

3.2 System Model

This section first introduces the platform and the application models used in this chapter. Then, the mapping of activities to resources is described, concluded by the problem statement.

3.2.1 Platform Model

The considered platform comprises a set of homogeneous *cores* on a single multi-core Electronic Control Unit (Electronic Control Unit (ECU)) with a *crossbar switch*, as shown in Figure 3.1c. This is similar to the TriCore architecture [1]. The crossbar switch provides point-to-point connection between the cores, and input ports act as communication endpoints and can receive only a single message at a time. Tasks on different cores communicate via the crossbar switch that writes variables in local

memories of the receiving cores. On the other hand, intra-core communication is realized through reading and writing variables that are stored in the local memory of each core. The set of m resources that include $\frac{m}{2}$ cores and $\frac{m}{2}$ crossbar switch input ports is denoted by $U = \{u_1, u_2, \cdots, u_m\}$. Moreover, the cores are characterized by their clock frequency and available memory capacity.

Although this work focuses on multi-core systems with crossbar switches, the current formulation is easily extensible to distributed architectures with multiple single-core processing units, connected by a bus, e.g. CAN [22]. Furthermore, assuming systems with fully switched networks, e.g. scheduling of time-triggered traffic in TTEternet [103] leads to a similar scheduling problem as shown in Chapter 4.

3.2.2 Application Model

The application model is based on characteristics of realistic benchmarks of a specific modern automotive software system, provided in [65]. We model the application as a set of periodic *tasks* T that communicate with each other via a set of *messages* M, transmitted over the crossbar switch. Then $A = T \cup M$, denotes the set of activities, which includes both the incoming messages on the input ports of the crossbar switch and the tasks executed on the cores. Each activity a_i is characterized by the tuple $\{p_i, e_i, jit_i\}$ representing its period, execution time and jitter requirements, respectively. Its execution may not be preempted by any other activity, since *non-preemptive scheduling* is considered. The release date of each activity equals the start of the corresponding period and the deadline is the end of the next period. This deadline prolongation extends the solution space. The period of a message is set to the period of the task that sends the message. Additionally, execution time of messages on the input ports correspond to the time it takes to write the data to the local memory of the receiving core. Thus, since the local memories are defined by both their bandwidth and latency, execution time for each message $a_i \in M$ is calculated as $e_i = \frac{sz_i}{bnd} + lat$, where sz_i is the size of the corresponding transmitted variable given by the application model, while bnd is the bandwidth of the memory and lat is its latency given by the platform model. This is the same as the conservative case of latency-rate server concept presented in Chapter 2, when each request starts a new busy period. The benefit of this being modeled as an Latency-Rate (LR) server is that it covers a variety of arbitration policies that could be used for the local memories.

We estimate required memory to store the schedule by the total number of jobs of all activities, assuming that 8 bytes is enough to store start time of a job. Although there is an overhead to store the schedule, this estimation is reasonable for our purposes, since we compare schedules with different number of jobs running on the same platform. Thus, we estimate the amount of memory required to store the schedule as in Equation (3.1), where the number of jobs in the schedule n^{total} is multiplied by 8.

$$mem = n^{total} \cdot 8 \tag{3.1}$$

Cause-effect chains are an important part of the model. A cause-effect chain comprises a set of activities that must be executed in a given order within predefined time bounds to guarantee correct behavior of the system. As one activity can be

a part of more than one cause-effect chain, the resulting dependency relations are represented by a directed acyclic graph (Directed Acyclic Graph (DAG)) that can be very complex in real-life applications [85], such as automotive engine control. Similarly to [29] and [50], activities of one cause-effect chain are assumed to have the same period. More generalized precedence relations with activities of distinct periods being dependent on each other can be found in e.g. [36]. Thus, the resulting graph of precedence relations consists of distinct DAG's for activities with different periods, although not necessarily only one DAG for each unique period. An example of a precedence relation is shown in Figure 3.2, which includes the activities from Figure 3.1. Note that many activities may not have any precedence constraints, since they are not part of any cause-effect chain. For instance, it could be simple logging and monitoring activities.

Lastly, each cause-effect chain has an *end-to-end deadline* constraint, i.e. the maximum time that can lapse from the start of the first activity till the end of the execution of the last activity in each chain equal to two corresponding periods. However, as the first activity in each chain can be scheduled at the beginning of the period at the earliest and the last activity of the chain at the end of the next period at the latest, the end-to-end latency constraint is automatically satisfied due to release and deadline constraints of the activities. Therefore, end-to-end latency constraints do not add further complexity to the model.

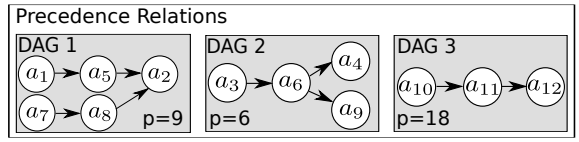


Figure 3.2: An example of the resulting precedence relations, where activities in DAG 1 have period 6, activities in DAG 2 have period 9 and activities in DAG 3 have period 18.

3.2.3 Mapping Tasks to Cores

The mapping $map : A \rightarrow U$, $map = \{map_1, \cdots, map_n\}$ of tasks to cores and messages to memories is assumed to be given by the system designer, which reflects the current situation in the automotive domain, e.g. for engine control. Note that for the previously discussed extension to fully switched network systems, both mapping and routing (i.e. define path through the network for each message) that respect some locality constraints are necessary. Since mapping influences routing and therefore message scheduling, for such systems it is advantageous to solve the three-steps at once, e.g., as done in [111].

To get a mapping for the problem instances used to validate the approaches in this chapter, a simple ILP model for mapping tasks to cores is formulated the following way. The variables $\mathbf{z}_{i,j} \in \{0, 1\}$ indicate whether task $i = 1, \cdots, |T|$ is mapped to resource $j = 1, \cdots, \frac{m}{2}$ ($\mathbf{z}_{i,j} = 1$) or not ($\mathbf{z}_{i,j} = 0$). Note that we consider only cores as resources and tasks on cores as activities while mapping. The mapping of messages follows implicitly form these. The mapping tries to balance the load, which is formulated as a sum of absolute values of utilization differences on two consecutive resources in Equation (3.2). Since the absolute operator is not linear, it needs to be linearized by introducing the load variables $o_j \in \mathbb{R}$ in

Equations (3.3) and (3.4).

$$\textit{Minimize:} \sum_{j \in 1,..,\frac{m}{2}} o_j \tag{3.2}$$

subject to:

$$o_j \geq \sum_{i=1,\cdots,|T|} \frac{e_i}{p_i} \cdot \mathbf{z}_{i,j} - \sum_{i=1,\cdots,|T|} \frac{e_i}{p_i} \cdot \mathbf{z}_{i,j+1},\ j = 1, \cdots, \frac{m}{2} \tag{3.3}$$

$$o_j \geq \sum_{i=1,\cdots,|T|} \frac{e_i}{p_i} \cdot \mathbf{z}_{i,j+1} - \sum_{i=1,\cdots,|T|} \frac{e_i}{p_i} \cdot \mathbf{z}_{i,j},\ j = 1, \cdots, \frac{m}{2}. \tag{3.4}$$

Moreover, each task must be mapped to a single resource, as stated in Equation (3.5).

$$\sum_{j=1,\cdots,\frac{m}{2}} \mathbf{z}_{i,j} = 1,\ i = 1, \cdots, |T|. \tag{3.5}$$

Note that this simple mapping strategy is not considered a contribution of this chapter, but only a necessary step to provide a starting point for the experiments, since the benchmark generator does not provide the mapping.

3.2.4 Problem Formulation

Given the above model, the goal is to find a schedule with a hyper-period $H = lcm(p_1, p_2, \cdots, p_n)$ with lcm being the least common multiple function, where the schedule is defined by start times $s_i^j \in \mathbb{N}$ of each activity $a_i \in A$ for every job $j = 1, 2, \cdots, n_i^{jobs}$, with $n_i^{jobs} = \frac{H}{p_i}$. The schedule must satisfy the periodic nature of the activities, the precedence relations and the jitter constraints. The considered scheduling problem is *multi-periodic non-preemptive scheduling of activities with precedence and jitter constraints on dedicated resources.*

The formal definition of a zero-jitter schedule is the following:

Definition 2 (Zero-jitter (ZJ) schedule)**.** *The schedule is a ZJ schedule if and only if for each activity a_i Equation* (3.6) *is valid, i.e. the difference between the start times s_i^j and s_i^{j+1} in each pair of consecutive periods j and $j+1$ over the hyper-period equals the period length.*

$$s_i^{j+1} - s_i^j = p_i,\ j = 1, 2, \cdots, n_i^{jobs} - 1. \tag{3.6}$$

Zero-jitter scheduling deals exclusively with ZJ schedules. If for some activity and some periods j and $j+1$ Equation (3.6) does not hold in the resulting schedule, we call it *jitter-constrained (JC) schedule.*

The scheduling problem, where a set of periodic activities are scheduled on one resource is proven to be NP-hard in [24] by transforming from the 3-Partition problem. Thus, the problems considered (both ZJ and JC) here are also NP-hard, since they are generalizations of the aforementioned NP-hard problem.

3.3 Exact Models

Due to significantly different timing behavior of the models on problem instances with varying complexity (see Section 3.5), both SMT and ILP models are formulated in this chapter. Moreover, the NP-hardness of the considered problem justifies using these approaches, since no polynomial algorithm exists to optimally solve the problem unless P=NP. This section first presents a minimal SMT formulation to solve the problem optimally, then continues with a linearization of the SMT model to get an ILP model. It concludes by providing improvements to both models that exploit problem-specific knowledge, reducing the complexity of the formulation and thus the computation time.

3.3.1 SMT Model

The SMT problem formulation is based on the set of variables $s_i^j \in \{1, 2, \cdots, H\}$, indicating a start time of job j of activity a_i. Following the problem statement in Section 3.2.4, we deal with a decision problem with no criterion to optimize. The solution space is defined by five sets of constraints. The first set of constraints is called *release date and deadline constraints* and it requires each activity to be executed in a given time interval of two periods, as stated in Equation (3.7).

$$\begin{aligned} (j-1) \cdot p_i \leq s_i^j \leq (j+1) \cdot p_i - e_i, \\ a_i \in A,\ j = 1, \cdots, n_i^{jobs}. \end{aligned} \tag{3.7}$$

The second set, Constraint (3.8), ensures that for each pair of activities a_i and a_l mapped to the same resource ($map_i = map_l$), it holds that either a_i^j is executed before a_l^k or vice-versa. These constraints are called *resource constraints*. Note that due to the extended deadline in Constraint (3.7), the resource constraints must be added also for jobs in the first period with jobs of the last period, since they can collide.

$$\begin{aligned} s_i^j + e_i \leq s_l^k \vee s_l^k + e_l \leq s_i^j, \\ s_i^1 + e_i + H \leq s_l^{n_l^{jobs}} \vee s_l^{n_l^{jobs}} + e_l \leq s_i^1 + H, \\ a_i,\ a_l \in A : map_i = map_l,\ j = 1, ..., n_i^{jobs},\ k = 1, ..., n_l^{jobs}. \end{aligned} \tag{3.8}$$

For the ZJ case, it is enough to formulate Constraints (3.8) for each pair of activities only for jobs in the least common multiple of their periods, i.e. $j = 1, \cdots, \frac{lcm(p_i,p_j)}{p_i}$ and $k = 1, \cdots, \frac{lcm(p_i,p_j)}{p_l}$. Moreover, the problem for ZJ scheduling is formulated using n variables. One variable s_i^1 is defined for the first job of each activity and other jobs are simply rewritten as $s_i^j = s_i^1 + p_i \cdot (j-1)$.

The next set of constraints is introduced to prevent situations where two consecutive jobs of one activity collide. Thus, Constraint (3.9) introduces precedence constraints between each pair of consecutive jobs of one activity, considering also

the last and the first job.

$$\begin{aligned} s_i^j + e_i &\leq s_i^{j+1}, \\ s_i^{n_i^{jobs}} + e_i &\leq s_i^0 + H, \\ a_i \in A,\ j &= 1, \cdots, n_i^{jobs} - 1. \end{aligned} \tag{3.9}$$

Next, due to the existence of cause-effect chains, *precedence constraints* are formulated in Equation (3.10), where $Pred_i$ denotes the set of directly preceding activities of a_i.

$$\begin{aligned} s_i^j + e_i &\leq s_l^j, \\ a_i, a_l \in A : a_l \in Pred_i,\ j &= 1, \cdots, n_i^{jobs}. \end{aligned} \tag{3.10}$$

The *jitter constraints* can be formulated either in terms of *relative jitter*, where we bound only the difference in start times of jobs in consecutive periods or in terms of *absolute jitter*, bounding the start time difference of any two jobs of an activity. Experiments have shown that defining jitter as absolute or relative does not significantly influence the resulting efficiency. The difference in terms of maximal achievable utilization is less than 1% on average with relative jitter showing higher utilization. Therefore, further in the chapter we use the relative definition of jitter. Note that the results for absolute jitter formulation do not differ significantly from the results presented in this chapter. The formulation of relative jitter is given in Equation (3.11), where the first constraint deals with jitter requirements of jobs inside one hyper-period and the second one deals with jobs crossing a border between two hyper-periods.

$$\begin{aligned} |s_i^j - (s_i^{j-1} + p_i)| &\leq jit_i, \\ |(s_i^1 + H) - (s_i^{n_i^{jobs}} + p_i)| &\leq jit_i, \\ j = 2, \cdots, n_i^{jobs},\ a_i &\in A. \end{aligned} \tag{3.11}$$

3.3.2 ILP Model

The formulation of the ILP model is very similar to the SMT model described above. The main difference in formulation is caused by the requirement of linear constraints for the ILP model. Thus, since Equations (3.7), (3.9) and (3.10) are already linear, they can be directly used in the ILP model. However, resource Constraints (3.8) are non-linear and to linearize them, we introduce new set of decision variables that reflect the relative order of each two jobs of different activities:

$$x_{i,l}^{j,k} = \begin{cases} 1, & \text{if } a_i^j \text{ starts before } a_l^k; \\ 0, & \text{otherwise.} \end{cases}$$

Therefore, resource constraints are formulated by Equation (3.12), which ensures that either a_i^j is executed before a_l^k (the first equation holds and $x_{i,l}^{j,k} = 1$) or vice-versa (the second equation holds and $x_{i,l}^{j,k} = 0$). However, exactly one of

these equations must always hold due to binary nature of $x_{i,l}^{j,k}$, which prevents the situation where two activities execute simultaneously on the same resource. Note that we use $2 \cdot H$ in the right part of the constraints, since the maximum difference between two jobs of distinct activities can be maximally $2 \cdot H$ due to release date and deadline constraints.

$$\begin{aligned} s_i^j + e_i &\leq s_l^k + 2 \cdot H \cdot (1 - x_{i,l}^{j,k}), \\ s_l^k + e_l &\leq s_i^j + 2 \cdot H \cdot x_{i,l}^{j,k}, \\ a_i,\ a_l \in A : map_i = map_l, j = 1, ..., n_i^{jobs},\ k &= 1, ..., n_l^{jobs}. \end{aligned} \tag{3.12}$$

Furthermore, to formulate the jitter constraints (3.11) in a linear form, the absolute value operator needs to be eliminated. As a result, Equation (3.13) introduces four sets of constraints, two for the jobs inside one hyper-period and two for the jobs on the border.

$$\begin{aligned} s_i^j - (s_i^{j-1} + p_i) &\leq jit_i, \\ s_i^j - (s_i^{j-1} + p_i) &\geq -jit_i, \\ (s_i^1 + H) - (s_i^{n_i^{jobs}} + p_i) &\leq jit_i, \\ (s_i^1 + H) - (s_i^{n_i^{jobs}} + p_i) &\geq -jit_i, \\ j = 2, \cdots, n_i^{jobs},\ a_i &\in A. \end{aligned} \tag{3.13}$$

Unlike the time-indexed ILP formulation [62] used in Chapter 2, where each variable $t_{i,j}$ indicates that the activity i is scheduled at time j (having $H \cdot n$ variables), the approach used here can solve problems with large hyper-periods when there are fewer jobs with longer execution time. Hence, it utilizes only $n_{jobs} + \frac{n_{jobs} \cdot (n_{jobs}-1)}{2}$ with $n_{jobs} = \sum_{i=1}^{n} n_i^{jobs}$ variables, which is a fraction of the variables that the time-indexed formulation requires for this problem.

3.3.3 Computation Time Improvements

While the basic formulations of the SMT and ILP models were presented previously, four computation time improvements for the models are introduced here to reduce the complexity of the formulation and computation time of the solver. Note that the improvements do not break the optimality of the solution.

The first improvement *removes redundant resource constraints*. Due to the release date and deadline constraints (3.7), it is known that for two activities a_i, a_l start times of their jobs j and k, respectively, are in the following ranges: $s_i^j \in [(j-1) \cdot p_i, (j+1) \cdot p_i - e_i]$ and $s_l^k \in [(k-1) \cdot p_l, (k+1) \cdot p_l - e_l]$. Therefore, it is necessary to include resource constraints only if the intervals overlap. This improvement results in more than 20% of the resource constraints being eliminated, reducing the computation time significantly since the number of resource constraints grows quadratically with the number of activities mapped to a given resource.

Instead of setting the release date and deadline constraint (3.7), the second improvement provides this information directly to the solver. Thus, each constraint is substituted by setting the lower bound of s_i^j on $(j-1) \cdot p_i$ and the upper bound

on $(j+1)\cdot p_i - e_i$. Hence, instead of assuming the variables s_i^j in interval $[1, \cdots, H]$ and pruning the solution space by the periodicity constraints, the solver starts with tighter bounds for each variable. This significantly cuts down the search space, thereby reducing computation time. Due to the different solver abilities for SMT and ILP, this optimization is only applicable to the ILP model.

We can further *refine the lower and upper bounds* of the variables by exploiting knowledge about precedence constraints, which is the third improvement. For each activity, the length of the longest critical path of the preceding and succeeding activities that must be executed *before* and *after* the given activity, t^b and t^a respectively, are computed. First, the values of $\underline{t_i^b}$ and $\underline{t_i^a}$ are obtained by adding up the execution times of the activities in the longest chain of successors and predecessors of the activity a_i, respectively, as proposed by [27]. For the example in Figure 3.2, assuming the execution times of all activities are equal to 1, $\underline{t_1^b} = 0$, $\underline{t_1^a} = 2$, $\underline{t_6^b} = 1$, $\underline{t_6^a} = 1$, $\underline{t_2^b} = 2$, $\underline{t_2^a} = 0$. Additionally, the bounds can be improved by computing the sum of execution times of all the predecessors, mapped to the same resource, i.e.

$$t_i^b = \max\Big(\sum_{l:\ l\in Pred_i,\ map_l=map_i} e_l, \underline{t_i^b}\Big), \tag{3.14}$$

$$t_i^a = \max\Big(\sum_{l:\ l\in Succ_i,\ map_l=map_i} e_l, \underline{t_i^a}\Big). \tag{3.15}$$

For the example in Figure 3.2 and a single core, the resulting values are the following: $t_1^b = 0$, $t_1^a = 2$, $t_6^b = 1$, $t_6^a = 2$, $t_2^b = 4$, $t_2^a = 0$. Hence, the lower bound of s_i^j can be refined by adding t_i^b and the upper bound can be tightened by subtracting t_i^a, i.e. $s_i^j \in [(j-1)\cdot p_i + t_i^b, (j+1)\cdot p_i - e_i - t_i^a]$. This can also be used in the first improvement, eliminating even more resource constraints.

The fourth and final improvement *removes jitter constraints* (3.13) *for activities with no freedom to be scheduled with larger jitter than required.* For instance, for jobs of a_2 from Figure 3.2 with $e_2 = 1$, $t_2^b = 4$, $t_2^a = 0$ and $p_2 = 9$, there are only 14 instants t, where it can be scheduled, i.e. $t \in \{4, \cdots, 17\}$. If $jit_2 \geq 13$, the jitter constraint can be omitted since the activity can be scheduled only at 14 instants due to the third improvement and it is not possible to have jitter bigger than 13 time units and still respect the periodicity of the activity. We denote by I_i the worst-case slack of the activity, i.e. the lower bound on the number of time instants where activity a_i can be scheduled and we compute it according to Equation (3.16). Hence, the jitter constraints are only kept in the model if Inequality (3.17) holds, i.e. the activity has space to be scheduled with larger jitter than required. We refer to an activity satisfying Equation (3.17) *jitter-critical*. Otherwise, it is a *non-jitter-critical* activity.

$$I_i = 2\cdot p_i - (t^b + t^a + e_i) + 1, \tag{3.16}$$

$$jit_i \leq I_i - 2,\ a_i \in A. \tag{3.17}$$

Experimental results have shown that even on smaller problem instances with 40-55 activities, the proposed improvements reduce computation time by up to 30 times for the ILP model and 12 times for the SMT model. Moreover, the first and the

third improvements result in the most significant reduction of the computation time. However, when experimentally comparing these two improvements, we see that the behavior is rather dependent on the problem instance characteristics, as both the first and the third improvements can be the most effective on different problem instances.

3.4 Heuristic Algorithm

Although the proposed optimal models solve the problem exactly, this section introduces a heuristic approach to solve the problem in reasonable time for *larger instances*, possibly sacrificing the optimality of the solution within acceptable limits.

3.4.1 Overview

The proposed heuristic algorithm, called *3-Level Scheduling (3-Level Scheduling (3-LS))* heuristic, creates the schedule constructively. It assigns the start time to every job of an activity. Moreover, it implements 3 levels of scheduling, as shown in Figure 3.3. The first level inserts activity after activity into the schedule, while removing some of the previously scheduled activities, a_u, if the currently scheduled activity a_s cannot be scheduled. However, in case the activity a_u to be removed had problems being scheduled in previous iterations, the algorithm goes to the second level, where two activities that were problematic to schedule, a_s and a_u are *scheduled simultaneously*. By scheduling these two activities together, we try to avoid problems with a sensitive activity further in the scheduling process. Simultaneous scheduling of two activities means that two sets of start times s_c and s_u are decided for activities a_s and a_u concurrently. The third scheduling level is initiated when even coscheduling two activities a_s and a_u simultaneously does not work. Then, the third level starts by removing all activities except the ones that were already scheduled by this level previously and their predecessors. Next, it co-schedules the two problematic activities again. Note that although there may be more than two problematic activities, the heuristic always considers maximally two at once.

Having three levels of scheduling provides a good balance between solution quality and computation time, since the effort to schedule problematic activities is reasonable to not prolong the computation time of the approach and to get good quality solutions. Experimental results show that 94% of the time is spent in the first scheduling level, where the fastest scheduling takes place. However, in case the first level does not work, the heuristic algorithm continues with the more time demanding second scheduling level and according to the experimental results it spends 3% of time in this level. The final 3% of the total computation time is spent in the third scheduling level that prolongs the computation time the most since it unschedules nearly all the activities scheduled before. Thus, three levels of scheduling is a key feature to make the heuristic algorithm cost efficient and yet still able to find a solution most of the time. As seen experimentally in Section 3.5, it suffices to find a good solutions for large instances within minutes.

Note that the advantage of scheduling all jobs of one activity at a time compare to scheduling by individual jobs lies in the significantly reduced number of entities

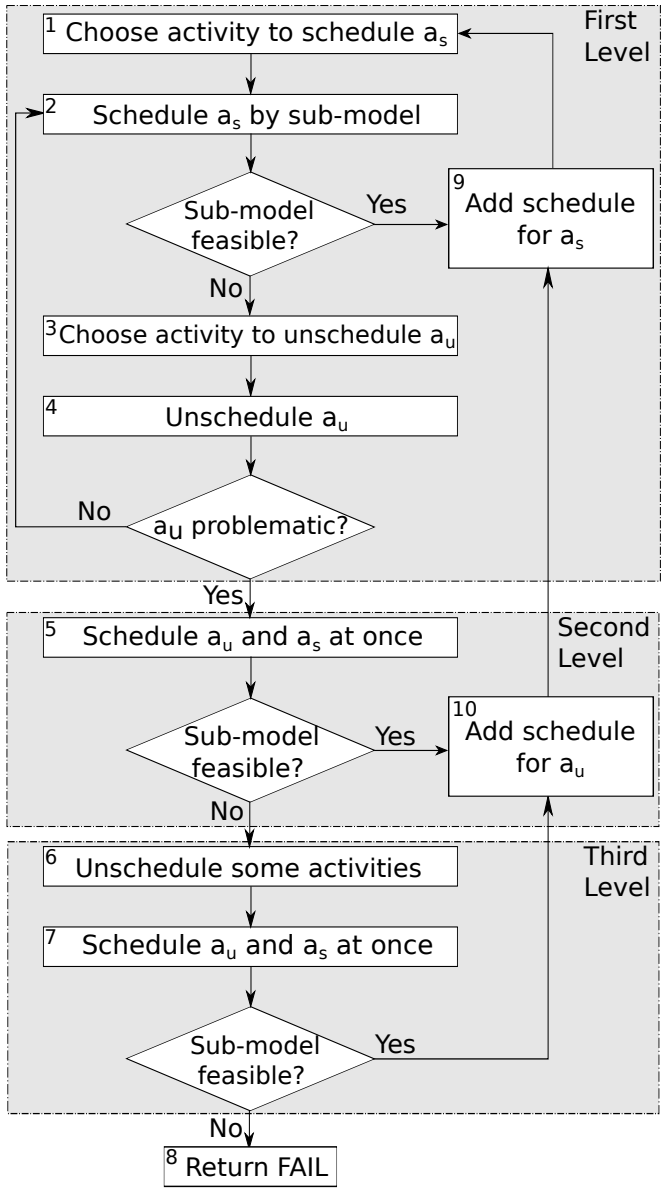


Figure 3.3: Outline of 3-Level Scheduling heuristic.

we need to schedule. Hence, unlike the exact model that focus on scheduling jobs for all of the activities at a time, the 3-LS heuristic approach decomposes the problem to smaller sub-problems for one activity. This implies that the 3-LS heuristic is not optimal and is also a reason why it takes significantly less time to solve the problem.

3.4.2 Sub-model

The schedule for a single activity or two activities at the same time, respecting the previously scheduled activities, is found by a so called *sub-model*, conceptually similar to the sub-model used in branch-and-price approach in Section 2.5. The sub-model for one activity a_i that is non-jitter-critical (i.e. a_i for which Inequality (3.17) does not hold) is formulated as follows. The minimization criterion is the sum of the start times of all a_i jobs (Equation (3.18)). The reasons for scheduling activities as soon as possible are twofold. Firstly, it is done for dependent activities to extend the time interval in which the successors of the activity can be scheduled, thereby increasing the chances for this DAG component to be scheduled. Secondly, scheduling at the earliest instant helps to reduce the fragmentation of the schedule, i.e. how much free space in the schedule is left between any two consecutively scheduled jobs, resulting in better schedulability in case the periods are almost harmonic, i.e. being multiples of each other, which is common in the automotive domain [65].

$$\textit{Minimize:} \sum_{j \in 1..n_i^{jobs}} s_i^j. \tag{3.18}$$

Note that the activity index i is always fixed in the sub-model since it only schedules a single activity at a time.

The start time of each job j can take values from the set D_i^j (Equation (3.20)), which is the union of intervals, i.e.

$$\begin{gathered} D_i^j = \{[l_1^{i,j}, r_1^j] \cup [l_2^{i,j}, r_2^{i,j}] \cup \cdots \cup [l_w^{i,j}, r_w^{i,j}]\}, \\ r_o^{i,j} < l_{o+1}^{i,j},\ l_o^{i,j} \leq r_o^{i,j},\ j = 1, \cdots, n_i^{jobs},\ o = 1, \cdots, w-1. \end{gathered} \tag{3.19}$$

and $\{l_1^{i,j}, r_1^{i,j}, \cdots, l_w^{i,j}, r_w^{i,j}\} \in \mathbb{Z}^{2 \cdot w}$, where w is the number of intervals in D_i^j and $l_o^{i,j}, r_o^{i,j}$ are the start and end of the corresponding interval o. This set of candidate start times is obtained by applying periodicity constraints (3.7) and precedence constraints (3.10) to already scheduled activities and changing the resulting intervals so that the activity can be executed fully with respect to its execution time. Note that since we insert only activities whose predecessors are already scheduled, all constraints are satisfied if the start time of the job s_i^j belongs to D_i^j.

For the example in Figure 3.2 with all the execution times equal to 1, with a single core, and with no activities scheduled, $D_2^1 = \{[0 \cdot p_2 + t_2^b; 2 \cdot p_2 - t_2^a - e_2 - 1]\} = \{[0 + 4; 18 - 0 - 1 - 1]\} = \{[4; 16]\}$, which is basically the application of the third improvement from Section 3.3. Now, suppose in the previous iterations a_8^1 is scheduled at time 4 and a_{10}^1 is scheduled at time 6. Then, the resulting $D_2^1 = \{[5; 5] \cup [7, 16]\}$, since a_2^1 must be scheduled after a_8^1 and it cannot collide with any other activity on the core.

$$s_i^j \in D_i^j,\ j = 1, 2, \cdots, n_i^{jobs}. \tag{3.20}$$

Furthermore, similarly to the ILP model in Section 3.3, the precedence constraints for consecutive jobs of the same activity must also be added.

$$\begin{aligned} s_i^j + e_i &\leq s_i^{j+1}, \\ s_i^{n_i^{jobs}} + e_i &\leq s_i^0 + H, \\ j &= 1, 2, \cdots, n_i^{jobs} - 1. \end{aligned} \tag{3.21}$$

The pseudocode of the sub-model is presented in Algorithm 3.1. As an input it takes the first activity to schedule a_1, and the optional second activity to schedule a_2 together with their requirements, and the set of intervals D. If a_2 is set to an empty object, the sub-model must schedule only activity a_1. In case a_1 is non-jitter-critical, this scheduling problem can be trivially solved by assigning $s_i^j = l_1^{i,j}$ and checking that it does not collide with the job in the previous period. If it does, we schedule this job at the finish time of the previous job if possible, otherwise at the end of the resource interval it belongs to. If the start time is more than the refined deadline of this job from Section 3.3.3, the activity cannot be scheduled.

This rule will always result in a solution, if one exists, while minimizing (3.18). Moreover, if for some job s_i^j, the interval D_i^j is empty, then there is no feasible assignment of this activity to time with the current set of already scheduled activities.

Algorithm 3.1 Sub-model used by 3-LS heuristic

```
Input: a_1, a_2, D
if a_2 = NULL then
    if a_1 is non-jitter-critical then
        S = min_{x ∈ D_i} x : Constraint (3.21) holds
    else
        S = ILP(a_1, D)
    end if
else
    S = ILP(a_1, a_2, D)
end if
Output: S
```

On the other hand, when a_1 is jitter-critical, the sub-model is enriched by the set of jitter constraints (3.13) and the strategy to solve it has to be more sophisticated. The sub-model in this case is solved as an ILP model, which has significantly shorter computation times on easier problem instances in comparison to SMT, as shown experimentally in Section 3.5. This is important for larger problem instances where sub-model is launched thousands of times, since the heuristic decomposes a large problem to many small problems by scheduling jobs activity by activity. Although this problem seems to be NP-hard in the general case because of the non-convex search space, the computation time of the sub-model is still reasonable due to the relatively small number of jobs of one activity (up to 1000) and the absence of resource constraints.

We formulate Constraint (3.20) as an ILP in the following way. First, we set $l_1^j \leq s_i^j \leq r_w^j$ defined earlier in this section, and for each $r_t^{i,j}$ and $l_{t+1}^{i,j}$ two new

Constraints (3.22) and a variable $y_{i,j,t} \in \{0,1\}$ is introduced, which handles the "$\vee$" relation of the two constraints similarly to the variable $x_{i,l}^{j,k}$ from the ILP model in Section 3.3.

$$\begin{aligned} s_i^j + (1 - y_{i,j,t}) \cdot H &\geq l_{t+1}^{i,j}, \\ s_i^j &\leq r_t^{i,j} + y_{i,j,t} \cdot H. \end{aligned} \tag{3.22}$$

Finally, when the sub-model is used to schedule two activities at once, i.e. a_2 is not an empty object, Criterion (3.18) is changed to contain both activities, and the resource constraints (3.12) for a_1 and a_2 are added. The resulting problem is also solved as an ILP model, but similarly to the previous case takes rather short time to compute due to small size of the problem. Note that the 3-LS heuristic also utilizes the proposed computation time improvements for the ILP model from Section 3.3 and always first checks non-emptiness of D_j for each job j before creating and running the ILP model.

3.4.3 Algorithm

The proposed 3-LS heuristic is presented in Algorithm 3.2. The inputs are the *set of activities* A, the *priority rule* Pr that states the order in which the activities are to be scheduled and the *rule to choose the activity to unschedule* Un if some activity is not schedulable with the current set of previously scheduled activities. The algorithm begins by initializing the interval set D for each a_i^j as $D_i^j = \{[(j-1) \cdot p_i + t_i^b; j \cdot p_i - t_i^a - e_i - 1]\}$ (Line 3). Then it sorts the activities according to the priority rule Pr (Line 4), described in detail Section 3.4.4. The rule always states that higher priority must be assigned to a predecessor over a successor, so that no activity is scheduled before its predecessors. Note that the first part of the first level of scheduling is similar to the list scheduling approach [121].

In each iteration, the activity with the highest priority a_s in the *priority queue of activities to be scheduled* Q, is chosen and scheduled by the sub-model (Line 7). If a feasible solution S is found, the interval set D is updated so that all precedence and resource constraints are satisfied. Firstly, for each a_l that is mapped to the same resource with a_s, i.e. if $map_l = map_s$, the intervals in which a_s is scheduled are taken out of D_l. Secondly, for each successor a_l of activity a_s the intervals are changed as $D_l^j = D_l^j \setminus \{[0, S_j + e_c - 1]\}$, since a successor can never start before a predecessor is completed. Next, the feasible solution is added to the *set of already scheduled activities, represented by their schedules* Sch, and Q is updated to contain previously unscheduled activities, if there is any. If the current activity a_s is not schedulable, at least one activity has to be unscheduled. The activity to be unscheduled a_u is found according to the rule Un and this activity with all its successors $Succ_u$ are taken out of Sch (Line 14). Next, the set of intervals D is updated in the inverse manner compared to the previously described new activity insertion. To prevent cyclic scheduling and unscheduling of the same set of activities, a set R of *activities that were problematic to schedule* is maintained. Therefore, the activity to schedule a_s has to be added to R (Line 16) if it is not there yet.

If the activity to be unscheduled is not problematic, i.e. $a_u \notin R$, the algorithm schedules a_s without a_u in the next iteration. Otherwise, the second level of scheduling takes place, as shown in Figure 3.3. In this case, the sub-model is called

Algorithm 3.2 3-Level Scheduling Heuristic

```
1: Input: A
2: Sch = ∅, R = ∅, Scratch = ∅
3: D.initialize()
4: Q = sort(A, Pr)
5: while |Sch| < |A| do
6:   a_s = Q.pop()                    // Schedule a_s alone
7:   S = SubModel(a_s, NULL, D)
8:   if SubModel found feasible solution then
9:     Q.update()                    // Add unscheduled activities to Q
10:    Sch.add(S)                    // First scheduling level
11:    D.update()
12:  else
13:    a_u = getActivityToUnschedule(Sch, Un)
14:    Sch = Sch \ {a_u ∪ Succ_u}
15:    D.update()
16:    R.add(a_s)
17:    if R.contains(a_u) then // Schedule a_s and a_u simultaneously
18:      S = SubModel(a_s, a_u, D)
19:      if SubModel found feasible solution then
20:        Sch.add(S)                // Second level
21:      else
22:        Sch = Scratch ∪ Pred_{a_s} ∪ Pred_{a_u}
23:        D.update()
24:        S = SubModel(a_s, a_u, D)
25:        if SubModel found feasible solution then
26:          Sch.add(S)              // Third level
27:          D.update()
28:          Scratch.add(a_s, a_u, Pred_{a_s}, Pred_{a_u})
29:        else
30:          Output: FAIL
31:        end if
32:      end if
33:    end if
34:  end if
35: end while
36: Output: t_i^{j,curr}
```

to schedule a_s and a_u simultaneously (Line 20) and the set of two schedules S are added to Sch.

Sometimes, even simultaneous scheduling of two problematic activities does not help and a feasible solution does not exist with the given set of previously scheduled activities Sch. If this is the case, we go to the third level of scheduling and try to schedule these two activities almost from scratch, leaving in the set of scheduled activities Sch only the set $Scratch$ of activities *that were previously scheduled in level 3* and the predecessors of a_s and a_u (Line 29). The set $Scratch$ is introduced to avoid the situation where the same pair of activities is scheduled almost from scratch more than once, which is essential to guarantee termination of the algorithm. At the third scheduling level, the algorithm runs the sub-model to schedule a_s and a_u with a smaller set of scheduled activities Sch. In case of success, the obtained schedules S are added to Sch (Line 29) and a_s together with a_u and their predecessors $Pred_{a_s}$ and $Pred_{a_u}$ are added to the set of activities $Scratch$, scheduled almost from scratch. If the solution is not found at this stage, the heuristics fails to solve the problem. Thus, the 3-LS heuristic proceeds iteration by iteration until either all activities from A are scheduled or the heuristic algorithm fails. Note that the same structure of the algorithm holds for both ZJ and JC cases.

3.4.4 Priority and Unscheduling Rules

There are two rules in the 3-LS heuristic: *Pr to set the priority* of insertion and *Un to select the activity to unschedule*. The rule to set the priorities considers information about activity periods P, activity execution times $\mathbf{E}$, the critical lengths of the predecessors execution before t^b and after t^a and the jitter requirements jit. However, not only the jitter requirements of the activity need to be considered, but also the jitter requirements of its successors. The reason is that if some non-jitter-critical activity would precede an activity with a critical jitter requirement in the dependency graph, the non-jitter-critical activity postpones the scheduling of the jitter-critical activity, resulting in the jitter-critical activity not being schedulable. We call this parameter *inherited jitter* of an activity, computed as $jit_i^{inher} = \min_{a_j \in Pred_i} jit_j$. Using the inherited jitter for setting the priority is similar to the concept of priority inheritance [99] in event-triggered scheduling.

Thus, the priority assignment scheme Pr sets the priority of each activity a_i to be a vector of two components $(\min(I_i, jit^{inher}), \max(I_i, jit^{inher}))$, where I_i is the worst-case slack of the corresponding DAG, defined in Equation (3.16). The priority is defined according to lexicographical order, i.e. by comparing the first value in the vector and breaking the ties by the second. We compare first by the most critical parameter, either jitter jit_i^{inher} or the worst-case slack I_i, since those two parameters reflect how much freedom the activity has to be scheduled and the activity with less freedom should be scheduled earlier. This priority assignment strategy considers all of the aforementioned parameters, by definition outperforming the strategies that compare based on only subsets of these parameters.

The rule Un to choose the activity to unschedule is a multi-level decision process. The general rules are that *only activities that are mapped to the resource where activity a_s is mapped are considered* and we do not unschedule the predecessors of a_s. Moreover, the intuition behind the Un rule is that unscheduling activities

with very critical jitter requirements or with already scheduled successors should be done only if no other options exist. The exact threshold for being very jitter-critical depends on the size of the problem, but based on experimental results we set the threshold of a high jitter-criticality level to the minimum value among all periods. Thus, whether or not an activity is very jitter-critical is decided by comparing its jitter to the threshold value $thresh = \min_{a_i \in A} p_i$.

The rule Un can hence be described by three-steps that are executed in the given order:

1. If there are activities without already scheduled successors and with $jit_i \geq thresh$, choose the one with the highest I_i.

2. If all activities have successors already scheduled, but activities with $jit_i \geq thresh$ exist, we choose the one according to the vector (number of successors scheduled, I_i) comparing lexicographically.

3. Finally, if all activities have $jit < thresh$, the step chooses the activity to unschedule according to the priority vector (jit^{inher}, I_i) comparing lexicographically.

Step 1 is based on the observation that activities with very critical jitter requirements are typically hard to schedule, unlike those with no jitter requirements or less critical ones. Besides, unscheduling many activities instead of one may cause prolongation of the scheduling process and possibly more complications with further scheduling of successors. Moreover, since only activities of cause-effect chains are a part of precedence relations, there are many activities with no predecessors and successors that can be unscheduled. This is typical for the automotive domain [65]. Step 2 allows unscheduling of activities with already scheduled predecessors, preferring to keep in the schedule activities with critical jitter requirements. Step 3 states that if all of the activities have very critical jitter requirements, the activity with the highest value of inherited jitter should be unscheduled. In all three steps, ties are broken by choosing the activity with higher worst-case slack I value by the same intuition as in the Pr rule.

We have experimentally determined that comparing to the unscheduling rule with only worst-case slack (I_i) considered, the gain of the presented unscheduling rule is 5% higher utilization achieved on average.

3.5 Experiments

This section experimentally evaluates and compares the proposed optimal models and 3-LS heuristic on synthetic problems with jitter requirements set differently to show the advantages and disadvantages of JC scheduling. Furthermore, we quantify the trade-off of additional cost in terms of memory to store the schedule and increase in computation time versus this gained utilization. The experimental setup is presented first, followed by experiments that evaluate the proposed exact and heuristic approaches for different jitter and period requirements. We conclude by demonstrating our approach on a case study of an Engine Management System with more than 10 000 activities to be scheduled.

Table 3.1: Generator parameters for the sets of problem instances

Set	$\|T\|$	*P [ms]*	*Variable accesses per task*	*Chains per task*
1	20	$1, 2, 5, 10$	4	4
2	30	$1, 2, 5, 10$	4	6
3	50	$1, 2, 5, 10, 20, 50, 100$	4	8
4	100	$1, 2, 5, 10, 20, 50, 100$	4	15
5	500	$1, 2, 5, 10, 20, 50, 100$	8	50

3.5.1 Experimental Setup

Experiments are performed on problem instances that are generated by a tool developed by Bosch [65]. There are five sets of 100 problem instances, each set containing 20, 30, 50, 100 and 500 tasks, respectively. The same problem instance is presented with different jitter requirements. The generation parameters for each dataset are presented in Table 3.1, and the granularity of the timer is set to be 1 μs. Message communication times are computed for the considered platform with the following parameters: bandwidth $bnd = 400$ MB/s and latency $lat = 50$ clock cycles.

The mapping is determined as described in Section 3.2.3 such that the load is balanced across the cores, i.e. the resulting mapping utilizes all cores approximately equally. The resulting problem instances contain 30-45, 50-65, 90-130, 180-250 and 1500-2000 activities (tasks and messages) for sets with 20, 30, 50, 100 and 500 tasks, respectively.

While we initially assume a system with 3 cores connected over a crossbar (resulting in 6 resources), inspired by the Infineon Aurix Tricore Family TC27xT, the approach can scale to a higher number of cores, as later shown in Section 3.5.2.

The metric for the experiments on the synthetic datasets is the maximum utilization for which the problem instance is still schedulable. The utilization is defined as $r_q = \sum_{a_i \in A: map_i = q} \frac{e_i}{p_i}$ on each resource $q = 1, \cdots, 6$. To achieve the desired utilization on each resource, the execution times of activities are scaled appropriately. The experiments always start from a utilization of 10%, increasing in steps of 1%, solving until the approach is not able to find a feasible solution. The last utilization value for which the solution was found is set as the *maximum utilization* of the approach on the problem instance. This approach to set the maximum schedulable utilization may not be completely fair, since a failure at a particular utilization doses not guarantee that there are instances with higher utilization that are schedulable. However, the utilization is monotonic in most cases, and we run each experiment 100 times, looking at the distributions and it is fair enough. Therefore, we have chosen to approximate the results by using this metric to get results that are easier to interpret.

Experiments were executed on a local machine equipped with Intel Core i7 (1.7 GHz) and 8 GB memory. The ILP model and ILP part of the 3-LS heuristic were implemented in IBM ILOG CPLEX Optimization Studio 12.5.1 and solved with the CPLEX solver using concert technology, while the SMT model was implemented in Z3 4.5.0. The ILP, SMT and heuristic approaches were im-

Table 3.2: Number of problem instances that optimal approaches failed to solve before the time limit of 3 000 seconds

jit_i	$p_i/2$		$p_i/5$		$p_i/10$		0	
	Set 1	Set 2	Set1	Set 2	Set 1	Set 2	Set 1	Set 2
ILP	14	76	9	53	6	45	4	27
SMT	2	51	3	13	2	9	2	7

plemented in the JAVA programming language. The source code of the exact approaches, the heuristic approach, and the problem instances are available at `https://github.com/CTU-IIG/CC_Scheduling_WithJitter`

3.5.2 Results

First, the experiments compare the computation time of the optimal ILP and SMT approaches to show for which problem instances it is advantageous to use each approach. Secondly, we evaluate the trade-off between the maximum achievable utilization and computation time of the 3-LS heuristic and the optimal approaches for differently relaxed jitter requirements. Thirdly, since memory consumption to store the final schedule is also a concern, the trade-off between solution quality and required memory is evaluated for systems of different sizes. Finally, a comparison of different period settings is presented to show the applicability of the approach to different application domains and to evaluate the behavior of both ZJ and JC approaches for different periods. A time limit of 3 000 seconds per problem instance was set for the optimal approaches to obtain the results in reasonable time. Note that the best solution found so far is used if the time limit is hit.

Comparison of the ILP and SMT models with different jitter requirements

First of all, we compare the computation time distribution for Set 1 and Set 2 (of smaller instance sizes with 30-45 activities and 50-65 activities, respectively) for the SMT and ILP approaches with jitter requirements of each activity $a_i \in A$ set to $jit_i = \frac{p_i}{2}$, $jit_i = \frac{p_i}{5}$, $jit_i = \frac{p_i}{10}$ and $jit_i = 0$. Since the first problem instance from Set 3 was computing for two days before it was stopped with no optimal solution found for both SMT and ILP models, the experiments with optimal approaches only use the first two sets. We will return to the larger sets in Section 3.5.2 when evaluating the 3-LS heuristic. The distribution is shown in the form of box plots similarly to Chapter 2.

The number of problem instances from Set 1 and Set 2 that the optimal approaches failed to solve within the given time limit is shown in Table 3.2. Moreover, Figure 3.4 displays the computation time distribution on Set 1, where only problem instances that both the ILP and SMT solvers were able to optimally solve all jitter requirements within the timeout period are included. For Set 1, it is 82 (out of 100), and for Set 2, it is 21 (out of 100) problem instances. The computation time distribution for Set 2 shows a similar trend, but since the sample is too small to be representative, we do not display them.

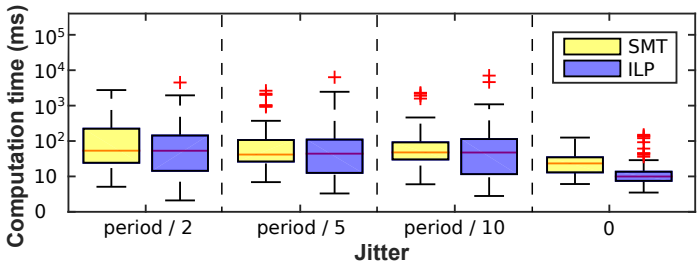


Figure 3.4: Computation time distribution for the SMT and ILP models with different jitter requirements for Set 1.

The results in Table 3.2 show that for more difficult problem instances the SMT model is significantly better than the ILP model in terms of computation time, since it is able to solve more problem instances within the given time limit. On the other hand, the comparison on the problem instances that both approaches were able to solve in Figure 3.4 indicates that the ILP runs faster on simpler problem instances that can be found at the bottom of the boxplots. As one can see, more relaxed jitter requirements result in longer computation time, which is a logical consequence of having larger solution space.

Thus, *the SMT model is more efficient than the ILP model for the considered problem on more difficult problem instances, while the ILP model shows better results for simpler instances, which justifies the usage of the ILP model in the sub-model of the 3-LS heuristic. Besides, more relaxed jitter requirements cause longer computation time for the optimal approaches.* Therefore, the SMT approach results are used for further comparison with the 3-LS heuristic.

Comparison of the optimal and heuristic solutions with different jitter requirements

Figure 3.5 shows the distribution of the maximum utilization on Set 1 for the SMT model and the 3-LS heuristic with different jitter requirements. For comparison, we use the solution with the highest utilization, while the low value of initial utilization guarantees that at least some solution is found. The time limit caused 3 problem instances in Set 1 not to finish when using the SMT approach and these instances are not included in the results. The results for Set 2 are similar to those of Set 1, but due to the small number of solvable instances we do not show them. The results for the optimal approach shows that stricter jitter requirements cause lower maximum achievable utilization. Namely, the average maximum utilization is 89%, 75%, 69%, 61% for Set 1 and 95%, 81%, 74%, 67% for Set 2 for the instances with jitter requirements equal to half, fifth, tenth of a period and zero, respectively. Meanwhile, the comparison of the 3-LS heuristic to the optimal solution reveals that the average difference goes from 17% and 23% for Set 1 and Set 2, respectively, with the most relaxed $jit_i = \frac{p_i}{2}$ to 0.1% for both sets with ZJ scheduling. This difference for problem instances with more relaxed jitter requirements is caused by very large complexity of the problem solved. One more reason for a significant difference in maximal utilization of the SMT model and the 3-LS heuristic is absence of flexibility of the heuristic, resting in assigning precise start times. It is not able to

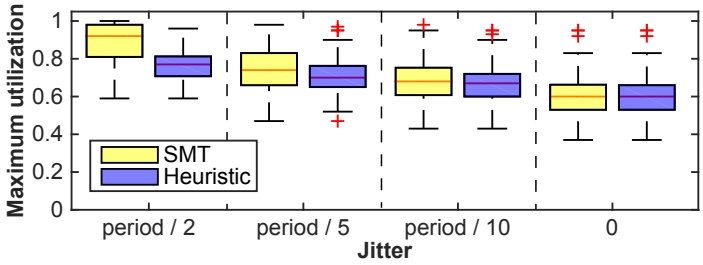


Figure 3.5: Maximum utilization distribution for the optimal SMT and 3-LS heuristic approaches with different jitter requirements for Set 1.

slightly move the already scheduled jobs of activities so that the currently scheduled activity still have a chance to be scheduled. Thus, a possible improvement is a more flexible solution representation that allows the existing schedule to adjust to the new activities to be scheduled better, which is applied to the heuristic in Chapter 4.

However, while the heuristic solves all problem instances in hundreds of milliseconds, the SMT model fails on 62 problem instances out of 200 within a time limit of 3 000 seconds. This reduction of the computation time by the heuristic is particularly important during design-space exploration, where many different mappings or platform instances have to be considered. In that case, it is not possible to spend too much time per solution. Hence, we conclude that the *3-LS heuristic performs better with decreasing jitter requirements and hence particularly well for ZJ scheduling, resulting in an average degradation of 7% for all instances. Moreover, unlike the SMT model, the 3-LS heuristic always finds feasible solutions in hundreds of milliseconds, hence providing a reasonable trade-off between computation time and solution quality.*

Comparison of the heuristic with ZJ and JC scheduling

While the previous experiment focused on comparing the optimal approach and the heuristic, therefore using only smaller problem instances, this experiment evaluates the 3-LS heuristic on all sets. Due to time restrictions, only two jitter requirements were considered, $jit_i = \frac{p_i}{5}$ and $jit_i = 0$. Figure 3.6 shows the distribution of the maximum utilization for the 3-LS heuristic on Sets 1 to 5. In all sets, 100 problem instances were used for this graph. The results show that with growing size of the problem instance, the maximum utilization generally increases. The average difference in maximum utilization of the 3-LS heuristic on the problem instances with JC and ZJ requirements is 15.3%, 9.7%, 8.6%, 4.2% and 7.5% for Sets 1 to 5, respectively, with JC achieving higher utilization. The decreasing difference with growing sizes of the problem is caused by the growing average utilization. For instance, the average maximum utilization for Set 5 is 89.1% for the problem instances with JC requirements and 82.6% for the problem instances with ZJ requirements, pushing how far the maximum utilization for the JC scheduling can go. This tendency of increasing maximum utilization for the ZJ scheduling can be intuitively supported by the fact that more and more activities are harmonic with each other, which results in easier scheduling. In reality, harmonization costs a significant amount of over-utilization, especially when activities with smaller periods

are concerned. On problem instances without harmonized activity periods, the JC scheduling can show notably better results compared to ZJ scheduling, as previously shown in Section 3.5.2.

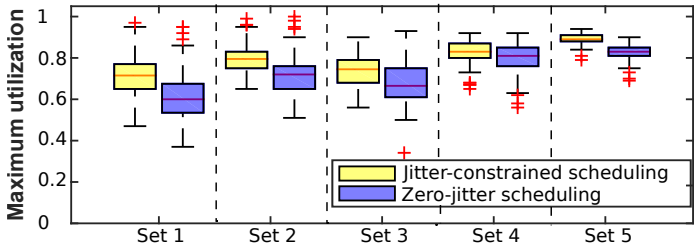


Figure 3.6: Maximum utilization distribution of the 3-LS heuristic for activities with jitter-constrained and zero-jitter requirements on the problem instance sets of increasing sizes.

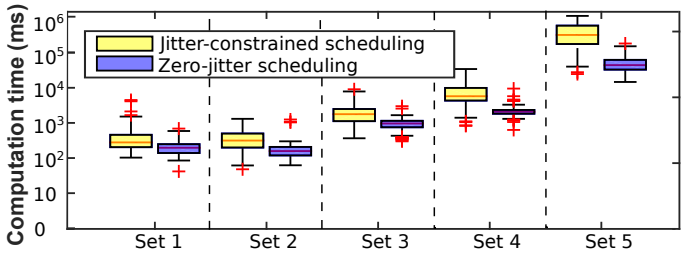


Figure 3.7: Computation time distribution for the 3-LS heuristic for activities with jitter-constrained and zero-jitter requirements on the problem instance sets of increasing sizes.

Figure 3.7 shows the computation time of ZJ and JC using the 3-LS heuristic. Similarly to the optimal approach, the 3-LS heuristic takes longer to solve problem instances with JC requirements due to a larger solution space. Specifically, the average computation time for JC heuristic for Sets 1 to 5 are 0.3, 0.6, 3.6, 14.5 and 1 003.6 seconds, respectively, while for ZJ scheduling it is 0.15, 0.28, 1.6, 4 and 109 seconds. Thus, solving a problem instance with JC requirements with 1 500 - 2 000 activities takes less than 17 minutes on average, which is still reasonable. Hence, *the 3-LS heuristic with JC scheduling provides better results, but requires more time than the 3-LS heuristic with ZJ scheduling.* Notice the increase in the computation time of the heuristic with increasing sizes, which can be caused by the usage of ILP solver for some activities. Thus, it is possible that for the hypothetical Set 7, the heuristic would run unreasonably long. This is supported by the results on the Engine Management Case Study with more than 10 000 activities in Section 3.5.3.

To summarize this experiment, *JC scheduling is promising in terms of maximum utilization, as it schedules with up to 55% higher resource utilization.* Besides, the computation time of the proposed heuristic is affordable even for larger problem instances, while the optimal models fail to finish in reasonable time already for much smaller instances. Moreover, *the proposed heuristic solves the problem instances with*

ZJ requirements near-optimally with a difference of 0.1% in schedulable utilization on average. Generally, *the JC heuristic provides more efficient solutions than the ZJ heuristic, while requiring longer computation time.*

Evaluation of maximum utilization and required memory trade-off with different number of cores

The trade-off between maximum achievable utilization and the amount of memory required to store the schedule is evaluated by this experiment. Figure 3.8 shows the average maximum utilization achieved on systems with different number of cores and with gradually increasing percentage of JC jobs on 50 problem instances from Set 2 (due to time restrictions). In the experiment, the jitter constraint is set to $jit_i = \frac{p_i}{5}$ and the instances are solved to optimality. Furthermore, the problem instances with different number of cores are solved in steps with the percentage of jobs of zero-jitter activities increasing by 5%. The percent of JC activities estimates amount of memory necessary to store the schedule, since more JC activities mean both a higher number of elements to store and a higher demand on memory as mentioned in Section 3.2. Note that the execution times of the activities are scaled so that each resource has the required utilization.

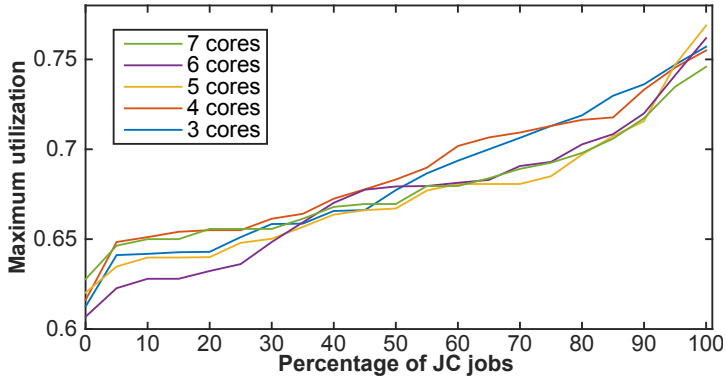


Figure 3.8: Utilization distribution for different percents of jitter-constrained activities for different architectures.

The results show that introducing more JC jobs and thus increasing memory requirements for storing the final schedules can significantly improve the average maximum utilization. For example, the architecture with 4 cores has a maximum utilization of 61% when all jobs are ZJ. Relaxing the jitter requirements of half the jobs results in 69% utilized resources, and relaxing all of the jobs increases the maximum utilization to 76%. Concerning the required memory to store the schedule, the problem instances with 4 cores on average contain 80 jobs with JC scheduling and 49 jobs while scheduling in ZJ manner. Thus, according to Equation (3.1), the memory overhead of relaxing jitter is $31 \cdot 8 = 248$ bytes, which is a reasonable price to pay for utilization gain of 15% on average on each resource.

The results demonstrate that on average there is no significant dependency on the number of cores we have in the system. Hence, *JC scheduling can result in high*

utilization gain, although at the cost of increased memory requirements to store the resulting schedule.

Comparison of the different period settings

To show that the approach is applicable to other domains, an experiment with different period settings is performed. All problem instances from Set 2 are solved monoperiodically ($p_i = 10$ ms for each activity $a_i \in A$), or with harmonic periods (activities with $p_i = 2$ ms are changed to $p_i = 5$ ms), or with initial periods (i.e. with periods 1, 2, 5, 10 ms), or with non-harmonic periods (with periods 2, 5, 7, 12 ms). Figure 3.9 displays the average maximum utilization achieved by the 3-LS heuristic with ZJ and JC scheduling ($jit_i = \frac{p_i}{5}$) on 100 problem instances from Set 2. Since the optimal approach was not able to solve 7 out of 10 first instances with non-harmonic periods within the given time limit, due to its complexity and extended hyper-period, the optimal approach results are not included in the figure.

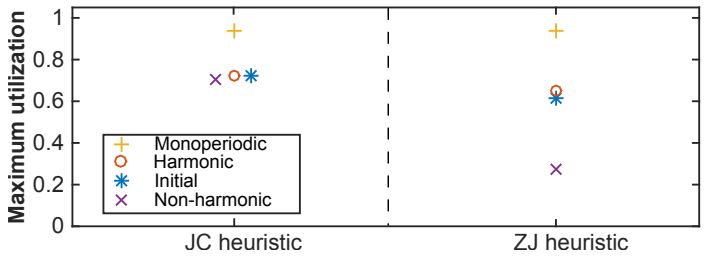


Figure 3.9: Utilization distribution for problem instances with different periods.

The results show that the maximum utilization for both ZJ and JC is achieved when scheduling monoperiodically, which is explained by having less possible collisions in the resulting schedule. An interesting observation is that for JC scheduling all other period settings on average resulted in very similar maximum utilization, while the ZJ approach shows the variation of 27% with non-harmonic periods, 62% with initial periods and 65% with harmonic period set. The relative insensitivity of JC scheduling to period variations can be caused by the significantly larger solution space resulting from relaxation of the strict jitter constraints. This allows solutions with high utilization to be found, even with the non-harmonic period setting. The order of computation time distribution using different period settings is the same as in Figure 3.9, i.e. monoperiodic is the fastest and non-harmonic the slowest.

From this experiment, we conclude that *the proposed approach is applicable to other domains, where the application periods have different degree of harmonicity. Furthermore, increasing harmonicity of the period set results in higher maximum utilization, lower computation time and lower gain of JC scheduling in comparison with ZJ scheduling in terms of maximum utilization.*

3.5.3 Engine Management System Case Study

We demonstrate the applicability of the proposed 3-LS heuristic on an Engine Management System (EMS). This system is responsible for controlling the time

and amount of air and fuel injected by the engine by considering the values read by numerous sensors in the car (throttle position, mass air flow, temperature, crankshaft position, etc). By design, it is one of the most sophisticated engine control units in a car consisting of 1 000-2 000 tightly coupled tasks that interact over 20 000 to 30 000 variables, depending on the features in that particular variant. A detailed characterization of such an application is presented by Bosch in [65], along with a problem instance generator that creates input EMS models in conformance with the characterization.

We consider such a generated EMS problem instance, comprising 2000 tasks with periods 1, 2, 5, 10, 20, 50, 100, 200 and 1 000 ms and with 30 000 variables in total, where each task accesses up to 12 variables. There are 60 cause-effect chains in the problem instance with up to 11 tasks in each chain. We consider the target platform to be similar to an Infineon AURIX Family TC27xT with a processor frequency of 125 MHz and an on-chip crossbar switch with a 16 bit data bus running at 200 MHz, thus having a bandwidth of 16-bit·200 MHz / 8 = 400 MB/s. The time granularity is 1 μs, and the resulting hyper-period is 1 000 ms. However, setting the hyper-period to be 100 ms results in a utilization loss of less than 0.5%, arising from shortening the scheduling periods of tasks with periods 200 ms and 1 000 ms and over-sampling, which is a reasonable sacrifice to decrease the memory requirements of the schedule. The tool in [65] provides the number of instructions necessary to execute each task, which is used to compute the worst-case execution time with the assumption that each instruction takes 3 clock cycles on average (including memory accesses that hit/miss in local caches).

The mapping of tasks to cores by the simple ILP formulation previously described in Section 3.2 requires minimally 3 cores with a utilization of approximately 89.6% on each core and approximately 30% on each input port of the crossbar. Moreover, the resulting scheduling problem has 10 614 activities with 104721 jobs for the JC assumptions in total. Neither SMT nor ILP can solve this problem in 24 hours, but the JC heuristic with $jit_i = \frac{p_i}{5}$ for all a_i solves the problem in 43 minutes. By gradually introducing more activities a_i with $jit_i = 0$, we have found a maximum value of 85% ZJ activities for which the 3-LS heuristic is still able to find a solution, which takes approximately 12 hours. Note that the computation time has increased by introducing more ZJ activities, due to the more restricted solution space. However, to store the schedule in the memory for 0% ZJ jobs, 104 721·8 = 818 Kbytes of memory is required according to Equation (3.1), while with 85% of ZJ jobs it is only 19 394·8 = 152 Kbytes. *Thus, for realistic applications the optimal approaches take too long, while the 3-LS heuristic approach is able to solve the problem in reasonable time. Moreover, increasing the percent of ZJ activities has shown to provide a trade-off between computation time and required memory to store the obtained schedule.*

3.6 Summary

This chapter introduces a coscheduling approach to find a time-triggered schedule for periodic tasks with hard real-time requirements executing on multiple cores and communicate over an interconnect such as buses, networks, or crossbars. More-

over, the tasks have precedence and jitter requirements due to the nature of such applications in the automotive domain. We concentrate specifically on the jitter requirements since existing works either forbid any jitter (zero-jitter (ZJ) approach) or put no constraints on it. In contrast, we propose a jitter-constrained (JC) approach that allows for more flexibility at the cost of higher memory required to store the schedule.

To optimally solve the considered problem, we propose both an Integer Linear Programming (ILP) model and a Satisfiability Modulo Theory (SMT) model that exploit precedence constraints to reduce the computation time by constraining variable domains. Furthermore, a three-step heuristic scheduling approach, called 3-LS heuristic, where the schedule is found constructively is presented. The 3-LS heuristic decomposes the problem into smaller sub-problems and utilizes the formulated optimal models to solve single sub-models. The heuristic works in three levels, where the scheduling complexity and the time consumption grow for each level, providing a right balance between solution quality and computation time.

We experimentally evaluate the efficiency of the proposed optimal and heuristic approaches with JC requirements, comparing to the widely used ZJ approach and quantify the gain in terms of maximum utilization of the resulting systems. The results show that optimal JC scheduling achieves higher utilization with an average difference of 28% compared to optimal ZJ scheduling. Moreover, the experimental evaluations indicate that the SMT model can solve more problem instances optimally within a given time limit than the ILP model, while the ILP model shows better computation time on simpler problem instances. We also show that the 3-LS heuristic solves the problem instances with ZJ requirements near-optimally, while the results for the JC case indicate that there is room for improvement due to lack of flexibility, which is later addressed in Chapter 4. However, the computation time of the proposed heuristic is acceptable even for larger problem instances, while the optimal models fail to finish in reasonable time already for smaller problem instances. For example, the 3-LS heuristic solves a problem instance from the three largest sets in slightly more than 3 minutes on average.

The approach is demonstrated on a case study of an Engine Management System, where 2000 tasks are executed on cores, sending around 8000 messages over the interconnect. Here, we show that for realistic applications, the proposed optimal solutions cannot find a feasible solution in 24 hours, while the 3-LS heuristic can find one solution in less than one hour. This provides a trade-off between required memory to store the schedule and computation time depending on the percent of activities with zero-jitter requirements.

Chapter

4 Coscheduling with Control Performance Optimization

The previous chapter focused on the problem of coscheduling computation on cores and on-chip communication over interconnect in safety-critical automotive systems with real-time requirements. In this chapter, we introduce a criterion that deals with the control performance of the applications to the coscheduling problem. Thus, we formulate the problem from the application view and simplify the work of the application engineer, who can focus on control issues and avoid translating it to timing requirements. Furthermore, in this chapter, we assume a more scalable platform architecture, being a distributed system with electronic control units (ECUs) connected by an off-chip switched time-triggered Ethernet network [18] as shown in Figure 1.2c of Chapter 1. To move with current trends, we also consider the presence of both control and video traffic in the system to address rapidly developing an advanced driver-assistance system (ADAS). In terms of the solution approach, we try out three optimal approaches to solve the problem: Integer Linear Programming (ILP), Satisfiability Modulo Theory (SMT), and Constraint Programming (CP) to compare their computational efficiency on this problem. Finally, we address the main drawback of the 3-Level Scheduling (3-LS) heuristic from the previous chapter, being the absence of flexibility, by considering the solution space of priority queues from which the solution is constructed rather than working directly with activity start times. Fixing the order (priority queue) of the activity start times in the solution instead of fixing a particular activity start times leaves the heuristic more room to decide which changes to apply to the schedule.

At present, car manufacturers face the problem of integrating applications that are typically received from different suppliers as (hardware) components. Sensors and actuators that are parts of such applications may be spatially distributed, and thus, these applications may require transmitting data over a network. A current trend in the automotive domain is to move from having multiple specific buses, such as CAN, FlexRay, and MOST, that have limited bandwidth to a common switched network, such as deterministic Ethernet. The goal is threefold: 1) to allow for the incorporation of applications with high bandwidth requirements, 2) to reduce the communication complexity, and 3) to increase the durability of the product [66]. Therefore, the platform model considered in this chapter consists of multiple ECUs connected by a time-triggered Ethernet as in Figure 4.1.

The control performance of the resulting system may significantly depend on how the applications are integrated into a car. Moreover, the control performance

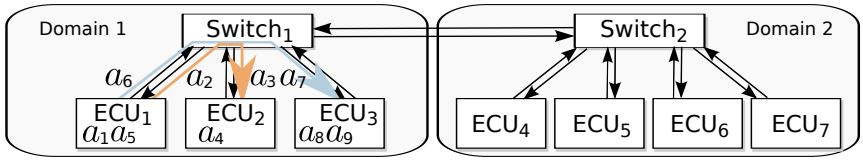


Figure 4.1: Assumed platform model.

of some applications is more sensitive to the changes in their timing behavior than others. Hence, it is essential to consider this aspect during the integration stage. The main factors that influence the control performance of an application are its jitter and end-to-end latency. The control performance degrades when sensing or actuation is done at different times within a period [25]. Therefore, it is important to avoid such degradation by scheduling it with zero jitter. Moreover, whereas for some applications the control performance degrades less with increasing end-to-end latency, others experience larger degradation. Thus, this issue should be considered when solving the scheduling problem. In this chapter, we address the problem of finding a periodic schedule that optimizes the high-level control performance, while satisfying hard real-time constraints, such as deadlines, jitter, data dependencies, and end-to-end latency constraints. This problem is challenging, as it belongs to the class of NP-hard problems [24].

Although the problem of control-scheduling codesign is a widely studied area [120], control algorithms are often provided by control engineers as runnables that cannot be modified during system scheduling. To the best of our knowledge, this is the first work that addresses Time-Triggered (TT) scheduling of applications with given sampling periods to optimize the control performance considering different sensitivities of applications to their end-to-end latency.

The main contributions of this chapter are the following: 1) ILP, CP and SMT models that solve the problem optimally and exploit particular properties of the problem to reduce the computation time. The three models are proposed to compare the efficiency of the approaches. 2) A heuristic approach that first constructively creates a schedule and then improves it via a local neighborhood search, thus ensuring the coarsest possible scheduling granularity at each stage of the algorithm to address the problem complexity. This approach provides a reasonable trade-off between computation time and solution quality compared to the optimal approaches. 3) An experimental evaluation of the proposed solution on datasets generated by a tool developed by Bosch [65] that examines the scalability of the proposed approaches and quantifies the computation time, control performance, and resource utilization of the heuristic and optimal approaches. The evaluation shows that the CP approach using solver-specific features significantly outperforms the ILP and SMT approaches. Furthermore, the heuristic approach has on average 17 times lower computation time, resulting in solutions on average 0.8% worse than the CP approach, while sacrificing on average 14% of utilization. Also, the proposed approaches are demonstrated on an automotive case study, for which the heuristic finds a solution in less than 4 minutes with objective value degradation of 1.6% from the solution, found by the best of the optimal approaches in 24 hours.

The rest of this chapter is organized as follows: the related work is discussed in Section 4.1. Section 4.2 proceeds by presenting the platform, application and control models, followed by Section 4.3, in which the problem formulation is given. A description of the optimal formulations and their computation time improvements follows in Section 4.4. Section 4.5 introduces the proposed heuristic approach for scheduling periodic activities, and Section 4.6 proceeds by presenting the experimental evaluation, before the chapter is concluded in Section 4.7.

4.1 Related Work

Works in TT domain focus on a wide variety of aspects, such as timing properties of the schedules [47, 83, 87, 124], application to specific platforms [89] and control performance of the applications [29, 123]. The authors in [87] and [124] address timing properties for the problem of finding a schedule with the maximum possible extension of activity processing times. Whereas the former work allows for limited preemption on ECUs, i.e., tasks can be executed in multiple separated parts, the latter assumes nonpreemptive scheduling, similar to this chapter. Puffitsch et al., in [89], on the other hand, concentrate on platform-specific features, introducing additional constraints to address nonperfect time synchronization.

While solving the problem of TT coscheduling of communication and computation in the automotive domain, jitter and end-to-end latency of the applications are often constrained to guarantee sufficient control performance. In the previos chapter, we consider jitter-constrained activities and show that strict jitter requirements may result in the significant underutilization of system resources. zero-jitter (ZJ) scheduling and bounded end-to-end latency are also considered in, e.g., [73] and [96]. To enlarge the solution space, the authors in [73] allow for limited preemption on ECUs and apply an ILP-based approach to solve the problem. In [96], on the other hand, no preemption is allowed, and a CP-based approach is used. However, unlike the end-to-end latency minimization performed in this chapter, constraining latency lacks flexibility. In reality, tight constraints may result in unschedulable instances, whereas loose ones may yield poor control performance. Meanwhile, minimization is able to set it as good as it is possible.

Some works have addressed the schedule integration problem, in which multiple components, sharing resources are integrated into one system. In [93], the authors solve the problem of integration of the applications with given schedules by keeping the end-to-end latency of the applications untouched. In [17], the authors integrate new applications, assuming the set of already configured applications on the resources. During the integration of the new applications, the schedules of the already-configured applications are fixed, whereas the end-to-end latency of the new applications is bounded. Although both approaches result in limited control performance degradation, this may cause low utilization of the resources. As a consequence, this may require the introduction of new ECUs and network links that increases cost of the resulting system.

Sometimes, it is possible to redesign controllers, which enables control-scheduling codesign. Following this direction, Aminifar et al. in [9] address designing high-quality embedded control systems, assuming Event-Triggered (ET) scheduling. The authors consider setting periods and processing times to be a part of the optimization problem. The period is also a changeable design parameter in [94], in which the authors solve the problem of optimizing control performance while scheduling in both a TT and a ET manner. The authors set application periods such that the resulting weighted sum of application control performances is optimized. In addition, Goswami et al. in [45] set the control performance to be a function of sampling periods and end-to-end latency, while addressing a similar TT scheduling and period-setting problem. However, since different control applications are designed by different suppliers, in the considered problem it is not possible to redesign the

controllers.

Finally, some works that address the same scheduling problem aim at minimizing end-to-end latencies, assuming the controller design to be given, similarly to this chapter. Whereas Craciunas and Oliver in [29] minimize the end-to-end latency of communication on the switched network, Zhang et al. in [123] propose multiobjective optimization with the goal to minimize the end-to-end latencies and response times of applications. These works consider scheduling tasks and messages, while optimizing application-level timing properties. However, in this work, we consider high-level control performance as the optimization objective.

4.2 System Model

This section introduces the platform, application, and control models used in this chapter.

4.2.1 Platform Model

Unlike the platform model in the previous chapter with multiple cores located on one Electronic Control Unit (ECU), communicating via a crossbar switch, the considered platform in this chapter includes multiple ECUs connected to each other by a switched network based on a *time-triggered automotive Ethernet* with a tree topology, similar to the one in Figure 4.1. This distributed architecture is commonly used for automotive systems [66]. ECUs are grouped into multiple domains, where ECUs in the same domain are interconnected by bidirectional links to a switch. We consider switches that are connected in chains by bidirectional links. Thus, there is only one path from any source to any destination ECU, and tasks on different ECUs in the same domain communicate via a switch and via multiple switches across the domains.

The set of resources comprises m_{ECU} ECUs and $m_{Links} = 2 \cdot m_{ECU} + 2 \cdot (m_{Dom} - 1)$ links, where m_{Dom} is the number of domains. The total number of resources is defined as $m = m_{ECU} + m_{Links}$.

4.2.2 Application Model

The system comprises components that are provided independently by different suppliers. Without loss of generality, each component is always represented by one *application* $app_w \in App$, which must be executed periodically with a certain period $p_w \in \mathbb{N}$. As in Chapter 3, the application model is based on the characteristics of realistic benchmarks of modern automotive software systems, provided in [65]. We model the application as a set of periodic *tasks* T that communicate with each other via a set of *messages* transmitted over the switched network. Each message comprises a set of *frames* M to be sent over the switch links. Then, $A = T \cup M$, denotes the set of activities, which includes the tasks executed on the ECUs and the frames transmitted over the network links.

Since an application is a sequence of sensing, computation, and actuation, the activities in an application are data-dependent. Similarly to the previous chapter, this dependency is represented as a general directed acyclic graph (DAG) of

precedence relations. We show examples of such graphs for a set of two applications in Figure 4.2. Note that although periods of these applications are harmonic, i.e., dividable by each other, in this chapter, we do not restrict ourselves to harmonic periods only. Also, we assume general DAGs without restricting neither to chains nor to out-trees.

Furthermore, we assume that the *mapping* $map_i : T \rightarrow \{1, 2, \cdots, m\}$ of tasks is given by manufacturers, while the routing of messages and mapping of frames is straightforwardly derived from the task mapping. Each activity $a_i \in A$ must be executed periodically with period $p_i \in P$ that is the same for all activities of an application, i.e., $p_i = p_w$ for $a_i \in app_w$. Only activities belonging to the same application can have precedence relations, since the applications are provided separately, i.e., *precedence relations apply only to activities within the same period* just like in the previous chapter.

We assume that time is discretized with sufficient precision and the processing time (either execution or transmission time) $e_i \in \mathbb{N}$ for each activity is provided. Unlike the previous chapter, the applications are heterogeneous regarding the volumes of data they transfer over the network, which is more realistic assumption for modern automotive systems. For example, an engine management systems transfers sensor values, and autonomous driving systems transfer video or lidar data. Thus, *traffic of both small and large size is always present in the system.*

The control functionality is designed robustly, such that an application can tolerate a certain *maximum end-to-end latency*, L_w, of the resulting schedule. The end-to-end latency is the time from the beginning of the earliest activity until the end of the latest activity. The maximum possible end-to-end latency is set as $\hat{L}_w = \mathbf{c} \cdot p_w$ for an application app_w, where $\mathbf{c}$ is a positive integer and a tunable parameter, which is tuned by the application provider. Note that in the previous chapter, this parameter is fixed to be 2, while here we generalize it to allow more flexibility in the solution. However, increasing this tunable parameter results in higher computation time.

Finally, the main differences of application model in this chapter are: heterogeneous traffic, generalized tunable parameter for the maximum end-to-end latency, and a fixed jitter requirements, described later in this section together with their benefits.

4.2.3 Control Model

In this chapter, we consider linear and time-invariant (LTI) systems. The mathematical model of such a system can be written as

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t),$$
$$y(t) = C \cdot x(t),$$

where $x(t)$, $u(t)$ and $y(t)$ represent the system states, control input and system output, respectively. A, B and C are constant system matrices.

Traditionally, an embedded controller is implemented according to a constant sampling period p. Assuming the system states are sensed at time instants $t_1, \cdots, t_n$ and are represented by states $x_1, \cdots, x_n$, the sampling period of an application is given by $p = t_{k+1} - t_k$. Further, the task execution time and frame transmission

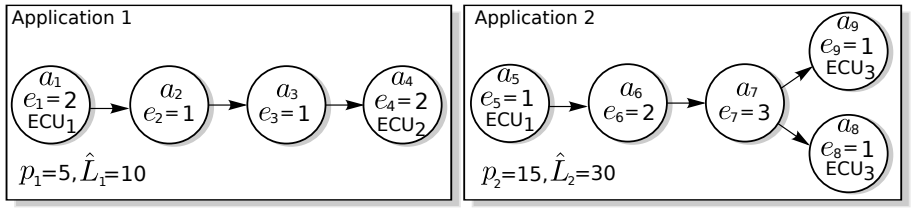


Figure 4.2: An example set of applications.

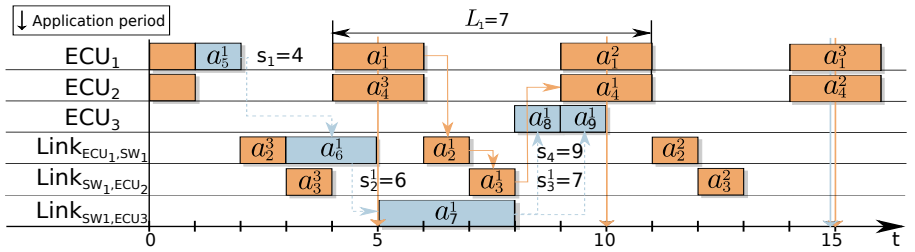


Figure 4.3: An example schedule for ECU_1, ECU_2, and ECU_3, and links $ECU_1 \rightarrow SW_1$, $SW_1 \rightarrow ECU_2$, and $SW_1 \rightarrow ECU_3$ for applications from Figure 4.2 with periods $p_1 = p_2 = p_3 = p_4 = 5$ and $p_5 = p_6 = p_7 = p_8 = p_9 = 15$.

time contribute to end-to-end latency L (also called delay). Corresponding to the end-to-end latency, new control inputs are applied at discrete time instants $t_k + L$ and are represented by $u[k]$, $k = 1, \cdots, n$.

From the above assumptions, the equivalent sampled data model can be derived as

$$\begin{aligned} x[k+1] &= \phi \cdot x[k] + \Gamma_0 \cdot u[k - \left\lfloor \frac{L}{p} \right\rfloor] + \Gamma_1 \cdot u[k - \left\lceil \frac{L}{p} \right\rceil], \\ y[k] &= C \cdot x[k], \end{aligned} \tag{4.1}$$

where ϕ, Γ_0 and Γ_1 for $\tau = L - \left\lfloor \frac{L}{p} \right\rfloor$ are given by [10] as:

$$\begin{aligned} \phi &= e^{A \cdot p}, \\ \Gamma_0 &= \int_0^{p-\tau} (e^{At} dt) \cdot B, \\ \Gamma_1 &= \int_{p-\tau}^{p} (e^{At} dt) \cdot B. \end{aligned}$$

It is assumed that the control input $u[k]$ in Equation (4.1) is computed according to the feedback control law given by

$$u[k] = K \cdot x[k] + F \cdot r, \tag{4.2}$$

where r is the reference input and K and F are feedback and feedforward gains, respectively. These gains are designed by control engineers satisfying certain control performance requirements assuming ideal implementation conditions, such as zero

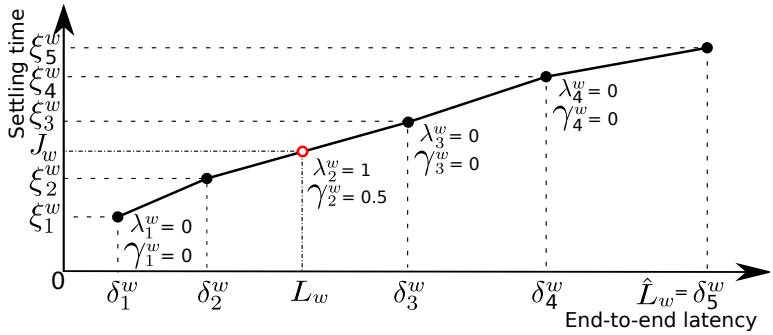


Figure 4.4: Piecewise linear control performance function for the application app_w. The hollow red dot indicates the actual end-to-end latency L_w.

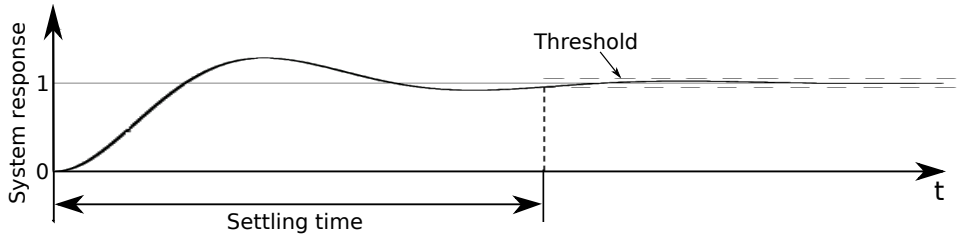


Figure 4.5: Settling time of the control system.

delay. The *performance metric here is the settling time*, ξ, of the system, which is defined as the time taken by the system to reach and stay within a threshold of the reference input, as shown in Figure 4.5. In most cases, the settling time is expected to increase with an increase in end-to-end latency.

Given the control law, continuous-time system matrices, sampling period and end-to-end latency, the closed-loop system can be simulated according to Equations (4.1) and (4.2), and the settling time can be calculated. Thus, it is possible to prepare a look-up table for each control application that contains the values of settling time ξ_k^w for N_w discrete values of end-to-end latency L_w that define the piecewise linear control function of the application, as in Figure 4.4. Such a look-up table can be represented as $LUT = \{(\delta_k^w, \xi_k^w)\}$, $k = 1, \cdots, N_w$. Here, $\xi_k^w \leq \xi_{k+1}^w$ for $\delta_k^w < \delta_{k+1}^w$. This table is used in the objective function of the scheduling algorithm to minimize performance degradation due to application end-to-end latency. Note that this technique elevates scheduling to consider the real application performance.

4.3 Problem Formulation

This section first presents the general problem formulation, followed by an introduction of the scheduling constraints, and concludes with the formulation of the minimization criterion. Given the above model, the aim is to find a schedule of length $H = lcm(p_i \in P)$, called the *hyperperiod* (*lcm* is the least common multiple function) for each considered resource, i.e. ECUs or network links. To preserve the control behavior of the resulting system, tasks are scheduled on ECUs with zero jitter. Definition 2 states precisely what is ZJ scheduling.

On the other hand, since resources can be allocated more efficiently by relaxing jitter constraints while scheduling meesages on the network links, we allow for non-zero-jitter (NZJ) scheduling for the network links, which are typically the scheduling bottleneck of the system. This strategy results in significantly higher utilization of the resources as shown in experiment evaluations of the previous chapter. Note that this decision does not influence control performance, since control performance is sensitive to input and output jitter only [25]. Moreover, unlike Jitter-Constrained (JC) scheduling assumed in the previous chapter, NZJ scheduling does not put any constraints on the jitter of the activities. However, jitters of the messages are implicitly constrained due to ZJ scheduling of the tasks, precedence relations and end-to-end latency constraints.

Similarly to Chapter 3, the schedule is defined by start times $s_i \in \mathbb{Z}$ for the first job (occurrence) of each task $a_i \in T$ and by $s_i^j \in \mathbb{Z}$ for each job of each frame $a_i \in M$, $j = 1, 2, \cdots, n_i^{jobs}$ with $n_i^{jobs} = \frac{H}{p_i}$. However, for the conciseness of the presentation, we define s_i^j for all $a_i \in A$ with $s_i^j = s_i + (j-1) \cdot p_i$ for $a_i \in T$.

4.3.1 Scheduling Constraints

First, *the constraints on activity start times* are given in Equation (4.3). Note that the constraint here is generalized compared to the previous chapter, since we introduced a tunable parameter **c**.

$$\begin{aligned} (j-1) \cdot p_i \leq s_i^j \leq (j + \mathbf{c} - 1) \cdot p_i - e_i, \\ a_i \in A,\ j = 1, 2, \cdots, n_i^{jobs}. \end{aligned} \tag{4.3}$$

Second, the *precedence relation constraints* that ensure the satisfaction of the data dependencies are set in Equation (4.4), where $Pred_i$ denotes the set of directly preceding activities.

$$\begin{aligned} s_i^j - s_k^j \geq e_k,\ a_i, a_k \in A : a_k \in Pred_i, \\ j = 1, 2, \cdots n_i^{jobs}. \end{aligned} \tag{4.4}$$

To prevent collisions of activities on resources, we formulate the *resource constraints*. Equation (4.5) presents these constraints for each pair of jobs of each two activities on the same resource. Here, either job j of activity i is executed after job l of activity k or vice versa.

$$\begin{aligned} s_i^j - s_k^l \geq e_k \ \vee \quad s_k^l - s_i^j \geq e_i, \\ a_i, a_k \in A : map_i = map_k, \\ j = 1, 2, \cdots, n_i^{jobs},\ l = 1, 2, \cdots, n_k^{jobs}. \end{aligned} \tag{4.5}$$

As in [87], we use the Bezout identity to formulate resource constraints for ZJ tasks on ECUs. This formula reduces the number of resource constraints on one ECU relative to Equation (4.5) from $\frac{n_{ECU}^{jobs} \cdot (n_{ECU}^{jobs} - 1)}{2}$ to $2 \cdot |A_{ECU}|$, with n_{ECU}^{jobs} and $|A_{ECU}|$ denoting the total number of jobs and activities on one ECU, respectively. For two tasks to satisfy $s_i^j - s_k^l \geq e_k$ from Equation (4.5), we are interested in all differences $s_i^j - s_k^l = (s_i + j \cdot p_i) - (s_k + l \cdot p_k)$ for $j \in \mathbb{Z},\ l \in \mathbb{Z}$. Bezout identity

states that if p_i and p_k are integers with greatest common divisor $g_{i,k} = \gcd(p_k, p_i)$, then there exist integers j and l such that $p_i \cdot j + p_k \cdot l = g_{i,k}$. More generally, the integers of the form $p_i \cdot j + p_k \cdot l$ are exactly the multiples of $g_{i,k}$. Then, due to the Bezout identity, the difference $(s_i + j \cdot p_i) - (s_k + l \cdot p_k)$ for $j \in \mathbb{Z},\ l \in \mathbb{Z}$ is equivalent to $s_i - s_k + w \cdot g_{i,k},\ w \in \mathbb{Z}$. The smallest positive representative of this set over w is $(s_i - s_k) \mod g_{i,k}$.

Thus, the resource constraints for tasks on ECUs are formulated in Equation (4.6), whereas Equation (4.5) is used for frames on the links.

$$\begin{aligned} (s_i - s_k) \mod g_{i,k} &\geq e_k, \\ (s_k - s_i) \mod g_{i,k} &\geq e_i, \\ a_i, a_k \in T : map_i &= map_k. \end{aligned} \tag{4.6}$$

The example in Figure 4.6 presents a schedule of two ZJ tasks a_1 and a_2 with periods $p_1 = 6$ and $p_2 = 9$. We can see that the value $(s_1 - s_2) \mod g_{1,2} = (1 - 6) \mod 3 = 1$ indicates the minimum distance among all jobs when the jobs in the schedule are in the order $a_2 \to a_1$ in time, whereas $(s_2 - s_1) \mod g_{1,2} = (6 - 1) \mod 3 = 2$ is the minimum distance when $a_1 \to a_2$, which correspond to the left and the right parts of the $\vee$ (OR) expression in Equation (4.5), respectively.

Next, we formulate and prove Theorem 1, which states the necessary and sufficient schedulability condition of two ZJ tasks. Although it is presented here to use the defined concepts, it is applied later in Section 4.5 to reduce problem complexity and computation time of the heuristic. This technique allows us to efficiently schedule ZJ task with a set of pre-scheduled activities on the same resource, which is especially useful since we perform it often.

Theorem 1 (Schedulability of two ZJ tasks)**.** *Two ZJ tasks a_i and a_k can be scheduled without collisions if and only if tasks $a_{i'}$ and $a_{k'}$ with execution times $e_{i'} = e_i$, $e_{k'} = e_k$ and periods $p_{i'} = p_{k'} = g_{i,k}$ can be scheduled without collisions.*

Proof. If the initial tasks a_i and a_k are schedulable, Equations (4.6) hold for some s_i and s_k. We set $s_{i'} = s_i \mod g_{i,k}$ and $s_{k'} = s_k \mod g_{i,k}$. Due to the additive property of modular arithmetic, $(s_{i'} - s_{k'}) \mod g_{i,k} = (s_i \mod g_{i,k} - s_k \mod g_{i,k}) \mod g_{i,k} = (s_i - s_k) \mod g_{i,k}$. Thus, Equations (4.6) also hold for $s_{i'}$ and $s_{k'}$.

In the other direction, if Equations (4.6) hold for $a_{i'}$ and $a_{k'}$ for some $s_{i'}$ and $s_{k'}$, we set $s_i = s_{i'}$ and $s_k = s_{k'}$. Then, Equations (4.6) trivially hold for s_i and s_k. □

Note that since tasks $a_{i'}$ and $a_{k'}$ have the same period, it is sufficient to check the collision of the first job of the tasks only.

Finally, the *constraints on end-to-end latency* are formulated in Equations (4.7) with $Succ_i$ being the set of directly succeeding activities. The end-to-end latency is the maximum time between start and end for any pair of root and leaf activities, i.e., activities with no predecessors and successors, respectively. Since the roots and

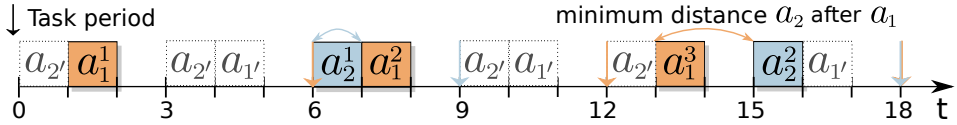


Figure 4.6: A schedule of two ZJ tasks a_1 and a_2 with periods $p_1 = 6$ and $p_2 = 9$ with the minimum distance between two jobs equal 1 from a_2 to a_1 and 2 from a_1 to a_2. Dotted lines are ZJ tasks $a_{1'}$ and $a_{2'}$ with $p_{1'} = p_{2'} = g_{1,2} = 3$ from Theorem 1.

leafs are always tasks, we omit job indices. Note that L_w is a variable completely determined by the start times of activities.

$$\begin{aligned} s_i + e_i - s_k &\leq L_w, \\ a_i, a_k \in app_w : \ Succ_i = \emptyset, \ Pred_k &= \emptyset, \\ L_w \leq \hat{L}_w = \mathbf{c} \cdot p_w, \ w &\in App \end{aligned} \tag{4.7}$$

4.3.2 Control Degradation Minimization

We minimize the accumulated performance degradation over all applications in Equation (4.8), where the control performance value J_w for each application is defined by Equations (4.9) and (4.10). This is essentially a linearization of an interpolation of the piecewise linear function defined by the look-up table LUT.

We define the binary variable λ_k^w for each interval $k = 1, \cdots, N_w$ and each application $w \in App$ as

$$\lambda_k^w = \begin{cases} 1, & \text{if latency } L_w \text{ is in interval } [\delta_k^w, \delta_{k+1}^w); \\ 0, & \text{otherwise,} \end{cases}$$

where $\lambda_{N_w}^w = 1$ states that $L_w = \delta_{N_w}^w$, since the end-to-end latency cannot be greater than $\delta_{N_w}^w$. As stated in Section 4.2, the end-to-end latency is bounded, therefore it always holds that $\delta_{N_w}^w = \hat{L}_w = \mathbf{c} \cdot p_w$.

Additionally, the rational variable $\gamma_k^w \in [0, 1)$ is the position of L_w in the interval $[\delta_k^w, \delta_{k+1}^w)$. Equation (4.11) guarantees that only one interval is chosen, whereas Equation (4.12) allows γ_k^w for to be nonzero only for the interval $[\delta_k^w, \delta_{k+1}^w)$, where L_w lies. Note that if $\lambda_{N_w}^w = 1$, the second summand of Equations (4.9) and (4.10) is zero, and the values are set for the last pair of values in the look-up table LUT.

$$\textit{Minimize:} \sum_{w \in App} J_w = \Phi \tag{4.8}$$

$$J_w = \xi_{N_w}^w \cdot \lambda_{N_w}^w + \sum_{k=1}^{N_w - 1} (\xi_k^w \cdot \lambda_k^w + \gamma_k^w \cdot (\xi_{k+1}^w - \xi_k^w)), \tag{4.9}$$

$$L_w = \delta_{N_w}^w \cdot \lambda_{N_w}^w + \sum_{k=1}^{N_w - 1} (\delta_k^w \cdot \lambda_k^w + \gamma_k^w \cdot (\delta_{k+1}^w - \delta_k^w)), \tag{4.10}$$

$$\sum_{k=1}^{N_w} \lambda_k^w = 1, \tag{4.11}$$

$$\begin{gathered} \gamma_k^w \leq \lambda_k^w, \\ w \in App,\ k = 1, \cdots, N_w. \end{gathered} \tag{4.12}$$

For the example in Figure 4.4, where the hollow red dot indicates the actual end-to-end latency value L_w, one can observe that $\gamma_2^w = 0.5$, since L_w lies in the middle of the interval (δ_2^w, δ_3^w).

4.4 Optimal Approaches

Two exact approaches are formulated in this section, ILP and CP models. We present two models to experimentally compare their efficiencies for the considered problem. Furthermore, we also implemented an SMT model for this problem, straightforwardly using the constraints as in the problem statement. To formulate the criterion, we used if-then statements. However, the introduction of non-integer arithmetic by piecewise linear functions resulted in a significant increase of computation time for Z3 Microsoft solver. Since it runs tens or even hundreds of times longer, we do not compare it further with ILP and CP models.

Following the problem statement, both approaches are based on decision variables s_i^j that indicate the start time of job j of activity a_i. Note that there are two strategies to formulate precedence relations allowing for a Directed Acyclic Graph (DAG) of data dependencies to span over several periods, as Application 1 in Figure 4.3. The first one is to bound start time variables as $s_i^j \leq p_i$. Then, a new variable σ_i^j per start time variable s_i^j would be introduced to indicate in which period the start time variable should be considered for precedence relations to hold. For the example schedule in Figure 4.3, the values for this strategy are $s_1^1 = 4$, and it is executed in period 1, i.e., $\sigma_1^1 = 1$, whereas $s_4^1 = 4$ and $\sigma_4^1 = 2$. Another strategy is to allow for start time variables s_i^j to go beyond its period. For the example in Figure 4.3, the values are $s_1^1 = 4$ and $s_4^1 = 9$. The advantage of this strategy is less variables and simpler constraints, whereas the disadvantage is that the new resource constraints (4.3) for frames in the next $\mathbf{c} + 1$ hyperperiods (a tunable parameter) and the first frame in hyperperiod $\mathbf{c} + 2$ should be added (assuming there is an activity with period equal to the hyperperiod). The reason is that one frame a_k can be scheduled at time s_k and another at time $H + s_k$ on the same network link. This scheduling causes a collision, since the schedule repeats after H time units. Due to a more general view of resource constraints (4.6) for tasks on ECUs, they guarantee the absence of collisions without introducing constraints in further hyperperiods.

Furthermore, since a start time variable can take values from multiple periods, consecutive jobs of the same frame can collide. To prevent this situation, we introduce Constraint 4.13 to both exact models, which creates a precedence constraint between each pair of consecutive jobs of one frame, considering the last and first jobs also.

$$\begin{gathered} s_i^j + e_i \leq s_i^{j+1}, \\ s_i^{n_i^{jobs}} + e_i \leq s_i^0 + H, \\ a_i \in T,\ j = 1, \cdots, n_i^{jobs} - 1. \end{gathered} \tag{4.13}$$

Although the criterion given in Equation (4.8) can be straightforwardly implemented in CPLEX [52]used to solve the ILP model, it also has a built-in function for piecewise linear functions. However, the ILP approach runs faster with the straightforward implementation, therefore we use it for the ILP model. On the other hand, since the CP approach works with integer variables only, it is both complicated and inefficient to implement the criterion straightforwardly. Therefore, we use a built-in function for the CP model.

4.4.1 Integer Linear Programming Model

Due to modulo operation, the only non-linearity in the problem formulation in Section 4.3 lies in the resource constraints for both tasks on the ECUs and frames on the network links. As in the previous chapter, we formulate resource Constraint (4.5) for frames with a set of decision variables that reflect the relative order of every pair of jobs of different activities:

$$x_{i,k}^{j,l} = \begin{cases} 1, & \text{if } a_i^j \text{ starts before } a_k^l; \\ 0, & \text{otherwise.} \end{cases}$$

Resource Constraint (4.14) ensures that either a_i^j is executed before a_k^l (the first equation holds and $x_{i,k}^{j,l} = 1$) or vice versa (the second equation holds and $x_{i,k}^{j,l} = 0$). We set $\mathbb{M} = \max(UB_i^j + e_i - LB_k^l,\ UB_k^l + e_k - LB_i^j)$, where UB_i^j and LB_i^j are the maximum and the minimum values that s_i^j can take, respectively. This is derived from Equation (4.14) using bounds on the start time variable.

$$\begin{aligned} s_i^j + e_i &\leq s_k^l + \mathbb{M} \cdot (1 - x_{i,k}^{j,l}), \\ s_k^l + e_k &\leq s_i^j + \mathbb{M} \cdot x_{i,k}^{j,l}, \\ a_i,\ a_k \in A, j = 1, ..., n_i^{jobs},\ l &= 1, ..., n_k^{jobs}. \end{aligned} \tag{4.14}$$

As the number of resource constraints for frames grows very quickly with the size of the problem instance, we use *lazy constraints*. At the beginning, the solver generates constraints for the jobs in the first hyperperiod only. Each time the solver finds a new solution, it looks for collisions in the schedule and, if necessary, adds the missing resource constraints.

Instead of non-linear Constraint (4.6) for tasks using the modulo operator, we use its linear version derived in [87] from Equation (4.6). A new integer variable called the *quotient variable* $q_{i,k} \in \mathbb{Z}$ is introduced for each ordered pair of different tasks $(a_i,\ a_k) \in T^2$. This variable is the quotient from the modulo operation, an artificial variable completely determined by the start times as follows:

$$q_{i,k} = \left\lceil \frac{s_i - s_k}{g_{i,k}} \right\rceil \tag{4.15}$$

Thus, Constraints (4.6) are linearized in Equations (4.16), which realize the definition of modulo operator by restricting the value to be less than $g_{i,k}$.

$$\begin{aligned} e_i &\leq s_k - s_i + q_{i,k} \cdot g_{i,k} < g_{i,k}, \\ e_k &\leq s_i - s_k + q_{k,i} \cdot g_{i,k} < g_{i,k}, \end{aligned} \tag{4.16}$$

We sum up the two Equations (4.16) and obtain $e_i+e_k \leq q_{i,k}\cdot g_{i,k}+q_{k,i}\cdot g_{i,k} < 2\cdot g_{i,k}$. The right inequality results in $q_{i,k} + q_{k,i} < 2$. Since the quotients are integer variables and the tasks have nonzero execution times, $q_{i,k} + q_{k,i} = 1$, which leads us to the substitution $q_{i,k} = 1 - q_{k,i}$. Moreover, we discard the strict inequality in Equation (4.16), since it is guaranteed by the substitution. This technique also reduces the number of variables and resource constraints by half.

For higher computational efficiency, we bound the quotient variables in the following manner: $\frac{LB_i-UB_k}{g_{i,k}} - 1 < q_{i,k} \leq \frac{UB_i-LB_k}{g_{i,k}}$. These values are derived straightforwardly from Equations (4.16).

4.4.2 Constraint Programming Model

Although it is possible to implement the model from Section 4.3 straightforwardly, experiments have demonstrated that the usage of solver-specific constraints results in a significantly lower computation time. Therefore, this section is tool-specific to provide the guidelines for the implementation in the CP solver. All decision variables s_i^j and L_w are implemented in the CP solver as interval variables, which is a special type of variable suited specifically for scheduling problems. Each activity job a_i^j is represented by a rectangle in a schedule, as in Figure 4.3, and has two fields, $a_i^j.startOf$ denotes the start time and $a_i^j.length$ denotes the execution time.

Constraint (4.7), which sets the end-to-end latency of each application, is realized with the $span(L_w, A_w)$ function. It states that the value of the variable L_w must be measured from the start of the variable with the lowest *startOf* in the set of variables A_w to the end of the variable with the highest *startOf* in A_w. In our case, $A_w = \{a_i \in A : Pred_i = \emptyset \vee Succ_i = \emptyset\}$. Thus, we set the end-to-end latency variable L_w to span all root-to-leaf pairs of variables for this application. Then, we set $L_w.length \leq \mathbf{c} \cdot p_w$ using *.setLengthMax* function to set bounds on the latency, as in Equation (4.7).

Constraints on activity start times (4.3) are set with the $s_i^j.setStartMin((j-1)\cdot p_i)$ and $s_i^j.setEndMax((j + \mathbf{c}) \cdot p_i - 1)$ functions, taking as argument the value to set. Then, precedence relation constraints (4.4) are formulated with function $endBeforeStart(s_i^j, s_k^j)$, which states that job j of activity a_i should end before job j of activity a_k may start.

Moreover, experimental evaluations have demonstrated that the most efficient method to implement resource Constraints (4.5) and (4.6) in the CP optimizer is to add a constraint $noOverlap(A_r)$ for set of activities $A_r \in A$ on resource r, i.e. $A_r = \{a_i \in A : map_i = r\}$. Additionally, to avoid collisions we create variables s_i^j for $j = 1, \cdots, (\mathbf{c}+1)\cdot n^{jobs}+1$. We link variables in the second, third, etc. hyperperiods with the variables in the first hyperperiod using constraint *startAtStart*.

Finally, we set the parameter *Workers* to 1, indicating the number of workers running in parallel to solve problem. It has experimentally shown significant reduction of the computation time. From our experience, setting this parameter to 1 typically reduces computation time in the current version of CP optimizer (IBM ILOG CPLEX Optimization Studio 12.8) independently of the problem and server characteristics on which it is run.

4.4.3 Computation Time Improvements

Since the periodic scheduling problem with precedence constraints is a generalization of the job shop scheduling (JSS) problem [12], we adjust a disjunctive graph representation [20] that has shown good results for JSS to *reduce the number of generated resource constraints for frames on network links* (4.5). While the optimizaition itself is presented later in this section, we first introduce the necessary concepts for this improvement. A disjunctive graph comprises a node for each activity job and an arc between pair of jobs either mapped to the same link or in precedence relations. The arcs are of two types, conjunctive and disjunctive. A conjunctive arc is a directed arc reflecting the order of jobs in time, whereas a disjunctive arc is an undirected arc between two jobs on the same resource with undefined order. The disjunctive graph for periodic applications from Figure 4.2 is shown in Figure 4.7, where conjunctive arcs are given by precedence relations, whereas disjunctive arcs are defined by all pairs of activity jobs on the same resource. Since the source of NP-hardness of the problem is related to disjunctive arcs, reducing the number of disjunctive arcs in the problem decreases the computation time. Observe that disjunctive arcs have the same meaning as the $x_{i,k}^{j,l}$ variables in the ILP problem, i.e., $x_{i,k}^{j,l} = 1$ means that the conjunctive arc is directed from a_i to a_k and $x_{i,k}^{j,l} = 1$, vice versa. Therefore, we create resource constraints (4.14) only for activity jobs, which are connected by disjunctive arcs, and for the conjunctive arcs we add precedence constraints (4.4), which are resource constraints (4.14) with variable x set to the value according the conjunctive arc direction.

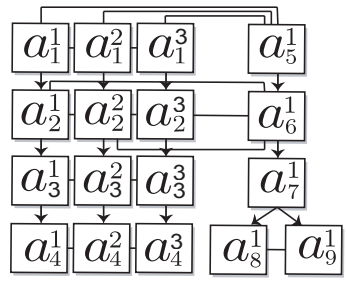


Figure 4.7: Disjunctive graph for applications from Figure 4.2.

To deduce disjunctive arc directions we first compute the length of the longest critical path of the preceding and succeeding activities that must be executed before, t^b, and after, t^a, the given activity, as in Equations (3.14) and (3.15) from Chapter 3, respectively. Then, we set $LB_i^j = (j-1) \cdot p + t_i^b$ and $UB_i^j = (j + \mathbf{c} - 1) \cdot p - t_i^a$. Moreover, to *reduce the symmetry in the solution space*, we set $UB_i = p_i - 1$ for roots, since the schedule with $s_i = p_i$ is equivalent to the one with start time $s_i = 0$ due to the ZJ nature of the tasks.

Finally, we deduce disjunctive arc directions with Condition (4.17), which checks whether the scheduling interval of a_i^j always precedes the scheduling interval of a_k^l. We check this condition for each pair of jobs with a disjunctive arc.

$$LB_k^l + e_k > UB_i^j \tag{4.17}$$

4.5 Heuristic Approach

This section introduces a conceptually easy-to-follow, yet efficient heuristic approach that first constructs a feasible solution and then optimizes it using local neighborhood search. Like in the previous chapter, the heuristic approach solves larger problem

instances than the optimal approaches in a reasonable time, although sacrificing the optimality of the solution within acceptable limits.

4.5.1 Overall Approach

The proposed heuristic approach comprises two stages, firstly looking for a feasible solution and, secondly, improving it. The stages are called *feasibility* and *optimization stage*, respectively. The feasibility stage is similar to 3-LS heuristic presented in Section 3.4 in the sense that both are constructive and solve similar scheduling problems. However, the feasibility stage of the heuristic proposed in this chapter addresses an important drawback of the 3-LS heuristic from the previous chapter, lacking flexibility.

The first possible reason for the inflexibility of the 3-LS heuristic is that it works with fixed start times of activities, i.e., with schedules. Here, we provide more flexibility by *changing the activity orders*, implemented as priority queues, from which schedules are constructed. While the 3-LS heuristic unschedules some activities if a feasible solution cannot be found, the heuristic presented in this chapter makes this decision itself after the order is changed. Although this may prolong the computation time, working with priority queues instead of start times results in a higher success rate of finding a feasible solution.

The second reason for inflexibility of the 3-LS heuristic is that it considers a single granularity level, i.e. scheduling and unscheduling by activities only. In contrast, this heuristic *adjusts the granularity of the elements in the priority queue*. Firstly, the activities of one applications are put together, then the activities of one application can be separated in the queue if the heuristic failed to schedule some activity. Finally, if only some jobs of a message cannot be scheduled, the message is split into jobs in the queue. Note that tasks on the ECUs are always scheduled as one entity, since a ZJ activity, according to Definition 2, derives the schedule for all jobs from the first one straightforwardly.

This procedure ensures the coarsest possible level of scheduling granularity at each stage of the algorithm, since we do not increase number of scheduling entities when it is not necessary. This results in a reduced computation time of the heuristic.

The feasibility stage is shown in Figure 4.8. It constructs a schedule element-by-element based on a current priority queue Q (Steps 2 and 3), being the *internal loop*. If the schedule is obtained, the optimality stage starts. However, when some element ϵ_i cannot be scheduled, it is split into jobs if it is a message and there are jobs that were scheduled (Step 5). Then, the element ϵ_d preventing ϵ_i from being scheduled, is found (Step 6), and problematic jobs of ϵ_i with its predecessors are put before ϵ_d in Q (Step 7). We call Steps (2-8) the *external loop* of our algorithm. The feasibility stage stops when either all jobs are scheduled or an infinite loop is detected.

When the feasibility stage finds a solution, the optimization stage iteratively applies a local neighborhood search [82] to improve the solution. In each iteration, it solves the optimization problem described in Section 4.3 for a chosen set of applications, a *neighborhood*, while fixing the schedule for the rest of the applications to the best one found thus far. Therefore, it looks *locally* in the neighborhood defined by the chosen set of applications. This local neighborhood search strategy

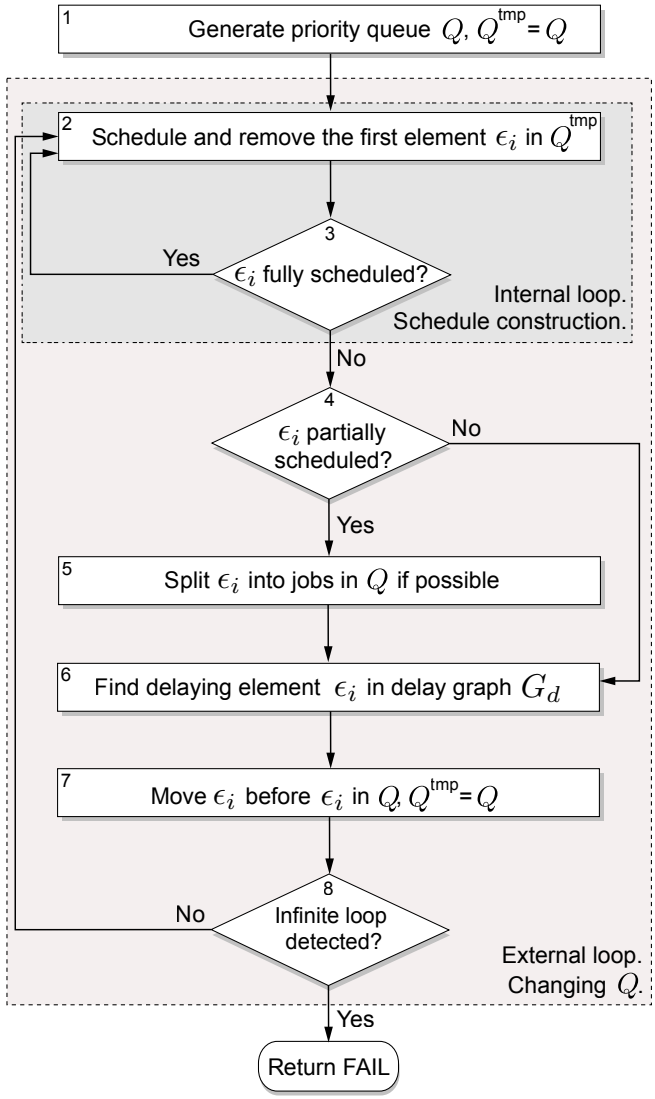


Figure 4.8: Outline of the feasibility stage of the heuristic approach.

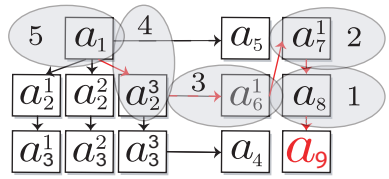


Figure 4.9: Example of a delay graph for applications from Figure 4.2 scheduled as in Figure 4.3. Solid lines indicate precedence relations, whereas dashed lines indicate resource constraints. Delay elements on levels 1 to 5 are shown for a_9.

reduces the computation time by squeezing the search space, considering only a subset of the decision variables. The optimization stage reuses the CP and ILP formulations from Section 4.4. We provide an experimental comparison of the efficiency of these two approaches in the optimization stage in Section 4.6.

4.5.2 Feasibility Stage

First, we introduce the novel concept of a *delay graph*, which is used to find the prescheduled element preventing the current element from being scheduled. Second, we outline the *sub-model* that schedules an element, given a partial schedule of the higher priority elements similarly to the previous chapter. Finally, we present the feasibility stage in detail.

Delay Graph to Modify Priority Queue

We introduce the novel concept of a *delay graph* to find an element ϵ_d preventing the current element ϵ_i from being scheduled. This concept also ensures termination of the external loop. A delay graph G_d is a directed acyclic graph with frame jobs and tasks as nodes. An edge is directed from one node to another if removing the former node from Q can result in an earlier start time of the latter node.

To make the scheduling process more straightforward, we do not allow an element to be considered by the scheduler before its predecessors in the priority queue Q. Therefore, the earliest time that ϵ_i can be scheduled is the maximum of the finish times of its predecessors, i.e., $\hat{s}_i = \max_{\epsilon_k \in Pred_i}(s_k + e_k)$.

If no activity prevents ϵ_i from being scheduled directly after the latest predecessor, i.e. at time $\hat{s}_i$, we add edge from its latest predecessor to ϵ_i in the delay graph. Otherwise, we add an edge from the element preventing the considered element from being scheduled at $\hat{s}_i$ to ϵ_i. Thus, we can only have an edge from an element on another resource if it is a predecessor. Otherwise, it is a resource constraint on the resource to which the element is mapped. If an element cannot be scheduled at $\hat{s}_i$ due to resource constraints, one element causing the delay is always found, namely, job ϵ_d^j, such that $s_d \leq \hat{s}_i \leq s_d + e_d$, i.e., scheduled at this time moment.

Figure 4.9 shows a delay graph for applications from Figure 4.2 scheduled as in Figure 4.3. The nodes are ZJ tasks and NZJ frame jobs, as mentioned before. Here, the priority queue is set as $Q = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9\}$. For all elements except a_5, a_6, and a_9, a corresponding parent is its predecessor. Observe that a_5 could be scheduled at time 0, since it has no predecessors. However, it is scheduled

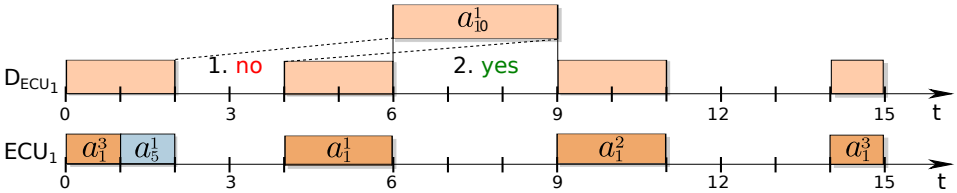


Figure 4.10: Example of the interval set D_{ECU_1} used to increase the efficiency of the heuristic for a given schedule on ECU_1.

at time 1 due to a_1 being scheduled at time 0. Therefore, we put an edge from a_1 to a_5. The same reasoning holds for pairs a^3_2 and a^1_6, and a_8 and a_9.

Sub-model to Schedule an Element

We find a start time for an element, respecting the previously scheduled set of elements by a sub-model. It uses the time interval representation of the occupied resource to find the earliest start time for the currently scheduled element, such that precedence, end-to-end latency, ZJ, and resource constraints hold. The sub-model respects the precedence and end-to-end constraints by setting the minimum and the maximum possible values for the start time. Finally, it finds the earliest possible start time for a given element going through the set of occupied intervals within the given bounds, applying a trick to reduce the time required to detect collisions using Theorem 1.

Algorithm 4.1 presents the sub-model. As input, it takes an element ϵ_i to be scheduled; $\hat{s}_i$ and $\check{s}_i$, the earliest and the latest possible start times of this element due to the predecessor and end-to-end latency constraints, respectively; the set of predecessor elements $\overline{Pred_i}$ finishing at $\hat{s}_i$; and the set of resource occupation intervals D_r, which is the union of time intervals in which prescheduled jobs are running on resource $r = map_i$. D_{ECU_1} for the schedule of ECU_1 from Figure 4.3 is shown in Figure 4.10.

Similarly to the sub-model of the 3-LS heuristic, the set D_r is a union of time intervals, for which resource r is occupied by already scheduled activities. For the example in Figure 4.10, $D_{ECU_1} = \{[0, 2], [4, 6], [9, 11], [14, 15]\}$. It is introduced to reduce computation time of the sub-model, since instead of going over all scheduled elements and checking that no element is already scheduled at a given time, we can iterate over intervals in D_r. Formally, D_r is a union of nonintersecting intervals, sorted in ascending order, i.e., $D_r = \cup_{f=1}^{n_{int}}[l_f, u_f]$ with $l_f, u_f \in \mathbb{N}$ due to discrete-time and integer execution times and $u_f < l_{f+1}$.

To schedule ZJ task a_i, we use Theorem 1, which states the necessary and sufficient condition for the schedulability of two ZJ tasks. For each prescheduled task a_k on the ECU, we add execution time intervals of task $a_{k'}$ with the same execution time to D_r, but with a period equal to $p_{k'} = g_{i,k}$ (Line 9). Thus, while scheduling a ZJ task, we only need to go over time intervals for the first task job instead of looking at time intervals for all the jobs, resulting in significantly reduced computation time, since we construct D for scheduling each ZJ task.

The sub-model always schedules an element as soon as possible. The motivation is twofold: first, it typically produces schedules with smaller fragmentation in each iteration. This is an especially sensitive issue here, since we have both short (control

Algorithm 4.1 Sub-model to schedule an element

1: Inputs: ϵ_i, $\hat{s}_i$, $\check{s}_i$, $\overline{Pred_i}$, D_r
2: **if** $\epsilon_i \in T$ **then** // Treat ZJ tasks
3: **for** $\epsilon_k \in T : map_k = map_i$ **do**
4: $s^1_{k'} = s_k \mod g_{i,k}$
5: $j = 1$
6: **while** $s^j_{k'} \leq \min\{\check{s}_i, H\}$ **do**
7: $s^j_{k'} = s^1_{k'} + (j-1) \cdot g_{i,k}$
8: $j = j + 1$
9: D_r.add($[s^j_{k'}, s^j_{k'} + e_k]$)
10: **end while**
11: **end for**
12: **end if**
13: **for** $f = 1$ **to** n_{int} **do** // Find the earliest start time of the element
14: **if** $\min(l_f, \check{s}_i) - \max(u_{f-1}, \hat{s}_i) \geq e_i$ **then**
15: $s_i = \max(u_f, \hat{s}_i)$
16: **if** $s_i = \hat{s}_i$ **then**
17: $parent_i = \overline{Pred_i}$
18: **else**
19: $c = \{z \in \{1, \cdots, n_{int}\} : u_z \leq s_i < l_{z+1}\}$
20: **end if**
21: **end if**
22: **end for**
23: **if** ϵ_i is not scheduled **then**
24: $c = \{z \in \{1, \cdots, n_{int}\} : l_z \leq \hat{s}_i < u_z\}$
25: **end if**
26: $parent_i = \{\epsilon_d \in A : s_d + e_d = u_c\}$
27: Output: s_i

traffic) and long (video traffic) transmission times of frames on the network. Thus, frames with long transmission times have higher risk of not being scheduled given a schedule with high fragmentation, whereas frames with short transmission times can heavily fragment the schedule. Although sometimes scheduling at the earliest time can result in higher fragmentation of the schedule because of precedence constraints, experiments have shown that this strategy shows a good success rate comparing to other strategies. The second reason is that this strategy results in solutions with lower end-to-end latency and, therefore, better objective value.

In the rest of this subsection, we describe the implementation details of the sub-model. The main loop of the sub-model iterates over intervals in D_r until it finds free space in the schedule after $\hat{s}_i$ to put the element respecting its execution time e_i. Then, if there is a free time interval of length e_i starting at $\hat{s}_i$, we set the parents in the delay graph to elements in $\overline{Pred_i}$ (i.e., to the predecessor(s) finishing at $\hat{s}_i$) (Line 17). Otherwise, the delaying element is the element scheduled last in the occupied interval before s_i (Line 19). If the current element cannot be scheduled, we choose the element finishing at the end of the interval to which $\hat{s}_i$ belongs (Line 24). In the example in Figure 4.10, we assume a new element a^1_{10}

to schedule on ECU_1 with execution time $e_{10} = 3$, and without predecessors. The sub-model first checks the interval $[2, 4]$, which is not enough, and then it finds that in interval $[6, 9]$ we can schedule a_{10}^1. The parent in the delay graph is set to a_1^1 applying Line 26, i.e. looking at the last element scheduled in the interval $[4, 6]$.

It may be necessary to go beyond one hyperperiod, since s_i^j can be maximally $(j+\mathbf{c})\cdot p_i$ due to Equation (4.3). For this purpose, we look at D_r in the hyperperiods, where the interval $[\hat{s}_i, \check{s}_i]$ lies. Due to periodicity of the schedule, we just add the corresponding number of hyperperiods to the interval edges l_f and u_f.

Algorithm

The proposed heuristic approach is presented in Algorithm 4.2. The inputs are the set of activities A, the rule to set the priority queue Pr_{feas}, and the maximum running time of the heuristic t_{max}^{feas}. After populating the priority queue Q (Line 2), the algorithm iterates over the external loop, where it adjusts the priority queue according to the feedback it gets from the internal loop. If some element ϵ_i failed to be scheduled by the sub-model (Line 12), we first find the delaying element ϵ_d (Line 13) on the current *delay level* l_d. Delay level defines which delay element ϵ_d for ϵ_i to set, depending on the distance (number of edges) in the delay graph.

We obtain an element ϵ_d on delay level l_d from ϵ_i in the delay graph G_d by a graph transformation in the following manner. We go from ϵ_i via edges in the opposite direction, storing an element on the second end of the edge only if it is not a predecessor of the node on the first end of the edge. Then, ϵ_d is the element on the l_d position in this sequence of stored elements. For the example in Figure 4.9, the parent on the first level for the activity a_9 is $\epsilon_d^1 = a_8$ and on the second level it is $\epsilon_d^2 = a_2^3$.

When at least one of ϵ_i and ϵ_d is different from the previous iteration, the delay level is reset to 1 (Line 18). In contrast, when the last prioritizing ϵ_i over ϵ_d in the priority queue did not have the desirable effect, we increase the delay level by one and set the new delaying element (Line 16). Finally, the current problematic element and all of its predecessors are put immediately before ϵ_d in Q. This strategy prevents us from becoming stuck with the same problematic element. Moreover, applying this strategy, we aim not to disturb the prescheduled elements, interfering more only if necessary.

The algorithm is terminated either when we have a complete solution or when we detect an infinite loop over iterations of the external loop. Since both the internal and external loops are deterministic, we identify an infinite loop when we encounter the same priority queue Q for the second time. However, since we aim for problem instances with more than 100000 jobs, we detect infinite loops in the sequence of pairs (ϵ_i, ϵ_d) to reduce the computation time. We detect an infinite loop if the same sequence of pairs (ϵ_i, ϵ_d) appears in Step 7 of Figure 4.8 twice. We additionally set a time limit t_{max}^{feas} on the running time of the heuristic, since sometimes detection of the infinite loop can be very time consuming, especially for larger instances. We set $t_{max}^{feas} = 300$ seconds in our experiments in Section 4.6. The time limit is hit only for 2% of the largest problem instances, with no effect on problem instances of smaller and medium sizes.

We exploit the advantage of a low running time of the feasibility stage of the

Algorithm 4.2 Feasibility stage of the heuristic approach

```
1: Inputs: A, Pr_feas, t_max^feas
2: Q = sort(A, Pr_feas)
3: l_d = 1
4: while Infinite loop not detected and running time less than t_max^feas do
   // External loop. Changing Q:
5:    ŝ_i = 0
6:    Pred_i = ∅
7:    Q' = Q
8:    while Q' is not empty or ε_i is scheduled do
      // Internal loop. Schedule construction:
9:       ε_i = Q'.pop()
10:      s_i = sub-model(ε_i, ŝ_i, Pred_i, D_map_i)
11:      D_map_i.update(), ŝ.update(), Pred.update()
12:      if ε_i is not scheduled then
13:         ε_d = G_d.getParent(l_r)
14:         if (ε_i, ε_d) is the same as previously then
15:            l_d = l_d + 1
16:            ε_d = G_d.getParent(l_r)
17:         else
18:            l_d = 1
19:         end if
20:         Q.putBefore({ε_i, AllPred_i}, ε_d)
21:      end if
22:   end while
23: end while
24: Output: S
```

heuristic and always run it with three strategies to set priorities Pr_{feas}:

1. in increasing order of the slack values $UB_i - LB_i$;
2. in decreasing order of amounts of transmitted data, since it is hard to schedule late large chunks non-preemptively. We break ties by the increasing order of the slack values;
3. in decreasing order of possibility to improve the objective value (settling time), i.e., $\xi^w_{N^w} - \xi^w_1$,

choosing the solution with the best objective value. This strategy is beneficial, since for instances with high utilization, we need to target the feasibility with strategies 1) and 2). On the other hand, for less-utilized systems, it is beneficial to use strategy 3).

4.5.3 Optimization Stage

The optimization stage works iteratively by searching in the neighborhood of solution S_{best} with the best objective value Φ_{best} found thus far. The neighborhood are

solutions with different start times of activities in a subset of applications only.

Algorithm 4.3 presents the optimization stage of the heuristic, where the input values are the schedule obtained by the feasibility stage S with objective value Φ_S, the number of applications considered in the neighborhood N_{apps}, the number of solutions in the neighborhood N_{sol}, the improvement tolerance τ_{opt} to stop the search, the priority rule Pr_{opt} to choose the neighborhood and the maximum computation time t^{opt}_{max} for one run of the optimal model (either CP or ILP), which is used to evaluate a neighbor.

Algorithm 4.3 Optimization stage of the heuristic approach

```
1: Inputs: S, N_apps, N_sol, τ_opt, Pr_opt, t^opt_max
2: Φ_best = Φ_S
3: Φ_cur = Φ_best + 2 · τ_opt   S_best = S
4: while Φ_cur − Φ_best > τ_opt do
5:    Φ_cur.initialize()
6:    for f = 1 to N_sol do
7:       S_neigh = App.getApps(N_apps, N_sol, Pr_opt)
8:       S = solve(S_best, t_max)
9:       Φ_cur.update(), S_cur.update()
10:   end for
11:   Φ_best.update(), S_best.update()
12: end while
13: Output: S^best
```

Start times of activities of applications that are not in the current neighborhood are fixed to the corresponding values in S_{best}, whereas start times of activities of applications chosen to be in the neighborhood can be changed. For each iteration of the inner loop, the neighborhood set $S_{neigh} \in App$ ($|S_{neigh}| = N_{apps}$) is generated (Line 7). Then, the resulting problem is solved by the CP or ILP approaches from Section 4.4. The computation time is limited by t_{max} to address the time complexity of optimal approaches.

While looking at the neighbors of the current solution, the neighbor with the best objective value is stored (Line 9). After evaluating all neighbors by CP or ILP, S_{best} is updated. Neighbors of the new solution are generated in the next iteration if the improvement over the previous iteration was larger than τ_{opt}. Otherwise, the search procedure is stopped and the best solution is returned.

We experimentally compared three priority rules Pr_{opt} to set applications in neighborhood:

1. random;

2. according to the application *room for improvement*, which is defined as the difference of the current control performance value and the minimal possible objective value of the application, i.e., $J_w - \xi_1^w$;

3. choosing one application w with the highest value $J_w - \xi_1^w$ and the other applications with the highest number of shared resources with w.

Table 4.1: Platform-generation parameters

Set	*P [ms]*	n_T	n_{exp}	m_{ECU}	r_{min}
1	$1, 2, 5, 10$	20	15	2	0.45
2	$1, 2, 5, 10$	30	15	2	0.5
3	$1, 2, 5, 10, 20, 50, 100$	50	30	2	0.6
4	$1, 2, 5, 10, 20, 50, 100$	100	30	3	0.65
5	$1, 2, 5, 10, 20, 50, 100$	500	50	8	0.7

Unlike the first and third strategies, which result in a very long computation times since it stagnates very slowly, the second strategy exhibits a reasonable trade-off between computation time and solution quality. Thus, we use only Strategy 2 to set application in neighborhood.

4.6 Experimental Results

This section experimentally evaluates and compares the proposed optimal models and the heuristic approach on synthetic problems regarding both the feasibility and optimality aspects. In addition, we quantify the trade-off between computation time and quality of the solution of heuristic and optimal approaches, where the quality is both maximal achievable utilization of the resources and control performance of the applications. The experimental setup is presented first, followed by two experiments, the first evaluating control performance and the second quantifying the utilization gain. We conclude by demonstrating our approach on a realistic automotive case study.

4.6.1 Experimental Setup

Experiments are performed on problem instances with the same application specification, as was used in the previous chapter, generated by a tool developed by Bosch [65]. The applications are executed on a platform similar to the one in Figure 4.1, i.e., there are a number of switches (domains) to which ECUs are connected. Moreover, we synthetically generate the control performance degradation values based on the simulated values for realistic control applications. There are five sets of 100 problem instances of different sizes. The generation parameters for each set are presented in Table 4.1, and the granularity of the timer is set to be 1 μs.

Each generated set of problem instances comprises a given number of tasks n_T, and we set the expected number of tasks executed on one ECU, n_{exp}. We compute the number of ECUs as $m_{ECU} = \lceil \frac{n_T}{n_{exp}} \rceil$. The number of tasks executed on one ECU is different among the sets to obtain a reasonable network utilization, not too low for sets with less tasks and still schedulable for the largest set. Although our approach can work with any number of domains m_{Dom}, there are 2 domains only for the set with the largest instances (Set 5), since typically, 5-6 ECUs are interconnected by one switch.

The mapping of tasks to ECUs is done in the following way. The probability of interdomain communication is set to 0.2, i.e., 20% of communicating tasks are situated on ECUs in different domains. Note that setting this parameter too high

results in unschedulable instances with overloaded links between switches. The mapping of tasks to ECUs is performed such that the load is balanced across the cores, i.e., the resulting mapping utilizes all cores approximately equally. The mapping of frames to the links, on the other hand, follows straightforwardly from the platform topology and the task mapping as there is always exactly one route between each pair of ECUs.

Each application represents a control function for which the plant model is derived from the automotive domain. These plants represent DC motor speed control [74, 97], DC motor position control [74], car suspension [97] and cruise control systems [74, 84]. Given a sampling period (equal to the component task and message repetition periods), we design a controller assuming a specific delay. Now, for given maximum and minimum possible delay values and the granularity of discretization, we compute a set of delay values. For each delay value δ_k^w and the given sampling period p_w, we simulate the closed-loop system (i.e., the controller and the plant) for step response, according to Equation (4.1) and (4.2), to analyze the settling time ξ_k^w. Thus, for each application, we compute a table showing the variation in the settling time with the delay.

We assume the network to be a time-triggered Ethernet with a bandwidth of 100 Mbps [105]. The frame transmission times are computed as the amount of data they transmit over the network divided by the network bandwidth. For messages that transmit video content, we set a maximum desirable utilization of one video message in its period to 0.1 to still be schedulable while reflecting typical message sizes. This corresponds to one message transmitting maximally 10 Mbps, which is a realistic assumption, since data that exceed this value are typically split into multiple messages due to decreasing reliability of the transmission with increasing message sizes.

For the optimality experiments, when the utilization of one of the resources is greater than 100%, or the utilization of each resource is less than r_{min} presented in Table 4.1, the problem instance is simply discarded and generated again. The resulting problem instances on average contain 54, 82, 168, 421 and 6 276 activities (tasks and frames) for Sets 1-5, respectively. We set the tunable latency coefficient from Section 4.3 to $\mathbf{c} = 2$, as it allows for some flexibility of the solution while not jeopardizing its control performance [98]. The number of discrete values for settling time ξ is $N_w = 10$ for all applications. Moreover, the parameters for the optimization stage of the heuristic are set in the following manner: $N_{apps} = (2, 2, 2, 2, 1)$, $N_{sol} = (3, 3, 3, 3, 1)$, $\tau_{opt} = 10^{-2}$, $t_{max}^{opt} = (10, 10, 10, 30, 300)$ seconds given for Sets 1 to 5. These values have experimentally demonstrated reasonable results in terms of computation time and criterion quality.

Experiments were executed on a local machine equipped with an Intel Core i7 (1.7 GHz) processor and 8 GB memory. The ILP and CP models were implemented in IBM ILOG CPLEX Optimization Studio 12.8 and solved with the CPLEX and CP optimizer solvers using concert technology. The ILP, CP and heuristic approaches were implemented in the JAVA programming language.

4.6.2 Results

First, the experiments compare the computation time of the two optimal ILP and CP approaches to show that the CP approach undoubtedly outperforms the ILP approach with growing problem sizes. Second, the computation time and solution quality trade-off in terms of control performance is evaluated for the heuristic approach and the CP approach. Finally, we evaluate the maximum achievable utilization of the heuristic and the optimal approach.

A time limit of 3 000 seconds per problem instance was set to obtain the results in reasonable time, and the best solution found thus far is used if the time limit is hit. Observe that the optimal approach can stop either due to the optimal solution being found or because of the time limit. If it hits the time limit, the optimal approach has either found a feasible solution or not. The heuristic finishes either when it is not able to find a feasible solution during the feasibility stage or when it has finished the optimality stage, in which case we have a solution. The heuristic approach never hit the time limit with these experimental settings. Notice that the evaluation results are dependent on the time limit value for the exact approaches, since it affects both the computation time and quality of the obtained solution.

Similarly to the previous chapters, the distribution in graphs is shown in the form of box plots [58], where the quartile, median and three quartiles together with outliers (diamonds) are shown. Outliers are numbers that lie outside 1.5×the interquartile range (three quartiles value minus quartile value) away from the top or bottom of the box that are represented by the top and bottom whiskers, respectively. Note that outliers were also successfully solved within the time limit.

Computation Time and Criterion Quality Trade-off Evaluation

Figure 4.11 shows the computation time comparison of the optimal ILP and CP approaches and the heuristic approach with CP and ILP used during the optimization stage on the problem instances from Sets 1 to 5. The computation time of the optimal ILP approach on Sets 1 and 2 is on average 2.5 times longer than the computation time of the optimal CP approach, whereas for most of the instances already for Set 3, the optimal ILP model cannot find a feasible solution within the time limit of 3 000 seconds, failing to find optimal solutions for all problem instances. To justify that the ILP scalability issue is not a result of the solver used, we also solved the ILP models by Gurobi Optimizer 8.0. It also failed to find feasible solutions for most of the instances of Set 3 within the time limit. The reason is a large number of resource constraints (4.14), which prevents the ILP approach from scaling. In particular, there are as many constraints as there are pairs of frame jobs that can collide as mentioned in Section 4.4.3. On the other hand, the usage of a built-in function that generates one noOverlap constraint per one resource in the CP approach results in a tremendous reduction of computation time thanks to elaborate constraint propagation techniques of the CP optimizer solver. The CP approach finds a solution even for the largest Set 5 (for 84 instances out of 100 todo: update
), with up to 8 397 activities within the given time limit, although failing to prove an optimal solution for all 100 problem instances. Therefore, only the CP approach is used for further comparison with the heuristic.

Table 4.2: Number of problem instances where feasibility stage of the heuristic, optimal ILP, and optimal CP approaches failed to find a feasible solution. For optimal approaches, the number after slash is number of time outs within the time limit of 3 000 seconds

Approach	Set 1	Set 2	Set 3	Set 4	Set 5
heuristic	4	1	20	20	13
optimal CP	0/0	0/11	1/71	0/69	16/100
optimal ILP	0/2	0/15	78/100	-	-

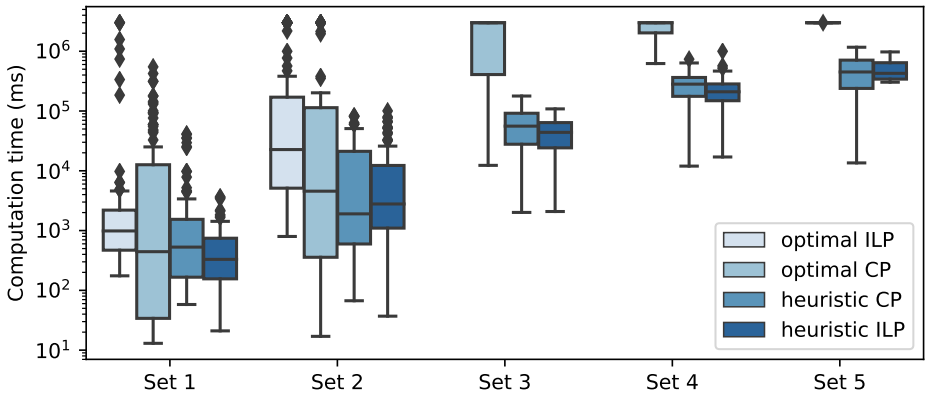


Figure 4.11: Computation time (on a logarithmic scale) of the optimal CP and ILP approaches and the heuristic CP and ILP approaches for Sets 1-5.

The number of problem instances for which the feasibility stage of the heuristic approach (that is the same for both heuristic ILP and heuristic CP), the optimal CP, and the optimal ILP failed to find a feasible solution, as well as the number of instances, for which the optimal approaches failed to prove optimality are reported in Table 4.2. These numbers do not reflect a fair evaluation of the approaches in terms of quality, since we could generate new instances with lower resource utilization, making all instances solvable by the heuristic approach. We present the fair evaluation of the maximum achievable utilization on resources by both approaches in the following subsection. However, this experiments fairly evaluate the approaches in terms of the computation time and the criterion value. The optimal CP approach fails to find any solution for the largest Set 5 within the time limit for 16 instances out of 100, which indicates that the approach starts having scalability issues for this size of problem.

As can be observed from in Figure 4.11, the computation time for problem instances that both the optimal CP and heuristic approaches were able to solve increases exponentially with increasing problem size. This tendency is less visible for the optimal CP approach due to the growing number of timeouts with increasing problem size. The heuristic approach exhibits significantly lower computational times on average: approximately 60, 33, 46, 11 and 6 times less for heuristic ILP and 10, 28, 33, 9, 6 times less for heuristic CP for Sets 1 to 5, respectively. In absolute values, heuristic CP requires on average 3, 13, 64, 279 and 501 seconds, whereas the optimal CP approach requires 35, 379, 2 070, 2 474 and 3 000 seconds for Sets 1 to 5, respectively. For the CP model, the difference decreases quickly because of the time limit, being completely determined by the time limit for Set 5.

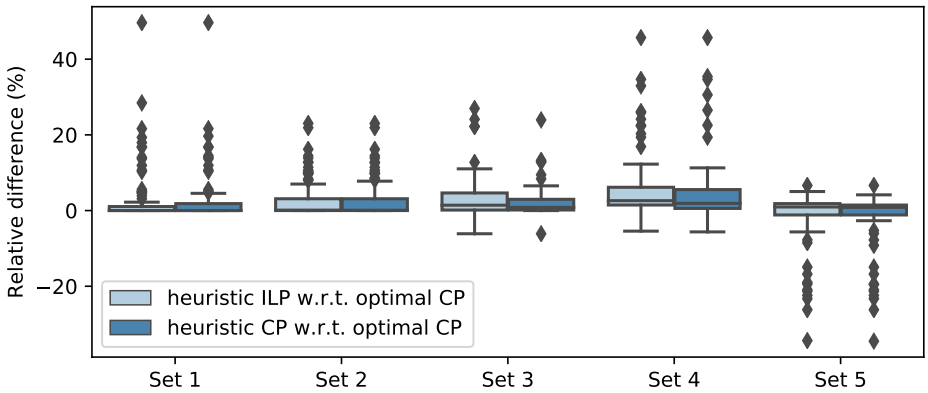


Figure 4.12: Relative difference of objective values for heuristic CP and heuristic ILP with optimal CP approaches on Sets 1-5 with the time limit of 3 000 seconds. Higher values mean that the optimal CP is better.

For Set 5, usage of the ILP in the optimization stage (heuristic ILP) does not result in any objective value improvement of the feasibility stage solution, since it always times out with no solution found. Therefore, although on smaller instances heuristic ILP results in shorter computation times, with increasing problem instance sizes it becomes too slow.

Figure 4.12 shows the relative difference in the objective values of heuristic ILP, heuristic CP and the optimal CP approaches for problem instances that all approaches were able to solve within the time limit. The optimal CP approach is used as the baseline for comparison due to its consistently better results over the results shown by the ILP approach. The median values for Sets 1 to 5 are 0, 0.15, 1.42, 2.7 and 1.2% for heuristic ILP and 0, 0.2, 0.84, 2 and 0.97% for heuristic CP. We can see that heuristic ILP degrades in solution quality with increasing problem instances sizes. Additionally, the optimal CP starts performing poorly in terms of the criterion for Set 5, as demonstrated by the prevalence of negative difference values in Figure 4.12.

As mentioned earlier, the time limit value affects both computation time and solution quality. Nevertheless, during the design exploration stage, it is typically crucial to have the solution within a reasonable time while possibly sacrificing some solution quality, and some time limit must be set. Moreover, for Set 5 with the largest problem instances, running the CP approach and the heuristic approach until the first feasible solution is found takes on average 1021 and 208 seconds, respectively, and the heuristic approach is on average 7% better, which demonstrates the clear advantage of heuristic CP over the optimal CP for larger problem instances.

Finally, *we conclude that the CP approach scales significantly better than the ILP approach both in the optimal and heuristic solutions. Additionally, the heuristic runs many times faster than the optimal CP approach while obtaining solutions with reasonable reduction of control performance. Finally, the criterion quality of the heuristic approach degrades moderately with increasing problem size, whereas the computation time remains reasonable.*

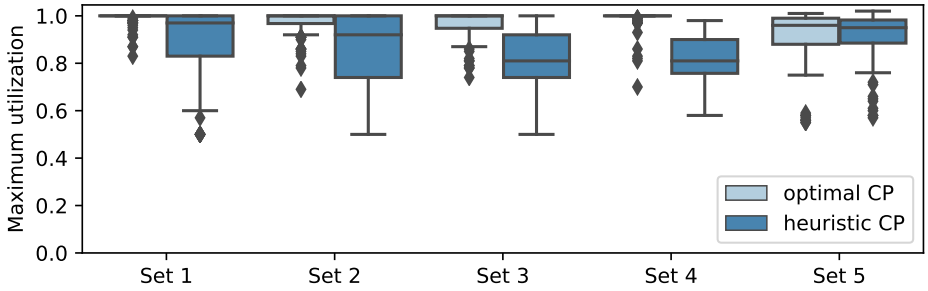


Figure 4.13: Maximally achievable utilization obtained by the optimal CP and heuristic approaches for Sets 1-4.

Comparison of Maximum Utilization

We evaluate the ability of the heuristic approach to find a feasible solution compared to the optimal approach with growing resource utilization of the problem instances as follows. The metric is the maximum utilization for which the problem instance is still schedulable. As in the previous chapter, the utilization of resource r is defined as $r_r = \sum_{a_i \in A: map_i = r} \frac{e_i}{p_i}$. We scaled the execution times of activities on the resources with maximum utilization to achieve the desired utilization on these resources. Thus, unlike the previous chapter, here we achieve the desired utilization only on a subset of resources with initially maximum utilization. There are much more resources than in the previous chapter here, and inter-switches links are clearly bottleneck resources. To both capture the reality and obtain reasonable results, it makes sense to increase utilization of only bottleneck resources. The experiments always start from a utilization of 10%, increasing by 1% in the next experiment, solving until the approach is unable to find a feasible solution. The *maximum utilization* value is the last one for which a feasible solution can be found. As argued in the previous chapter, although this strategy may not be completely fair, due to monotonic behavior of the feasibility in most cases it is indeed the highest value. Moreover, the results are easier to interpret with this strategy.

Figure 4.13 shows the distribution of the maximum utilization values for the optimal CP and heuristic approaches that were cut off after the time limit of 3000 seconds. The difference in the maximum utilization is on average 9, 13, 16, 17, and 0.05% for problem instances from Sets 1 to 5, respectively. Note that for Sets 3, 4, and 5 there are instances on which the heuristic approach achieves higher utilization, located under zero value of relative difference axis. Due to the time limit, the heuristic CP approach yields maximum achievable utilization similar to the optimal CP approach for the largest set of instances.

Thus, *the heuristic approach exhibits a reasonable degradation of the maximum achievable utilization with increasing problem sizes for smaller sets, whereas for the largest set, it achieves the same utilization as the optimal CP approach.*

4.6.3 Automotive Case Study

We proceed by demonstrating the practical applicability of our proposed heuristic and optimal approaches on an automotive case study, which is based on an auto-

Table 4.3: The use-case characteristics

Domain	$\|App\|$	$\|T\|$	$\|M\|$	m_{ECU}	P [ms]	$\sum e_i$ [ms]
body	3	15	10	7	10-20	5.9 - 12.1
chassis	4	26	11	8	5-10	2.5 - 11.4
information	3	13	10	5	20	7.9 - 13.9
electric	4	17	9	7	5-20	0.8 - 8.4
safety	6	27	16	10	5-20	1.4 - 5.9
telematics	1	6	6	6	20	10.1

motive architecture similar to the one in Figure 4.1 with 31 ECUs connected by a time-triggered Ethernet network with 4 switches. Thus, there are 68 network links that connect 31 ECUs. The functionality of the case study is implemented by 21 applications consisting of 104 tasks and 62 messages with 652 frames. The application characteristics are presented in Table 4.3. The control settling values for the applications are obtained by simulating control behavior of a given application in MATLAB, i.e., the same manner as for the synthetic problem instances previously presented in this section. Finally, the resulting utilization of the resources ranges from 0 to 70% with zero value for not used network links.

We apply our approaches to find a schedule with a low control performance degradation, such that zero-jitter, precedence, and end-to-end latency constraints are satisfied. The objective values of the optimal and heuristic approaches as a function of computation time are shown in Figure 4.14, in which both the objective value and the computation time axes are cut for better visibility. The optimal CP and ILP approaches failed to find the optimal solution within 24 hours. The optimal CP approach found its best solution with the criterion 65.73 after 500 seconds, whereas the optimal ILP approach found a solution with the same criterion after 8000 seconds. Thus, the results of the optimal approaches on the case study are consistent with the trends of the main experiments. On the contrary, the heuristic using ILP outperformed the CP heuristic on this case study unlike for the datasets in Section 4.6.2. Therefore, *the optimal CP approach yields the best results if there is some time, whereas the heuristic using ILP was a good choice on this use-case when time is bounded, such as during the design exploration phase.*

Note that this case study corresponds to an automotive system of a relatively low complexity, which is the reason why the optimal approaches exhibit good results comparing to the heuristic approach. Nonetheless, the heuristic approach is required to scale to future systems of larger complexity, as demonstrated in Section 4.6.2.

4.7 Summary

This chapter presents an approach to find a time-triggered schedule of periodic applications with hard real-time requirements while minimizing their control performance degradation. The control performance degradation of an application is defined as its settling time, and it depends on its end-to-end latency. To preserve the control performance of the applications, we also put jitter requirements on activities. Particularly, we require tasks to be executed with zero jitter, whereas we do not constraint the jitter of messages to allow more flexibility. Similarly to the previous

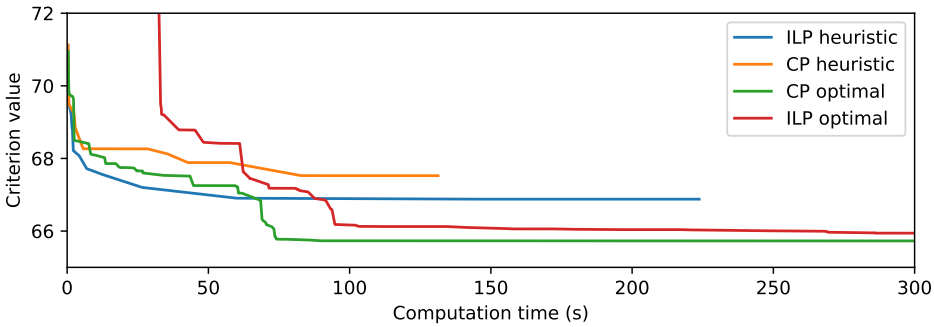


Figure 4.14: Objective function values progressing with time for the heuristic and optimal approaches applied to an automotive case study.

chapter, there are data dependencies among activities (tasks and message frames). However, here we consider a distributed architecture with electronic control units connected by a switched time-triggered Ethernet network, which results in a higher complexity of the problem due to a higher number of message frames to schedule.

We solve the considered problem optimally using ILP, CP, and SMT models, applying computation time improvements that reduce the number of variables and constraints in the model. Moreover, we propose a heuristic to solve the problem quickly and efficiently. The heuristic comprises two stages called feasibility and optimality stages. The first stage works similarly to the 3-LS heuristic from Chapter 3, which builds a schedule constructively with a given order of activities. However, the heuristic here addresses a critical drawback of the 3-LS heuristic, being the absence of flexibility. Unlike the 3-LS heuristic that makes changes directly to activity start times, this heuristic applies changes to the activity order, from which the schedule is generated. This leaves more space for the heuristic to decide which changes should be made to improve the schedule. Moreover, the feasibility stage works in three granularity levels of elements to be scheduled: applications, activities, and jobs, starting from the application level and going to the next level only if necessary. This allows us to address the least possible number of items to reduce the computation time while obtaining high-quality solutions. We also increase the time efficiency of the heuristic by using a schedulability condition of two zero-jitter tasks, proven in this chapter, which results in a significantly reduced search space. In the second stage, the heuristic approach searches for a better solution by applying a local neighborhood search, iteratively decomposing the problem to manage complexity.

We experimentally evaluate the scalability of the optimal and heuristic approaches and quantify the trade-off between computation time and solution quality of the proposed approaches. The results show that although the ILP approach in both the optimal and heuristic forms behaves slightly better on smaller instances, it fails to scale to larger instances. Moreover, the heuristic approach requires on average 6 times less computation time than its optimal counterpart, sacrificing 1% of the solution quality and 1% of the utilization for the dataset with 5 000 to 10 000 activities. Particularly, the heuristic approach solves a problem instance from the three largest sets in less than 5 minutes on average. We also demonstrate the practical relevance of our approach by applying it to an automotive case study

with 21 applications running on a platform with 31 ECUs and 68 network links. Here, we show that the heuristic approach can find a solution in less than 4 minutes while sacrificing 1.6% of the best objective value found by the optimal approaches in 24 hours.

Chapter 5
Conclusions and Future Work

This section concludes by summarizing what was learned and how it can be applied, before discussing how we fulfilled the stated goals. Finally, it presents possible directions for future research.

5.1 Conclusions

The number of applications sharing platform components in embedded real-time systems grows rapidly due to consumer demand for integrating multiple functionalities into one device. Moreover, the number of platform components are minimized to reduce the cost of the final system. This results in many applications sharing platform resources, which can cause significant collisions, and timing requirements may not be satisfied. Resources are sometimes scheduled in a time-triggered manner to prevent this situation. Finding time-triggered schedules is a challenging problem lying in a class of NP-hard problems, involving non-trivial timing requirements and optimization criteria. To solve this problem, we propose a three-step approach corresponding to three problems to capture different aspects of the underlying integration problem. Optimal approaches to solve these problems have limited scalability, failing to find a solution in reasonable time for larger and more difficult problem instances. To cope with this issue, we propose heuristic approaches to solve instances of modern and future sizes and complexities efficiently and in a reasonable time.

However, on a higher level of abstraction, the main contribution of this thesis is a methodology to work with optimization problems. The three steps of our approach show the application of the methodology on three optimization problems: first, the problems are formulated; then, the optimal models are formulated and improved using problem-specific properties; next, the heuristic to solve the problem faster but sub-optimally is proposed that reuses the existing optimal formulations; finally, the heuristic is compared on smaller instances with optimal approaches, and its behavior is evaluated on larger problem instances.

We continue this section by discussing the contributions of this thesis in details. First, we discuss the design and implementation of scalable and efficient heuristic algorithms, followed by characteristics and comparisons of the optimal approaches used in this thesis. Then, the experimental evaluation of the proposed optimal and heuristic approaches is presented, before we discuss demonstration of practical applicability of the proposed approaches on realistic case studies.

5.1.1 Scalable and Efficient Heuristic Algorithms

We propose three efficient heuristics to solve industrial-sized instances of the considered problems in a reasonable time in Sections 2.7, 3.4, and 4.5. The first heuristic for the problem of TDM arbiter scheduling in Section 2.7 is a generative one. It constructs the whole solution at once, allowing a time slot to be allocated by multiple clients. As this can result in collisions, it assigns a cost for allocating each

time slot for each client and runs the schedule generation, respecting the costs in the next iteration. The cost assigning procedure involves memory state, which reflects how many times a given slot has already been allocated to the considered client and other clients in the previous iterations. The cost is set low if this slot was allocated only to this client before, higher if it was empty and even higher if it was allocated to other clients in previous iterations. This strategy is aimed to change the current client schedule only if it collides with some other client, avoiding unnecessary changes. However, as shown in Section 2.8, this heuristic has room for improvement, especially for problem instances where it is challenging to find any feasible solution.

Therefore, in the second step of our approach, we propose heuristic called 3-Level Scheduling (3-LS) heuristic for safety-critical systems scheduling problem. It works in three levels, the first being constructive creation of a schedule by setting activity start times. The heuristic backtracks if some activity cannot be scheduled, first trying to remove one of the prescheduled activities. However, if the activity to be removed already had problems being scheduled, the 3-LS heuristic schedules the two activities, the problematic one and the one to be unscheduled, at the same time, which is the second level. Finally, if this does not result in a conflictless solution, in the third level it removes almost all activities and schedules the two activities from scratch. The three levels provide a good trade-off between solution quality and computation time, since the effort for scheduling problematic activities is reasonable to not unnecessary prolong the computation time and to get the solution of a good quality.

However, the main drawback of 3-LS heuristic is lack of flexibility, caused by looking at the solution space of fixed schedules. To address it, in the third step of our approach we propose a heuristic that works in the solution space of activity orders, from which the schedules are constructed. Since working with orders is more flexible than working with activity start times, a smart interchange of two activities in a given order has higher chances to result in a feasible solution than a smart change in activity start times. The heuristic works the following way. Firstly, it tries to construct the schedule from the given order. If it fails to find a feasible solution, we promote the activity or job that could not be scheduled, using a novel concept of a reason graph that helps us to define which changes to do in the priority queue. In this heuristic, we make use of the theorem, proven in Section 4.3, which reduces computation time of the heuristic. Moreover, the heuristic has three levels of scheduling granularity: it schedules by applications, by activities or by jobs, going to the level with more elements only if necessary. This provides a reasonable trade-off between computation time and solution quality.

Finally, Table 5.1 presents a comparison of the three proposed heuristics: the heuristic to solve TDM problem from Chapter 2, 3-LS heuristic from Chapter 3, and the heuristic solving the coscheduling problem with control regarded in Chapter 4. Whereas the heuristic to solve TDM problem uses an ILP solver for every problem instance, 3-LS heuristic can work without using one if there are only activities with relaxed jitter requirements. Meanwhile, the feasibility stage of the control heuristic does not use any exact solver, applying it for the optimization of the obtained solution in the second stage only. As there is a risk of rapidly decreasing scalability with increasing problem sizes with exact solvers, getting a feasible solution without

Table 5.1: Comparison of the heuristic approaches from Chapters 2, 3, and 4.

	Deterministic behavior	Coarsest Granularity	Dependence on exact solvers	Element of a search space	Main concepts to increase efficiency
Stage 1	no	activities	fully dependent	fixed schedule	1. memory state, 2. lazy constraints
Stage 2	yes	activities	sometimes dependent	fixed schedule	1. interval representation of a search space, 2. 3 scheduling levels
Flexi	yes	occurrences	partially independent	order	1. Theorem 1 of schedulability, 2. 3 granularity levels

using exact solvers is an advantage.

5.1.2 Characteristics and Comparison of Exact Approaches

In this section, we look at three different formalisms to optimally solve our problem: Integer Linear Programming (ILP) used in Chapters 2, 3, and 4; Satisfiability Modulo Theory (SMT) used in Chapters 3 and 4; and Constraint Programming (CP) used in Chapter 4. We present a comparison and our experience with both the formalisms and the tools for solving the corresponding models that are IBM ILOG CPLEX and Gurobi solvers for the ILP, Microsoft Z3 solver for SMT and IBM CPLEX CP optimizer solver for CP. Note that some of these conclusions are problem-dependent and do not need to apply to problems in different domains, since adding or removing specific constraints or reformulating criterion can change the computation time of the solvers significantly.

Computational Efficiency

An improvement of the computational efficiency of the exact approaches can belong to one of the following 4 categories: 1) *reducing the number of constraints and decision variables* without changing the search space of feasible solutions (used in Chapters 2, 3, and 4); 2) *reducing the search space* by introduction of additional constraints (called cuts in ILP), reducing variable domains, or breaking symmetries in the solution (used in Chapters 2, 3, and 4); 3) *Reformulating the constraints* so they can be used by the solver more efficiently (used in Chapters 2 and 4); 4) adjusting solver-specific features (used in Chapter 4).

Improvements from the first two categories apply to all formalisms and in terms of computation time they are similar across all formalisms and solvers. On the other hands, techniques in Categories 3 and 4 are generally solver-specific regarding both implementation and resulting efficiency. For instance, reformulation of resource constraints using the modulo operation of the Bezout identity in Section 4.3 resulted in significant time reduction of the ILP model, while the SMT and CP problems have shown longer computation time with the Bezout inequality applied. The intuition behind this is that instead of accelerating the solving process, some techniques can worsen pruning and propagation of the SMT and CP approaches. As far as the space reduction is concerned, it should be done consciously, since it can cause enormous prolongation of the computation time. An example of this situation was adding an upper bound on the criterion as a constraint to the CPLEX solver for the

TDM configuration problem in Chapter 2. Adding an upper bound resulted in some instances running ten to hundred times slower, most probably due to the absence of the first feasible solution that enables bounding in the search tree. Instead, setting this value as a solver parameter resulted in computation time reduction since it discarded only a part of feasible solutions of the worse quality. This enabled smaller search tree by smart bounding in nodes, i.e., not continuing further in the nodes that cannot result in the optimal solution.

The main reason for the high efficiency of the CP approach in Chapter 4 is the usage of solver-specific variable and constraint representation that resulted in thousands times faster solving process. Another example of an improvement belonging to the fourth category is the usage of lazy constraints for ILP when only crucial constraints are generated in the beginning, and others are added if needed during the search process. This significantly improved the efficiency of the ILP approach in Chapters 2 and 4. Finally, we set a single thread solving process for the CP approach in Chapter 4, which belongs to the fourth category. Setting a single thread in CP optimizer often shows good results on scheduling problems.

Finally, we observed that the SMT solver works efficiently on a fully discrete domains on our problem in Chapter 4. However, merely introducing rational coefficients without changing neither decision variables nor constraints resulted in an efficiency reduction of roughly 50%. Generally, the SMT approach has shown the most unstable behavior regarding computation time, such as a tendency to run infinitely long for smaller problem instances with specific constraints. However, since we only have experience with a single solver for this formalism, we do not know if the problem is inherent to the formalism or to the solver.

User Experience

Since CPLEX, CP optimizer, and Gurobi are commercial products unlike Microsoft Z3, the first three tools provide much better user support. There are documentations, examples and special forums for users of CPLEX, CP Optimizer, and Gurobi, where questions are answered promptly, and tool developers are interested in resolving problems and bugs. These three tools provide clear documentation for each supported programming language. Development with Microsoft Z3 can be more challenging, since there are tutorials for its usage only as a native script and in Python. Although there are examples of codes for Z3 in each programming language it supports, not every feature is covered in the examples, and sometimes it takes time to understand how certain things can be implemented. Also, it has less features provided to the user. For instance, although defining a piece-wise linear function is possible in SMT using if-then-else statements, the computation efficiency is low due to the continuous nature of this function. Thus, while the conceptual ideas behind the Z3 solver for SMT are easily accessible, programming not in a native script may be an adventure full of hidden surprises.

User experience can be measured by simplicity to model, program and debug. Unlike ILP that works only with linear expressions, CP and SMT handle non-linear constraints, such as OR, modulo, multiplication of variables, etc. Thus, whereas for ILP we need to transform constraints and criterion to linear form, for both CP and SMT it is possible to have expressions in a natural form. This saves user time

Table 5.2: Supported API of different optimization tools.

CPLEX	C, C++, C#, Java, Visual Basic, Python, Matlab, FORTRAN
CP optimizer	C++, C#, Java, Python
Gurobi	C, C++, C#, Java, Visual Basic, Python, Matlab, R
Z3	C, C++, .Net, Java, Python, and OCaml

and often simplifies later modification and debugging. Although ILP tools have build-in transformations of some non-linear expressions, such as piece-wise linear functions or OR expressions, often they are not as efficient as transforming it by hand as shown in Chapter 4 with criterion formulated as piece-wise linear function.

Finally, Table 5.2 contains the list of supported API of the considered tools by the time of writing this thesis.

Portability

Since CPLEX and CP optimizer are both developed by IBM, they use very similar syntax. However, to implement the solution efficiently, it is inevitable to get acquainted with available language features, such as interval variables or global constraints, mentioned in Section 4.4. Furthermore, since both CPLEX and Gurobi solve the same type of problem, they support export and import from and to .lp files that contain a problem in an algebraic form. Therefore, it is possible to solve by models generated by CPLEX using Gurobi or vice-versa quickly. However, to implement some computation time improvements, such as lazy constraints or built-in piecewise linear function, as in Chapter 4, it is still required to implement the model in one of the supported languages. For the Z3 solver, the problem formulation cannot be straightforwardly transformed from one of the other tools. However, with example codes and basic knowledge of how Z3 works in the used programming language (e.g., constants cannot be directly used by the solver, they need to be wrapped into either integer (mkInt()) or real (mkReal()) objects in Java), the models can be programmed quite intuitively, unless particular features are necessary that are not covered in the examples.

Appropriate Usage

Finally, this subsection presents what we have learned during these four years of work with the optimal formalisms and their solvers. First of all, every formalism is the best for the things for which it was designed. Therefore, *for problems in a linear form that do not require applying additional techniques for linearization, there is a good chance that ILP will outperform both SMT and CP approaches.* However, since NP-hard problems always contain non-linearities, this statement is not supported by any experiments presented in this thesis. Moreover, ILP also shows lower computation times on instances of smaller sizes and lower complexity, making it suitable for use in heuristic algorithms, where a large number of small problems are to be solved. Generally, with growing size and complexity, CP and SMT approaches typically start outperforming ILP on real-time systems scheduling problems.

We have also noticed that *the CP approach often finds the first feasible solution much faster than ILP, whereas ILP is typically better at proving optimality and infeasibility* due to its elaborated propagation techniques, especially for scheduling problems. Thus, if finding a feasible solution is challenging and a system designer can afford sub-optimality of the solution, it is better to choose the CP formalism, whereas if it is critical to prove optimality of the solution or that the problem is infeasible with given input parameters, it is more appropriate to use ILP.

Regarding making changes in the problem, ILP typically behaves more predictably than CP and SMT regarding the number of introduced constraints and variables. In other words, we observed that adding or reformulating constraints and criterion could cause significantly larger changes in running time of CP and SMT than of ILP. Thus, *if the model is under changes, it is better to choose ILP and only when the final version of the model is formulated, to implement it in CP or SMT efficiently. On the other hand, CP and SMT approaches provide framework for formulating non-linear expressions efficiently, which both simplifies implementation and reuse of the existing models.* However, the effort in implementing the same problem using a different formalism is limited, as mentioned in the previous subsection.

5.1.3 Quantitative Comparison of the Proposed Approaches

We compared optimal and heuristic algorithms on realistic problem instances for the three steps of our approach. The criteria are the computation time of the approach, maximum achievable utilization on the resources, and the criterion value itself, being slack utilization for the first step and control performance for the third step. As expected, the experimental results have shown that optimal approaches are far less scalable than the heuristic approaches. Average improvements in the maximal size of the instance that heuristic approaches can solve in slightly less than 1 hour over that of optimal approaches are approximately 10 times. Regarding criterion value, the average difference is 0.9% for all three steps, while the heuristics lost on average 11% of maximal utilization, considering the instances that the optimal approaches solved within the time limit. Furthermore, the average computation time of the heuristic solutions on the largest datasets are 2, 3, and 5 minutes for the heuristics from Steps 1, 2, 3 of our approach, respectively. These values are reasonable for one run and acceptable for design exploration phase of the modern and future systems.

5.1.4 Practical Applicability

We demonstrated practical applicability of the proposed approaches on the case studies typical for the corresponding domains in Sections 2.8.3, 3.5.3, and 4.6.3. For the first step of the approach, we focused on a case study of an HD video and graphics processing system, where we first derive the client requirements and present a detailed description of the components. Then, we apply our configuration methodologies to find the optimal schedule, each of which runs in less than a second. Next, for the second step of our approach dealing with safety-critical systems, we consider a case study of Engine Management System (EMS) in Section 3.5.3. It

controls the time and amount of air and fuel injected by the engine by working with the values read by numerous sensors in the car. Engine Management System (EMS) is one of the most sophisticated engine control units with more than 10000 tasks and messages. While the proposed optimal approaches fail to find any schedule in 24 hours, the 3-LS heuristic solves the case study in less than an hour, showing its applicability to industrial-sized problem instances. Finally, in Section 4.6.3, we verify the proposed optimal and heuristic solutions for the third step of our approach, where we consider a distributed automotive architecture with 31 ECUs connected by 68 time-triggered Ethernet network links. All the proposed approaches were able to find a feasible solution, although both the ILP and CP models failed to prove optimality of the solution found in a week. In contrast, the heuristic approach has found a solution in less than 4 minutes with the criterion value within 1.6% relative difference of the solution found by the optimal approaches.

5.2 Fulfillment of the Goals

This section addresses the fulfillment of the stated goals:

1. *Study the existing literature for real-time embedded systems scheduling and determine its weak points regarding problem statement and solution approaches.*

   For the first step of our approach, the study of existing literature presented in Section 2.1 revealed that no existing solutions provide latency-rate guarantees and minimize the total allocated rate to increase the performance of non-real-time applications. Moreover, the state-of-the-art approach for TDM scheduling with latency-rate guarantees provided is highly inefficient regarding the total allocated rate, as we showed in Section 2.8. For the second step of the approach, state-of-the-art solutions for hard real-time systems scheduling either eliminate any activity (task or message) jitter or do not put any constraints on it, as discussed in Section 3.1. However, constraining the jitter instead of eliminating it can result in significant maximum utilization gain, as demonstrated in Section 3.5, reducing the cost of the system. As far as the third step of our approach is concerned, as discussed in Section 4.1, there are no scheduling approaches that look at the control behavior of the system globally, considering different importance of applications control behavior and without activity periods being changeable. These aspects are especially useful in the automotive domain since there are many control applications there. Finally, only a few existing works on real-time embedded systems scheduling provide scalable approaches, as discussed in Sections 2.1 and 4.1. However, none of them addresses the points discussed above.

2. *Devise mathematical models, reflecting the most important constraints and criteria of the embedded real-time systems scheduling problem both in consumer electronics and in safety-critical domains.*

   The mathematical models are presented in Sections 2.3, 3.2, and 4.3. For the first step of our approach applicable to the consumer-electronics domain domain, we formulate the model regarding the latency-rate characteristics

and minimizing utilized rate. The main benefit of the considered latency-rate abstraction is that it enables performance analysis of systems with shared resources in a unified manner, irrespective of the chosen arbiter and configuration. For the second step of our approach, the devised mathematical model considers periodicity, precedence relations, end-to-end latency, and jitter constraints. In the third step of our approach, the model devised in the second step is changed in these three ways: 1) we fix jitter requirements on the computational resources and relax them on the network, 2) we generalize end-to-end latency requirements, and 3) we introduce an optimization criterion, being the aggregated control performance of the applications in the system.

3. *Propose exact and heuristic algorithms to solve industrial-sized problems of embedded real-time systems scheduling that use problem-specific knowledge.*

   Section 2.4 presents an ILP model for the first step of our approach that uses problem-specific knowledge to reduce the solution space and improve computation time. Then, in Section 2.5, we introduce a solution that wraps the ILP model into a branch-and-price framework to enhance its scalability. Moreover, a heuristic is proposed in Section 2.7 to solve the same problem faster, but with a possible loss in criterion value. Later, Section 3.3 introduces an ILP and SMT models for the second step of our approach that shows different behavior on problem instances of different complexity. However, none of the proposed optimal approaches scale to the large sizes of modern and future systems. Therefore, Section 3.4 proposes a three-step heuristic scheduling algorithm, called 3-LS heuristic, where the schedule is found constructively. Finally, in Section 4.4, ILP and CP models are presented to deal with the third step of the approach. Moreover, in Section 4.5 we present a heuristic approach that constructs the solution based on a priority queue, changing it if necessary.

4. *Verify the proposed algorithms on benchmark instances and compare the quality of the obtained solutions by heuristic with the solutions, obtained by the exact approaches. Show applicability of the proposed approaches on realistic use-cases.*

   For the first step of our approach, we present experiments in Section 2.8, confirming that the branch-and-price approach is more scalable than the pure ILP approach. Namely, it solves instances with 64 clients, while the ILP approach only scales to 16 clients. Moreover, the proposed heuristic provides a reasonable trade-off between computation time and solution quality, solving the problem near-optimally in 86% of the use-cases in half of the time required by the branch-and-price approach. Also, we present a case study of an HD video and graphics processing system that demonstrates the practical applicability of the approach.

   For the second step of our approach, Section 3.5 presents experiments showing the efficiency of both scheduling with zero jitter and constraining it reasonably. It shows that up to 28% higher resource utilization can be achieved by at cost of 10 times longer computation time. Furthermore, it shows that the SMT approach outperforms the ILP approach on larger problem instances, while ILP

shows better results on smaller instances. However, neither approach scales to industrial-sized problems, whereas the proposed 3-LS heuristic provides a reasonable trade-off of solution quality and computation time compared to the optimal approaches. It shows an average degradation of maximum resource utilization of 7%, while running tens of times faster on the sets with the smallest problem instances. Besides, the proposed solutions for the second step of the approach are demonstrated on a case study of an EMS with more than 10000 activities, a heuristic solves that in less than an hour.

Finally, for the third step of the approach, in Section 4.6, we show that the CP model outperforms the ILP and SMT models, scaling to significantly larger instances. Moreover, the heuristic approach provides a solution on average 6 times faster than CP approach, while sacrificing 1% of the solution quality and 1% utilization for the most complex problem instances. Also, we demonstrate the applicability of the approaches on an automotive case-study with 21 applications running on the platform with 31 ECUs and 68 network links, for which the heuristic finds a solution within less than 4 minutes with objective value degradation of 1.6% from the best solution, found by the optimal approach in 24 hours.

5.3 Future Work

This section presents two directions to extend this thesis in the future to increase the efficiency and applicability of the solution.

5.3.1 Mapping and Routing Problems

The number of components in complex real-time embedded systems, including scalable interconnects, on-chip, and off-chip switched networks is increasing rapidly. The mapping and routing problems that concern activity assignment to resources and finding a route that a message should take in the switched network, respectively, go along with the scheduling problem. Although sometimes mapping and routing are already given and we need to deal with scheduling only, in the other cases there is a freedom to set it. Mapping and routing done efficiently, for instance in switched networks or networks-on-chip, can significantly increase the utilization of the system and potentially reduce cost. However, when we introduce the additional complexity to the scheduling problem which is NP-hard on its own, the solution process becomes complicated. Thus, the fourth step of our approach would be to efficiently solve the optimization problem, merging these three problems: scheduling, routing, and mapping, together. Solving these three problems at once would result in a more efficient solution for the price of the enormous complexity of the resulting problem.

5.3.2 Incorporation of Platform-Specific Constraints

When applying the solution proposed in this thesis, some platform-specific constraints can arise. The first source of such constraints is the network. In this thesis,

we assume a general time-triggered network that can have additional constraints for specific network types. We can observe an example of such constraint for Time-Sensitive Networks [88] that, for instance, require a frame of a flow to be fully transmitted on one link before transmission is initiated on the subsequent link of the route or buffer limit constraint in switches. The second source of platform-specific constraints is a limited memory of platform resources. While scheduling in a time-triggered manner, the schedule length could become very large. We can prevent it by limiting the schedule length, which also introduces a lot of complexity to the solution. Finally, in this work we consider the activities of one application to have the same period. However, sometimes tasks with different periods are data dependent. This requires definition of an appropriate framework since end-to-end latency can have multiple interpretations when tasks of different periods are data dependent [54].

Bibliography

[1] TriCore pipeline behaviour and instruction execution timing, June 2019.

[2] T. F. Abdelzaher and K. G. Shin. Combined task and message scheduling in distributed real-time systems. *IEEE Transactions on parallel and distributed systems*, 10(11):1179–1191, 1999.

[3] B. Akesson and K. Goossens. Architectures and modeling of predictable memory controllers for improved system integration. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[4] B. Akesson and K. Goossens. *Memory Controllers for Real-Time Embedded Systems*. Embedded Systems Series. Springer, 2011.

[5] B. Akesson, A. Minaeva, P. Sucha, A. Nelson, and Z. Hanzálek. An efficient configuration methodology for time-division multiplexed single resources. In *2015 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 161–171. IEEE, 2015.

[6] B. Akesson, L. Steffens, E. Strooisma, and K. Goossens. Real-time scheduling using credit-controlled static-priority arbitration. In *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA'08. 14th IEEE International Conference on*, pages 3–14, 2008.

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

[8] A. Albert et al. Comparison of event-triggered and time-triggered concepts with regard to distributed control systems. *Embedded world*, 2004:235–252, 2004.

[9] A. Aminifar, S. Samii, P. Eles, Z. Peng, and A. Cervin. Designing high-quality embedded control systems with guaranteed stability. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 283–292. IEEE, 2012.

[10] K. J. Åström and B. Wittenmark. *Computer-controlled systems: theory and design*. Courier Corporation, 2013.

[11] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber. Minimizing service and operation costs of periodic scheduling. *Mathematics of Operations Research*, 27(3):518–544, 2002.

[12] J. R. Barker and G. B. McMahon. Scheduling the general job-shop. *Management Science*, 31(5):594–598, 1985.

[13] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari. Scheduling periodic task systems to minimize output jitter. *Real-Time Computing Systems and Applications, 1999. RTCSA'99. Sixth International Conference on*, pages 62–69, 1999.

[14] S. Baruah and G. Fohler. Certification-cognizant time-triggered scheduling of mixed-criticality systems. In *2011 IEEE 32nd Real-Time Systems Symposium (RTSS 2011)*, pages 3–12. IEEE, 2011.

[15] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[16] M. Becker, D. Dasari, B. Nikolic, B. Akesson, V. Nelis, and T. Nolte. Contention-free execution of automotive applications on a clustered many-core platform. *Euromicro Conference on Real-Time Systems*, 2016.

[17] S. Beji, A. Gherbi, J. Mullins, and P.-E. Hladik. Model-driven approach to the optimal configuration of time-triggered flows in a TTEthernet network. In *International Conference on System Analysis and Modeling*, pages 164–179. Springer, 2016.

[18] L. L. Bello. Novel trends in automotive networks: A perspective on Ethernet and the IEEE audio video bridging. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, pages 1–8. IEEE, 2014.

[19] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee. Synthesis of embedded software from synchronous dataflow specifications. *Journal of VLSI signal processing systems for signal, image and video technology*, 21(2):151–166, 1999.

[20] J. Błażewicz, E. Pesch, and M. Sterna. The disjunctive graph machine representation of the job shop scheduling problem. *European Journal of Operational Research*, 127(2):317–331, 2000.

[21] A. Bonatto, A. Soares, and A. Susin. Multichannel SDRAM controller design for H.264/AVC video decoder. *In Programmable Logic (SPL), 2011 VII Southern Conference on*, pages 137–142, 2011.

[22] R. Bosch. CAN specification version 2.0, 1991.

[23] G. C. Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[24] Y. Cai and M. Kong. Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems. *Algorithmica*, 1996.

[25] A. Cervin. Stability and worst-case performance analysis of sampled-data control systems with input and output jitter. In *American Control Conference (ACC), 2012*, pages 3760–3765. IEEE, 2012.

[26] J. Chen, C. Du, F. Xie, and Z. Yang. Schedulability analysis of non-preemptive strictly periodic tasks in multi-core real-time systems. *Real-Time Systems*, 52(3):239–271, 2016.

[27] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, pages 181–194, 1990.

[28] C. Connelly. World's first 'smartphone' celebrates 20 years. `https://www.bbc.com/news/technology-28802053`, 2014. [Online; accessed 24-October-2018].

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

[30] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

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

[32] M. Di Natale and J. A. Stankovic. Applicability of simulated annealing methods to real-time scheduling and jitter control. In *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*, pages 190–199. IEEE, 1995.

[33] M. Di Natale and J. A. Stankovic. Scheduling distributed real-time tasks with minimum jitter. *IEEE Transactions on Computers*, 49(4):303–316, 2000.

[34] J. Dvořák and Z. Hanzálek. Using two independent channels with gateway for FlexRay static segment scheduling. *IEEE Transactions on Industrial Informatics*, 12(5):1887–1895, 2016.

[35] D. Feillet. A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR*, 8(4):407–424, 2010.

[36] M. Forget, E. Grolleau, C. Pagetti, and P. Richard. Dynamic priority scheduling of periodic tasks with extended precedences. *ETFA, 2011 IEEE 16th Conference on*, pages 1–8, 2011.

[37] S. Foroutan, B. Akesson, K. Goossens, and F. Petrot. A general framework for average-case performance analysis of shared resources. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 78–85. IEEE, 2013.

[38] S. Fürst, J. Mössinger, S. Bunzel, T. Weber, F. Kirschke-Biller, P. Heitkämper, G. Kinkelin, K. Nishikawa, and K. Lange. AUTOSAR–a worldwide standard is on the road. In *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, volume 62, 2009.

[39] G. Giannopoulou, N. Stoimenov, P. Huang, L. Thiele, and B. D. de Dinechin. Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources. *Real-Time Systems*, 52(4):1–51, 2015.

[40] M. D. Gomony, B. Akesson, and K. Goossens. A real-time multichannel memory controller and optimal mapping of memory clients to memory channels. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(2):25, 2015.

[41] M. D. Gomony, J. Garside, B. Akesson, N. Audsley, and K. Goossens. A globally arbitrated memory tree for mixed-time-criticality systems. *IEEE Transactions on Computers*, 66(2):212–225, 2017.

[42] S. Goossens, B. Akesson, and K. Goossens. Conservative open-page policy for mixed time-criticality memory controllers. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 525–530. EDA Consortium, 2013.

[43] S. Goossens, K. Chandrasekar, B. Akesson, and K. Goossens. Power/performance trade-offs in real-time SDRAM command scheduling. *IEEE Transactions on Computers*, 65(6):1882–1895, 2016.

[44] S. Goossens, J. Kuijsten, B. Akesson, and K. Goossens. A reconfigurable real-time SDRAM controller for mixed time-criticality systems. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2013 International Conference on*. IEEE, 2013.

[45] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty. Time-triggered implementations of mixed-criticality automotive software. *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1227–1232, 2012.

[46] A. Hansson, K. Goossens, and A. Rădulescu. A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic. *VLSI design*, 2007.

[47] Z. Hanzalek, P. Burget, and P. Sucha. Profinet IO IRT message scheduling with temporal constraints. *Industrial Informatics, IEEE Transactions on*, 6(3):369–380, 2010.

[48] M. Hassan, H. Patel, and R. Pellizzoni. A framework for scheduling DRAM memory accesses for multi-core mixed-time critical systems. *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*, pages 307–316, 2015.

[49] R. Holte, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: A realtime scheduling problem. *Proceedings of the 22 Hawaii International Conference on System Sciences*, pages 693–702, 1989.

[50] M. Hu, J. Luo, Y. Wang, and B. Veeravalli. Scheduling periodic task graphs for safety-critical time-triggered avionic systems. *Aerospace and Electronic Systems, IEEE Transactions on*, 51(3):2294–2304, 2015.

[51] D. Huisman, R. Jans, M. Peeters, and A. Wagelmans. Combining generation and lagrangian relaxation. *ERIM Report Series Research in Management*, 2003.

[52] IBM ILOG Cplex. 12.2 user's manual. *Book 12.2 User's Manual, Series 12.2 User's Manual*, 2010.

[53] ILOG AMPL CPLEX. System Version 7.0 User's Guide. *ILOG CPLEX Division, Incline Village, NV*, 2000.

[54] P. Jayachandran and T. Abdelzaher. End-to-end delay analysis of distributed systems with cycles in the task graph. In *ECRTS'09. 21st Euromicro Conference on*, pages 13–22. IEEE, 2009.

[55] JEDEC Solid State Technology Association. *DDR3 SDRAM Specification*, JESD79-3F edition, 2012.

[56] K. Jeffay, D. Stanat, and C. Martel. On non-preemptive scheduling of period and sporadic tasks. *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 129–139, Dec 1991.

[57] W. Jiang, P. Pop, and K. Jiang. Design optimization for security-and safety-critical distributed real-time applications. *Microprocessors and Microsystems*, 52:401–415, 2017.

[58] P. Kampstra et al. Beanplot: A boxplot alternative for visual comparison of distributions. *Journal of statistical software*, 28(1):1–9, 2008.

[59] D.-I. Kang, R. Gerber, and M. Saksena. Parametric design synthesis of distributed embedded systems. *Computers, IEEE Transactions on*, 49(11):1155–1169, 2000.

[60] O. Kermia. An efficient approach for the multiprocessor non-preemptive strictly periodic task scheduling problem. *Journal of Systems Architecture*, 79:31–44, 2017.

[61] P. Kollig, C. Osborne, and T. Henriksson. Heterogeneous multi-core platform for consumer multimedia applications. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1254–1259, 2009.

[62] O. Koné, C. Artigues, P. Lopez, and M. Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13, 2011.

[63] H. Kopetz. Event-triggered versus time-triggered real-time systems. In *Operating Systems of the 90s and Beyond*, pages 86–101. Springer, 1991.

[64] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[65] S. Kramer, D. Ziegenbein, and A. Hamann. Real world automotive benchmark for free. *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems*, 2015.

[66] H.-T. Lim, L. Völker, and D. Herrscher. Challenges in a future IP/Ethernet-based in-car network for real-time applications. In *Proceedings of the 48th Design Automation Conference*, pages 7–12. ACM, 2011.

[67] J. Lin, A. Gerstlauer, and B. Evans. Communication-aware heterogeneous multiprocessor mapping for real-time streaming systems. *Journal of Signal Processing Systems*, 69(3):279–291, 2012.

[68] K.-J. Lin and A. Herkert. Jitter control in time-triggered systems. *System Sciences, 1996., Proceedings of the Twenty-Ninth Hawaii International Conference on*, 1:451–459, 1996.

[69] W. Liu, M. Yuan, X. He, Z. Gu, and X. Liu. Efficient SAT-based mapping and scheduling of homogeneous synchronous dataflow graphs for throughput optimization. *Real-Time Systems Symposium, 2008*, pages 492–504, 2008.

[70] Z. Lu and A. Jantsch. Slot allocation using logical networks for TDM virtual-circuit configuration for network-on-chip. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 18–25, 2007.

[71] M. Lukasiewycz and S. Chakraborty. Concurrent architecture and schedule optimization of time-triggered automotive systems. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 383–392. ACM, 2012.

[72] M. Lukasiewycz, R. Schneider, D. Goswami, and S. Chakraborty. Modular scheduling of distributed heterogeneous time-triggered automotive systems. *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 665–670, January 2012.

[73] M. Lukasiewycz, R. Schneider, D. Goswami, and S. Chakraborty. Modular scheduling of distributed heterogeneous time-triggered automotive systems. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 665–670. IEEE, 2012.

[74] W. C. Messner, D. M. Tilbury, and A. P. R. Hill. Control tutorials for MATLAB® and Simulink®, 1999.

[75] A. Minaeva, B. Akesson, Z. Hanzalek, and D. Dasari. Time-triggered co-scheduling of computation and communication with jitter requirements. *IEEE Transactions on Computers*, 2017.

[76] A. Minaeva, P. Sucha, and B. Akesson. Source codes for the problem of resource scheduling with latency-rate abstraction. https://github.com/CTU-IIG/BandP_TDM, 2015.

[77] A. Minaeva, P. Šůcha, B. Akesson, and Z. Hanzálek. Scalable and efficient configuration of time-division multiplexed resources. *Journal of Systems and Software*, 113:44 – 58, 2016.

[78] Y. Monnier, J.-P. Beauvais, and A.-M. Déplanche. A genetic algorithm for scheduling tasks in a real-time distributed system. *Euromicro Conference, 1998. Proceedings. 24th*, pages 708–714, 1998.

[79] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion. Multisource software on multicore automotive ECUs—combining runnable sequencing with task scheduling. *Industrial Electronics, IEEE Transactions on*, 59(10):3934–3942, 2012.

[80] O. Moreira, F. Valente, and M. Bekooij. Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, pages 57–66. ACM, 2007.

[81] A. Nelson, K. Goossens, and B. Akesson. Dataflow formalisation of real-time streaming applications on a composable and predictable multi-processor SOC. *Journal of Systems Architecture*, 61(9):435–448, 2015.

[82] A. Novak, Z. Hanzalek, and P. Sucha. Scheduling of safety-critical time-constrained traffic with f-shaped messages. In *Factory Communication Systems (WFCS), 2017 IEEE 13th International Workshop on*, pages 1–9. IEEE, 2017.

[83] R. S. Oliver, S. S. Craciunas, and W. Steiner. IEEE 802.1 Qbv gate control list synthesis using array theory encoding. 2018.

[84] K. Osman, M. F. Rahmat, and M. A. Ahmad. Modelling and controller design for a cruise control system. In *Signal Processing & Its Applications, 2009. CSPA 2009. 5th International Colloquium on*, pages 254–258. IEEE, 2009.

[85] M. Panić, S. Kehr, E. Quiñones, B. Boddecker, J. Abella, and F. J. Cazorla. Runpar: An allocation algorithm for automotive applications exploiting runnable parallelism in multicores. *International Conference on Hardware/Software Codesign and System Synthesis*, page 29, 2014.

[86] D.-T. Peng and K. G. Shin. Static allocation of periodic tasks with precedence constraints in distributed real-time systems. *Distributed Computing Systems, 1989.*, pages 190–198, 1989.

[87] C. Pira and C. Artigues. Line search method for solving a non-preemptive strictly periodic scheduling problem. *Journal of Scheduling*, 19(3):227–243, 2016.

[88] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner. Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks. *IET Cyber-Physical Systems: Theory & Applications*, 1(1):86–94, 2016.

[89] W. Puffitsch, E. Noulard, and C. Pagetti. Off-line mapping of multi-rate dependent task sets to many-core platforms. *Real-Time Systems*, 51(5):526–565, 2015.

[90] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *Parallel and Distributed Systems, IEEE Transactions on*, 6(4):412–420, April 1995.

[91] F. Reimann, M. Lukasiewycz, M. Glass, C. Haubelt, and J. Teich. Symbolic system synthesis in the presence of stringent real-time constraints. In *Proceedings of the 48th Design Automation Conference*, DAC '11, pages 393–398, New York, NY, USA, 2011. ACM.

[92] V. Rodrigues, B. Akesson, M. Florido, S. Melo de Sousa, J. P. Pedroso, and P. Vasconcelos. Certifying execution time in multicores. *Science of Computer Programming*, 111:505–534, 2015.

[93] F. Sagstetter, S. Andalam, P. Waszecki, M. Lukasiewycz, H. Stähle, S. Chakraborty, and A. Knoll. Schedule integration framework for time-triggered automotive architectures. *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6, 2014.

[94] S. Samii, A. Cervin, P. Eles, and Z. Peng. Integrated scheduling and synthesis of control applications on distributed embedded systems. In *Proceedings of the conference on design, automation and test in Europe*, pages 57–62. European Design and Automation Association, 2009.

[95] T. Schenkelaars, B. Vermeulen, and K. Goossens. Optimal scheduling of switched FlexRay networks. *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6, 2011.

[96] K. Schild and J. Würtz. Scheduling of time-triggered real-time systems. *Constraints*, 5(4):335–357, 2000.

[97] R. Schneider, D. Goswami, S. Zafar, M. Lukasiewycz, and S. Chakraborty. Constraint-driven synthesis and tool-support for flexray-based automotive control systems. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 139–148. ACM, 2011.

[98] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl. ILP-based joint routing and scheduling for time-triggered networks. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, pages 8–17. ACM, 2017.

[99] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *Computers, IEEE Transactions on*, 39(9):1175–1185, 1990.

[100] H. Shah, A. Knoll, and B. Akesson. Bounding SDRAM interference: Detailed analysis vs. latency-rate analysis. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pages 308–313. IEEE, 2013.

[101] S. Sriram and S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. CRC, 2000.

[102] L. Steffens, M. Agarwal, and P. Wolf. Real-time analysis for memory access in media processing SoCs: A practical approach. In *Real-Time Systems, 2008. ECRTS'08. Euromicro Conference on*, pages 255–265. IEEE, 2008.

[103] W. Steiner. TTEthernet specification. *TTTech Computertechnik AG, Nov*, 39:40, 2008.

[104] W. Steiner. An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks. *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, 31:375–384, November 2010.

[105] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch. Time-triggered ethernet. In *Time-Triggered Communication*, pages 209–248. CRC Press, 2011.

[106] A. Stevens. QoS for high-performance and power-efficient HD multimedia. *ARM White paper, http://wwww.arm.com*, 2010.

[107] D. Stiliadis and A. Varma. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on Networking (ToN)*, 6(5):611–624, 1998.

[108] S. Stuijk, M. Geilen, and T. Basten. Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs. *Computers, IEEE Transactions on*, 57(10):1331–1345, 2008.

[109] A. Syed and G. Fohler. Efficient offline scheduling of task-sets with complex constraints on large distributed time-triggered systems. *Real-Time Systems*, pages 1–39, 2018.

[110] D. Tamas-Selicean, P. Pop, and W. Steiner. Synthesis of communication schedules for TTEthernet-based mixed-criticality systems. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 473–482. ACM, 2012.

[111] D. Tamas-Selicean, P. Pop, and W. Steiner. Design optimization of TTEthernet-based distributed real-time systems. *Real-Time Systems*, 51(1), 2015.

[112] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and microprogramming*, 40(2-3):117–134, 1994.

[113] L. T. Truong, C. De Gruyter, G. Currie, and A. Delbosc. Estimating the trip generation impacts of autonomous vehicles on car travel in Victoria, Australia. *Transportation*, 44(6):1279–1292, 2017.

[114] C. Van Berkel. Multi-core for mobile phones. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1260–1265, 2009.

[115] P. Van Der Wolf and J. Geuzebroek. SoC infrastructures for predictable system integration. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[116] R. J. Vanderbei. *Linear programming: foundations and extensions*. Kluwer Academic Publishers, 1996.

[117] J. P. Vink, K. van Berkel, and P. van der Wolf. Performance analysis of SoC architectures based on latency-rate servers. *Design, Automation and Test in Europe, 2008. DATE'08*, pages 200–205, 2008.

[118] M. H. Wiggers, M. J. Bekooij, and G. J. Smit. Modelling run-time arbitration by latency-rate servers in dataflow graphs. In *Proceedingsof the 10th international workshop on Software & compilers for embedded systems*, pages 11–22. ACM, 2007.

[119] S. Wildermann, M. Glaß, and J. Teich. Multi-objective distributed run-time resource management for many-cores. In *Proceedings of the conference on Design, Automation & Test in Europe*, pages 221:1–221:6, 2014.

[120] F. Xia and Y. Sun. Control-scheduling codesign: A perspective on integrating control and computing. *arXiv preprint arXiv:0806.1385*, 2008.

[121] T. Yang and A. Gerasoulis. List scheduling with and without communication delays. *Parallel Computing*, 19(12):1321–1344, 1993.

[122] Y. Yi, W. Han, X. Zhao, A. Erdogan, and T. Arslan. An ILP formulation for task mapping and scheduling on multi-core architectures. *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09*, pages 33–38, 2009.

[123] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty. Task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems. *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pages 119–124, 2014.

[124] W. Zheng, J. Chong, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Extensible and scalable time triggered scheduling. In *Application of Concurrency to System Design, 2005. ACSD 2005. Fifth International Conference on*, pages 132–141. IEEE, 2005.

Chapter A

Nomenclature – Chapter 2

Background and Problem Statement Symbols

c_i	Real-time client.
C	Set of real-time clients.
n	Number of clients.
ϕ_i	Number of slots, allocated to client c_i.
r_i^j	Minimum provided service to client c_i during a busy period of duration j.
sz_i^k	Size of k-th request by client c_i.
arr_i^k	Arrival time of k-th request by client c_i.
fin_i^k	Finishing time of k-th request by client c_i.
$\hat{\Theta}_i$	Required service latency of client c_i.
Θ_i	Service latency of client c_i provided by a schedule.
$\hat{\rho}_i$	Required bandwidth (bandwidth) of client c_i.
ρ_i	Allocated bandwidth of client c_i provided by a schedule.
H	Schedule length.
F	Set of time slots.
D	Schedule.
d_j	Schedule element at time j.
Φ	Total allocated bandwidth of all real-time clients in C.

Symbols Used in ILP Model and Branch-and-Price Approach

t_i^j	Variable indicating that time slot j is allocated to client c_i.
$\underline{r_i^j}$	Worst-case provided service to client c_i during a busy period of duration j.
$\underline{\phi_i}$	Lower bound on the number of slots allocated to client c_i.
Ω	Set of all possible columns.
$MM(\Omega)$	Master model that contains Ω.
$\omega_{i,k}$	Variable indicating that column z_k is included in the schedule for client c_i.
z_k	column from set Ω.

$h_{i,k}^{j}$ Coefficient that indicates column z_k allocates slot j to client c_i.

Ω^R Subset of columns.

$MM(\Omega^R)$ Restricted master model that works on subset of columns Ω^R only.

Φ^{LB} Lower bound for the optimal integral solution.

Φ^{UB} Upper bound for the optimal solution.

y_j Variable reflecting the over-allocation of slot j in the solution.

v_j The number of clients to which slot j is allocated in the solution.

$\mathbb{M}$ Big number used to penalize criterion of master model for over-allocated slots.

Φ^{MM} Criterion value of the master model.

$b_{i,k}$ Coefficient indicating that column z_k was constructed for client c_i.

$DMM(\Omega^R)$ Dual master model.

λ_j Variable of dual master model equal a potential gain of criterion Φ^{MM} if slot j is allowed to be allocated more than once.

σ_i Variable of dual master model equal to reduction of Φ^{MM} if client c_i has no columns in the solution.

$\phi_{i,k}$ Number of slots allocated to client c_i in column z_k.

Φ_i^{sub} Criterion value of sub-model for client c_i.

ω_j^{total} the total "probabilities" of including each slot $j \in F$ in the solution.

$\overline{\Phi}^{LB}$ estimated lower bound on Φ^{MM}.

Φ_{curr}^{MM} current criterion value of the master model.

Symbols Used in Heuristic and Experimental Evaluation

β Interval from which bandwidth requirements for each client are uniformly drawn.

γ Interval from which latency requirements for each client are uniformly drawn.

Chapter B

Nomenclature – Chapter 3

System Model Symbols

m	Number of resources.
U	Set of resources (cores + input ports).
u_k	Resource k.
T	Set of tasks.
M	Set of messages.
A	Set of activities (tasks and messages), $A = T \cup M$.
a_i	i-th activity.
n	Number of activities.
P	Set of activity periods.
p_i	Period of a_i.
e_i	Execution time of a_i.
jit_i	Jitter of a_i.
map_i	Number of resource to which task a_i is mapped.
$\mathbf{z}_{i,j}$	Variable that indicates task i is mapped to resource u_j.
o_j	Variable equal to the load on resource u_j.
H	Hyper-period (schedule length).
s_i^j	Start time of job j of activity a_i.
n_i^{jobs}	Number of jobs of activity a_i.
mem	Amount of memory required to store the schedule in bytes.

Symbols Used in ILP and CP Models

x_i^j	Variable indicating that time slot j is allocated to client c_i.
$Pred_i$	Direct predecessors of a_i.
$Succ_i$	Direct successors of a_i.
t_i^b	Length of the longest critical path of the preceding activities that must be executed before activity a_i.
t_i^a	Length of the longest critical path of the succeeding activities that must be executed after activity a_i.
I_i	Worst-case slack of activity a_i.

Symbols Used in Heuristic and Experimental Evaluation

Q	Priority queue comprising activities to be scheduled.
D	Set of already scheduled activities, represented as a union of intervals on each resource.
Sch	Set of already scheduled activities, represented by their schedules.
R	Set of activities that were problematic to schedule.
jit_i^{inher}	Jitter inherited by activity a_i.
a_u	Scheduled activity to remove from the schedule.
a_s	Activity to schedule.
$Scratch$	Set of activities that were previously scheduled from scratch and the predecessors of a_s and a_u.
S	Schedule obtained by the heuristic.
r_y	Utilization of resource y.

Chapter C

Nomenclature – Chapter 4

System Model Symbols

m_{ECU}	Number of ECUs in a problem instance.
m_{Dom}	Number of domains in a problem instance.
m_{Links}	Number of network links in a problem instance.
m	Number of resources.
app_w	Application w.
App	Set of applications.
p_i	Period of a_i.
T	Set of tasks.
M	Set of message frames.
A	Set of activities (tasks and messages), $A = T \cup M$.
a_i	i-th activity.
n	Number of activities.
map_i	Number of resource to which task a_i is mapped.
e_i	Execution time of a_i.
L_w	End-to-end latency of app_w.
$\hat{L}_w$	Bound on end-to-end latency of application app_w.
$\mathbf{c}$	Tunable coefficient to bound end-to-end latency.
LUT_w	Look-up table for application app_w with settling time ξ values.
ξ_k^w	Control performance value of application app_w for end-to-end latency value δ_k^w.
δ_k^w	End-to-end latency value of application app_w for which k-th settling time of the system are measured.
N	Number of measurements for one application in LUT.

Problem Formulation Symbols

H	Hyper-period (schedule length).
s_i^j	Start time of job j of activity a_i.
n_i^{jobs}	Number of jobs of activity a_i.

$Pred_i$	Direct predecessors of a_i.
$Succ_i$	Direct successors of a_i.
$g_{i,k}$	Greatest common divisor of periods of activities a_i and a_k.
LB_i^j	Lower bound on start time of a_i^j.
UB_i^j	Upper bound on start time of a_i^j.
J_w	Settling time of application app_w for end-to-end latency value $L_w = \delta_k^w$.
λ_k^w	Variable indicating that L_w is in interval $[\delta_k^w, \delta_{k+1}^w)$.
γ_k^w	Variable reflecting the position of L_w in the interval $[\delta_k^w, \delta_{k+1}^w)$.

Symbols Used in ILP and CP Models

x_i^j	Variable indicating that time slot j is allocated to client c_i.
q	Quotient variable.
t_i^b	Length of the longest critical path of the preceding activities that must be executed before activity a_i.
t_i^a	Length of the longest critical path of the succeeding activities that must be executed after activity a_i.
I_i	Worst-case slack of activity a_i.

Symbols Used in Heuristic and Experimental Evaluation

Q	Priority queue comprising activities to be scheduled.
ϵ_i	Element to be scheduled, either activity or frame job.
G_d	Directed acyclic graph with frame jobs and tasks being nodes. An edge is directed from one node to another if removing the former node from Q can result in an earlier start time of the latter node.
ϵ_d	An element, which removing from the schedule can result in earlier start time of currently scheduled element.
l_d	Number of parent of the currently scheduled activity in the reason graph G_d to set as ϵ_d.
D	Set of already scheduled activities, represented as a union of intervals on each resource.
n_{int}	Number of intervals in D.
$\hat{s}_i$	Earliest possible start time of the currently scheduled element due to data dependency constraint.

$\overline{Pred_i}$	Set of predecessors finishing at $\hat{s}_i$.
$\check{s}_i$	Latest possible start time of the currently scheduled element ϵ_i due to end-to-end latency constraint.
Φ	Objective function value.
S_{neigh}	The neighborhood set.
r_y	Utilization of resource y.
n_T	Number of tasks in a problem instance.
n_{exp}	Expected number of tasks executed on one ECU.

Chapter D

Curriculum Vitae

Anna Minaeva was born in Krasnoyarsk, Russia in 1989. She received the B.Sc. degree in applied mathematics and informatics at Siberian Federal University in 2010. Next, she received the M.Sc. degree in systems analysis at Siberian State Aerospace University in 2012. The same year, she started studying artificial intelligence at Czech Technical University in Prague and successfully graduated with M.Sc. degree in 2014 and the master project was related to scheduling of real-time memory controllers. In September 2014, Anna started her journey towards a Ph.D. degree at the same university supported by Eaton Corporation. During her research work, she has published two papers in impacted international journals (Journal of Systems and Software and IEEE Transactions on Computers), and another one which is currently under the review in IEEE Transactions on Computers. Her areas of research interests are time-triggered periodic resource scheduling, automotive real-time embedded systems, and optimization problems.

Anna Minaeva
Prague, February 2019

Chapter E

List of Author's Publications

List of publications related to this thesis is included in this appendix.

Publications in Journals with Impact Factor

Anna Minaeva, Premysl Sucha, Benny Akesson, and Zdenek Hanzalek. Scalable and efficient configuration of time-division multiplexed resources. Journal of Systems and Software, 113: 44 – 58, March 2016. ISSN 0164-1212. DOI: http://dx.doi.org/10.1016/j.jss.2015.11.019. **Coauthorship 30%, indexed in Web of Science, 11 citations in Google Scholar**.

Anna Minaeva, Benny Akesson, Zdenek Hanzalek, and Dakshina Dasari. Time-triggered co-scheduling of computation and communication with jitter requirements. IEEE Transactions on Computers, 67(1): 115–129, July 2018a. ISSN 1557-9956. DOI: 10.1109/TC.2017.2722443. **Coauthorship 40%, indexed in Web of Science, 3 citations in Google Scholar.**

Anna Minaeva, Debayan Roy, Benny Akesson, Zdenek Hanzalek, and Samarjit Chakraborty. Efficient heuristic approach for control performance optimization in time-triggered periodic scheduling. IEEE Transactions on Computers, September 2018b. **Under the review, the decision will be known soon.**

International Conferences and Workshops

Benny Akesson, Anna Minaeva, Premysl Sucha, Andrew Nelson, and Zdenek Hanzalek. An efficient configuration methodology for time-division multiplexed single resources. In 2015 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pages 161–171, April 2015. DOI: 10.1109/RTAS.2015.7108439. **Coauthorship 22%, indexed in Web of Science, 21 citations in Google Scholar.**

Anna Minaeva, Benny Akesson, and Zdenek Hanzalek. Towards scalable configuration of time-division multiplexed resources. In 27th Euromicro Conference on Real-Time Systems July 7–10, 2015, page 1, July 2015.

Publications not Related to this Thesis

Katerina Brejchova, Jitka Hodna, Lucie Halodova, Anna Minaeva, Martin Hlinovsky, and Tomas Krajnik. Two-stage approach for long-term motivation of children to study robotics. In International Conference on Robotics and Education RiE 2017, pages 137–148. Springer, 2018.

Anna Minaeva
Prague, February 2019