Optimized Scheduling in Real-Time Environments with Column Generation

Dissertation
zur Erlangung des Grades

„Doktor der Naturwissenschaften“

am Fachbereich Physik, Mathematik und Informatik
der Johannes Gutenberg-Universität
in Mainz

Sebastian Hoffmann

geboren in Bad Homburg v.d.H.

Mainz, 2014

D77 – Mainzer Dissertation

Datum des Kolloquiums: 8. Oktober 2014

To my wife

Abstract

The design process of embedded applications in safety-critical systems is very complex. With regard to the distributed hardware architecture, computing units can be upgraded to execute all existing tasks and transmit all arising messages on time. As the timing requirements are strict and have to be fulfilled in every periodic occurrence of the tasks, the guarantee of schedulability is of highest importance. Existing approaches are able to quickly generate possible design alternatives, but they come with no guarantee of minimal cost for the applied hardware modifications.

We propose an approach that provides cost-minimal solutions to the distributed scheduling problem while satisfying all timing requirements. Our algorithm uses linear programming with column generation encapsulated in a branch and bound framework to hold lower and upper bounds available during the optimization process. The complex constraints which ensure the periodic schedulability are transferred to independent pricing problems by a decomposition of the master problem and are stated as integer linear programs.

Both task schedulability analyses and message transmission policies are examined and linear representations are given. Additionally, we present a new formulation for a fixed priority scheduling policy, which computes worst-case response times for tasks that are necessary for scenarios, where timing restrictions are specified over subsets of tasks and messages.

We prove the applicability of our methods by analysing instances with task networks containing data from real-world applications. Our results show that computed lower bounds are quickly available to prove the optimality of heuristic solutions. When providing optimal solutions with worst-case response times, our new formulation turns out to be advantageous over other approaches in terms of running times. The best results are obtained with a hybrid approach combining heuristic start solutions, non-sufficient presolving, and a heuristic phase with a small subsequent exact phase of computation.

Zusammenfassung

Im Bereich sicherheitsrelevanter eingebetteter Systeme stellt sich der Designprozess von Anwendungen als sehr komplex dar. Entsprechend einer gegebenen Hardwarearchitektur lassen sich Steuergeräte aufrüsten, um alle bestehenden Prozesse und Signale pünktlich auszuführen. Die zeitlichen Anforderungen sind strikt und müssen in jeder periodischen Wiederkehr der Prozesse erfüllt sein, da die Sicherstellung der parallelen Ausführung von größter Bedeutung ist. Existierende Ansätze können schnell Designalternativen berechnen, aber sie gewährleisten nicht, dass die Kosten für die nötigen Hardwareänderungen minimal sind.

Wir stellen einen Ansatz vor, der kostenminimale Lösungen für das Problem berechnet, die alle zeitlichen Bedingungen erfüllen. Unser Algorithmus verwendet Lineare Programmierung mit Spaltengenerierung, eingebettet in eine Baumstruktur, um untere und obere Schranken während des Optimierungsprozesses bereitzustellen. Die komplexen Randbedingungen zur Gewährleistung der periodischen Ausführung verlagern sich durch eine Zerlegung des Hauptproblems in unabhängige Unterprobleme, die als ganzzahlige lineare Programme formuliert sind.

Sowohl die Analysen zur Prozessausführung als auch die Methoden zur Signalübertragung werden untersucht und linearisierte Darstellungen angegeben. Des Weiteren präsentieren wir eine neue Formulierung für die Ausführung mit fixierten Prioritäten, die zusätzlich Prozessantwortzeiten im schlimmsten anzunehmenden Fall berechnet, welche für Szenarien nötig sind, in denen zeitliche Bedingungen an Teilmengen von Prozessen und Signalen gegeben sind.

Wir weisen die Anwendbarkeit unserer Methoden durch die Analyse von Instanzen nach, welche Prozessstrukturen aus realen Anwendungen enthalten. Unsere Ergebnisse zeigen, dass untere Schranken schnell berechnet werden können, um die Optimalität von heuristischen Lösungen zu beweisen. Wenn wir optimale Lösungen mit Antwortzeiten liefern, stellt sich unsere neue Formulierung in der Laufzeitanalyse vorteilhaft gegenüber anderen Ansätzen dar. Die besten Resultate werden mit einem hybriden Ansatz erzielt, der heuristische Startlösungen, eine Vorverarbeitung und eine heuristische mit einer kurzen nachfolgenden exakten Berechnungsphase verbindet.

Acknowledgments

I thank my advisor for introducing me to the interesting area of linear programming and combinatorial optimization, and for his guidance and help throughout my doctoral studies.

I would like to thank my second reviewer for his instant commitment to evaluate this dissertation and his helpful feedback.

Special thanks belong to the members of the Institute for Computer Science at the Johannes Gutenberg-University of Mainz for the interesting discussions, fascinating projects and pleasant working atmosphere.

I thank all members of the Transregional Collaborative Research Center SFB/TR 14 "Automatic Verification and Analysis of Complex Systems" (AVACS) [9], especially the members of the project group R2 "Timing Analysis and Distribution of Real-Time Tasks" for the fruitful discussions about the topic.

I am very thankful for the assistance and advice of all my friends and colleagues.

My deepest thanks go to my wife for always being a calm anchor and unwavering supporter in many ways. Finally, I thank my sister-in-law for her endurance in motivating me, and my entire family for all they have done for me.

Sebastian Hoffmann Mainz, 9 July 2014

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

That something is difficult must be one more reason to do it.
(Dass etwas schwer ist, muss ein Grund mehr sein, es zu tun.)
— Rainer Maria Rilke, Rome 14. Mai 1904
[Letter to Franz Xaver Kappus]

The design process of safety-critical embedded applications in the automotive industry is very complex. Starting with a structural model of a new functionality and the existing hardware architecture, the next step is to generate a task network consisting of widely independent tasks that communicate via signals. The crucial step is to allocate the new tasks to computational units so that their execution is guaranteed and satisfies all timing constraints. The allocation affects the load and arrangement of the communication buses which also have to ensure the transmission on time.

The given hardware architecture is allowed to be modified by exchanging computational units which affects the total cost of the system, as well as the execution time and accessible memory resources of the tasks.

The design space exploration seeks for a schedulable allocation of task networks that causes cost-minimal modifications of the existing hardware architecture and is known to be $\mathcal{NP}$-hard.

1.1 Motivation

The described design space exploration refers to the wide field of operations research where mathematical techniques are used to obtain near-optimal – superior optimal – solutions to practical applications. These solutions result in the automation of decision processes or at least in the computational guidance in complex scenarios. The requirements on algorithms in this field highly depend on the available time scale and the requested level of optimality.

On the one hand, heuristic approaches provide results without deeper knowledge about their quality within a small time window. Standard approaches, like evolutionary algorithms or simulated annealing methods, are also applicable but yield no information about reaching an optimal solution or sticking in a local optimum.

On the other hand, global optimal approaches, such as integer linear programming, terminate with an optimal solution. During the solving process of an *integer linear program (ILP)* additional information about the current objective value and bounds on the objective function are available. Therefore, statements about the quality of detected solutions are possible and the process can be terminated if the quality measure falls below a given threshold. The major drawback is the theoretically exponential computational complexity of ILP solvers for $\mathcal{NP}$-hard problems.

In general when developing algorithms for $\mathcal{NP}$-hard problems, a trade-off between accuracy and computational running time has to be made. In this dissertation we aim to develop an optimal approach based on integer linear programming, but also apply heuristic strategies to keep the running times acceptable. The challenging task is to find a suitable formalization so that state-of-the-art ILP solvers achieve a reasonable running time on instances close to reality.

As a naive ILP formulation for solving the examined problem is too complex and would lead to unacceptable running times, we develop an algorithm based on ILP methods with column generation. The results show that instances related to real-world scenarios can be solved within reasonable time limits and our new ILP formulation for the fixed scheduling policy turns out to be advantageous compared to other analyses.

1.2 Related Work

The work of Büker et al. [15, 16] describes the process from designing safety-critical embedded-systems to extracting all data to perform schedulability and memory analyses. The exploration of the design space is done with a two-tier approach that consists of a heuristic global and an exact local phase. In the

first, heuristic global phase the allocation of tasks to processing subsystems is realized. In subordinate, exact local phases schedulability and memory analyses are performed to ensure that the allocation is executable, otherwise the process starts again. The modular architecture allows the exchange of analyses in both phases.

In the PhD thesis of Thaden [55] a semi-automatic user-driven optimization algorithm is described that adds the element of motivation, that engineers can guide the exploration of the design space by their expert knowledge, to the modular two-tier approach. The software architecture captures the two-tier approach with pre-allocations in the heuristic global phase and exact analyses in the exact local phase. The local analyses are designed to schedule the most suitable tasks and non-allocated tasks are handled in later iterations.

The work of Zhu et al. [63] also follows a two-tier approach, but the operating principles of the phases are transposed to the approaches of Büker and Thaden. Zhu et al. describe a first exact phase that allocates the most important tasks by an ILP formalization, whereas the second phase consists of several heuristic steps including the reallocation of tasks. In their work they focus on the realization of task and message sequences with end-to-end deadlines.

An exact local analysis that fits in the setting of the two-tier approaches of Büker and Thaden and is based on ILPs is achieved by extending the work of Althaus et al. [2]. Therein the scheduling problem of a set of processing units is treated as an ILP, embedded in a branch and bound algorithm for *linear program (LP)* relaxations with column generation of similar architecture to the one in this dissertation.

Similarly, the work of Eisenbrand et al. [31] can be extended to serve as an exact local analysis within the two-tier approach of Büker and Thaden. The authors describe a linear formulation of the well-known fix-point equality to determine worst-case response times for preemptive, fixed priority scheduling policies.

1.3 Contributions and Objectives

The main contribution of this dissertation is the development of an approach based on LP methods with column generation that can solve the distributed scheduling problem to optimality. Therefore, we show that all necessary real-time requirements of the considered scenarios are expressible with linear formulations, including constraints on the hardware architecture, on the periodic schedulability of tasks, and on the transmission of messages via the global bus. Additionally, we present a new ILP formulation for the computation of worst-case response times that directly uses the structure of the applied column generation approach.

Our algorithm design allows the application of several scheduling policies, as well as heuristic strategies, so that it is capable to quickly provide safe lower bounds on the objective function to assess the quality of heuristic solutions. Moreover, valid heuristic solutions from external applications – even if they are only partial – can be included into our algorithm and serve as generated variables in the column generation approach that reduce the solution space. Regardless of the objective, either providing lower bounds or computing the optimal solution, a hybrid approach using heuristics and optimal LP methods can be evolved from our algorithm design.

A naive linear formulation for the $\mathcal{NP}$-hard distributed scheduling problem is not feasible within reasonable running times due to its complexity. The structure of our problem allows a Dantzig-Wolfe decomposition into several independent subproblems for the individual subsystems if we apply a column generation approach. Previous works, especially Althaus et al. [2], showed that this strategy is promising in a similar scenario. A more detailed insight into our algorithm design is given in Chapter 4 with the linear formulations for the ILP subproblems presented in Chapter 5.

Parts of this work were honored with a Best Paper Award at the International Conference on Computer Science and Applications at the World Congress on Engineering and Computer Science [3], and were extended for publication in the Lecture Notes in Electrical Engineering [4].

The requirements of real-world applications and the research on methods to handle them lead to the following objectives and challenges in the development of this work.

1.3.1 Assessing Quality of Solutions by Providing Lower Bounds

Existing heuristic approaches are able to generate solutions, but they come with no measurement of quality, as they are not approximation algorithms. This raises the question how tight lower bounds can be provided to assess the quality of heuristic solutions.

To estimate a lower bound we apply necessary but non-sufficient scheduling policies in a first attempt. One advantage of the developed algorithm is that we solve LP relaxations of a minimization problem within a branch and bound framework, so that lower bounds are obtained besides reaching for the overall optimal solution, see Section 1.3.2. Therefore, the computation of lower bounds comes as a byproduct of the optimization process regardless of the applied scheduling policy. Nevertheless the strategy of applying non-sufficient scheduling policies that consume less running time can be included in the approach of reducing the solution space, see Section 1.3.3.

1.3.2 Providing Global Optimal Allocations

In the situation where heuristic approaches exist and a complex ILP formalization can be used to provide lower bounds on the objective value, naturally, the question arises whether or not an ILP formalization is able to provide global optimal solutions in reasonable running time. These global optimal solutions consist of an allocation of tasks and messages that ensures that every task and message meets its deadline in every periodic cycle and the overall hardware costs are minimized.

In this thesis we show that the considered model is able to express all requirements as ILPs with a huge number of variables. To keep the running times reasonable several strategies are applied. Starting with an LP relaxation, the structure of the problem allows a decomposition leading to a column generation approach that consists of several independent subproblems. Encapsulated in a branch and bound framework, some strategies in the processing of the branch and bound nodes are promising to handle real-world applications in reasonable running time. A similarly working heuristic phase is added to these combined ideas to reduce the search space of the LP approaches by providing upper bounds.

1.3.3 Development of Hybrid Approaches

The existing two-tier approaches with heuristic algorithms and the previously introduced global optimal approach point in different directions. This raises the question how both fields of research benefit each other.

Whereas lower bounds can assess the quality of heuristic approaches, heuristic solutions can reduce the search space of LP approaches by providing upper bounds. This idea leads to an inclusion of heuristic start solutions as well as presolving methods into our algorithm. Due to the column generation structure, partial heuristic solutions are also valuable as they can be combined to full solutions in the LP relaxation. This hybrid approach promises to further reduce the running time and offers the possibility of compound benefits.

1.4 Context

This thesis is located in the context of the Transregional Collaborative Research Center SFB/TR 14 “Automatic Verification and Analysis of Complex Systems” (AVACS) [9]. It is part of the subproject R2 of AVACS “Timing Analysis and Distribution of Real-Time Tasks” that focuses on the development of task distributions and scheduling techniques in the environment of distributed computer architectures.

1.5 Outline

The hereafter chapters of this thesis are structured as follows.

Chapter 2 contains the basic preliminaries to understand the context of the work and summarizes well-known results from the relevant academic literature. The main focus is on linear optimization and real-time scheduling.

In the sections about linear programming we focus on developing the necessary theory to finally state the established simplex method to solve LPs and ILPs by some extensions. The operating principle of the simplex method is linked to the development of our column generation approach in Chapter 4. As the simplex method is used widely in commercial LP and ILP solvers, it offers the possibility to benefit from the performance and efficiency of these state-of-the-art solvers in our implementation.

In the section about real-time scheduling we give a short introduction into embedded systems, but mainly focus on task and message scheduling policies and analyses. A variety of policies and analyses is presented, and applied in different scenarios later on.

Chapter 3 presents the formal definitions of both variants of the *distributed scheduling problem (DSP)* of this thesis. The general formulation contains numerous conditions concerning the hardware architecture, the tasks and relations between them, as well as messages and the transmission on the global bus. The more complex chain formulation additionally respects timing requirements on the execution of sequences of tasks and messages with end-to-end deadlines.

Chapter 4 examines our developed algorithm for solving the DSP to optimality. We present an ILP formulation of the problem and proceed to its LP relaxation which is iteratively solved within a branch and bound framework. As we handle large LPs we apply a column generation approach that shifts most of the complexity to independent subproblems. These subproblems are again ILPs and take the formulations of the schedulability tests of Chapter 5. The chapter closes with several strategies that are also used in the implementation, such as multiple columns generation as well as a heuristic phase.

Chapter 5 presents the linear formulations that are contained in the subproblems. Besides general constraints, the analyses are divided into formulations for tasks, messages, and more complex scenarios as provided in the chain formulation. The task schedulability analyses cover the processor utilization, the response time, and the workload demand, whereas the message schedulability analyses deal with several buses, like TAN, CAN, and a simple TDMA bus. In the scenario of complex chain constraints, we present a new ILP formulation to compute worst-case response times that directly uses the structure of our column generation process.

Chapter 6 starts with the description of the processed instances from a project of a driver assistance system. We present several experiments structured by the main three objectives introduced in Section 1.3 and compare our results to related approaches where it is possible.

Chapter 7 shortly summarizes the results and leads over to a discussion that focuses on the algorithm design. We also comment on the scalability of our approach, the use of symmetry in the data, the opportunities for parallelization, as well as the possibilities of parameter tuning. For every previously mentioned objective, we present an outlook of further improvements and a line of possible future research.

Chapter 2

Preliminaries

This chapter starts with a collection of general preliminaries, followed by a short introduction to computational complexity, and some basics in linear algebra. We then give an introduction to linear programming focusing on the simplex method to solve linear programs, as well as several algorithm schemes for integer linear programming. We conclude this chapter with fundamentals of real-time scheduling in the scenario of embedded systems.

The content of this chapter is a collection from various sources and not original contribution: Bertsimas and Tsitsiklis [10], Schrijver [53, 54], Cormen et al. [21] Beutelspacher [11], and Burns and Wellings [17].

2.1 General Preliminaries

The symbols $\mathbb{Z}, \mathbb{Q}$ and $\mathbb{R}$ denote the set of integers, rationals and real numbers. $\mathbb{N}$ denotes the set of natural numbers, that is, the restriction of $\mathbb{Z}$ to non-negatives, and $\mathbb{B} = \{0, 1\}$ is the integer representation of the Boolean values true and false. The *power set* of a set S is denoted by 2^S.

For a real number $r \in \mathbb{R}$ we define the absolute value $|r| = \max\{r, -r\}$.

For any real numbers a, b we define the *closed interval* $[a, b] = \{x \in \mathbb{R} \mid a \leqslant x \leqslant b\}$, the *open interval* $(a, b) = \{x \in \mathbb{R} \mid a < x < b\}$, the *left half-open interval* $(a, b] = \{x \in \mathbb{R} \mid a < x \leqslant b\}$, and the *right half-open interval* $[a, b) = \{x \in \mathbb{R} \mid a \leqslant x < b\}$.

For two sets X and Y we denote a *function* f from X to Y by $f : X \longrightarrow Y$,

satisfying that for every argument $x \in X$ there is a unique function value $y = f(x)$. If there is at least one argument $x \in X$ where $f(x)$ is undefined, i.e., if $f : X' \longrightarrow Y$ with $X' \subset X$ is a function, we call f from X to Y *partial function* and denote it by $f : X \nrightarrow Y$.

For every $x \in \mathbb{R}$ we denote the *floor function* by $\lfloor x \rfloor = \max\{k \in \mathbb{Z} \mid k \leqslant x\}$, and respectively the *ceiling function* by $\lceil x \rceil = \min\{k \in \mathbb{Z} \mid k \geqslant x\}$, so the inequalities $x - 1 < \lfloor x \rfloor \leqslant x \leqslant \lceil x \rceil < x + 1$ hold.

For a finite set of positive integers $\{a_1, a_2, \ldots, a_n\}$ the *lowest common multiple* $m = \operatorname{lcm}(a_1, a_2, \ldots, a_n)$ is the smallest positive integer m that is divisible by all a_i for $i = 1, 2, \ldots, n$.

2.2 Computational Complexity

When we discuss the computational complexity of algorithms, we have to distinguish between problems, like linear programming, and instances of these, to work out criteria for calling an algorithm efficient.

A *problem* consists of a collection of *instances*, where instances are described according to a common format.

The following definitions of the *size* of data depend on the way the data is represented, therefore, we present three common models to represent natural numbers. The extension to integers, as a tuple of a sign and a natural number, and rational numbers as the fraction of two integers, is straightforward.

Unary model A natural number $n \in \mathbb{N}$ is represented by a sequence of n ones followed by a zero, so we obtain $\operatorname{size}(n) = n + 1$.

Bit model (Turing machine model) Natural numbers $n \in \mathbb{N}$ are represented by a sequence of $\lfloor \log_2(n) \rfloor + 1$ binary scalars using their binary representation $n = \sum_{i=0}^{k} \alpha_i 2^i$ with $\alpha_i \in \mathbb{B}$, and zero is represented by $\alpha_0 = 0$. Thus we obtain $\operatorname{size}(n) = \lfloor \log_2(|n|) \rfloor + 1$ for $n > 0$ and $\operatorname{size}(0) = 1$.

Arithmetic model The size is independent of the value, thus $\operatorname{size}(n) = 1$ for all $n \in \mathbb{N}$.

When we talk about efficiency of algorithms, we have to realize that algorithms are designed to solve problems – but are applied to individual instances – so an *algorithm* is a finite set of instructions of a common programming language.

The *running time* of an algorithm is the total number of steps involved in carrying out these instructions until a termination statement is reached, thus the running

time depends on the underlying representation of data, because complex instructions in the arithmetic model have to be decomposed into elementary single-bit instructions in the bit model, respectively, unary operations in the unary model.

To simplify the counting of operations, we introduce the *order of magnitude* notation to categorize real-valued functions. Considering a given real-valued function $g(n)$ we write

$$f(n) = O(g(n)) \Longleftrightarrow \exists\, c > 0, n_0 \in \mathbb{N} \text{ so that } 0 \leqslant f(n) \leqslant cg(n)\ \forall\ n \geqslant n_0,$$
$$f(n) = \Omega(g(n)) \Longleftrightarrow \exists\, c > 0, n_0 \in \mathbb{N} \text{ so that } 0 \leqslant cg(n) \leqslant f(n)\ \forall\ n \geqslant n_0,$$
$$f(n) = \Theta(g(n)) \Longleftrightarrow f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n)).$$

The *worst-case* running time of an algorithm is determined by the largest running time over all inputs up to a given size n, thus we obtain different definitions of worst-case running times and what is meant by a polynomial-time algorithm depending on the underlying model.

An algorithm is *pseudo polynomial* if its worst-case running time in the unary model is $T_u(n) = O(n^k)$ for some $k \in \mathbb{N}$; *weakly polynomial* if its worst-case running time in the bit model is $T_b(n) = O(n^k)$ for some $k \in \mathbb{N}$; and *strongly polynomial* if its worst-case running time in the arithmetic model is $T_a(n) = O(n^k)$ for some $k \in \mathbb{N}$.

If an algorithm takes polynomial time under the arithmetic model, i.e., $T_a(n) = O(n^k)$ for some $k \in \mathbb{N}$, and any integer produced during the execution is also bounded by a polynomial in n, then the algorithm runs in polynomial time under the bit model as well, i.e., $T_b(n) = O(n^\ell)$ for some $\ell \in \mathbb{N}$. Therefore, we talk about running times in the arithmetic model most of the times, unless otherwise stated[1].

An algorithm is considered *efficient* if its running time in the arithmetic model grows polynomially with the size of the input, otherwise it is considered *inefficient*.

One of the most interesting and challenging problems in computer science is the question if both complexity classes $\mathcal{P}$ and $\mathcal{NP}$ are equal or not. Here we consider the bit model. $\mathcal{P}$ denotes the class of decision problems solvable in polynomial time by a deterministic Turing machine, and $\mathcal{NP}$ denotes the class of decision problems solvable in polynomial time by a non-deterministic Turing machine. Thus $\mathcal{P} \subset \mathcal{NP}$ is obvious.

[1] Examples: the algorithm of Ford and Fulkerson to compute the maximum flow in a flow network is pseudo polynomial, but the algorithm of Edmonds and Karp is strongly polynomial; the Euclidean algorithm to compute the greatest common divisor of two integers is weakly polynomial.

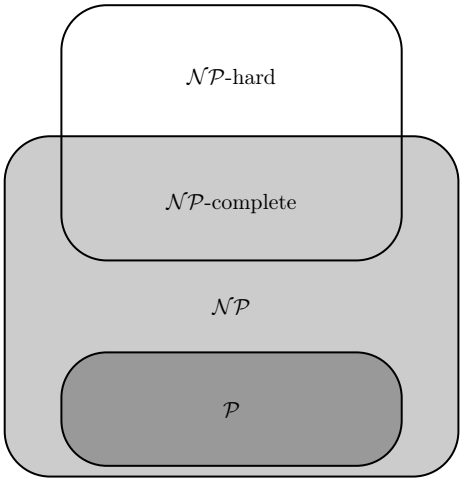


Figure 2.1: Complexity classes.

Complexity classes $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-complete, and $\mathcal{NP}$-hard under the assumption $\mathcal{P} \neq \mathcal{NP}$ with raising complexity from bottom to top. If $\mathcal{P} = \mathcal{NP}$, the classes $\mathcal{P}$, $\mathcal{NP}$, and $\mathcal{NP}$-complete would be equal and a subset of $\mathcal{NP}$-hard.

A problem K is *polynomial-time reducible* to a problem L, denoted by $K \leqslant_p L$, if there is a total polynomial-time computable function f from the alphabet of K into the alphabet of L that fulfills $x \in K \Leftrightarrow f(x) \in L$. Problem L is called *$\mathcal{NP}$-hard* if all $\mathcal{NP}$ problems can be polynomial-time reduced to L, i.e., $K \leqslant_p L$ for all $K \in \mathcal{NP}$. $\mathcal{NP}$-hard problems that belong to $\mathcal{NP}$ build the complexity class *$\mathcal{NP}$-complete*, see Figure 2.1 for an overview.

The *satisfiability (SAT)* problem, i.e., the question if there is a truth assignment to a Boolean expression, was the first problem shown to be $\mathcal{NP}$-complete by Cook [20] in 1971. Cook and Levin [48] laid the foundations of the theory of $\mathcal{NP}$-completeness.

In 1972, Karp [41] showed the $\mathcal{NP}$-completeness of 21 basic decision problems. Nowadays thousands of problems are known to be $\mathcal{NP}$-complete and all of them would be solvable in polynomial time if any single $\mathcal{NP}$-complete problem can be shown to be in $\mathcal{P}$, i.e., $\mathcal{P} = \mathcal{NP}$. Figure 2.1 shows the relation under the assumption $\mathcal{P} \neq \mathcal{NP}$.

2.3 Linear Algebra

A real *matrix* $A \in \mathbb{R}^{m \times n}$ is an array of real numbers of dimension $m \times n$ with its (i, j)-th entry a_{ij} or $[A]_{ij}$. By $a_i \in \mathbb{R}^{1 \times n}$ we denote the i-th row and by $A^{(j)} \in \mathbb{R}^m$ we denote the j-th column of A. Consequently, *vectors* are considered as one-

column matrices with $n = 1$ unless otherwise stated. The *transpose* A^T of matrix $A \in \mathbb{R}^{m \times n}$ is the $n \times m$ matrix A^T with $[A^T]_{ij} = [A]_{ji}$, respectively column vectors transpose into row vectors and vice versa.

The i-th *unit vector* e_i is the vector with all components equal to zero except for the i-th component which is equal to one. We use I to denote the square *identity matrix* with all diagonal entries equal to one and the others equal to zero. For a vector x we use the notation $x \geqslant 0$ or $x \leqslant 0$ in a component-wise meaning, i.e., every entry fulfills the inequality.

The *inner product* of two vectors $c, x \in \mathbb{R}^n$ is given by $c^T x = x^T c = \sum_{i=1}^{n} c_i \cdot x_i$ and defines both a *linear function* f in x by $f(x) = c^T x$ and a linear function $g(c) = x^T c$. *Linear constraints* can occur as equalities $f(x) = b$ or inequalities of the form $f(x) \leqslant b$ or $f(x) \geqslant b$ for any real number b. The *matrix product* of two matrices $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$ is the matrix $AB \in \mathbb{R}^{m \times n}$ given by $[AB]_{ij} = \sum_{\ell=1}^{k} [A]_{i\ell} \cdot [B]_{\ell j}$, the *matrix-vector* product is defined accordingly for $B \in \mathbb{R}^{k \times 1}$.

Given a square matrix A, we say that A is *invertible* if and only if there exists a square matrix A^{-1} of the same dimension that satisfies $AA^{-1} = A^{-1}A = I$ and call it the *inverse matrix* of A. If we are given a system $Ax = b$ with an invertible matrix A, the solution $x = A^{-1}b$ is given by *Cramer's rule*, i.e., the j-th component of x is obtained by $x_j = \frac{\det(A^j)}{\det(A)}$, where A^j is the same matrix as A, except that its j-th column is replaced by b. The *determinant* $\det(A)$ of a square n-dimensional matrix A can be computed by the *Laplace expansion* for any row $i = 1, 2, \ldots, n$ by $\det(A) = \sum_{j=1}^{n} (-1)^{i+j} [A]_{ij} \cdot \det(A_{ij})$, where A_{ij} denotes the $(n-1) \times (n-1)$ matrix that equals A, but the i-th row and j-th column are deleted.

We say a finite collection of vectors $x_1, x_2, \ldots, x_k \in \mathbb{R}^n$ is *linearly dependent* if there exist $\lambda_1, \lambda_2, \ldots, \lambda_k \in \mathbb{R}$, not all of them zero, so that $\sum_{i=1}^{k} \lambda_i x_i = 0$; otherwise, we call them *linearly independent*.

To quote the number of arithmetic operations performed when using basic algebraic operations, consider a square matrix $A \in \mathbb{R}^{n \times n}$ and vectors $x, b \in \mathbb{R}^n$. Computing the inner product $b^T x$ takes $O(n)$, the matrix-vector product Ax takes $O(n^2)$, the inverse of A or solving a linear system $Ax = b$ takes $O(n^3)$ arithmetic operations.

2.4 Linear Programming

In a general *linear program* (LP) we minimize a linear *objective function* $c^T x$ given by a vector $c \in \mathbb{R}^n$ over a set of *optimization variables* $x \in \mathbb{R}^n$ subject to

a set of linear equality and inequality constraints. Therefore, consider index sets I_1, I_2, I_3 and for every i in these sets we are given a vector $a_i \in \mathbb{R}^n$ and a scalar $b_i \in \mathbb{R}$ to build the i-th constraint. Furthermore, bounds to the optimization variables can be given. Consider index sets $J_1, J_2 \subseteq \{1, 2, \ldots, n\}$ and for every $j \in J_1$ we are given a lower bound $l_j \in \mathbb{R}$ to x_j; and for every $j \in J_2$ we are given an upper bound $u_j \in \mathbb{R}$ to x_j. We can formulate the LP in its most *general form* as

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & a_i^T x \geqslant b_i \qquad i \in I_1 \\ & a_i^T x \leqslant b_i \qquad i \in I_2 \\ & a_i^T x = b_i \qquad i \in I_3 \\ & x_j \geqslant l_j \qquad j \in J_1 \\ & x_j \leqslant u_j \qquad j \in J_2. \end{aligned} \tag{2.1}$$

The case of a maximization problem is equivalent to minimizing the negative objective function.

A *feasible solution* is a vector x satisfying all of the constraints including the bounds on the variables, and the set of feasible solutions is called *feasible set*. A feasible solution $x^\star$ that minimizes the objective function, i.e., $c^T x^\star \leqslant c^T x$ for all feasible solutions x, is called an *optimal solution* and the corresponding value of the objective function $c^T x^\star$ is called *optimal value*. If we find a feasible solution whose objective value is less than every real number, the optimal value is $-\infty$ and we call the problem *unbounded*; in the case we cannot find a feasible solution, we call the problem *infeasible*.

2.4.1 Canonical Form

To unify the notation we introduce the *canonical form* of an LP that only takes linear greater-than-or-equal constraints. Starting with the general form (2.1), we replace all equality constraints $a_i^T x = b_i$ by two symmetric inequalities $a_i^T x \geqslant b_i$ and $a_i^T x \leqslant b_i$, afterwards, we replace all inequality constraints of the form $a_i^T x \leqslant b_i$ by $-a_i^T \geqslant -b_i$. The conditions given by lower and upper bounds are interpreted as constraints and are handled just the same.

Suppose that a total of m such constraints remains, indexed by $1, 2, \ldots, m$; we write the scalars b_i into a vector $b \in \mathbb{R}^m$ and the corresponding vectors a_i^T are

building the rows of the matrix $A \in \mathbb{R}^{m \times n}$, i.e.,

$$b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \quad \text{, respectively,} \quad A = \begin{bmatrix} — \; a_1^T \; — \\ \vdots \\ — \; a_m^T \; — \end{bmatrix}.$$

Then we obtain the canonical form of the LP as

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \geqslant b. \end{aligned} \tag{2.2}$$

2.4.2 Standard Form

The following LP is in *standard form*, i.e., it only takes linear equalities rather than inequalities, and all optimization variables have to be non-negative

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geqslant 0. \end{aligned} \tag{2.3}$$

The standard form is a special case of the general form (2.1), because we can replace the equality constraints by two symmetric inequality constraints as shown above. The converse is also true and both forms are equivalent, if we introduce new variables.

Linear inequality constraints If we are given a linear inequality constraint of the form $a_i^T x \leqslant b_i$ for some i, then we introduce a new *slack variable* s_i with the non-negativity constraint $s_i \geqslant 0$ to close the gap between $a_i^T x$ and b_i by stating $a_i^T x + s_i = b_i$. Similarly, if the constraint is of the form $a_i^T x \geqslant b_i$ then we call $s_i \geqslant 0$ a *surplus variable* and state $a_i^T x - s_i = b_i$.

Unbounded variables Given an unbounded (*free*) variable x we can express x as the difference of two non-negative variables, x^+ and x^-, by replacing $x = x^+ - x^-$ with two non-negativity constraints $x^+ \geqslant 0$ and $x^- \geqslant 0$.

We will often use the canonical or general form to develop LPs, but when we talk about algorithms the standard form is more convenient.

2.4.3 Algebra and Geometry

For the development and discussion of the different methods to solve LPs we recall some basic definitions and results of the algebra and geometry of linear

programming to simplify the representation.

In the sequel $P \subseteq \mathbb{R}^n$ denotes a *polyhedron* given by $P = \{x \in \mathbb{R}^n \mid Ax \geqslant b\}$ with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, and thus equals the intersection of the *half spaces* $\{x \in \mathbb{R}^n \mid a_i^T x \geqslant b_i\}$ for $i = 1, 2, \ldots, m$. A *polytope* is a bounded polyhedron, i.e., there exists a constant $K \in \mathbb{R}$, so that the absolute value of every component of every element of the polyhedron is less than or equal to K.

A set $S \subseteq \mathbb{R}^n$ is *convex* if for any two points $x, y \in S$ and any scalar $\lambda \in [0, 1]$, we have $\lambda x + (1 - \lambda)y \in S$, hence, polyhedra are convex[2]. A *convex combination* of the vectors $x_1, x_2, \ldots, x_k$ is the vector $\sum_{i=1}^k \lambda_i x_i$ with $\lambda_i \in [0, 1]$ for $i = 1, 2, \ldots, k$ and $\sum_{i=1}^k \lambda_i = 1$, thus every convex combination of points of a convex set lies within the set.

We call $x \in P$ an *extreme point* of P if we cannot find two vectors $y, z \in P$ with $y \neq x \neq z$ and a scalar $\lambda \in [0, 1]$, so that $x = \lambda y + (1 - \lambda)z$, see point D in Figure 2.2 for a counterexample.

A vector $x \in P$ is a *vertex* of P if there exists some vector $c \in \mathbb{R}^n$, so that $c^T y > c^T x$ for all $y \in P$ with $y \neq x$, i.e., there exists a *hyperplane* $\{y \in \mathbb{R}^n \mid c^T y = b\}$ with $b = c^T x$ that meets P only at the point x, see point C in Figure 2.2.

We say a constraint is *active* or *binding* at a point, if the constraint holds with equality at that point. Now consider that the definition of the polyhedron P can also include equality constraints. A vector $x \in \mathbb{R}^n$ is called *basic solution* if all equality constraints are active at x and out of the active constraints there are n of them that are linearly independent. If x is a basic solution that satisfies all of the constraints, i.e., $x \in P$, we say that x is a *basic feasible solution*. A basic solution x is called *degenerate* if more than n constraints are active at x.

Bringing together the geometric definitions of vertices and extreme points with the algebraic definition of feasible solutions, we obtain the equivalence by Theorem 2.1, see Figure 2.2 for an example.

Theorem 2.1 (Equivalence of the definitions). *Let $P \subseteq \mathbb{R}^n$ be a non-empty polyhedron and let $x \in P$, then the following are equivalent: x is a vertex; x is an extreme point; and x is a basic feasible solution.*

We remark that the property of being a basic solution depends on the representation of the polyhedron[3] P, but with Theorem 2.1 we conclude that a basic

[2] Consider $x, y \in P$ satisfying $Ax \geqslant b$, respectively, $Ay \geqslant b$, and any $\lambda \in [0, 1]$, then every point between x and y fulfills $A(\lambda x + (1 - \lambda)y) = \lambda(Ax) + (1 - \lambda)(Ay) \geqslant yb + (1 - \lambda)b = b$, hence it lies in P.

[3] Consider a point x satisfying the constraint $a_i^T x \leqslant b_i$ but violating $a_i^T x \geqslant b_i$ of P. Then x is a basic solution in this representation, but if we replace both inequalities by the equality constraint $a_i^T x = b_i$, then x is no longer a basic solution.

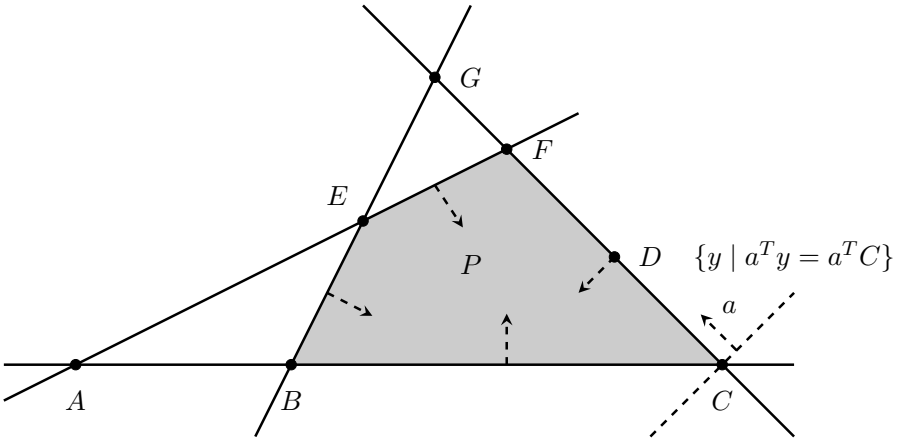


Figure 2.2: Polyhedron with basic feasible solutions.

A polyhedron P with the basic solutions A, B, C, E, F, G and the basic feasible solutions B, C, E, F that are also vertices and extreme points. Point D is neither a basic solution (only one active constraint), nor a vertex (there is no hyperplane that meets P only at D), nor an extreme point (center of C and F).

feasible solution is independent of the representation.

To get a more algebraic point of view, we consider non-empty polyhedra in standard form $\{x \in \mathbb{R}^n \mid Ax = b, x \geqslant 0\}$ with $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and m linearly independent rows, thus $m \leqslant n$ holds and we say A has *full rank* $\operatorname{rank}(A) = m$. The full rank assumption on A results in no loss of generality by Theorem 2.2, showing that if there were less than m linearly independent rows describing a non-empty polyhedron, the linearly dependent rows can be discarded and m can be reduced.

Theorem 2.2 (Full rank assumption). *Let $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geqslant 0\}$ be a non-empty polyhedron with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Suppose that $\operatorname{rank}(A) = k < m$, and that the rows $a_{i_1}, \ldots, a_{i_k}$ are linearly independent, then P equals the polyhedron given by $\{x \in \mathbb{R}^n \mid a_{i_1}^T x = b_{i_1}, \ldots, a_{i_k}^T x = b_{i_k}, x \geqslant 0\}$.*

Consider an m-dimensional set of indices $B \subseteq \{1, 2, \ldots, n\}$, so that the columns $A^{(j)}$ for $j \in B$ are linearly independent. We call B a *basis*[4], and build the invertible $m \times m$ matrix A_B consisting of the columns of A with indices in B; likewise we can constrain a vector b to these indices and write b_B. With these definitions we can compute basic solutions by Theorem 2.3.

Theorem 2.3 (Computation of basic feasible solutions). *Consider a polyhedron in standard form $\{x \in \mathbb{R}^n \mid Ax = b, x \geqslant 0\}$ with $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and an m-dimensional basis $B \subseteq \{1, 2, \ldots, n\}$. A vector $x \in \mathbb{R}^n$ is a basic solution*

[4]We also use the term *basis* for the corresponding set of variables x_j with $j \in B$.

if and only if $x_i = [A_B^{-1} b_B]_i$ *for* $i \in B$ *and* $x_i = 0$ *for* $i \notin B$*. Moreover, a basic solution* x *is a basic feasible solution if it satisfies the non-negativity constraints, too.*

A polyhedron P contains an infinite *line* if there exist a vector $x \in P$ and a non-zero vector $d \in \mathbb{R}^n$, so that every vector $x + \lambda d \in P$ for all scalars λ. Therefore, non-empty polytopes as well as non-empty polyhedra in standard form do not contain lines, because they are bounded in every component, respectively their feasible set is contained in the positive orthant $\{x \mid x \geqslant 0\}$. The existence of extreme points is achieved by Theorem 2.4 and with Theorem 2.1 we achieve the existence of basic feasible solutions.

Theorem 2.4 (Existence of extreme points). *Consider a non-empty polyhedron* $P = \{x \in \mathbb{R}^n \mid Ax \geqslant b\}$ *with* $A \in \mathbb{R}^{m \times n}$ *and* $b \in \mathbb{R}^m$*, then the following are equivalent:* P *has at least one extreme point;* P *does not contain a line; and there exist* n *linearly independent constraints.*

Now we consider that we are also given a linear objective function $c^T x$ with $c, x \in \mathbb{R}^n$. The optimality of extreme points is achieved by Theorem 2.5.

Theorem 2.5 (Optimality of extreme points). *Consider an LP of minimizing* $c^T x$ *over a polyhedron* $P \subseteq \mathbb{R}^n$ *and suppose that* P *has at least one extreme point. Then either the optimal cost is equal to* $-\infty$*, or there exists an extreme point which is optimal.*

In the case that the feasible set has no extreme points, the existence of an optimal solution is given by Corollary 2.6 by transforming the given LP into standard form and applying Theorem 2.4.

Corollary 2.6 (Optimal solution). *Consider the LP of minimizing* $c^T x$ *over a non-empty polyhedron. Then either the optimal cost is equal to* $-\infty$*, or there exists an optimal solution.*

2.4.4 Duality

Consider an LP in standard form $\min\{c^T x \mid Ax = b, x \geqslant 0\}$ that we call *primal* problem, and assume that an optimal solution $x^\star$ exists. We obtain a related problem by replacing the constraint $Ax = b$ by a penalty $y^T(b - Ax)$ in the objective function

$$g(y) = \min\{c^T x + y^T(b - Ax) \mid x \geqslant 0\},$$

Table 2.1: Conversion between primal and dual LPs.

The table shows the conversion between primal minimization LPs and dual maximization LPs with corresponding constraints and variables. If we minimize an LP with equality constraints and non-negative variables, we obtain a dual maximization LP with free variables and less-than-or-equal inequalities.

Primal LP	Minimize	Maximize	Dual LP
Constraints	$\geqslant b$	$\geqslant 0$	Variables
	$\leqslant b$	$\leqslant 0$	
	$= b$	free	
Variables	$\geqslant 0$	$\leqslant c$	Constraints
	$\leqslant 0$	$\geqslant c$	
	free	$= c$	

where g is a function of the price vector y of the same dimension as b. With the optimal solution $x^\star$ of the primal LP we obtain

$$g(y) = \min\{c^T x + y^T(b - Ax) \mid x \geqslant 0\} \leqslant c^T x^\star + y^T(b - Ax^\star) = c^T x^\star,$$

thus each y leads to a lower bound $g(y)$ for the optimal cost $c^T x^\star$, and the search for the tightest possible lower bound of this type is called the *dual* problem $\max g(y)$. With the definition of g we obtain that

$$g(y) = y^T b + \min\{(c^T - y^T A)x \mid x \geqslant 0\} = y^T b + \begin{cases} 0, & y^T A \leqslant c^T \\ -\infty, & \text{otherwise} \end{cases},$$

and as we are maximizing $g(y)$ we only get useful information from these lower bounds in the non-infinity case.

In summary, if we consider the standard form for the primal LP, we have the following equivalent pair of primal and dual problems:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geqslant 0 \end{aligned} \qquad\qquad \begin{aligned} \max \quad & y^T b \\ \text{s.t.} \quad & y^T A \leqslant c^T \end{aligned}$$

Starting with the canonical form (2.2) for the primal LP, we achieve the following equivalent pair of primal and dual problems by similar calculations:

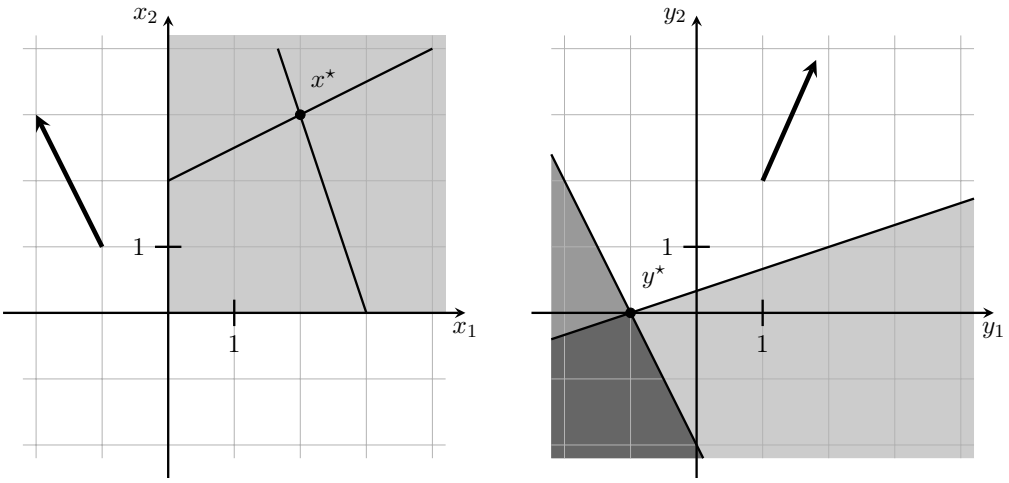


Figure 2.3: Example of a primal and dual LP.

The primal LP is given in standard form by $\min\{x_1 - 2x_2 \mid -x_1 + 2x_2 = 4, 3x_1 + x_2 = 9, x_1 \geqslant 0, x_2 \geqslant 0\}$, the dual by $\max\{4y_1 + 9y_2 \mid -y_1 + 3y_2 \leqslant 1, 2y_1 + y_2 \leqslant -2\}$. The optimal primal solution is $x^\star = (x_1, x_2) = (2, 3)$, the optimal dual solution is $y^\star = (y_1, y_2) = (-1, 0)$, both objective values are -4. The arrows indicate the directions of an increasing objective value.

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax \geqslant b \end{array} \qquad \begin{array}{ll} \max & y^T b \\ \text{s.t.} & y^T A = c^T \\ & y \geqslant 0 \end{array}$$

The dual problem is built by introducing a variable for every constraint (other than the non-negativity constraints); and a constraint for every variable of the primal problem, see Table 2.1 for an overview of the conversion between primal and dual LPs, and Figure 2.3 for a geometric example.

Dualizing an LP twice leads to Theorem 2.7.

Theorem 2.7 (Dual of dual is primal). *If we convert the dual problem into an equivalent minimization problem and form its dual, we obtain a problem equivalent to the original primal problem.*

The given motivation to search for lower bounds to the primal objective function with the help of the dual problem is called weak duality, see Theorem 2.8.

Theorem 2.8 (Weak duality). *Consider the standard form for the primal LP* $\min\{c^T x \mid Ax = b, x \geqslant 0\}$. *If x is a feasible solution to the primal problem and y is a feasible solution to the dual problem, then* $y^T b \leqslant c^T x$.

The central theorem on linear programming duality is the strong duality Theorem 2.9.

Theorem 2.9 (Strong duality). *If an LP has an optimal solution, its dual has an optimal solution, and furthermore, the respective optimal values are equal.*

The weak duality Theorem 2.8 implies that if one problem is unbounded its dual must be infeasible; and the strong duality Theorem 2.9 implies that if one problem has an optimal solution, so does its dual. Consequently, there are four possible situations out of nine combinations that can occur for primal and dual problems as shown in Corollary 2.10.

Corollary 2.10 (Possible primal and dual situations). *Given a primal LP and its dual, only the following situations are possible:*

1. *The objective values of the primal and dual are finite and equal.*
2. *The primal is unbounded and the dual is infeasible.*
3. *The dual is unbounded and the primal is infeasible.*
4. *The primal and dual are infeasible.*

The complementary slackness conditions of Theorem 2.11 show an important relation between the primal and dual optimal solution.

Theorem 2.11 (Complementary slackness). *Let x and y be feasible solutions to the primal and dual problem, respectively. Then they are optimal solutions for the two respective problems if and only if*

$$y^T(Ax - b) = 0,$$
$$(c^T - y^T A)x = 0.$$

By the first complementary slackness condition we see that if an inequality constraint of the form $a_i^T x \geqslant b_i$ is not active for a primal feasible solution x, then the corresponding dual variable y_i must be zero. We transformed this type of inequality constraints by introducing surplus variables s_i to obtain equality constraints of the form $a_i^T x - s_i = b_i$ and $s_i \geqslant 0$. In standard form every feasible solution must satisfy the first condition, so either the surplus variable or the dual variable is zero.

2.4.5 Degeneracy

With our definition of basic solutions at hand, we need n linearly independent active constraints, but there is a possibility that more than the necessary ones

are active, leading to degeneracy.

A basic solution $x \in \mathbb{R}^n$ is said to be *degenerate* if more than n of the constraints are active at x. If we additionally consider a polyhedron $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geqslant 0\}$ in standard form with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, then a basic solution $x \in \mathbb{R}^n$ is a *degenerate basic solution* if more than $n - m$ of the components of x are zero.

In our setting of standard form polyhedra, a degenerate basic feasible solution remains degenerate under every standard form representation of the original polyhedron.

2.4.6 Simplex Method

Overview There are numerous classes of algorithms to solve linear programs, several with high theoretical importance and several with good performances in practical applications.

We focus on the *simplex method*, first described around 1948 by Dantzig [23–25], because our column generation approach, see Chapter 4, is a direct modification of the search routine as described later. Further developments of Dantzig [26] lead to the *revised simplex method* with higher computational efficiency in 1953. In 1954 both Lemke [46] and Beale [8] developed the *dual simplex method* that benefits from the powerful duality theory.

In theory, the simplex method has exponential complexity, but well-implemented simplex methods show up to be the best LP solvers in practical applications. From that point of view, it was a breakthrough when in 1979 Khachiyan [42] introduced the *ellipsoid method* with weakly polynomial complexity, although this approach performs poorly for most practical uses. In 1984 Karmarkar [40] also introduced a weakly polynomial time algorithm, called *interior-points method*. This approach can compete against the simplex methods depending on the structure of the LP. The question whether LPs can be solved by a strongly polynomial time algorithm is still open.

Algebraic Derivation Throughout this section we consider $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c, x \in \mathbb{R}^n$ for an LP in standard form

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geqslant 0 \end{aligned}$$

and denote the corresponding feasible set by $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geqslant 0\}$.

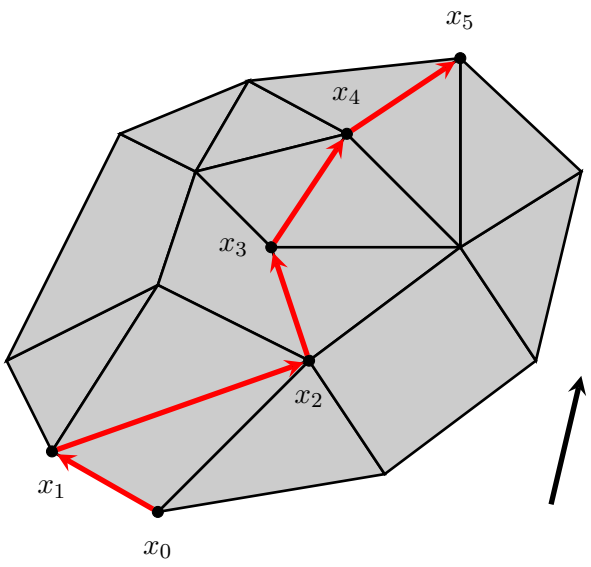


Figure 2.4: Simplex method.
The simplex method starts from a basic feasible solution x_0 and optimizes the objective value by moving along the edges of the polyhedron, shown as the red path $x_0, x_1, x_2, \ldots$, until reaching x_5 where no further improvement is possible. The black arrow indicates the direction of optimization.

Since we are optimizing a convex function $c^T x$ over a convex set P, local optimality implies global optimality, thus a local search routine for points in the neighborhood of a basic feasible solution that reduces the objective value can be applied.

Starting from a point x in the polyhedron P, a vector $d \in \mathbb{R}^n$ is a *feasible direction* at x, if there is a scalar $\lambda > 0$ for which $x + \lambda d \in P$. Assuming that $x \in P$ is a basic feasible solution, we obtain the existence of a m-dimensional basis B from Theorem 2.3, and especially, the formula

$$x_B = A_B^{-1} b_B \text{ and } x_i = 0 \; \forall \; i \notin B.$$

We select a non-basic variable x_j that is initially zero and try to increase the value to $\lambda > 0$ by moving in direction $d \in \mathbb{R}^n$ with $d_j = 1$, and $d_i = 0$ for every non-basic index i other than j, so we reach $x_B + \lambda d_B$, where d_B is the constrained vector d to basis B. Since we want to move from feasible to feasible solutions, we require $A(x + \lambda d) = b$ and know that $Ax = b$, thus we obtain $0 = Ad = A_B d_B + A^{(j)}$ which leads to a formula for the *j-th basic direction*

$$d_j = 1, d_B = -A_B^{-1} A^{(j)}, \text{ and } d_i = 0 \; \forall \; i \notin B, i \neq j.$$

The non-basic variable x_j is increased, all other non-basic variables stay at zero,

hence we only have to check the basic variables x_B. If x is a non-degenerate basic feasible solution, i.e., $x_B > 0$, we obtain that $x_B + \lambda d_B \geqslant 0$ and this is also feasible if λ is sufficiently small, thus d is a feasible direction. If x is degenerate, d is not always a feasible direction, because it is possible that a basic variable is zero and the corresponding component is negative, so following d would violate the non-negativity of $x + \lambda d$.

We define the *reduced cost* $\overline{c}_j$ of variable x_j as the change in the objective function when moving in the j-th basic direction d of unit length given by

$$\overline{c}_j = c^T d = c_B^T d_B + c_j = c_j - c_B^T A_B^{-1} A^{(j)},$$

thus the reduced cost of a basic variable is zero. If we move in this direction, we say that x_j *enters the basis*.

Theorem 2.12 (Reduced costs). *Consider a basic feasible solution x associated with a basis matrix A_B, and let $\overline{c}$ be the corresponding vector of reduced costs. If $\overline{c} \geqslant 0$, x is optimal. If x is optimal and non-degenerate, then $\overline{c} \geqslant 0$.*

The maximal length λ^* of the feasible direction d is determined by the constraints $x + \lambda^* d \in P$, so in the non-degenerate case we obtain

$$\lambda^* = \min_{i|d_i<0} \left(-\frac{x_i}{d_i} \right) = \min_{i \in B|d_i<0} \left(-\frac{x_i}{d_i} \right),$$

whereas in the degenerate case the constraint is never violated by setting $\lambda^* = \infty$. We obtain a new basis and a new associated basic feasible solution in the non-degenerate case by Theorem 2.13.

Theorem 2.13 (Changing the basis). *Consider a basis B, the j-th basic direction d, and let ℓ be the index where the minimum of $\lambda^* = \min_{i \in B|d_i<0} \left(-\frac{x_i}{d_i} \right)$ is attained. Then the index set $\overline{B} = B \setminus \{\ell\} \cup \{j\}$ is a basis, the columns of $A_{\overline{B}}$ are linearly independent, and the vector $\overline{x} = x + \lambda^* d$ is a basic feasible solution associated with $\overline{B}$.*

The Algorithm The previous results merge into Algorithm 2.1 that gives a structural overview of a single simplex iteration, whereas the correctness of the simplex method is stated in Theorem 2.14.

Theorem 2.14 (Correctness of the simplex method). *Assume that $P \neq \emptyset$ and that every basic feasible solution is non-degenerate, then the simplex method terminates after a finite number of iterations. At termination, we either have an optimal basis B and an associated basic feasible solution x which is optimal, or we have found a vector d satisfying $Ad = 0$, $d \geqslant 0$, and $c^T d < 0$ which means the optimal cost equals $-\infty$.*

1 $(\overline{B}, \overline{x})$ `SimplexIteration(`B`,`x`)`
  Data: m-dimensional basis B, associated basic feasible solution x
  Result: m-dimensional basis $\overline{B}$, associated basic feasible solution $\overline{x}$ with reduced cost
2 compute reduced cost $\overline{c}_j = c_j - c_B^T A_B^{-1} A^{(j)}$ for all $j \notin B$
3 **if** $\overline{c} \geqslant 0$ **then** x is optimal **terminate**
4 **else**
5 choose some $j \notin B$ with $\overline{c}_j < 0$ and compute $u = -d_B = A_B^{-1} A^{(j)}$
6 **if** $u \leqslant 0$ **then** $\lambda^* = \infty$ and the optimal cost $= -\infty$ **terminate**
7 **else**
8 determine $\lambda^* = \min_{i \in B | u_i > 0} \frac{x_i}{u_i}$ and choose ℓ, so that $\lambda^* = \frac{x_\ell}{u_\ell}$
9 set $\overline{x}$ with $\overline{x}_j = \lambda^*$ and $\overline{x}_i = x_i - \lambda^* u_i$ for all $i \in B \setminus \{\ell\}$
10 **return** $(B \setminus \{\ell\} \cup \{j\}, \overline{x})$

Algorithm 2.1: Iteration of the simplex method.

The iterations terminate with the optimal solution x in line 3 if all reduced costs are non-negative, otherwise, the LP is unbounded, see line 6; in the default case, the variable x_j enters and x_ℓ leaves the basis, see line 7 and following. The decisions of choosing j in line 5 and ℓ in line 8 are driven by various pivoting rules.

Pivoting Rules The decision of choosing an entering variable x_j in Algorithm 2.1 is called *pivot selection* and is driven by various *pivoting rules* that we will shortly summarize, keeping in mind that for choosing an exiting variable x_ℓ a similar discussion applies.

The simplest strategy is *Bland's rule* [13] that chooses the smallest index $j \notin B$ with $\overline{c}_j < 0$ to enter the basis, and respectively the smallest index ℓ to exit, so that we can abort the computation of further reduced costs, but it can increase the number of iterations. *Dantzig's rule* [27] chooses an index $j \notin B$ with $\overline{c}_j < 0$ and $\overline{c}_j \leqslant \overline{c}_i$ for all $i \notin B$, i.e., the direction of decreasing costs at the fastest rate, however, the actual cost decrease depends on how far we can move in this direction. A rule taking into account the distance traveled along the search edge is called *steepest edge rule*, but it leads to higher complexity, see Goldfarb and Reid [34], and Forrest and Goldfarb [32]. An efficient approximation of the steepest edge rule was developed earlier by Harris [37] and is called *Devex rule*.

Degeneracy In the case of degeneracy, there are two possible settings. First, we start at a degenerate basic feasible solution x and λ^* can be equal to zero, in which case the new basic feasible solution $\overline{x}$ is the same as x. This occurs if for some basic variable $x_\ell = 0$ and $d_\ell < 0$ hold, but we can still compute a new basis

$\overline{B}$ and Theorem 2.13 remains valid. Second, even if $\lambda^* > 0$ holds, it can happen that more than one of the original basic variables become zero if we move in the direction of d, and since only one variable will exit the basis, the new basic solution is degenerate.

Cycling Taking the previous paragraph into account, it can happen that we stay at the same basic feasible degenerate solution in spite of changing the basis. If we do not find a new cost-reducing direction, we can fall back to the initial basis in an infinite loop; this is called *cycling* and has to be avoided. Even if Bland's rule behaves poorly in practical applications, it does guarantee that cycling never occurs and the simplex method is guaranteed to terminate after a finite number of iterations. In most implementations the simplex method uses hybrid pivoting rules in different iterations to prevent cycling and gain the best performance.

Implementations The computation of the vectors $A_B^{-1}A^{(j)}$ in lines 2 and 5 of Algorithm 2.1 plays a key role in the efficiency of different implementations of the simplex method. A comparison of the methods is shown in Table 2.2.

In a *naive implementation* we compute the so-called *simplex multipliers* $p^T = c^T A_B^{-1}$ by solving $p^T A_B = c^T$. To compute the reduced cost we use $\overline{c}_j = c_j - c_B^T A_B^{-1} A^{(j)} = c_j - p^T A^{(j)}$, and to determine the vector u we solve $A_B u = A^{(j)}$. If we have to compute the reduced cost for all n variables, this leads to a total computational effort per iteration of $O(m^3 + nm)$.

The *revised simplex implementation* provides the inverse matrix A_B^{-1} at the beginning of each iteration to compute $c_B^T A_B^{-1}$ and $A_B^{-1}A^{(j)}$ by favorable matrix-vector products. Therefore, it is necessary to update A_B^{-1} after a change in the basis only with a few operations. This is achieved by a sequence of m *elementary row operations*, i.e., adding a constant of one row to the same or another row, each of them takes $O(m)$ operations, see Bertsimas and Tsitsiklis [10] for details. In total this leads to a computational effort of $O(m^2 + nm)$.

In the *full tableau implementation* we maintain and update a special $(m+1)\times(n+1)$ matrix, called *simplex tableau*. The simplex tableau first extends the matrix $A_B^{-1}A$ with a zeroth column taking $x_B = A_B^{-1}b$, and then with a zeroth row taking the current negative objective value $-c_B^T x_B$ and the reduced cost vector $\overline{c}^T = c^T - c_B^T A_B^{-1} A$:

$-c_B^T x_B$	$\overline{c}^T = c^T - c_B^T A_B^{-1} A$
$x_B = A_B^{-1} b$	$A_B^{-1} A$

Table 2.2: Comparison of different simplex implementations.

The time and memory requirements of the naive, revised and full tableau implementation for one iteration of the simplex method with $m \leqslant n$. In the worst-case the revised simplex cannot be slower than the full tableau implementation, but when considering a pivoting rule that only evaluates one reduced cost at a time until a variable with negative reduced cost is found, the revised simplex can be much faster. The advantage of the full tableau implementation in less memory demand is obvious.

Implementation	Naive	Revised simplex	Full tableau
Memory	$O(m^2)$	$O(m^2)$	$O(nm)$
Worst-case time	$O(m^3 + nm)$	$O(nm)$	$O(nm)$
Best-case time	$O(m^3)$	$O(m^2)$	$O(nm)$

Updating the tableau can also be achieved by elementary row operations in the size of the tableau, i.e., in $O(nm)$, see Bertsimas and Tsitsiklis [10] for details.

The two-phase Simplex Method In the first phase we provide an initial basic feasible solution to be able to continue the second phase with the simplex iterations of Algorithm 2.1.

We are still considering an LP in standard form. By multiplying some constraints with -1 we attain that $b \geqslant 0$, then we introduce auxiliary variables $y_1, y_2, \ldots, y_m$ and state the auxiliary LP

$$\begin{aligned} \min \quad & y_1 + y_2 + \ldots + y_m \\ \text{s.t.} \quad & Ax + y = b \\ & x \geqslant 0 \\ & y \geqslant 0 \end{aligned}$$

with the basic feasible solution $x = 0$ and $y = b$. If the optimal solution in the auxiliary problem is non-zero, we conclude that the original problem is infeasible[5], otherwise, we obtain a zero cost solution with $y = 0$ and x is a feasible solution to the original problem, but possibly with some artificial variables in the basis.

If we cannot simply eliminate the artificial variables and respectively the corresponding columns from the basis matrix, we have to *drive the artificial variables out of the basis* to obtain a basis consisting only of the original variables. Therefore, we assume that x is a basic feasible solution to the auxiliary problem, but

[5] If there exists a feasible solution x' to the original problem, then x' and $y = 0$ lead to a zero cost solution to the auxiliary problem.

only $k < m$ basis variables belong to the final basis of the original problem. With the result of Theorem 2.2, we can assume without loss of generality that matrix A has full rank, i.e., $\text{rank}(A) = m$, so we can choose $m - k$ additional columns to obtain a new basis consisting of m linearly independent columns of A exclusively, thus all non-basic components of x are at zero level and x is a basic feasible solution associated with the new basis as well.

The basis and corresponding basic feasible solution of the first phase is passed to the second phase and Algorithm 2.1 is applicable.

The Dual Simplex Method The *dual simplex method* is very similar to the preceding (primal) simplex method, thus we only give a brief motivation by Theorem 2.15.

Theorem 2.15 (Karush-Kuhn-Tucker optimality conditions). *Given an LP in standard form, the vector x is an optimal solution if and only if there exist vectors y and r so that:*

1. $Ax = b$ *and* $x \geqslant 0$ *(primal feasibility)*
2. $y^T A + r^T = c^T$ *and* $r \geqslant 0$ *(dual feasibility)*
3. $r^T x = 0$ *(complementary slackness)*

In Theorem 2.15, the vector r can be seen in a primal and dual fashion. In the dual problem it is simply the slack from the constraints $y^T A \leqslant c^T$. In the primal problem it is referred to as the vector of reduced costs, previously denoted by $\overline{c}$. We remark that the second complementary slackness condition, see Theorem 2.11, provides $y_B^T = c_B^T A_B^{-1}$, so the reduced costs of non-basic variables can be simplified to $\overline{c_j} = c_j - y_B^T A^{(j)}$ with the help of the dual feasibility condition of Theorem 2.15.

The previously described (primal) simplex method moves along basic feasible solutions satisfying both the primal feasibility and the complementary slackness condition in every iteration until dual feasibility is obtained at the optimal solution. The dual simplex method ensures both the dual feasibility and complementary slackness condition in every iteration, and obtains the primal feasibility only at the optimal solution.

Computational Efficiency The computational effort at each simplex iteration, as discussed above, and the number of iterations determine the computational efficiency of the simplex method. In theory the number of extreme points of a polyhedron can increase exponentially with the number of variables n and constraints m, but the observation for practical applications is that the simplex methods typically takes only $O(m)$ iterations.

The number of iterations depends on the chosen pivoting rule, but for most common pivoting rules examples where the simplex methods visits every extreme point already exist, see Klee and Minty [43] for the first known example for Dantzig's pivoting rule. Nevertheless, it is still an open question if for every pivoting rule there are examples where the simplex method takes an exponential number of iterations.

2.4.7 Dantzig-Wolfe Decomposition

In 1960 Dantzig and Wolfe [28] published decomposition principles that lead to a decomposition algorithm to handle large-scaled LPs. Consider an LP of the form

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \in X, \end{aligned} \tag{2.4}$$

with numerous variables $x \in X$ given by the bounded polyhedron $X = \{x \geqslant 0 \mid Dx = d\}$. Let x_i, $i \in I$, be the extreme points of X, then every point $x \in X$ can be represented by a convex combination of the extreme points by

$$x = \sum_{i \in I} \lambda_i x_i, \quad \sum_{i \in I} \lambda_i = 1, \quad \lambda_i \geqslant 0 \qquad \forall\, i \in I. \tag{2.5}$$

Replacing x in (2.4) by its convex combination (2.5) provides the *master problem*

$$\begin{aligned} \min \quad & \sum_{i \in I} \lambda_i c^T x_i \\ \text{s.t.} \quad & \sum_{i \in I} \lambda_i A x_i = b \\ & \sum_{i \in I} \lambda_i = 1 \\ & \lambda_i \geqslant 0 \qquad \forall\, i \in I. \end{aligned} \tag{2.6}$$

If we assume a basic solution of (2.6) exists, we obtain a vector of free dual variables q in the size of the initial constraints and a free dual variable r of the convex combination constraint. The reduced cost of the new optimization variables λ_i, $i \in I$ in (2.6) are then given by

$$\overline{c}_i = c^T x_i - q^T A x_i - r,$$

as introduced subsequent to Theorem 2.15. Instead of computing the reduced cost $\overline{c}_i$ for every variable λ_i, $i \in I$, the key idea is to determine the variable with the most negative reduced cost by the *subproblem*

$$\begin{aligned} \min \quad & (c^T - q^T A)x - r \\ \text{s.t.} \quad & x \in X. \end{aligned} \tag{2.7}$$

Solving the subproblem (2.7) with the simplex method, on the one hand can terminate with a negative objective value. Thus it provides us with an extreme point x_i, $i \in I$ that can be added to the master problem (2.6) by generating a new column with the coefficients Ax_i in the initial constraints and 1 in the convex combination constraint. On the other hand, if the objective value is non-negative, then every extreme point has non-negative reduced cost, thus the solution of the master problem is optimal, since no variable will enter the basis anymore.

Obviously, this decomposition algorithm is also applicable if there are more independent polyhedra $X^{(1)}, \ldots, X^{(k)}$, so that their variables are only linked in a number of *linking constraints* that remain in the master problem. The computational advantages are that the subproblems can be solved independently, usually they are easier to solve preferably with a combinatorial approach, and in the best case there are several identical subproblems.

2.5 Integer Linear Programming

We continue with discrete optimization, where we optimize a linear objective function over a set of variables that can take both rational and discrete values, so that all linear constraints are fulfilled. Therefore, a vector y of non-negative integers with an appropriate matrix and cost vector is added to the LP formulation (2.3).

2.5.1 Introduction

To obtain an *integer linear program (ILP)* in standard form, we consider $A \in \mathbb{Q}^{m \times n}$, $B \in \mathbb{Q}^{m \times k}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $d \in \mathbb{Q}^k$, $x \in \mathbb{Q}^n$, $y \in \mathbb{N}^k$ and

$$\begin{aligned} z_{ILP} = \min \quad & c^T x + d^T y \\ \text{s.t.} \quad & Ax + By = b \\ & x, y \geqslant 0 \\ & y \text{ integer}. \end{aligned} \tag{2.8}$$

To emphasize the structure with real and integer variables, (2.8) is also called *mixed integer linear program (MILP)*, but we adhere to ILP. The special case where y only takes binary values $y \in \mathbb{B}$ is called *binary integer linear program (BILP)* or *zero/one integer linear program*.

The decision variants of BILP and ILP neglect the objective function, thus they

only search for a feasible assignment for the optimization variables.

Theorem 2.16 (Computational complexity of ILP and BILP). *The decision variants of BILP and ILP are $\mathcal{NP}$-complete. The optimization variants of BILP and ILP are $\mathcal{NP}$-hard.*

The proofs of Theorem 2.16 are given by Schrijver [53, Chapter 18] and go back to Cook [20] and Karp [41], see Section 2.2 for an overview of the complexity classes.

Since solving ILPs is inherently hard, we show three classes of algorithms to manage ILPs, but focus on *exact algorithms* that terminate with the optimal solution but probably consume exponential time. Furthermore, there are *approximation algorithms* that compute a suboptimal solution in polynomial time together with a bound on suboptimality, and *heuristic algorithms* that provide a suboptimal solution in polynomial time without a guarantee of its quality.

Several approaches use the *LP relaxation* of an ILP, where the integrality constraints are relaxed, thus the relaxation of (2.8) is the polynomial feasible LP

$$\begin{aligned} z_{LP} = \min \quad & c^T x + d^T y \\ \text{s.t.} \quad & Ax + By = b \\ & x, y \geqslant 0. \end{aligned} \tag{2.9}$$

In the case of BILP, we introduce additional constraints of the form $0 \leqslant x \leqslant 1$ for every binary variable x. If an optimal solution to the relaxation (2.9) is integral, then it is also an optimal solution to the ILP (2.8); if not, it is at least a lower bound to the objective value of the ILP (2.8) when minimizing, i.e.,

$$z_{LP} \leqslant z_{ILP}.$$

2.5.2 Exact Algorithms

The class of *exact algorithms* includes algorithms that terminate with the optimal solution, but have an exponential running time in the worst-case.

Cutting Plane Algorithms A generic *cutting plane algorithm* solves the LP relaxation and – if an optimal solution $x^\star$ is not integer – adds a linear inequality constraint to separate $x^\star$ from the feasible set of the LP relaxation. The so-called *cut* constraint has to be satisfied by all integer solutions, and especially, the fractional solution $x^\star$ has to violate the cut. The expanded LP is the new LP relaxation and the process starts from beginning, until an integral solution is found.

Probably the best cuts can be obtained by studying the given problem and develop specific constraints, but this takes a lot of insight. The most prominent general approach was introduced by Gomory [35] and is called *Gomory cuts*.

Consider a basic feasible solution $x^\star$ of the LP relaxation, with $x_i^\star$ fractional. Let B be the basis and N denotes the set of non-basic variables. Every feasible solution $x = (x_B, x_N)$ satisfies

$$A_B x_B + A_N x_N = b,$$

thus also

$$A_B^{-1} A_B x_B + A_B^{-1} A_N x_N = A_B^{-1} b,$$

where the i-th row reads as

$$x_i + \sum_{j \in N} [A_B^{-1} A^{(j)}]_i x_j = [A_B^{-1} b]_i.$$

Since $x_j \geqslant 0$, we can floor the coefficients and obtain the Gomory cut

$$x_i + \sum_{j \in N} \left\lfloor [A_B^{-1} A^{(j)}]_i \right\rfloor x_j \leqslant \left\lfloor [A_B^{-1} b]_i \right\rfloor,$$

that is valid for all integral solutions, but is violated by $x^\star$, since $x_i^\star = [A_B^{-1} b]_i$ is fractional by assumption, and therefore $x_i^\star > \left\lfloor [A_B^{-1} b]_i \right\rfloor$.

Branch and Bound The background of *branch and bound* algorithms is the divide and conquer design paradigm to explore the set of feasible integer solutions. In 1960 the method was first proposed by Land and Doig [44], then extended, implemented and tested by Dakin [22] in 1965. Instead of exploring the entire search space, branch and bound algorithms use bounds on the optimal objective value to skip subspaces that cannot improve the current optimal value.

Assume F to be the set of feasible solutions to an ILP of the form

$$\begin{aligned} z_{OPT} = \min \quad & c^T x \\ \text{s.t.} \quad & x \in F. \end{aligned}$$

A partition of $F = \bigcup_{i \in I} F_i$ into a finite collection of subsets F_i, $i \in I$ leads to separately solvable subproblems of the same structure

$$\begin{aligned} z_i = \min \quad & c^T x \\ \text{s.t.} \quad & x \in F_i \end{aligned}$$

and $z_{OPT} = \min\{z_i \mid i \in I\}$. Consequently, the subproblems can be partitioned

again leading to a tree of subproblems. We additionally assume that there is a tolerably efficient algorithm to compute lower bounds $lb(i) \leqslant z_i$ on the subproblems, e.g., by solving the LP relaxation, and when solving a subproblem to optimality we get an upper bound $ub \geqslant z_{OPT}$ to the initial problem.

A generic branch and bound algorithm selects a subproblem i out of the list of subproblems and computes $lb(i)$. If the problem is infeasible or $lb(i) \geqslant ub$, we delete this subproblem, otherwise, we either compute the optimal solution to the subproblem and obtain improved bounds, or we partition the subproblem into further subproblems that are enqueued in the list.

There are several strategies to obtain lower bounds, to decide whether to solve or to partition a subproblem, and especially which subproblem to consider first. In most practical applications it is beneficial to choose the subproblem with the lowest lower bound first, called *best first search* strategy, because these subproblems have to be computed anyway, whereas subproblems with higher lower bounds could be skipped later on because the upper bound decreases. Other extreme strategies are *breadth first search*, respectively *depth first search* that inspects neighboring subproblems in sequence, respectively subproblems along each branch.

A typical way to partition a subproblem when an optimal fractional solution $x^\star$ to the LP relaxation has been obtained, is by adding either of the constraints $x_i \leqslant \lfloor x_i^\star \rfloor$ or $x_i \geqslant \lfloor x_i^\star \rfloor$ with $x_i^\star$ fractional. As the extended new subproblems only differ by a single constraint, we can expect to find the optimal solution by a small number of iterations with the dual simplex method, because this is equivalent to introducing a new variable in the dual problem.

Duality Unlike linear programming, integer linear programming does not have a strong duality theory, but we can obtain tight bounds from weak duality. Consider the ILP

$$\begin{aligned} z_{ILP} = \min \quad & c^T x \\ \text{s.t.} \quad & Ax \geqslant b \\ & x \text{ integer} \end{aligned} \tag{2.10}$$

with integer entries and the feasible set $X = \{x \text{ integer}\}$. Introducing a dual variable for every constraint leads to a vector of *Lagrange multipliers* $p \geqslant 0$ in the size of vector b and for a fixed p we introduce the function

$$\begin{aligned} z(p) = \min \quad & c^T x + p^T(b - Ax) \\ \text{s.t.} \quad & x \in X. \end{aligned} \tag{2.11}$$

Furthermore, we can maximize the function $z(p)$ and denote this as the *La-*

grangian dual

$$\begin{aligned} z_D = \max \quad & z(p) \\ \text{s.t.} \quad & p \geqslant 0 \\ & x \in X. \end{aligned} \tag{2.12}$$

Obtaining tight lower bounds from this dualizing is called *Lagrangian relaxation*, see Theorem 2.17.

Theorem 2.17 (Weak duality). *If z_{ILP} is the optimal solution to* (2.10) *and $p \geqslant 0$, then $z(p) \leqslant z_{ILP}$. By Lagrangian relaxation, see* (2.11) *and* (2.12)*, we obtain the weak duality $z_D \leqslant z_{ILP}$.*

The relation between the objective values of the ILP, its Lagrangian dual, and its LP relaxation is examined in Bertsimas and Tsitsiklis [10] and stated in Theorem 2.18.

Theorem 2.18 (Ordering). *If z_{ILP} is the optimal solution to* (2.10)*, z_D is the optimal solution to the Lagrangian dual* (2.12)*, and z_{LP} is the optimal solution to the LP relaxation of* (2.10) *$z_{LP} = \{\min c^T x \mid Ax \geqslant b\}$, the following ordering holds:*

$$z_{LP} \leqslant z_D \leqslant z_{ILP}.$$

Branch and Cut In practical applications it can take numerous iterations of adding cuts, as well as improving the bounds on the variables, therefore, the cutting plane and branch and bound strategies are combined to an approach called *branch and cut*. When solving subproblems we use additional cuts to obtain better bounds on the variables and improve the running time.

Dynamic Programming *Dynamic Programming* is an exact method to solve complex optimization problems by breaking them down into smaller subproblems if two properties are given: overlapping subproblems and optimal substructure. *Overlapping subproblems* are given if a recursive algorithm can solve the same subproblems multiple times instead of generating different subproblems all the time, thus a solution has to be computed only once and can be reused if it is stored, i.e., *memoization*. *Optimal substructure* is given if an optimal solution can be determined efficiently by the optimal solutions of the subproblems, comparable to the strategy pursued in greedy algorithms.

Developing dynamic programming algorithms is crucially depending on the given problem, so we only give same basic guidelines. The choice of a feasible solution equals a sequence of single decisions and the cost of the feasible solution equals

the sum of the cost of the particular decisions. A sequence of decisions is saved in a state that summarizes the past decisions. Changing from one state to another is a transition and it takes the cost of the corresponding decision. The optimal solution is then provided by writing a recursive method to obtain the optimal objective value from the origin state to a destination state.

2.5.3 Approximation Algorithms

Approximation algorithms compute a suboptimal solution in polynomial time together with a bound on suboptimality. As the development of good approximation algorithms depends highly on the given problem, we only show what is meant by the bound on suboptimality.

An algorithm A is a *ρ-approximation algorithm* with $\rho > 1$ for a minimization problem with optimal value z_{OPT}, if A has a polynomial running time and returns a feasible solution z_A with $z_A \leqslant \rho \cdot z_{OPT}$ for every instance.

There are minimization problems where an $(1 + \varepsilon)$-approximation algorithm for every fixed $\varepsilon > 0$ exists, so that we can approximate the problem arbitrarily closely. These family of algorithms is called a *polynomial time approximation scheme (PTAS)*. It is still an open field of research which properties of integer programming problems allow an approximation, especially, an arbitrarily accurate one, and which make it unlikely to exist.

2.5.4 Heuristic Algorithms

Heuristic algorithms provide a suboptimal solution in polynomial time without a guarantee on their quality.

Most heuristic algorithms are based on inspecting the neighborhood of a feasible solution, therefore, it is important to define neighboring points. Given an optimization problem of the form

$$\begin{aligned} z_{OPT} = \min \quad & c(x) \\ \text{s.t.} \quad & x \in F, \end{aligned}$$

we start at some $x \in F$ and evaluate $c(x)$. Then we explore the neighborhood of x, i.e., some points $y \in F$, and repeat the process starting from a point y with $c(y) < c(x)$. If we cannot make any improvement, this *local search* terminates at a *local optimum*, i.e., a point that is at least as good as any of its neighbors.

In practical applications there are two contrary properties of local search: on the one hand in a big neighborhood there are less local optima and the search

probably provides better results at the expense of slower running times, and on the other hand in a small neighborhood there are fewer points to evaluate at and the search is faster. In the case of linear programming two vertices are neighbors if they are connected by an edge, thus the simplex method from Section 2.4.6 can be seen as a local search method, however, it terminates at a global optimum.

The generic method of *simulated annealing* tries to remedy the disadvantage that local search can get stuck in a local optimum, by allowing moves to a point y in the neighborhood with higher or equal costs, i.e., $c(y) \geqslant c(x)$. To guarantee further that the algorithm will terminate, the probability of doing such an adverse movement is constantly decreased by a *cooling function*, e.g., $\exp\left(-\frac{c(y)-c(x)}{T(t)}\right)$ with a positive function $T(t)$, called *temperature schedule* that is mostly physically motivated and tends to zero for $t \to \infty$.

For unstructured problems with initially unknown optimization approaches, simulated annealing can achieve good results after some tuning of the parameters in appropriate running times.

2.6 Real-Time Scheduling

The scheduling of real-time tasks and corresponding communication in this thesis is done for a simplified embedded system model related to systems in the automotive industry. We give a short insight into embedded systems and the design phase, then we focus on scheduling policies and analyses in real-time scenarios.

2.6.1 Real-Time Embedded Systems

Most definitions cover a wide range of what is meant by a real-time embedded system, see Burns and Wellings [17, 1.1] for some examples. From our viewpoint, a *real-time embedded system* is a combination of a hardware architecture, software and optionally external parts, to realize a specific functionality within hard timing restrictions.

The characteristics of such systems are numerous, so we focus on complex, safety-critical real-time systems in the automotive environment, where the reliability of the system is of high importance, and there are hard timing requirements that do not allow a delay for generated output due to safety reasons.

The *hardware architecture* is composed of various classes such as sensors, actuators, microprocessors, communication channels, and so on, but we only consider *electronical control units (ECUs)* that represent the processing units for computation which are clustered in *subsystems*, and a *global bus* for transmitting

information between ECUs of different subsystems.

The *software* to realize the specific functionality is given as a *task network*, consisting of several tasks and communication messages that have to be scheduled to ECUs, respectively buses, ensuring their real-time requirements and hardwares restrictions.

The following sections give insight to the particular components of real-time embedded systems that we work with throughout the thesis. A detailed formal definition is given in the problem formulation, see Sections 3.1 and 3.2.

2.6.1.1 Electronical Control Units (ECUs) and ECU types

In this thesis the expression *electronical control unit (ECU)* is used as a placeholder in the hardware architecture of an embedded system. A physical microprocessor is first assumed if an *ECU type* is assigned to an ECU. All properties of the microprocessor, e.g., connected I/O ports, the available memory or the computational speed, are given by the ECU type, and influence the executability of several tasks up to their execution times. Installing a new ECU type or upgrading an existing one during the design phase of an embedded system is accounted by the cost of the new ECU type.

2.6.1.2 Communication Buses

Multiple ECUs in an embedded system are connected via communication channels, such as simple wires or more complex structures for bidirectional communication like buses. ECUs inside a subsystem are connected by a *local bus* and subsystems are connected by a *global bus* in such a way that access is only given by a single *gateway ECU* in every subsystem. These buses operate using different communication protocols, e.g., *token area network (TAN)*, *controller area network (CAN)* or *FlexRay*.

From an abstract view the TAN protocol gives a token to linked units in a round-robin fashion and the unit holding the token can transmit a message, whereas the CAN protocol assigns priorities to messages for non-preemptive transmission. A FlexRay consists of two segments, the deterministic static segments resembling a *time-division multiple access (TDMA)* fashion, and the CAN bus-like dynamic segment. A *bus slot* is a time slice of the TDMA scheme where units can transmit messages.

In this thesis we assume that the local bus is sufficiently fast and of sufficient capacity to transmit every occurring message in time. Consequently, the message stream – from an ECU to the gateway ECU over the local bus, then over the global

bus to another gateway ECU, and finally to the destination ECU – is compressed to the transmission of the message over the global bus, whereby every ECU has direct access to the global bus.

2.6.1.3 Tasks

The definition of a task depends on the level of granularity we are looking at. In this thesis, a *task* is a sequence of computing instructions that is executed by an ECU as a basic unit. If a task is allocated to a specific ECU and started, there is no chance of reallocating, or restarting the task. If the task is executed, it has exclusive access to the ECUs components, only its execution can be interrupted for some time by tasks assigned to this ECU with higher priority, what is called *preemption*. On closer examination of preemption, problems can arise concerning blocking shared resources, costs of context switches, and exception handling that are out of scope of this thesis.

We consider a task model with *periodic tasks* that come with information about their *periodicity*, i.e., the time after which a task is initiated again, their *worst-case execution time (WCET)* depending on the ECU type, i.e., the worst-case time that a task has to use the ECU for computation, mostly their *deadline*, i.e., the maximal time limit for the computation, and their *memory consumption* also depending on the ECU type.

There are three scenarios that have to be separated, first, the case of *implicit deadlines*, i.e., if the deadline equals the period, second *explicit* or *constrained* deadlines, i.e., if the deadline is less than or equal to the period, and third *unknown deadlines* or *end-to-end deadlines*, i.e., if only a sequence of task and message executions is bounded by an overall deadline. Besides periodic tasks, there are *sporadic* or *aperiodic* tasks that do not arrive at a constant rate, but rather come sporadic, and are not part of the discussion.

More details about several approaches for automatically deriving tasks early in the design phase of an embedded system from its abstract model to improve the development are described in Büker et al. [15, 16]. Another challenging problem in this scenario is the safe over-estimation of WCETs and memory consumption, see Wilhelm et al. [61] for an overview.

2.6.1.4 Messages

Tasks can trigger *messages* to send data to other tasks, such as computational results or instructions. If the destination tasks are located on the same ECU, the data flow can be handled directly via shared resources, but if the destination task

is distributed to another ECU or located in another subsystem, the message has to be sent over the corresponding buses.

We consider a message model with *periodic messages* that come with information about their *periodicity*, i.e., the time after which the message is initiated again, the unique *source task*, several *destination tasks*, their *worst-case transmission time (WCTT)*, i.e., the time that they need for transmission depending on the size of the bus slots, and their *deadline*, i.e., the maximal time limit for transmission.

2.6.1.5 Task Networks

A *task network* is the combination of tasks and messages showing the data flow between these entities. In this context, the timing requirements can be formulated to sequences of tasks and messages. We consider tasks and messages without *release jitter*, i.e., without variations in their timing behaviour that leads to unpredictability.

2.6.2 Task Scheduling Policies and Analyses

If a set of concurrent tasks is assigned to be executed on a single ECU, they can be ordered in several ways, especially if they can be interrupted by some tasks. A *scheduling policy*, *scheme* or *algorithm* is a set of rules for ordering the execution of the tasks at any time, represented by a *schedule*, while ensuring the hard real-time constraints in the worst-case behaviour. Predicting the *worst-case response time (WCRT)*, i.e., the maximal time between the initiation and completion of a task execution in every periodic instance, is important because for the in-time completion of a task it is necessary that its WCRT is less than or equal to its deadline.

A schedule is called *feasible* if all considered tasks can be executed subject to their constraints, and a set of tasks is called *schedulable* if there exists at least one scheduling policy that produces a feasible schedule.

Following Buttazzo [18], scheduling algorithms can be classified in four main classes:

Preemptive vs. non-preemptive Preemptive algorithms allow that a task that is currently executed can be interrupted at any time for assigning another waiting task. In non-preemptive algorithms the running task is executed without interruptions until its completion.

Static vs. dynamic In static algorithms the scheduling decisions are made based on fixed parameters before task activation, whereas dynamic scheduling de-

cisions are based on dynamic parameters that can change during execution.

Offline vs. online Offline algorithms develop the schedule before task activation contrary to online algorithms that decide at execution time about new arriving and waiting tasks.

Optimal vs. heuristic An optimal algorithm provides a solution with the minimal/maximal value of a predefined objective function under all solutions of the search space, e.g., a cost function on the used hardware or a penalty measurement on missed deadlines. A heuristic algorithm cannot guarantee finding an optimal solution.

In this thesis we develop and examine a preemptive, static, offline, optimal algorithm that provides a cost minimal assignment of tasks, ECU types and bus allocation. Therefore, we have to check the schedulability before tasks' execution to obtain reliable assertions about worst-case scenarios.

A *critical instant* of a task t is defined to be the point in time where the request for a task will lead to its largest response time. Liu and Layland [49] showed that a critical instant for preemptive and periodic tasks occurs when all higher priority tasks are also requested, see Theorem 2.19.

Theorem 2.19 (Critical instant). *A critical instant for any preemptive, periodic task occurs when the task is requested simultaneously with all higher priority tasks.*

We assume that tasks are released *synchronously* at time $\delta = 0$, i.e., they are in phase and do not have an offset in their initiation. Therefore, the critical instant will be at time $\delta = 0$ and if we achieve schedulability for the critical instant, we can ensure it at every time.

Since we deal with periodic tasks, the schedule will repeat itself for the first time after a time interval called *hyperperiod* H that is obtained by the lowest common multiple of all task periods, i.e., $H = \text{lcm}(P_t \mid t \in T)$, because of the synchronous releases.

In the following sections we introduce several approaches in the analysis of scheduling policies and conclude with established task scheduling policies for different scenarios as shown in Table 2.3.

For several scheduling policies and analyses, tasks, respectively messages, receive pairwise disjoint priorities by the function $\text{pr} : \mathcal{T} \longrightarrow \mathbb{N}$, respectively $\text{pr} : \mathcal{M} \longrightarrow \mathbb{N}$, to determine the order of execution in the case of preemption. The set of higher priority tasks is denoted by

$$\text{hp}(t) = \{t' \in \mathcal{T} \mid \text{pr}(t') > \text{pr}(t)\},$$

Table 2.3: Scheduling policies in different scenarios for periodic tasks.
The shown scheduling policies are earliest deadline first (EDF), rate monotonic scheduling (RMS), and deadline monotonic scheduling (DMS) classified into dynamic priority scheduling (DPS) and fixed priority scheduling (FPS) algorithms. The scenarios cover implicit deadlines (deadline = period) and explicit/constrained deadlines (deadline $\leqslant$ period). EDF is also optimal in both scenarios for aperiodic tasks.

	DPS / FPS	Implicit deadlines	Explicit deadlines
EDF	DPS	optimal [38, 49]	optimal [7]
RMS	FPS	optimal under all FPS [30, 49]	not applicable
DMS	FPS	optimal equal to RMS	optimal under all FPS [47]

and tasks with a priority higher than or equal to t are given by the set

$$\text{hep}(t) = \{t' \in \mathcal{T} \mid \text{pr}(t') \geqslant \text{pr}(t)\}.$$

The denotation applies straightforward to lower priority tasks $\text{lp}(t)$ and lower and equal priority tasks $\text{lep}(t)$, as well as to messages.

2.6.2.1 Processor Utilization Analysis

The *processor utilization* is the fraction of processor time spent in executing a set of tasks $T \subseteq \mathcal{T}$, i.e.,

$$U(T) = \sum_{t \in T} \frac{C(t)}{P(t)}, \tag{2.13}$$

with the computational running time $C(t)$ and period $P(t)$ of tasks t.

Let H be the hyperperiod, then $\frac{H}{P(t)}C(t)$ is the total computational time requested by task t in the hyperperiod, and summation over all tasks yields

$$\sum_{t \in T} \frac{H}{P(t)} \cdot C(t) = H \cdot \sum_{t \in T} \frac{C(t)}{P(t)} = H \cdot U(T),$$

which is greater than H if the total demand exceeds the available processor time for $U(T) > 1$, see Corollary 2.20.

Corollary 2.20. *A task set $T \subseteq \mathcal{T}$ is not schedulable if* $U(T) = \sum_{t \in T} \frac{C(t)}{P(t)} > 1$.

2.6.2.2 Processor Demand Analysis

We still consider that tasks are activated synchronously at time 0, so the *processor demand* in an interval $[0, \delta]$ for $\delta \geqslant 0$ of a subset $T \subseteq \mathcal{T}$ is the sum of computational running time of instances that contribute processor demand in this interval, i.e., formally expressed by the *demand bound function*

$$\mathrm{dbf}(\delta, T) = \sum_{t \in T} \left\lfloor \frac{\delta + P(t) - D(t)}{P(t)} \right\rfloor \cdot C(t), \tag{2.14}$$

where task instances with a relative deadline $D(t)$ greater than δ are not considered.

2.6.2.3 Response Time Analysis

As we are considering preemption, the response time is the sum of the computational time $C(t)$ and the *interference* $I(t)$ by higher priority tasks.

A schedulability test for a set of tasks $T \subseteq \mathcal{T}$ based on response time analysis is to check

$$R(t) = C(t) + I(t) \leqslant D(t) \qquad \forall\, t \in T, \tag{2.15}$$

see Table 2.4 and Figure 2.5 for an example with three tasks on a single ECU.

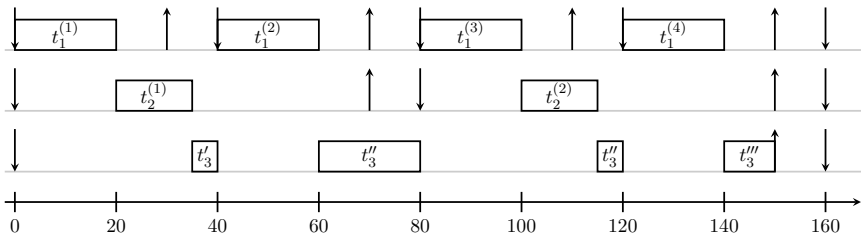


Figure 2.5: Response time analysis on a single ECU.
The schedule of the execution of the three exemplary tasks from Table 2.4 is shown. The down arrows indicate the activation times, the up arrows the corresponding deadlines. We see that the execution of task t_3 is splitted into four parts.

Table 2.4: Example for response time analysis.

The three exemplary tasks are ordered highest priority first. We present the WCETs, the deadlines, the periods, and the WCRTs by two computations. A sufficient, but not necessary computation of the interferences is given by (2.17), whereas (2.19) gives the correct WCRTs. We see that task t_3 is not schedulable with the not necessary formula because the WCRT of 165 exceeds the deadline of 150.

Task	WCET $C(t_i)$	Deadline $D(t_i)$	Period $P(t_i)$	WCRT $R(t_i)$ by (2.15) & (2.17)	(2.19)
t_1	20	30	40	20	20
t_2	15	70	80	35	35
t_3	40	150	160	165	150

2.6.2.4 Workload Analysis

The workload analysis approach of Lehoczky et al. [45] is very similar to the response time analysis, but differs in handling interferences.

The *workload* $W_t(\delta)$ is the cumulative computation time of task $t \in T$ and all higher priority tasks $t' \in \mathrm{hp}(t)$ in the interval $(0, \delta]$ requested to a processor, i.e.,

$$W_t(\delta) = C(t) + \sum_{t' \in \mathrm{hp}(t)} \left\lceil \frac{\delta}{P(t')} \right\rceil \cdot C(t'). \tag{2.16}$$

If there is a time $\delta \leqslant D(t)$ where the demand is less than or equal to the supply of the processor, i.e., $W_t(\delta) \leqslant \delta$, then task t can be scheduled.

An example of the workload analysis with four tasks is given in Figure 5.1.

2.6.2.5 Earliest Deadline First (EDF)

Earliest deadline first (EDF) is a dynamic scheduling policy that executes tasks according to their *absolute deadlines*, i.e., the time from the first initiation of task t over all periods of further instances up to the current deadline, formally $(j-1)P(t) + D(t)$ for the j-th instance of task t, $j \in \mathbb{N}$.

Processor Utilization Analysis There is a simple schedulability test by Liu and Layland [49] based on the fact that a processor can be fully stretched by tasks scheduled by EDF, see Theorem 2.21.

Theorem 2.21 (Processor utilization for EDF with implicit deadlines). *A set of preemptive, periodic tasks $T \subseteq \mathcal{T}$ with implicit deadlines is schedulable on a single processor by EDF if and only if $U(T) = \sum_{t \in T} \frac{C(t)}{P(t)} \leqslant 1$.*

Processor Demand Analysis With the help of the processor demand analysis we can formulate a schedulability test by Baruah et al. [7] for explicit deadlines, and improve on the number of test intervals from Buttazzo [18, Sec. 4.6.2], see Theorem 2.22.

Theorem 2.22 (Processor demand for EDF with explicit deadlines). *A set of preemptive, periodic tasks $T \subseteq \mathcal{T}$ with explicit deadlines is schedulable on a single processor by EDF if and only if $\mathrm{dbf}(k, T) \leqslant k$ for all $k > 0$. It is sufficient to test for*

$$k \in K = \{D(t) + zP(t) \mid z \in \mathbb{N}_0, D(t) + zP(t) \leqslant \min\{H, \max\{D_{\max}, K^*\}\}\}$$

with hyperperiod H, maximal deadline $D_{\max}$, and $K^* = \frac{\sum_{t \in T}(P(t) - D(t)) \cdot \frac{C(t)}{P(t)}}{1 - \sum_{t \in T} \frac{C(t)}{P(t)}}$.

2.6.2.6 Fixed Priority Scheduling (FPS)

A *fixed priority scheduling (FPS)* algorithm determines fixed priorities $\mathrm{pr}(t)$ for all tasks $t \in \mathcal{T}$, therefore the order of execution is fixed in advance.

Processor Utilization Analysis The original paper of Liu and Layland [49] and its revision from Devillers and Goossens [30] give a sufficient, but not necessary schedulability test for preemptive, periodic tasks with implicit deadlines by the utilization of the processor, see Theorem 2.23 and Figure 2.6.

Theorem 2.23 (Processor utilization for FPS). *A set of preemptive, periodic tasks $T \subseteq \mathcal{T}$ with implicit deadlines is schedulable on a single processor by a FPS policy, if $U(T) \leqslant U^*$ with $U^* = |T| \cdot \left(2^{\frac{1}{|T|}} - 1\right)$ tending to be $\ln(2) \simeq 0.69$ for large sets T.*

Response Time Analysis The interference I_t of the execution of task t can be roughly estimated if we summarize the computational time of all tasks t' with higher priority than t, i.e., $t' \in \mathrm{hp}(t)$, by

$$I(t) = \sum_{t' \in \mathrm{hp}(t)} \left\lceil \frac{D(t)}{P(t')} \right\rceil \cdot C(t'), \tag{2.17}$$

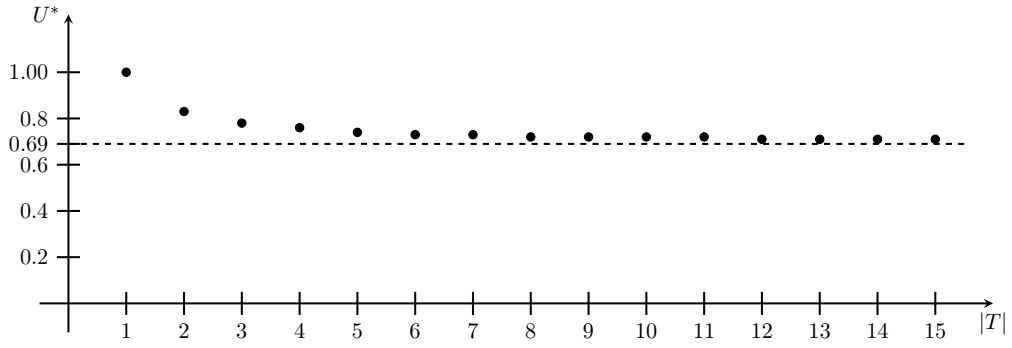


Figure 2.6: Processor utilization by Liu and Layland.
A sufficient, but not necessary schedulability test for preemptive, periodic tasks with implicit deadlines is given by the utilization of the processor. The maximal utilization for larger sets of tasks tends to $\ln(2) \simeq 0.69$.

but this only provides a sufficient, but not necessary schedulability test. A higher priority task t' can be finished before the deadline $D(t)$, therefore, the exact interference is determined by the response time $R(t)$. Audsley et al. [5] showed that a necessary and sufficient test is obtained with an exact computation of the interference leading to a recursive formula for the response time, see Theorem 2.24.

Theorem 2.24 (Response Time Analysis for FPS). *A set of preemptive, periodic tasks $T \subseteq \mathcal{T}$ with explicit deadlines is schedulable on a single processor by a FPS policy if and only if $R(t) = C(t) + I(t) \leqslant D(t)$ for all $t \in T$ with*

$$I(t) = \sum_{t' \in \mathrm{hp}(t)} \left\lceil \frac{R(t)}{P(t')} \right\rceil \cdot C(t'). \tag{2.18}$$

Based on (2.18) we obtain a recursive formula for the response time, i.e., the smallest value $R(t)$ satisfying

$$R(t) = C(t) + \sum_{t' \in \mathrm{hp}(t)} \left\lceil \frac{R(t)}{P(t')} \right\rceil \cdot C(t'). \tag{2.19}$$

We remind that only a finite number of points need to be checked in Theorem 2.24, because the interference can only increase when there is a release of a higher priority task.

An example with three tasks on a single ECU is given in Table 2.4 and Figure 2.5.

Workload Analysis Lehoczky et al. [45] developed a necessary and sufficient schedulability test for FPS algorithms with the help of the workload analysis

(2.16), see Theorem 2.25, and Bini and Buttazzo [12] reduced the number of points in time for checking the constraints, see Theorem 2.26.

Theorem 2.25 (Workload demand for FPS). *A set of preemptive, periodic tasks $T \subseteq \mathcal{T}$ with explicit deadlines is schedulable on a single processor by a FPS algorithm if and only if there exists $\delta \in (0, D(t)]$ with*

$$W_t(\delta) = C(t) + \sum_{t' \in \mathrm{hp}(t)} \left\lceil \frac{\delta}{P(t')} \right\rceil \cdot C(t') \leqslant \delta \qquad \forall\ t \in T.$$

Theorem 2.26 (Testing set for workload demand). *Consider a set of preemptive, periodic tasks $T \subseteq \mathcal{T}$ with explicit deadlines that is enumerated in decreasing priority, i.e., $T = \{t_1, t_2, \ldots, t_n\}$ with $i < j \Leftrightarrow \mathrm{pr}(t_i) > \mathrm{pr}(t_j)$. Then T is schedulable on a single processor by a FPS algorithm if and only if there exists $\delta \in \Delta_i$ with*

$$W_i(\delta) = C(t_i) + \sum_{t_j \in \mathrm{hp}(t_i)} \left\lceil \frac{\delta}{P(t_j)} \right\rceil \cdot C(t_j) \leqslant \delta$$

for all $i : t_i \in T$, where Δ_i is the testing set *for task t_i defined by $\Delta_i = F_{i-1}(D_i)$ with*

$$F_i(\delta) = F_{i-1}\left(\left\lfloor \frac{\delta}{P(t_i)} \right\rfloor P(t_i) \right) \cup F_{i-1}(\delta) \text{ and } F_0(\delta) = \{\delta\}.$$

See Figure 5.1 for an example on the workload analysis.

2.6.2.7 Rate Monotonic Scheduling (RMS)

The *rate monotonic scheduling (RMS)* policy is a FPS policy for tasks with implicit deadlines, so tasks gain a priority reciprocal to their periods, ties are broken arbitrary, i.e.,

$$\mathrm{pr}(t) > \mathrm{pr}(t') \Leftarrow P(t) < P(t'). \tag{2.20}$$

Consequently, all theorems from Section 2.6.2.6 apply.

2.6.2.8 Deadline Monotonic Scheduling (DMS)

The *deadline monotonic scheduling (DMS)* policy is also a FPS policy, but covers tasks with explicit deadlines, so tasks gain a priority reciprocal to their deadlines, ties are broken arbitrary, i.e.,

$$\mathrm{pr}(t) > \mathrm{pr}(t') \Leftarrow D(t) < D(t'). \tag{2.21}$$

It is obvious that DMS equals RMS if the tasks' deadline is equal to its period, and

especially, all theorems from Section 2.6.2.6 are valid. Leung and Whitehead [47] showed that DMS is optimal under all FPS policies, see Theorem 2.27.

Theorem 2.27 (DMS is optimal under all FPS). *If a task set is schedulable by some FPS policy, it is also schedulable by DMS.*

2.6.3 Message Scheduling Policies and Analyses

We assume that internal communication of an ECU and communication of ECUs inside a single subsystem is of minor interest and can always be ensured by sufficiently fast internal buses. Communication between distributed ECUs is managed by a global bus, for which several bus types exist and specific analyses are necessary. As the buses can appear in several settings, we talk about *nodes* representing tasks, ECUs or subsystems as well, depending on the level of granularity.

We assume that every message is sent by its own, i.e., we do not consider packing signals to messages, and every message fits into one frame/slot to be transmitted in a single step. We also assume that the *release jitter* equals 0, i.e., we neglect the time needed for enqueuing a message ready for transmission.

In *event-triggered* systems, message events are the consequences of occurring events, e.g., the completion of a task, and therefore are of low deterministic manner. However, in *time-triggered* systems, message events are initiated periodically at predetermined points in time, and therefore have high deterministic character and good predictability.

In the following sections we introduce the TAN bus, the CAN bus as candidate for event-triggered systems, and the time division multiple access protocol with their schedulability analyses.

2.6.3.1 Token Ring Local Area Network (TAN)

If the nodes are arranged in a *token ring local area network (TAN)*, transmitting data is handled in a round robin fashion. Therefore, a token is moving from node to node, and the node holding the token can transmit data solely.

Asynchronous messages are scheduled after synchronous messages with the strategy that if a node has nothing to transmit, the token is passed earlier, allowing subsequent nodes to transmit more data, however, the opposite case can occur, i.e., a node holding the token too long leading to a late token for the next node. In this scenario the time for one round of the token can be almost two-times larger than aspired, see Johnson [39] and Burns et al. [62].

Since we are not considering asynchronous messages or any technical overheads,

we can compute the *token rotation time (TRT)*, i.e., the time for one round of the token, by summarizing the WCTT of the messages on the bus

$$TRT = \sum_{\substack{m \in \mathcal{M}: \\ m \text{ on the bus}}} C(m). \tag{2.22}$$

When a node has a message to transmit, but the token was just passed to the next node that is now transferring a message, the worst-case delay appears. The node has to wait for the token coming around in its next round, thus it has to wait for the TRT. A necessary condition to the TRT, respectively the deadlines of the transmitted messages is given by

$$D(m) \geqslant TRT. \tag{2.23}$$

Accordingly, the WCRT $R(m)$ of a transmitted message m is obtained by

$$R(m) \geqslant TRT + C(m). \tag{2.24}$$

2.6.3.2 Controller Area Network (CAN)

In this section we examine the schedulability analysis of priority-based, event-triggered buses, such as a *controller area network (CAN)* that is very similar to the analysis of ECUs. The order of transmission is given by message priorities, but the actual transmission of a message cannot be preempted by others, as CAN works non-preemptively. For this reason there are two effects that will delay the transmission of a message: on the one hand, the transmission of higher priority messages delays the transmission, and on the other hand, the blocking by a message with lower priority that is actually being transmitted.

Response Time Analysis Starting in 1994 Tindell, Burns, Wellings and Hansson [56, 58, 59] presented the first approaches to schedulability analysis of CAN buses that were later revised by Davis et al. [29]. The computation of the WCRT $R(m)$ for a message m, i.e., the longest time from initiation of a message to receiving it by its destination tasks, is given by

$$R(m) = C(m) + W(m), \tag{2.25}$$

where $C(m)$ denotes its WCTT, and $W(m)$ denotes the *busy period*, i.e., the time a message is waiting for the transmission of other messages. In detail we obtain

$$W(m) = B(m) + I(m) = \max_{m' \in \text{lep}(m)} \{0, C(m')\} + \sum_{m' \in \text{hp}(m)} \left\lceil \frac{R(m)}{P(m')} \right\rceil \cdot C(m'), \quad (2.26)$$

where the first term is referred to as the *blocking time* $B(m)$ by lower or equal priority messages, i.e., $m' \in \text{lep}(m)$, and the second term measures the *interference time* $I(m)$ by higher priority messages.

A necessary and sufficient schedulability test is given by

$$R(m) = C(m) + B(m) + I(m) \leqslant D(m), \quad (2.27)$$

where the blocking time and interference time are correctly determined due to the condition that the response time is less than or equal to the deadline, see Davis et al. [29].

2.6.3.3 Time Division Multiple Access (TDMA)

The *time division multiple access (TDMA)* protocol describes time-triggered communication between processors over a bus that is organized in repetitive *TDMA rounds*. These rounds are of equal length and divided into *time slots*, where each time slot is assigned to at most one processor that has exclusive access to the bus during this time slot. A processor may occupy more than one slot. The multiple access allows that a time slot is used for transmitting different messages in each TDMA round in a repetitive fashion, thus the utilization is increased.

An example of a TDMA bus is the static segment of the *FlexRay* bus that is used in series-production vehicles. Influenced by the hardware manufacturers, there are $2^6 = 64$ rounds, and messages in a slot can be transmitted in several rounds. The offset from round zero is called *base cycle* and their periodicity is called *cycle repetition*, i.e., a power of two $2^0, \ldots 2^6$. The schedulability of the static segment is $\mathcal{NP}$-hard and can be formulated as a two-dimensional bin packing problem, see Lukasiewycz et al. [50].

To reduce the complexity in this thesis, we assume release jitter to be 0, and we focus on a single TDMA cycle with equally sized time slots and messages that fit exclusively in these slots, thus we ignore the multiple access option of a TDMA bus.

Chapter 3

Problem Definition

This chapter comes up with the formal definition of the distributed scheduling problem discussed in this thesis. Two scenarios are characterized and will be evaluated in what follows. For the definition of the problem we introduce a formalism to describe the hardware architecture, i.e., subsystems with ECUs of predefined ECU types connected by a global bus, and the task networks, i.e., tasks and messages between tasks.

We will neglect units throughout the thesis, because we consider all units to be consistent, i.e., times are measured in seconds and sizes of memory and messages are measured in bytes, so that non-basic terms like bandwidth in bytes/second translate naturally. A *partial function* f from A to B is denoted by $f : A \nrightarrow B$.

3.1 Hardware Architecture

The hardware architecture is given by subsystems that are connected with a global bus, see Figure 3.1. These subsystems can be provided with several ECUs of predefined ECU types that have to be assigned.

An ECU type represents a physical processing unit with all its properties, such as its hardware cost and available memory, see Definition 3.1.

Definition 3.1 (ECU type). *An* ECU type *et is an object from the finite collection of ECU types $\mathcal{ET}$ and given by*

cost(et) *defined by a function* cost : $\mathcal{ET} \longrightarrow \mathbb{N}$ *that specifies the cost of the hardware,*

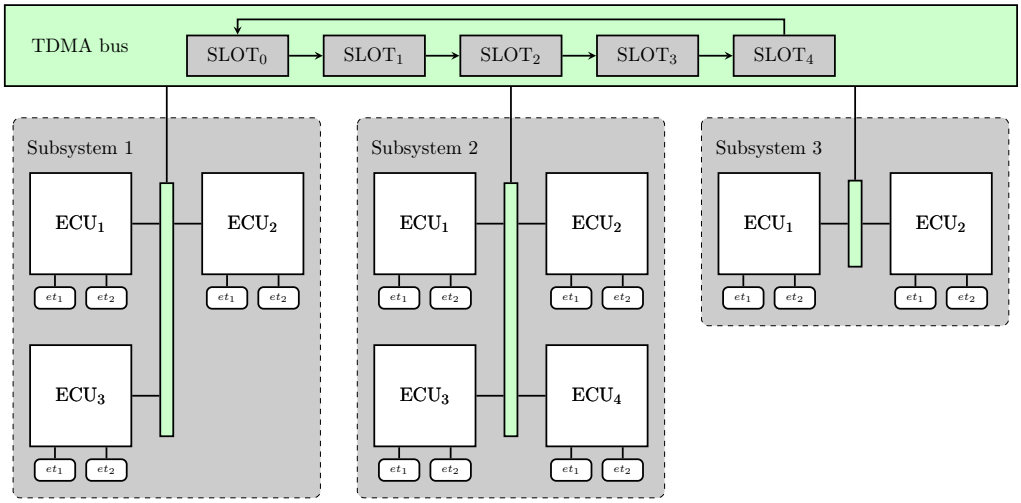


Figure 3.1: Hardware architecture with subsystems connected by a TDMA bus. In our hardware architecture each ECU has direct access to the global TDMA bus, whereas local message transmission is not taken into account. Each ECU can be of a specific ECU type or is not present.

mem(et) *defined by a function* mem : $\mathcal{ET} \longrightarrow \mathbb{N}$ *that specifies the size of available memory.*

A subsystem represents a cluster of computational processors and has constraints of the applicable ECU types due to space or construction issues, see Definition 3.2.

Definition 3.2 (Subsystem). *A* subsystem s *is an object from the finite collection of subsystems* $\mathcal{S}$ *and given by*

$\max_{\text{ECU}}(s)$ *a natural number* $\max_{\text{ECU}}(s) \in \mathbb{N}$ *defining the maximal applicable number of ECUs in this subsystem. We denote the set of applicable ECUs by* $\mathcal{E} = \{e_1, \ldots, e_{\max_{\text{ECU}}(s)}\}$.

Considering the communication, we come up with our model of a TDMA bus with a single TDMA round, see Definition 3.3.

Definition 3.3 (TDMA bus). *A* TDMA bus *is a finite, ordered collection of bus slots* $\mathcal{B}$ *and given by*

N *a natural number* $N \in \mathbb{N}$ *defining the number of slots* $\mathcal{B} = \{b_1, \ldots, b_N\}$ *where messages can be exclusively allocated to,*

size *a natural number size* $\in \mathbb{N}$ *defining the size of every slot* b_i *for* $i = 1, \ldots, N$.

As a consequence of Definition 3.3, the size of the TDMA round is given by

$$N \cdot size.$$

A lot of additional constraints are possible and they are mostly given by partial assignments, e.g., an a priori deployment of a task to a specific subsystem, or the condition that two tasks cannot be executed in the same subsystem due to safety issues. For a detailed examination of additional constraints see Section 3.4.

3.2 Task Network

The task network is given by tasks, messages and chains, the latter describe sequences of tasks and corresponding messages. We consider periodic tasks without release jitter and with explicit deadlines, i.e., the deadline is smaller than or equal to the period, see Definition 3.4.

Definition 3.4 (Task). *A* task t *is an object from the finite collection of tasks* $\mathcal{T}$ *and given by*

$C(t, et)$ *defined by a partial function* $C : \mathcal{T} \times \mathcal{ET} \nrightarrow \mathbb{N}$ *that specifies the WCET of task* t *on ECU type* et*, if the task is executable there,*

$D(t)$ *defined by a partial function* $D : \mathcal{T} \nrightarrow \mathbb{N}$ *that specifies the explicit deadline of task* t*, if it is known,*

$P(t)$ *defined by a function* $P : \mathcal{T} \longrightarrow \mathbb{N}$ *that specifies the period of task* t*,*

$\mathrm{mem}(t, et)$ *defined by a function* $\mathrm{mem} : \mathcal{T} \times \mathcal{ET} \nrightarrow \mathbb{N}$ *that specifies the size of required memory per ECU type.*

A new instance of task t *is initiated every period* $P(t)$*. The instance must be completely executed before the next period* $P(t)$*, and if an explicit deadline* $D(t) \leqslant P(t)$ *is given, then the task must already be completed at* $D(t)$ *the latest.*

If a task t has a WCET on a specific ECU type et greater than its deadline, it is not executable there and $C(t, et) = \perp$. If the deadline of task t is unknown, i.e., $D(t) = \perp$, its computation must only be finished before its period $P(t)$.

The communication between tasks is handled by messages in *multicast* fashion, i.e., one task transmits information to many others. A message has to be sent over the global bus, if at least one of the destination tasks is assigned to a different subsystem than its sender task, see Definition 3.5. In our model there are no gateway ECUs, thus every ECU has direct access to the global bus, and local communication between ECUs of the same subsystem is neglected.

Definition 3.5 (Message). *A multicast* message m *is an object from the finite collection of messages* $\mathcal{M}$ *and given by*

$C(m)$ *defined by a function* $C : \mathcal{M} \longrightarrow \mathbb{N}$ *that specifies the WCTT of message* m*,*

$D(m)$ *defined by a partial function* $D : \mathcal{M} \nrightarrow \mathbb{N}$ *that specifies the deadline of message* m *if it is known,*

$\text{send}(m)$ *defined by a function* $\text{send} : \mathcal{M} \longrightarrow \mathcal{T}$ *that specifies the sender task of message* m*,*

$\text{dest}(m)$ *defined by a function* $\text{dest} : \mathcal{M} \longrightarrow 2^{\mathcal{T}}$ *that specifies the non-empty set of destination tasks disjoint to the sender task* $\text{send}(m)$ *of message* m*, i.e.,* $\text{send}(m) \notin \text{dest}(m)$*.*

A message m *initiates a new instance every period of its sender task* $P(\text{send}(m))$*. If at least one of the destination tasks* $t' \in \text{dest}(m)$ *is not scheduled in the same subsystem as the sender task* $\text{send}(m)$*, then message* m *has to be transmitted over the global bus. The transmission must be completed before the period* $P(\text{send}(m))$*. If an explicit deadline* $D(m) \leqslant P(\text{send}(m))$ *is given, message* m *must already be transmitted before the deadline* $D(m)$*.*

The combined representation of tasks and messages is done in a task network, see Definition 3.6 and Figure 3.2.

Definition 3.6 (Task network). *A* task network $G = (\mathcal{T}, \mathcal{M})$ *is a bipartite, directed graph with nodes* $\mathcal{T} \dot{\cup} \mathcal{M}$ *that represents the tasks and messages. The edges are of the form* $(\text{send}(m), m)$ *and* (m, t) *for every* $t \in \text{dest}(m)$ *that represents a message* m *from its sender* $\text{send}(m)$ *to all of its destination tasks in* $\text{dest}(m)$*.*

The deadline of a task, respectively a message, can be unknown due to the fact that they are contained in a chain, see Definition 3.7 and Figure 3.2, and only an overall end-to-end deadline for the execution of the sequence is given.

Definition 3.7 (Chain). *A* chain c *is an object from the finite collection of chains* $\mathcal{C}$ *and given by*

$\text{seq}(c)$ *defined by an alternating path of tasks* $\mathcal{T}$ *and messages* $\mathcal{M}$ *in the task network* $G = (\mathcal{T}, \mathcal{M})$ *starting from a task* $t_1 \in \mathcal{T}$ *and ending at a task* $t_{k+1} \in \mathcal{T}$ *of the form* $\langle t_1, m_1, t_2, m_2, t_3, \ldots, t_k, m_k, t_{k+1} \rangle$*, with the properties* $\text{send}(m_i) = t_i$*,* $t_i \notin \text{dest}(m_i)$*, and* $t_{i+1} \in \text{dest}(m_i)$ *for* $i = 1, \ldots, k$*,*

$D(c)$ *defined by a function* $D : \mathcal{C} \longrightarrow \mathbb{N}$ *that specifies the end-to-end deadline of chain* c*.*

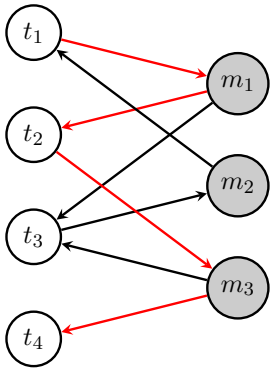


Figure 3.2: Task network with chain.

A task network is a bipartite, directed graph showing the correspondence of tasks and messages. A chain with the sequence $\langle t_1, m_1, t_2, m_3, t_4 \rangle$ is depicted in red.

A chain c with the above-named sequence $\mathrm{seq}(c)$ *with its root task t_1 initiates a new instance every period $P(t_1)$.*

3.3 Assignments

The variables of our optimization are the value assignments of two functions that determine the allocations of tasks and messages, see Definitions 3.8 and 3.9.

Definition 3.8 (Task assignment). *A (partial)* task assignment *is given by a (partial) function*

$$\tau : \mathcal{T} \nrightarrow \mathcal{S} \times \mathcal{E} \times \mathcal{ET}$$

that specifies in which subsystem, on which ECU of which ECU type a task is executed.

Definition 3.9 (Message assignment). *A (partial)* message assignment *is given by a (partial) function*

$$\mu : \mathcal{M} \nrightarrow 2^{\mathcal{B}}$$

that specifies which bus slots are used by a message.

3.4 Additional Constraints

Additional constraints can be categorized regarding their object of restriction, thus there are hardware, task, and message constraints, see Definitions 3.10 – 3.12.

Definition 3.10 (Additional hardware constraints). Additional hardware constraints *are given by*

$\min_{\text{ECUtype}}(s, et)$ *defined by a function* $\min_{\text{ECUtype}} : \mathcal{S} \times \mathcal{ET} \longrightarrow \mathbb{N}$ *that specifies the minimal number of applicable ECUs of every ECU type per subsystem,*

$\max_{\text{ECUtype}}(s, et)$ *defined by a function* $\max_{\text{ECUtype}} : \mathcal{S} \times \mathcal{ET} \longrightarrow \mathbb{N}$ *that specifies the maximal number of applicable ECUs of every ECU type per subsystem.*

Definition 3.11 (Additional task allocation constraints). Additional task allocation constraints *are given by*

$\text{pre}_{\text{task}}(t)$ *defined by a partial function* $\text{pre}_{\text{task}} : \mathcal{T} \nrightarrow \mathcal{S}$ *that specifies whether a task is predeployed to a subsystem,*

$\text{for}_{\text{task}}(t)$ *defined by a partial function* $\text{for}_{\text{task}} : \mathcal{T} \nrightarrow 2^{\mathcal{S}}$ *that specifies whether a task is forbidden to run on a subset of subsystems.*

Definition 3.12 (Additional message allocation constraints). Additional message allocation constraints *are given by*

$\text{pre}_{\text{msg}}(m)$ *defined by a partial function* $\text{pre}_{\text{msg}} : \mathcal{M} \nrightarrow 2^{\mathcal{B}}$ *that specifies whether a message is predeployed to some global bus slots,*

$\text{for}_{\text{msg}}(m)$ *defined by a partial function* $\text{pre}_{\text{msg}} : \mathcal{M} \nrightarrow 2^{\mathcal{B}}$ *that specifies whether a message is forbidden to be assigned to some global bus slots.*

We remark that a task cannot be predeployed to a subsystem and forbidden to run on the same subsystem simultaneously, similarly a message cannot be predeployed to a set of bus slots and simultaneously be forbidden for these slots, see Definition 3.15. The additional message allocation constraints that define requirements to bus slots are only considered for a global TDMA bus, see Section 5.2.3.

On one hand, there is the option to allocate two or more tasks to the same subsystem, and on the other hand to distribute them to different subsystems, see Definition 3.13.

Definition 3.13 (Additional task clustering constraints). Additional task clustering constraints *are given by*

$\text{cluster}_{\text{task}}(t)$ *defined by a function* $\text{cluster}_{\text{task}} : \mathcal{T} \longrightarrow 2^{\mathcal{T}}$ *that specifies a set of tasks that has to be allocated to the same subsystem as* t*,*

$\text{separate}_{\text{task}}(t)$ *defined by a function* $\text{separate}_{\text{task}} : \mathcal{T} \longrightarrow 2^{\mathcal{T}}$ *that specifies a set of tasks that has to be allocated to different subsystems than* t*.*

The function $\text{cluster}_{\text{task}}$ *can be seen as an* equivalence relation*, meaning that it is* reflexive*, i.e.,* $t \in \text{cluster}_{\text{task}}(t)$*,* symmetric*, i.e., if* $t' \in \text{cluster}_{\text{task}}(t)$ *then*

$t \in \text{cluster}_{\text{task}}(t')$, *and* transitive, *i.e., if* $t' \in \text{cluster}_{\text{task}}(t)$ *and* $t'' \in \text{cluster}_{\text{task}}(t')$ *then* $t'' \in \text{cluster}_{\text{task}}(t)$, *whereas the function* $\text{separate}_{\text{task}}$ *is non-reflexive, i.e.,* $t \notin \text{separate}_{\text{task}}(t)$, *and symmetric.*

3.5 Objective Function

The goal of optimization in our model is the minimization of the cost caused by the ECU types chosen for installing in the subsystems, see Definition 3.14.

Definition 3.14 (Objective function). *The* objective function *of optimization measures the cost of the selected ECU types and is consequently dependent on the task assignment* τ. *The cost of an assignment is given by*

$$\text{cost}(\tau) = \sum_{s \in \mathcal{S}} \sum_{e \in \mathcal{E}} \sum_{\substack{et \in \mathcal{ET}: \\ \exists t \in \mathcal{T}: \\ \tau(t) = (s,e,et)}} \text{cost}(et). \tag{3.1}$$

The cost of the hardware caused by the task assignment τ is obtained by (3.1), where we sum over all subsystems, ECUs and ECU types and only count the cost that arise, if there is an assignment of a task t to ECU e in subsystem s, where the chosen ECU type for e is et.

3.6 Feasibility

In this section we state what is meant by a valid instance, see Definition 3.15, a feasible solution, see Definition 3.16, and an optimal solution, see Definition 3.17.

Definition 3.15 (Valid instance). *An* instance I *is an object from the set of instances* $\mathcal{I}$ *and is given by a tuple*

$$I = (\mathcal{ET}, \mathcal{S}, \mathcal{B}, \mathcal{T}, \mathcal{M}, \mathcal{C}, \mathcal{F}),$$

where $\mathcal{F}$ *denotes the set of additional functions given by the additional constraints from Section 3.4, i.e.,* $\text{min}_{\text{ECUtype}}$, $\text{max}_{\text{ECUtype}}$, pre_{task}, for_{task}, pre_{msg}, for_{msg}, $\text{cluster}_{\text{task}}$, $\text{separate}_{\text{task}}$. *An instance is called* valid *if it satisfies the following consistency conditions:*

1. Consistency of a subsystem $s \in \mathcal{S}$ *is given by*

$$0 \leqslant \text{min}_{\text{ECUtype}}(s, et) \leqslant \text{max}_{\text{ECUtype}}(s, et) \leqslant \text{max}_{\text{ECU}}(s),$$

for every ECU type $et \in \mathcal{ET}$.

2. Consistency of a task $t \in \mathcal{T}$ *is given by*

$$0 \leqslant C(t, et) \leqslant D(t) \leqslant P(t),$$

for every ECU type et*, if* t *is executable on* et*, i.e.,* $C(e, et) \neq \perp$.

3. Consistency of the task allocation *for task* $t \in \mathcal{T}$ *is given by*

$$\text{pre}_{\text{task}}(t) \notin \text{for}_{\text{task}}(t).$$

4. Consistency of the message allocation *for message* $m \in \mathcal{M}$ *is given by*

$$\text{pre}_{\text{msg}}(m) \cap \text{for}_{\text{msg}}(m) = \emptyset.$$

5. Consistency of the task clustering/separation *for task* $t \in \mathcal{T}$ *is given by*

$$\text{cluster}_{\text{task}}(t) \cap \text{separate}_{\text{task}}(t) = \emptyset.$$

We search for assignments of tasks and messages as defined in Section 3.3 under the guarantee of schedulability for every ECU and the global bus, see Definition 3.16.

Definition 3.16 (Feasible solution). *A* solution *A is a tuple*

$$A = (\tau, \mu)$$

with a partial task assignment $\tau : \mathcal{T} \nrightarrow \mathcal{S} \times \mathcal{E} \times \mathcal{ET}$ *and a partial message assignment* $\mu : \mathcal{M} \nrightarrow 2^{\mathcal{B}}$*, as defined in Definitions 3.8 and 3.9.*

A solution is called partial *if at least one assignment is partial, i.e., at least one task or message is not assigned.*

A solution is called feasible *if the assignments are not partial and lead to schedulable allocations of all tasks and messages, i.e., every deadline is guaranteed to be met in every periodic cycle – including adequate memory resources and satisfiability of all additional constraints added into the model. Otherwise, the solution is called* infeasible.

Feasible solutions obtained by our algorithm or heuristics can be ordered by their objective value given in Section 3.5, leading to the definition of an optimal solution, see Definition 3.17. If the detailed assignments of a solution are of minor interest for the discussion and there is no risk of confusion, we often denote a solution by its objective value to simplify the reading.

Definition 3.17 (Optimal solution). *The value of a given feasible solution*

Table 3.1: Characterization of DSP variants.

The table shows both DSP variants with their restrictions examined in this thesis: the general formulation DSP-GF and the chain formulation DSP-CF. Deadlines for tasks and messages are not mandatory in DSP-CF if they are contained in the sequence of a chain with an end-to-end deadline. Additional hardware constraints as well as additional task or message allocation constraints are applicable in both scenarios.

	Task			Message	Chains
	Deadline	Memory	Cluster	Deadline	
DSP-GF	+	+	+	+	−
DSP-CF	(−)	+	+	(−)	+

$A = (\tau, \mu)$ is determined by its cost

$$\text{cost}(A) = \sum_{s \in \mathcal{S}} \sum_{e \in \mathcal{E}} \sum_{\substack{et \in \mathcal{ET}: \\ \exists t \in \mathcal{T}: \\ \tau(t) = (s,e,et)}} \text{cost}(et),$$

in relation to Definition 3.14. According to their cost, feasible solutions can be ordered and an optimal solution *is a feasible solution with minimal cost under all feasible solutions.*

3.7 Characterization of Problems

We distinguish between the *general formulation (DSP-GF)* without chains, and the *chain formulation (DSP-CF)* that considers chains and their given end-to-end deadlines, see Table 3.1.

The general formulation DSP-GF does not consider chains. From Theorem 2.27 we know that if a set of tasks is schedulable by a FPS policy it is also schedulable by the DMS policy. The advantage of this result is that the costly analyses for the WCRT computation are redundant in DSP-GF and we can achieve a dramatic reduction of running times, as we will see in Chapter 6.

In the scenario of the chain formulation DSP-CF with end-to-end deadlines the above argument is wrong. The crucial step is that the fulfillment of end-to-end deadlines leads to more stringent requirements on the WCRT of tasks than regular task deadlines do, see Section 5.3.1.

If a set of tasks is schedulable by DMS due to their deadlines, a chain can force

the tasks to swap their execution order since the later scheduled task needs to respond earlier – contrary to their priorities in the DMS policy. Therefore, the schedulability analyses need to take the WCRTs into account leading to more complex ILP formulations, see Sections 5.1.5.2 and 5.3.4, and consequently to larger running times, see Chapter 6.

Chapter 4

Column Generation Approach for the Distributed Scheduling Problem

In Section 4.1 we show the hardness of DSP introduced in Chapter 3. The derivation of our algorithm starts by stating DSP-GF as ILP with pattern variables that are advantageous in applying a column generation approach, see Section 4.2. Then we state the relaxation to an LP, see Section 4.3, and encapsulate it into a branch and bound framework, see Section 4.4, to still provide integral, optimal solutions. The details of the column generation approach are examined in Section 4.5, especially the partition into the master problem and the independent pricing problems for tasks and global communication. Important strategies of the implementation are described in Section 4.6.

When introducing our optimal algorithm to solve DSP, we restrict to the general formulation DSP-GF with implicit or explicit deadlines, i.e., we do not handle chains. The discussion of DSP-CF follows in Section 5.3, because it is handled by a combined formulation and modifications of the master and pricing problems.

An overview of our solving process is given in Figure 4.1.

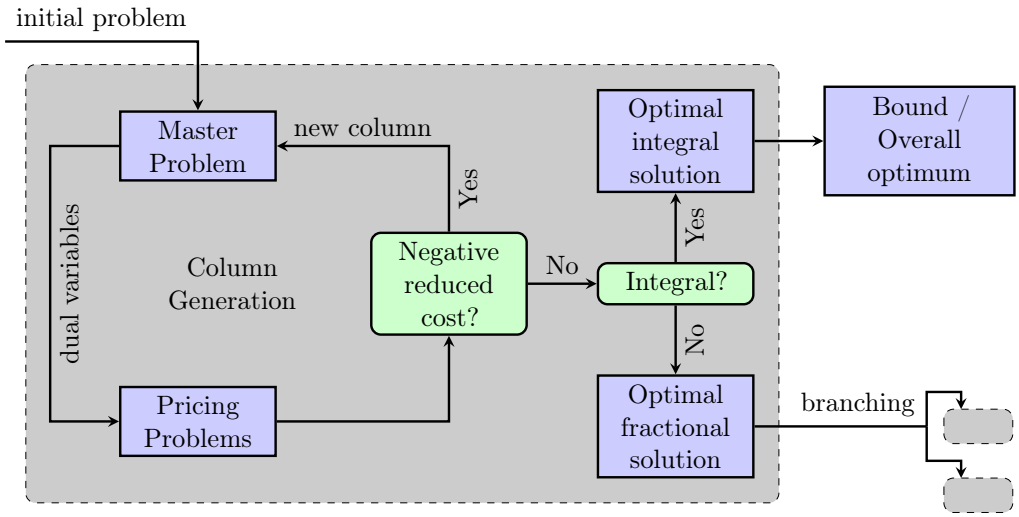


Figure 4.1: Algorithm for solving DSP.
The column generation with its master and pricing problems is encapsulated into a branch and bound framework to provide an optimal solution to DSP. If there are no more variables with negative reduced cost, an optimal solution to the master problem is detected. If the solution is integral, the branch is processed, otherwise subsequent branching nodes are generated with the help of the fractional solution.

4.1 Computational Complexity

In Chapter 3 we defined two variants of DSP: the general formulation DSP-GF without chains and the chain formulation DSP-CF considering chains.

We give a proof that both modifications are $\mathcal{NP}$-hard, see Theorem 4.1, by a reduction from the optimization variant of the strong $\mathcal{NP}$-hard *bin packing* problem[1], see Coffman et al. [19] for a survey of approximation algorithms on this topic.

An instance of the optimization variant of bin packing is given by a set of items $\{a_j\}_{j=1,\dots,n}$, each of size size$(a_j) \in \mathbb{N}$, and the capacity of the bins $B \in \mathbb{N}$. The question is to minimize the number of bins so that every item is assigned to a bin, regarding the size of the items and capacity of the bins. Formally we can define the problem by the BILP

[1]The complexity class *strong $\mathcal{NP}$-hard* is defined similarly to the (weak) $\mathcal{NP}$-hard class, however in the unary model. See Garey and Johnson [33] for some strong $\mathcal{NP}$-complete results.

$$
\begin{aligned}
\min \quad & \sum_{k=1}^{n} y_k \\
\text{s.t.} \quad & \sum_{j=1}^{n} \text{size}(a_j) \cdot x_{jk} \leqslant B \cdot y_k && \forall\ k = 1, \ldots, n \\
& \sum_{k=1}^{n} x_{jk} = 1 && \forall\ j = 1, \ldots, n \\
& x_{jk}, y_k \in \mathbb{B} && \forall\ j, k = 1, \ldots, n.
\end{aligned}
\tag{4.1}
$$

Theorem 4.1 (DSP is $\mathcal{NP}$-hard). *The distributed scheduling problem DSP in both its variants DSP-GF and DSP-CF is $\mathcal{NP}$-hard.*

The proof works with the same reduction for both variants, since the complexity of schedulability on a single ECU is given in every formulation.

Proof. We reduce a bin packing instance $(\{a_j, \text{size}(a_j)\}_{j=1,\ldots,n}, B)$ to an instance $I = (\mathcal{ET}, \mathcal{S}, \mathcal{B}, \mathcal{T}, \mathcal{M}, \mathcal{C}, \mathcal{F})$ of DSP with one ECU type $\mathcal{ET} = \{et\}$ of unit cost, i.e., $\text{cost}(et) = 1$, and memory capacity $\text{mem}(et) = B$, and only one subsystem $\mathcal{S} = \{s\}$ with at most n ECUs, i.e., $\max_{\text{ECU}}(s) = n$. Furthermore, we ignore the global bus, messages, chains and additional constraints by setting $\mathcal{B} = \mathcal{M} = \mathcal{C} = \mathcal{F} = \emptyset$.

The idea of the reduction is to identify each item a_j with a task t_j for $j = 1, \ldots, n$. The WCET and memory of task t_j are $C(t_j, et) = \text{mem}(t_j, et) = \text{size}(a_j)$, the deadline and period are $D(t_j) = P(t_j) = B$.

The number of used bins is equal to the cost of used ECUs. A set of items $J \subseteq \{1, \ldots, n\}$ can be packed into a bin if and only if

$$\sum_{j \in J} \text{size}(a_j) \leqslant B \Leftrightarrow \sum_{j \in J} C(t_j) \leqslant B = D(t_j) = P(t_j)$$

for every $j \in \{1, \ldots, n\}$. The tasks in J can be scheduled in arbitrary order, because their deadline is large enough, so that every task finishes its execution before C.

Therefore, the bin packing instance is solvable if and only if the DSP instance is solvable, thus DSP is $\mathcal{NP}$-hard. □

4.2 ILP formulation

In this section we develop the ILP formulation for DSP-GF with the help of *patterns*.

A straightforward formulation would contain variables to express the allocation of tasks to ECUs that are grouped to subsystems, ECU types to ECUs, and messages to the global bus, as well as it would contain constraints for the hardware requirements and to ensure the schedulability of the tasks and the global bus. Even if this complex naive ILP would be solvable with a state-of-the-art ILP solver, the work of Althaus et al. [2] shows that a decrease of running times is achieved when applying a Dantzig-Wolfe decomposition with column generation, especially for large-scaled instances.

We will not present the just mentioned naive ILP formulation since it is too complex and does not bring deeper knowledge. We start by defining patterns that correspond to schedulable subsets of tasks, respectively schedulable subsets of messages, see Definitions 4.1 and 4.2. With the help of these patterns we can hide the complexity of scheduling the ECUs, managing the global bus, and satisfy several hardware restrictions. In the context of the following column generation approach, the patterns correspond to the extreme points of the solution space.

Definition 4.1 (Task pattern). *A* task pattern $p \subseteq \mathcal{T}$ *is a set of tasks that can be scheduled together in a specific subsystem. The collection of task patterns for a subsystem* $s \in \mathcal{S}$ *is denoted by* $\mathcal{P}^s \subseteq 2^{\mathcal{T}}$.

Definition 4.2 (Message pattern). *A* message pattern $q \subseteq \mathcal{M}$ *is a set of messages that can be scheduled together on the global bus. The collection of message patterns is denoted by* $\mathcal{Q} \subseteq 2^{\mathcal{M}}$.

We introduce binary optimization variables $X_{s,p} \in \mathbb{B}$ indicating whether task pattern $p \in \mathcal{P}^s$ consists of a set of tasks that are schedulable in subsystem s or not. Similarly, we introduce $Z_q \in \mathbb{B}$ indicating whether all messages in the message pattern $q \in \mathcal{Q}$ are sent over the global bus or not.

Additional variables are $Y_m \in \mathbb{B}$ indicating whether message m has to be sent over the global bus due to distributed source and destination tasks – it does not indicate whether it has been sent or not, therefore the corresponding message pattern variables Z_q with $m \in q$ have to be inspected. This is due to the fact that a message can be sent over the global bus without the necessity by at least one distributed destination task.

The ILP given in (4.2) – (4.10) will be examined constraint by constraint to prove the correct representation of our DSP model. We stick neither to the canonical nor

to the standard form, since the general form allows the most intuitive formulation of our model, see Section 2.4.

$$z_{ILP} = \min \quad \sum_{p \in \mathcal{P}^s} \text{cost}(s,p) \cdot X_{s,p} \tag{4.2}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}^s} X_{s,p} = 1 \qquad \forall\ s \in \mathcal{S} \tag{4.3}$$

$$\sum_{s \in \mathcal{S}} \sum_{p \in \mathcal{P}^s : t \in p} X_{s,p} = 1 \qquad \forall\ t \in \mathcal{T} \tag{4.4}$$

$$\sum_{q \in \mathcal{Q}} Z_q = 1 \tag{4.5}$$

$$-Y_m + \sum_{q \in \mathcal{Q} : m \in q} Z_q \geqslant 0 \qquad \forall\ m \in \mathcal{M} \tag{4.6}$$

$$-Y_m + \sum_{s \in \mathcal{S}} \sum_{\substack{p \in \mathcal{P}^s : \\ \text{send}(m) \in p \wedge \\ \text{dest}(m) \setminus p \neq \emptyset}} X_{s,p} \leqslant 0 \qquad \forall\ m \in \mathcal{M} \tag{4.7}$$

$$X_{s,p} \in \mathbb{B} \qquad \forall\ s \in \mathcal{S}, p \in \mathcal{P}^s \tag{4.8}$$

$$Z_q \in \mathbb{B} \qquad \forall\ q \in \mathcal{Q} \tag{4.9}$$

$$Y_m \in \mathbb{B} \qquad \forall\ m \in \mathcal{M}. \tag{4.10}$$

Objective function (4.2) The objective function is to minimize the overall cost that is determined by the aggregated cost of the assignments of task pattern p to subsystem s. The coefficients $\text{cost}(s,p)$ indicating these cost will be determined by the selected ECU types in the subsystem, see Section 5.1.3. For the moment, the term $\text{cost}(s,p)$ comes as a non-negative coefficient of the variable $X_{s,p}$.

Subsystem constraints (4.3) The constraints ensure that there is exactly one task pattern that is scheduled on every subsystem. A subsystem is not used if and only if the empty task pattern with zero cost runs on this subsystem.

Task constraints (4.4) The constraints ensure that every task is contained in exactly one task pattern that is used for the solution schedule.

Bus constraint (4.5) The constraint ensures that there is exactly one message pattern that is scheduled on the global bus. The global bus is not used if and only if the empty message pattern is transmitted.

Message pattern constraints (4.6) The constraints ensure that if a message has to be sent over the global bus indicated by $Y_m = 1$, then m has to be contained

in at least one message pattern $q \in \mathcal{Q}$. In combination with (4.5) we know that there can be only one q in the solution. However, a message can be contained in the message pattern, but their contained tasks are not distributed to different subsystems.

Distributed message constraints (4.7) Consider a message $m \in \mathcal{M}$. For each subsystem $s \in \mathcal{S}$ the inner sum concerns every task pattern $p \in \mathcal{P}^s$ where the sender task is included, i.e., $\text{send}(m) \in p$, but at least one of the destination tasks is not included, i.e., $\text{dest}(m) \setminus p \neq \emptyset$. We call this case a *distributed message*, and this message has to be sent over the global bus which is enforced by the constraint to set Y_m equal to 1.

Integrality constraints for the optimization variables (4.8) – (4.10) The optimization variables are Boolean, indicating which task pattern is used for the solution schedule $X_{s,p}$, respectively which message pattern is used for the solution schedule Z_q. The variables Y_m indicating distributed messages are auxiliary.

4.3 LP relaxation

Instead of solving the $\mathcal{NP}$-hard optimization problem (4.2) – (4.10) as an ILP, a standard procedure is to obtain fractional lower bounds on the objective value first to speed-up the search for integral solutions. The method of *LP relaxation* was introduced by Agmon [1] in 1954. The LP relaxation ignores the integrality constraints on the optimization variables of the initial ILP and comes up with an LP that can be solved in polynomial time in general. However, our LP relaxation has exponentially many variables, leading to inefficient worst-case running times.

Consider the optimal objective value of a minimization ILP to be z_{ILP} and the optimal objective value of the corresponding LP relaxation to be z_{LP}, then the relation

$$z_{LP} \leqslant z_{ILP}$$

holds, because the feasible set of the LP relaxation contains the feasible set of the ILP, see Figure 4.2. The gap between these two objective values is called *integrality gap*.

The LP relaxation of (4.2) – (4.10) is given by (4.11) – (4.19). The constraints stay unchanged, however the optimization variables become fractional.

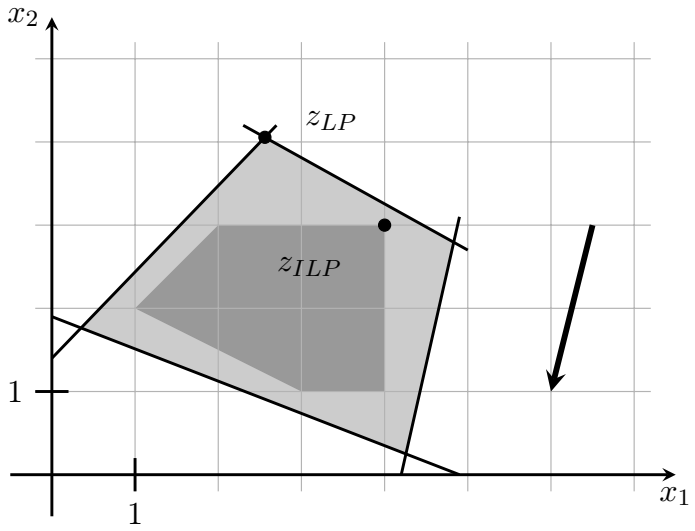


Figure 4.2: LP relaxation.
The constraints of a minimization ILP are shown as black lines and the convex closure of the feasible set is shaded in dark gray. The LP relaxation leads to the light gray shaded feasible set and the optimal relaxed solution z_{LP}, whereas the optimal integral solution z_{ILP} has an objective value that is greater than the relaxed solution. The direction of the objective function is given by the black arrow.

$$z_{LP} = \min \quad \sum_{p \in \mathcal{P}^s} \text{cost}(s,p) \cdot X_{s,p} \tag{4.11}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}^s} X_{s,p} = 1 \qquad \forall\, s \in \mathcal{S} \tag{4.12}$$

$$\sum_{s \in \mathcal{S}} \sum_{p \in \mathcal{P}^s : t \in p} X_{s,p} = 1 \qquad \forall\, t \in \mathcal{T} \tag{4.13}$$

$$\sum_{q \in \mathcal{Q}} Z_q = 1 \tag{4.14}$$

$$-Y_m + \sum_{q \in \mathcal{Q} : m \in q} Z_q \geqslant 0 \qquad \forall\, m \in \mathcal{M} \tag{4.15}$$

$$-Y_m + \sum_{s \in \mathcal{S}} \sum_{\substack{p \in \mathcal{P}^s : \\ \text{send}(m) \in p \wedge \\ \text{dest}(m) \setminus p \neq \emptyset}} X_{s,p} \leqslant 0 \qquad \forall\, m \in \mathcal{M} \tag{4.16}$$

$$X_{s,p} \geqslant 0 \qquad \forall\, s \in \mathcal{S}, p \in \mathcal{P}^s \tag{4.17}$$

$$Z_q \geqslant 0 \qquad \forall\, q \in \mathcal{Q} \tag{4.18}$$

$$Y_m \geqslant 0 \qquad \forall\, m \in \mathcal{M}. \tag{4.19}$$

If we analyze the LP (4.11) – (4.19), we see the master problem of a Dantzig-Wolfe decomposition, see Section 2.4.7, by the bounded polyhedra $\mathcal{P}^s$, $s \in \mathcal{S}$, and $\mathcal{Q}$. The independent extreme points are the task patterns $p \in \mathcal{P}^s$ for every

subsystem $s \in \mathcal{S}$ and the message patterns $q \in \mathcal{Q}$. The corresponding coefficients for the convex combinations are given by $X_{s,p} \geqslant 0$, see (4.17), and $Z_q \geqslant 0$, see (4.18). The convex combination constraints are (4.12) and (4.14), whereas the linking constraints can be identified as (4.13), (4.15), and (4.16).

Upper bounds on the optimization variables (4.17) – (4.19) The optimization variables are relaxed to non-negative fractional variables, where the upper bounds are given implicitly by the other constraints. The task pattern variables $X_{s,p}$ are bounded below 1 by (4.12) because

$$X_{s,p} \leqslant \sum_{p \in \mathcal{P}^s} X_{s,p} = 1 \qquad \forall\ s \in \mathcal{S}, p \in \mathcal{P}^s.$$

The message pattern variables Z_q are bounded below 1 by (4.14) since

$$Z_q \leqslant \sum_{q \in \mathcal{Q}} Z_q = 1 \qquad \forall\ q \in \mathcal{Q}.$$

The variables Y_m are bounded below 1 by (4.15) and (4.14) since

$$Y_m \leqslant \sum_{q \in \mathcal{Q}: m \in q} Z_q \leqslant \sum_{q \in \mathcal{Q}} Z_q = 1 \qquad \forall\ m \in \mathcal{M}.$$

The obtained LP relaxation (4.11) – (4.19) can be solved with a state-of-the-art LP solver using interior points, the ellipsoid method or the extensively discussed simplex method from Section 2.4.6. Our implementation utilizes the simplex method due to the column generation approach, see Section 4.5.

Since we are interested in an optimal integral solution in the end, we embed the LP relaxation into a branch and bound framework, see Section 4.4.

4.4 Branch and Bound Framework

An overview of exact algorithms, such as branch and bound as a divide and conquer design paradigm, is given in Section 2.5.2. Our approach encapsulates the introduced LP relaxation (4.11) – (4.19) into a branch and bound framework to provide an optimal integral solution to the initial ILP formulation (4.2) – (4.10).

The occurring subproblems are arranged as nodes of a search tree that are stored in a priority queue starting with the *root* node corresponding to the initial LP relaxation (4.11) – (4.19). Solving a node means solving the corresponding LP

relaxation to optimality. The strategy of extracting nodes is called *best first search* and is explained in Section 4.4.5.

As we are minimizing the overall cost, every integral solution during the branch and bound process can be used as an upper bound to the cost function, see Section 4.4.1. Therefore, we store the best known integral solution A_I with the integral objective value z_I. Additional lower and upper bounds can be carried along branches with use of the integrality of the objective cost function, see sections 4.4.1 and 4.4.2. Thus a node is only processed if there is a possible combination of ECU types leading to hardware cost between the lower and upper bound for this node.

The evaluation of the LP solution is explained in Section 4.4.3 and the applied branching rules are introduced in Section 4.4.4.

The objective value of the LP relaxation is denoted by z with its lower bound $\underline{z}$ and upper bound $\overline{z}$.

4.4.1 Upper Bounds on the Objective Function

We improve the solving process by pruning nodes with greater or equal objective values and trade on the integrality of the cost function.

For every node except the root node[2], the objective value z_{parent} of the parent node and the objective value z_I of an already known integral solution A_I can be used as upper bound on the objective value z

$$z \leqslant \overline{z} = \min\{z_I - 1, \lfloor z_{parent} \rfloor\}. \tag{4.20}$$

We remark that the value of z_{parent} is known when new branching nodes are generated, but the value z_I can decrease until the node is being processed, and therefore, must be updated prior to the solving process.

4.4.2 Lower Bounds on the Objective Function

Lower bounds on the objective value z can also be used to prune the search tree. We obtain the LP relaxation solution z_{parent} that fulfills $z_{parent} \leqslant z_{ILP}$ and by integrality of the cost function we can conclude that

$$\underline{z} = \lceil z_{parent} \rceil \leqslant z \tag{4.21}$$

[2]Due to the limited number of ECU types per subsystem the overall costs are bounded below by the maximal occurring costs, if every possible ECU is equipped with the most expensive ECU type, i.e., $\sum_{s \in \mathcal{S}} \max_{\mathrm{ECU}}(s) \cdot \max_{et \in \mathcal{ET}} \mathrm{cost}(et) \geqslant z_{root}$.

holds as well, so we add this lower bound to the LP relaxation of generated child nodes.

4.4.3 Evaluation of Solutions

Solving the node's LP relaxation can lead to infeasibility and termination of the branch at this node, or the LP is feasible and we receive a feasible solution A with objective value z. In this case we have to distinguish two cases, namely if A is integral or not.

If A is integral, we ensured by (4.20) that the solution is better than the already known one, so we store the new solution and update $\overline{z}$ to obtain a new upper bound on the cost function of subsequent nodes.

If A is fractional, we start generating new child nodes by a branching rule, see Section 4.4.4, and enqueue these in the priority queue of nodes that have to be examined. The lower bounds are updated according to Section 4.4.2.

4.4.4 Branching Rules

We assume that the solution process of a node provides a fractional solution A, so a branching decision has to be done. The optimization variables are the task and message patterns that have to be forced to be Boolean, for obtaining an integral solution in the end.

The reason we focus on the task pattern is that if the task pattern of the final solution is integral, the message pattern can be turned integral too, as we will show at the end of this section. The intention is to select a branching task and to fix it to every possible subsystem in different child nodes. The process of selecting a branching task is repeated, because a feasible task in the first choice can violate conditions later on.

First, we determine the number of fractional variables for every task indicating the *degree of distribution* of task t, i.e.,

$$\text{degree}(t) = \left|\{X_{s,p} \mid 0 < X_{s,p} < 1, s \in \mathcal{S}, p \in \mathcal{P}^s, t \in p\}\right| .$$

According to constraint (4.13), the degree of a task will satisfy $\text{degree}(t) > 1$, otherwise, the task assignment is integral. Out of the tasks with the highest degree, we select the branching task randomly

$$t \in \left\{t' \in \mathcal{T} \mid \text{degree}(t') \geqslant \text{degree}(t'') \; \forall \; t'' \in \mathcal{T}\right\} .$$

On the basis of the above selection, it is guaranteed that branching task t can be assigned to at least $1 <$ degree(t) many subsystems. In detail we generate child nodes with the restriction that task t is predeployed to exactly one of the available subsystems $s \in \mathcal{S} \setminus \text{for}_{\text{task}}(t)$. For a specific child node, the restriction to subsystem s is formulated by the additional task allocation constraints

$$\text{pre}_{\text{task}}(t) = s \text{ and } \text{for}_{\text{task}}(t) = \{s' \in \mathcal{S} \mid s' \neq s\},$$

thus we ensure that the predeployment and forbidden constraints are consistent. Additionally, we only generate child nodes that lead to valid instances according to Definition 3.15.

Now we turn to the integrality of the message patterns. By our branching strategy, we guarantee that a solution consists of integral task pattern in the end. Thus, the distributed messages $m \in \mathcal{M}$ can be determined by evaluating (4.16) and check which auxiliary optimization variable Y_m is forced to be one.

Constraint (4.14) implies that we terminate with integral message patterns if and only if there is exactly one variable equal to one. Let us assume that there are more than one message patterns with fractional value, i.e., $Q = \{q \in \mathcal{Q} \mid 0 < Z_q \leqslant 1\}$. By constraint (4.16) and the above argument, we know that every distributed message $m \in \mathcal{M}$ with $Y_m = 1$ is included in Q, and satisfies $\sum_{q \in Q : m \in q} Z_q = 1$ by (4.15). Remaining are non-distributed messages $m \in Q$ that do not have to be transmitted, thus we can drop them from the resulting message pattern. Therefore, we select only the message pattern $q = \{m \in \mathcal{M} \mid m \text{ is distributed } \} \in \mathcal{Q}$, and obtain $Z_q = 1$.

4.4.5 Best First Search Strategy

Instead of using a depth or breadth first search strategy when selecting the next node from the priority queue, we apply a best first search strategy, i.e., we choose the node with the lowest lower bound on the objective value first.

Consider that the nodes are ordered increasing on their lower bound $\underline{z}$ on the objective value z. From Sections 4.4.1 and 4.4.2 we know that these bounds are controlled by the objective value of the parent node z_{parent}, and additionally, the upper bound $\overline{z}$ is controlled by the best known integral solution so far z_I. We now show that this order is advantageous in two ways as depicted in Figure 4.3.

Firstly, assume that a node j is erroneously ordered before node i, even if $\underline{z}_i \leqslant \underline{z}_j$. A feasible solution z_j of node j satisfies $\underline{z}_j \leqslant z_j \leqslant \overline{z}_j$, so we can probably lower the upper bound $\overline{z}_i$ to z_j if the solution is integral, but we still have to examine node i since it can lead to a better solution due to the relation $\underline{z}_i \leqslant \underline{z}_j$.

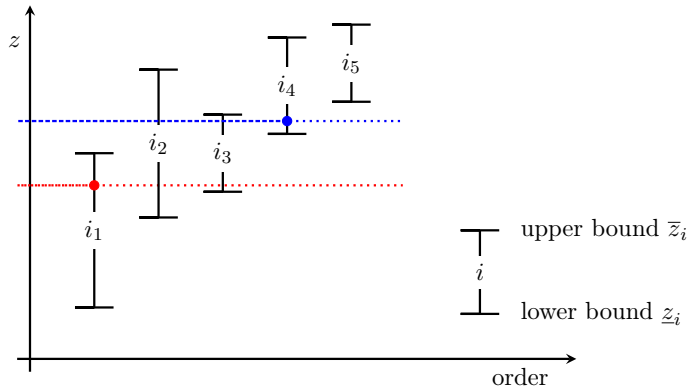


Figure 4.3: Best first search strategy.

The nodes are ordered according to their lower bounds on the objective value $\underline{z}$. If node i_1 can be solved integral, we can decrease the following upper bounds according to the new integral objective value and delete nodes i_4 and i_5, as shown in red. Solving node i_4 before node i_1 is only of minor advantage, because nodes i_1, i_2, and i_3 have to be solved nevertheless, especially, for node i_1 the situation does not change, only for nodes i_2 and i_3 the upper bound would be decreased, as shown in blue.

Secondly, assume that node i can be solved to integrality with objective value z_i, then the upper bounds on every node j after i can be decreased to $\overline{z}_j = z_i - 1$. In the case $\overline{z}_j < \underline{z}_j$, node j can provide no better solution, thus it can be deleted.

4.5 Column Generation Approach

Column generation is applied when solving LPs with a huge number of variables that are not allowed to be basic. In the column generation scenario, the LP is called *master problem*. The master problem in the beginning only consists of a restricted number of variables, and after it is solved to optimality, the values of the dual variables are known. Following the simplex method from Section 2.4.6, we now search for a variable not currently in the LP, so that its reduced cost in the current basic solution is negative. The search of this variable is called *pricing problem*, and the theory is examined in Section 4.5.1.

On the one hand, if the pricing problem can determine a variable with negative reduced cost, the master problem is updated with a new column and solved again. On the other hand, if the pricing problem finds variables with non-negative reduced cost only, it is guaranteed that the current solution of the master problem is optimal, because no variable that was not considered will ever enter the basis.

In our LP relaxation, see (4.11) – (4.19), both the number of task pattern vari-

ables is exponential in the number of tasks for every subsystem and the number of message pattern variables is exponential in the number of messages, see Definitions 4.1 and 4.2. Obviously, this is due to the fact that we introduced task and message patterns, but the advantage of this approach is the separation in independent subproblems with the shift of complexity to these subproblems, as we will see in Sections 4.5.2 and 4.5.3.

4.5.1 Algebraic Derivation

Following Section 2.4.6 we consider an LP in standard form

$$\begin{aligned} z = \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geqslant 0 \end{aligned} \tag{4.22}$$

with $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c, x \in \mathbb{R}^n$. Let B be a m-dimensional basis, then the reduced cost of a non-basic variable x_j, $j \notin B$ is given by

$$\overline{c}_j = c_j - c_B^T A_B^{-1} A^{(j)}. \tag{4.23}$$

With x_B being an optimal primal solution and y_B being an optimal dual solution, we can apply the second complementary slackness condition $(c^T - y_B^T A)x_B = 0$, see Theorem 2.11, to basic variables $x_B > 0$ provides $y_B^T = c_B^T A_B^{-1}$ and the reduced cost from (4.23) simplifies to

$$\overline{c}_j = c_j - y_B^T A^{(j)} \tag{4.24}$$

with free dual variables $y_B \in \mathbb{R}^m$.

In the following, the pricing problems will determine a variable with the most negative reduced cost, i.e.,

$$\begin{aligned} \min \quad & \overline{c}^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geqslant 0 \end{aligned} \tag{4.25}$$

with $\overline{c} = [c_j - y_B^T A^{(j)}]_{j \notin B}$ being the vector of reduced cost for basis B. We can determine the Lagrangian lower bound to the objective value z of (4.22) by

$$z = \min_{\substack{Ax=b, \\ x \geqslant 0}} c^T x \geqslant \min_{\substack{Ax=b, \\ x \geqslant 0}} c^T x - c^T x_B$$

$$
\begin{aligned}
&= \min_{\substack{Ax=b,\\ x\geqslant 0}} c^T x + y_B^T(b - Ax) - c^T x_B \\
&= \min_{\substack{Ax=b,\\ x\geqslant 0}} (c^T - y_B^T A)x + y_B^T b - c^T x_B \\
&= \min_{\substack{Ax=b,\\ x\geqslant 0}} \overline{c}^T x,
\end{aligned}
$$

where the first inequality follows from $c \geqslant 0$ and $x_B > 0$, and the last equality follows from (4.24) and strong duality, see Theorem 2.9.

If we start with an LP with inequalities, the corresponding dual variables are not free anymore, and if the variables are not bounded from below, the dual constraints change, too, see Table 2.1 for an overview of the conversion between primal and dual LPs, and Figure 2.3 for a geometric example.

4.5.2 Master Problem

The master problem must be held feasible all the time to obtain the optimal solution with the restricted number of variables in the beginning. Therefore, artificial variables are introduced to fulfill all constraints with the highest cost possible.

We introduce a *super task pattern* variable $X_{\hat{s},\mathcal{T}}$ with the highest cost possible $\text{cost}(\hat{s}, \mathcal{T})$ expressing that the entire set of tasks $\mathcal{T}$ can be scheduled in an artificial *super subsystem* $\hat{s} \notin \mathcal{S}$. As all tasks are scheduled together, there is no message to send over the global bus, thus we introduce the *empty message pattern* variable $Z_\emptyset$.

The LP relaxation (4.11) – (4.19) is complemented by the two new optimization variables accordingly. In detail, the objective function takes the high cost of the super task pattern, see (4.26), however every subsystem and task constraint is satisfied if the super task pattern is equal to one, see (4.27) and (4.28). The empty message pattern was already included in $\mathcal{Q}$, but to emphasize the formulation we changed (4.29) and excluded $Z_\emptyset$ from the sum. Finally, we obtain the LP relaxation of the root node of the branch and bound search tree (4.26) – (4.35).

$$z_{LP} = \min \quad \text{cost}(\hat{s}, \mathcal{T}) \cdot X_{\hat{s},\mathcal{T}} + \sum_{p \in \mathcal{P}^s} \text{cost}(s,p) \cdot X_{s,p} \tag{4.26}$$

$$\text{s.t.} \qquad X_{\hat{s},\mathcal{T}} + \sum_{p \in \mathcal{P}^s} X_{s,p} = 1 \qquad \forall\, s \in \mathcal{S} \tag{4.27}$$

$$X_{\hat{s},\mathcal{T}} + \sum_{s \in \mathcal{S}} \sum_{p \in \mathcal{P}^s : t \in p} X_{s,p} = 1 \qquad \forall\, t \in \mathcal{T} \tag{4.28}$$

$$Z_\emptyset + \sum_{\emptyset \neq q \in \mathcal{Q}} Z_q = 1 \tag{4.29}$$

$$-Y_m + \sum_{q \in \mathcal{Q} : m \in q} Z_q \geqslant 0 \qquad \forall\, m \in \mathcal{M} \tag{4.30}$$

$$-Y_m + \sum_{s \in \mathcal{S}} \sum_{\substack{p \in \mathcal{P}^s : \\ \text{send}(m) \in p \wedge \\ \text{dest}(m) \setminus p \neq \emptyset}} X_{s,p} \leqslant 0 \qquad \forall\, m \in \mathcal{M} \tag{4.31}$$

$$X_{\hat{s},\mathcal{T}} \geqslant 0 \tag{4.32}$$

$$X_{s,p} \geqslant 0 \qquad \forall\, s \in \mathcal{S}, p \in \mathcal{P}^s \tag{4.33}$$

$$Z_q \geqslant 0 \qquad \forall\, q \in \mathcal{Q} \tag{4.34}$$

$$Y_m \geqslant 0 \qquad \forall\, m \in \mathcal{M}. \tag{4.35}$$

Both pattern variables build the first artificial, feasible solution $X_{\hat{s},\mathcal{T}} = Z_\emptyset = 1$ to (4.26) – (4.35) with $z_{LP} = \text{cost}(\hat{s}, \mathcal{T})$ in the beginning, when $\mathcal{P}^s = \emptyset$ for all $s \in \mathcal{S}$ and $\mathcal{Q} = \{\emptyset\}$.

We introduce the dual variables for the constraints to build the pricing problems in Section 4.5.3. In Table 4.1 we present the constraints of the LP relaxation (4.26) – (4.35) by name and reference with their dual variable and domain corresponding to Table 2.1.

Table 4.1: Dual variables of the LP relaxation.

The table enumerates the constraints (4.27) – (4.31) of the LP relaxation (4.26) – (4.35) with their corresponding dual variables for computing the reduced cost.

Constraint name	Dual variable	Reference
Subsystem $s \in \mathcal{S}$	π_s free	(4.27)
Task $t \in \mathcal{T}$	π_t free	(4.28)
Bus	π_Q free	(4.29)
Message pattern $m \in \mathcal{M}$	$\pi_m \geqslant 0$	(4.30)
Distributed message $m \in \mathcal{M}$	$\pi'_m \leqslant 0$	(4.31)

4.5.3 Pricing Problems

In Section 4.5.1 we derived a formula to compute the reduced cost of a variable via the dual variables. To evaluate (4.24), we need to determine the column of the coefficient matrix of the LP corresponding to the variable.

At this step we separate the examination into two independent subproblems, namely the pricing problem for a task pattern in Section 4.5.3.1, and the pricing problem for a message pattern in Section 4.5.3.2. The task pricing problem comes as a separate and independent problem for every subsystem $s \in \mathcal{S}$.

4.5.3.1 Task Pricing Problems

For every subsystem $s \in \mathcal{S}$, the task pricing problem has to determine a task pattern with negative reduced cost. We construct an ILP that returns a task pattern with the most negative reduced cost. The advantage of this approach is that if the reduced cost is negative, the task pattern will be considered in the next iteration of the simplex method for the master problem. Otherwise, there is no pattern that can enter the basis without increasing the objective value, hence the solution is optimal.

For the remainder of this section we fix the subsystem to $s \in \mathcal{S}$ and determine the column of the coefficient matrix of the LP relaxation (4.26) – (4.35) for a new variable $X_{s,p}$ with $p \subseteq \mathcal{T}$.

For $X_{s,p}$ we get coefficient 1 by the subsystem constraint (4.27), coefficient 1 for every task $t \in p$ by the task constraints (4.28), and coefficient 1 for every message $m \in \mathcal{M}$ with $\text{send}(m) \in p$ and $\text{dest}(m) \cap p \neq \emptyset$, i.e., every distributed message with sender task in p, by (4.31). Since the tasks $t \in p$ have to be determined by the pricing problem, and this decision influences whether a message is distributed or not, we introduce binary variables for this decision, $p_t \in \mathbb{B}$ for every $t \in \mathcal{T}$, respectively $y_m \in \mathbb{B}$ for every $m \in \mathcal{M}$. Variable p_t equals one if and only if task t is included in the resulting pattern p, respectively y_m equals one if message m is distributed.

To ensure the correct determination of y_m we can formulate the constraints $y_m - p_{\text{send}(m)} + p_b \geqslant 0$ for every $m \in \mathcal{M}$ and $b \in \text{dest}(m)$. On the one hand, if $\text{send}(m)$ is not included in p, then $p_{\text{send}(m)} = 0$ and the constraints are redundant to the lower bounds on y_m and every p_b with $b \in \text{dest}(m)$. On the other hand, if $\text{send}(m)$ is included in p, then $p_{\text{send}(m)} = 1$ and the constraints are $y_m + p_b \geqslant 1$ for every $b \in \text{dest}(m)$ which means that if just one $b \in \text{dest}(m)$ is not included in pattern p, i.e., $p_b = 0$, then message m is distributed and this forces $y_m = 1$.

Finally, the reduced cost $\bar{c}_{s,p}$ of pattern $p = \{t \in \mathcal{T} \mid p_t = 1\}$ in subsystem $s \in \mathcal{S}$

can be determined by the task pricing problem ILP

$$\begin{aligned}
\overline{c}_{s,p} = \min \quad & \text{cost}(s,p) - \pi_s - \sum_{t \in \mathcal{T}} \pi_t \cdot p_t - \sum_{m \in \mathcal{M}} \pi'_m \cdot y_m && && (4.36)\\
\text{s.t.} \quad & y_m - p_{\text{send}(m)} + p_b \geqslant 0 && \forall\ m \in \mathcal{M}, b \in \text{dest}(m) && (4.37)\\
& p = [p_t]_{t \in \mathcal{T}} \in \mathcal{P}^s && && (4.38)\\
& p_t \in \mathbb{B} && \forall\ t \in \mathcal{T} && (4.39)\\
& y_m \in \mathbb{B} && \forall\ m \in \mathcal{M}. && (4.40)
\end{aligned}$$

The objective function (4.36) expresses the computation of the reduced cost by (4.24). The crucial condition is stated in $p \in \mathcal{P}^s$, see (4.38), meaning that pattern p will be regarded in the master problem if and only if it is schedulable in this subsystem, corresponding to Definition 4.1. We remark that the task pricing problems for different steps of solving the master LP do not differ in the constraints, but only in the coefficients of the objective function as the dual variables are changing – except for our new ILP formulation, see Section 5.3.4.

In Chapter 5 we focus on ILP formulations for the schedulability conditions.

4.5.3.2 Message Pricing Problem

For the global communication we state the message pricing problem that has to determine a message pattern $q \in \mathcal{Q}$ with negative reduced cost. The strategy is similar to the construction of the task pricing problems in Section 4.5.3.1. We derive an ILP that returns a message pattern with the most negative reduced cost, and in the case the reduced costs are negative, this pattern will be regarded in the next simplex iteration of the master problem.

For a new variable Z_q of the master problem we get coefficient 1 by the bus constraint (4.29), and coefficient 1 for every message $m \in q$ by the message pattern constraint (4.30). To determine which messages m are included in the message pattern q we introduce binary optimization variables $q_m \in \mathbb{B}$ for $m \in \mathcal{M}$, expressing that $m \in q$ if and only if $q_m = 1$.

The reduced cost $\overline{c}_q$ of message pattern $q = \{m \in \mathcal{M} \mid q_m = 1\}$ can be determined by the message pricing ILP

$$\begin{aligned}
\overline{c}_q = \min \quad & -\pi_Q - \sum_{m \in \mathcal{M}} \pi_m \cdot q_m && && (4.41)\\
\text{s.t.} \quad & q = [q_m]_{m \in \mathcal{M}} \in \mathcal{Q} && && (4.42)\\
& q_m \in \mathbb{B} && \forall\ m \in \mathcal{M}. && (4.43)
\end{aligned}$$

The objective function (4.41) expresses the computation of the reduced cost by (4.24), where the cost of transmitting a message pattern is considered to be zero.

Again, the crucial condition is hidden in $q \in \mathcal{Q}$, i.e., the schedulability of the global bus due to a specific bus protocol. We remark that also the message pricing problems for different steps of solving the master LP only differ in the coefficients of the objective function.

In Chapter 5 we present ILP formulations for the schedulability of the global bus.

4.6 Implementation Strategies

In this section we introduce implementation strategies applied to reduce the search space, accelerate the solution process, and decrease the running time of our algorithm.

4.6.1 Integrality of the Objective Function

We can use the integrality of the objective function in two ways that we will examine, where the computation of tighter upper bounds is not as important as the raising of lower bounds.

On the one hand, the objective value is given by the sum over the cost of installed ECU types, see Section 3.5 and (3.1). Since the maximal number of ECUs in a subsystem is given by the description of the hardware architecture, see Definition 3.2, we can easily compute an overall upper bound to an instance by assuming that the most expensive ECU type is used for every available ECU slot.

This upper bound is rather pessimistic since there can be restrictions on the number of used ECU types, too. To make use out of these additional information we start calculating with the most expensive ECU types, unless the ECU slots are exhausted or the given ECU type reaches its maximum. Then we continue with the cheaper ones to obtain a tighter upper bound. This strategy is also applied to obtain a value for the cost of the artificial super pattern $\text{cost}(\hat{s}, \mathcal{T})$, see Section 4.5.2, that has to be larger than realizable.

On the other hand, we are solving LP relaxations in the master problem and do not assume to reach an integral solution in the first nodes of the branch and bound tree. Therefore, it is advantageous if we can stop the computation after a few iterations. This happens if the lower bound reaches the current LP value.

In Section 4.5.1 we computed the Lagrangian lower bound by the sum of the objective values of the pricing problems, see (4.5.1), since this is the maximal possible decrease of the objective function. Additionally, we trade on the integral coefficients of the objective function to compute all possible resulting cost,

sort them, and push the lower bound to the next possible cost value if it is fractional. Therefore, we pre-compute a pessimistic set of possible costs $A(n)$ for the maximal quantity of available ECU slots $n = \sum_{s \in \mathcal{S}} \max_{\text{ECU}}(s)$ with a dynamic programming approach by the recursive formula

$$A(n) = A(n-1) \cup \{a + \text{cost}(et) \mid a \in A(n-1), et \in \mathcal{ET}\} \text{ and } A(0) = \{0\}.$$

4.6.2 Multiple Columns Generation

In Section 4.5.1 we derived the column generation approach that adds one single variable to the master problem, namely the one with the most negative reduced cost computed by the pricing problems. In general there will be even more variables that can enter the basis due to negative reduced cost, but we do not want to examine all of them like in the simplex method. Nevertheless, we can benefit from adding some more variables to the master problem than just one.

One advantage of solving the pricing ILPs by a state-of-the-art ILP solver is that they provide suboptimal solutions without extra time since they initiate a branching process themselves to solve the pricing ILPs. In detail they provide an allocation of the optimization variables and the objective value is computable in linear time by evaluation of the objective function. If the size and running time of solving the master LP is still acceptable in comparison to the pricing ILPs, there is no need to care about removing unused columns from the master LP.

4.6.3 Heuristic Phase

The branch and bound framework explained in Section 4.4 is referred to as the *exact phase*. A disadvantage of the exact phase is that numerous branches have to be examined if the upper bound is weak or decreases poorly. In the other case, if we find a good upper bound during optimization, we can prune branches and delete nodes where the lower bound is greater than the currently obtained upper bound.

Therefore, a slight modification of the exact phase is processed previously, called *heuristic phase*. The goal is to generate heuristic solutions and useful pattern variables for the master problem with a similar framework.

The heuristic phase also starts with solving the root LP relaxation. In general we expect large running times for the computation of the initial nodes since they start with the smallest pattern sets, and an improvement of the lower or upper bound in these nodes is rare. To avoid large running times with steady bounds we terminate the solving process of a node according to several stopping criteria

explained below. Also the branching rule and the processing order of the branch and bound nodes varies from the exact phase.

The best outcome of such a heuristic phase would be a large set of integral solutions after just a minimum of invested time, followed by one single iteration of the exact phase that confirms the best solution to be optimal. To guarantee the termination of the heuristic phase both the number of processed nodes and the number of obtained integral solutions is limited, however an additional limitation of the running time per node is conceivable to keep the heuristic phase controllable.

Stopping Criteria We define the absolute error κ_{abs} of the objective value by the difference of the upper $\overline{z}$ and lower bound $\underline{z}$ in a branch and bound node, i.e.,

$$\kappa_{abs} = \overline{z} - \underline{z} \geqslant 0.$$

Consequently, the relative error κ_{rel} for minimization problems is given by division by the best known solution, i.e., the upper bound $\overline{z}$

$$\kappa_{rel} = \frac{\kappa_{abs}}{\overline{z}},$$

so that a relative error of ε guarantees that the optimal objective value z satisfies

$$z \in [(1-\varepsilon) \cdot \overline{z}, \overline{z}] .$$

The computation of a branch and bound node can be terminated if the absolute error κ_{abs} underruns an absolute limit, or the relative error κ_{rel} underruns a relative limit. Additionally, we terminate the solving process of a node if the absolute error κ_{abs} remains constant for a number of iterations. This happens if the lower bound does not increase and the optimal solution to the LP is the artificial solution with the highest cost possible.

Heuristic Branching Rule In the case a node is not solved by an integral allocation, a slightly modified branching rule than in Section 4.4.4 is applied. Instead of predeploying only a single task to a subsystem, the heuristic phase tries to predeploy as many tasks as possible to the same subsystem. We assume that the branching task $t \in \mathcal{T}$ is included in the task pattern $p \in \mathcal{P}^s$ for some $s \in \mathcal{S}$ with the largest value $X_{s,p} < 1$ of the current optimal solution. The heuristic phase generates child nodes similarly to Section 4.4.4, but predeploys as many tasks $t' \in p$ as possible to the specific subsystem, taking care of generating valid instances according to Definition 3.15 only.

Processing Order In Section 4.4.5 we introduced the best first search order for processing the nodes of the branch and bound tree in the exact phase. In the heuristic phase, the prevalent goal is to quickly generate feasible pattern variables to find integral solutions. Due to this fact, we traverse the heuristic branch and bound tree by a depth first search since the nodes become easier and faster to solve with the additional predeployments of the modified heuristic branching rule.

Chapter 5

Formulations for Different Scenarios

Up to now, the crucial details in the task pricing problems (4.36) – (4.40), respectively (4.41) – (4.43), are hidden in $p \in \mathcal{P}^s$, respectively $q \in \mathcal{Q}$. We derive formulations that express both the schedulability of a set of preemptive, periodic tasks, as well as messages on a global bus.

This chapter is divided into formulations for the task pricing problems, see Section 5.1, formulations for the message pricing problem, see Section 5.2, and formulations that lead to modifications of the master and the pricing problems, so called combined formulations, see Section 5.3.

5.1 Task Pricing Formulations

As the task pricing problems are independent, we consider a fixed subsystem $s \in \mathcal{S}$ for the remainder of this section. We restrict our presentation to the case that there is at most one ECU and one ECU type in every subsystem, i.e., $\max_{\text{ECU}}(s) = 1$ for every $s \in \mathcal{S}$ and $\mathcal{ET} = \{et\}$. Therefore, the definition of a task assignment $\tau : \mathcal{T} \rightarrowtail\!\!\!\!\!\!\rightarrow \mathcal{S} \times \mathcal{E} \times \mathcal{ET}$, see Definition 3.8, collapses to

$$\tau : \mathcal{T} \rightarrowtail\!\!\!\!\!\!\rightarrow \mathcal{S},$$

without loss of generality by the argument following in Section 5.1.1. Except for the additional constraints from Section 3.4 that we will discuss in Section 5.1.2, we stick to the collapsed formulation.

The remainder of this section starts with the derivation of constraints to deter-

mine the cost of executing a task pattern in a subsystem, see Section 5.1.3. A constraint for the memory consumption is determined in Section 5.1.4. Section 5.1.5 is split into several formulations for FPS policies of preemptive, periodic tasks.

5.1.1 Simplifications

In this section we present the formulation for the complex, but general case with multiple ECUs and ECU types.

In the general case, we introduce consistent variables $a(t, e, et) \in \mathbb{B}$ that express if task $t \in \mathcal{T}$ is assigned to ECU $e \in \mathcal{E}$ of ECU type $et \in \mathcal{ET}$, variables $b(e, et) \in \mathbb{B}$ that express if ECU e is of ECU type et, and variables $u(e) \in \mathbb{B}$ that express if ECU e is used in the subsystem.

By

$$\sum_{e \in \mathcal{E}} \sum_{et \in \mathcal{ET}} a(t, e, et) = p_t \qquad \forall\, t \in \mathcal{T}$$

we ensure that every task is exactly assigned to one ECU and ECU type if and only if it is included in pattern p. The constraints

$$a(t, e, et) \leqslant b(e, et) \qquad \forall\, t \in \mathcal{T}, e \in \mathcal{E}, et \in \mathcal{ET}$$

enforce that the ECU type of an ECU has to be set if a task is assigned to this ECU of this ECU type. Finally, we ensure that every ECU is of exactly one ECU type if and only if it is used by

$$\sum_{et \in \mathcal{ET}} b(e, et) = u(e) \qquad \forall\, e \in \mathcal{E}.$$

Consequently, the cost of the subsystem is given by

$$\text{cost}(s, p) = \sum_{e \in \mathcal{E}} \sum_{et \in \mathcal{ET}} b(e, et) \cdot \text{cost}(et).$$

To reduce complexity, we restrict our discussion to at most one ECU of a given ECU type in every subsystem. Therefore, the variable $u \in \mathbb{B}$ satisfies $p_t \leqslant u$ for all $t \in \mathcal{T}$.

5.1.2 Formulations for the Additional Constraints

According to Section 3.4 we present ILP formulations for all additional constraints, using the expanded formulation given in the argument above. We remark

that we consider the subsystem $s \in \mathcal{S}$.

In Definition 3.10 we defined additional hardware constraints to the minimal and maximal number of used ECU types in a subsystem. These requirements are formulated as linear constraints of the form

$$\sum_{e \in \mathcal{E}} b(e, et) \geqslant \min_{\text{ECUtype}}(s, et) \qquad \forall\ et \in \mathcal{ET} \tag{5.1}$$

$$\sum_{e \in \mathcal{E}} b(e, et) \leqslant \max_{\text{ECUtype}}(s, et) \qquad \forall\ et \in \mathcal{ET}. \tag{5.2}$$

In Definition 3.11 we defined additional task allocation constraints that are of great importance, especially for the branch and bound process. The predeployment of a task to subsystem s is realized in a change of the lower bound, respectively the upper bound is changed if the task is forbidden to be executed in subsystem s. Since the optimization variables p_t are Boolean, we fix them according to

$$p_t = \begin{cases} 1, & s = \text{pre}_{\text{task}}(t) \\ 0, & s \in \text{for}_{\text{task}}(t) \end{cases}, \tag{5.3}$$

where it is satisfied that the cases are disjoint.

In Definition 3.13 we defined additional task clustering constraints. $\text{cluster}_{\text{task}}(t)$ specifies a set of tasks that has to be assigned to the same subsystem as t, and by $\text{separate}_{\text{task}}(t)$ we specify a set of tasks that has to be assigned to other subsystems than t. We formulate this by

$$p_t = p_{t'} \qquad \forall\ t' \in \text{cluster}_{\text{task}}(t), t' \neq t \tag{5.4}$$

$$p_t + p_{t'} \leqslant 1 \qquad \forall\ t' \in \text{separate}_{\text{task}}(t). \tag{5.5}$$

5.1.3 Cost of Hardware

For computing the reduced cost of a task pattern, see (4.36), we need to determine the cost that arises by assigning a set of tasks to this subsystem. Therefore, we introduce an auxiliary binary variable $u \in \mathbb{B}$ that is equal to one if and only if the ECU in the subsystem is used, see Section 5.1.1. The cost of a subsystem is zero if no task is assigned to, otherwise the ECU type determines the cost by

$$\text{cost}(s,p) = \text{cost}(et) \cdot u \tag{5.6}$$

$$u \leqslant \sum_{t \in \mathcal{T}} p_t \tag{5.7}$$

$$p_t \leqslant u \qquad \forall\, t \in \mathcal{T} \tag{5.8}$$

$$u \in \mathbb{B}. \tag{5.9}$$

We remark that the lower bounds on u can be stated as one linear constraint by

$$\frac{1}{|\mathcal{T}|} \cdot \sum_{t \in \mathcal{T}} p_t \leqslant u \tag{5.10}$$

instead of (5.8). However, formulation (5.10) is weaker[1], because the feasible set of the relaxation of (5.10) is a superset of the relaxation of (5.8).

Consider the relaxation of the task variables p_t, i.e., $p_t \in [0,1]$, and assume that not all p_t are equal, thus they can be sorted in ascending order $\{p_{\min}, \ldots, p_{\max}\}$ with $p_{\min} < p_{\max}$. The alternative formulation (5.10) leads to the lower bound $\frac{1}{|\mathcal{T}|} \cdot \sum_{t \in \mathcal{T}} p_t < p_{\max}$, whereas (5.8) always implies the lower bound $p_{\max}$. Therefore, the formulation (5.8) is stronger than (5.10).

5.1.4 Memory Consumption

The memory consumption of the scheduled set of tasks has to be less than or equal to the available memory of the ECU type. This constraint translates to an inequality constraint of the form

$$\sum_{t \in \mathcal{T}} \text{mem}(t) \cdot p_t \leqslant \text{mem}(et). \tag{5.11}$$

5.1.5 Scheduling

We consider FPS policies, i.e., we have to determine a fixed priority $\text{pr}(t)$ for every task $t \in \mathcal{T}$ in advance of the execution, because FPS policies result in safe and reliable schedules, see Section 2.6.2.6. Therefore, we recall the analyses introduced in Section 2.6.2 and express the schedulability tests in linear constraints to replace $p \in \mathcal{P}^s$ in the task pricing problems, see (4.38).

[1] More guidelines on weak and strong ILP formulations can be found in Bertsimas and Tsitsiklis [10, 10.2].

5.1.5.1 Processor Utilization Analysis

We can determine the processor utilization $U(T)$ for a set of tasks $T \subseteq \mathcal{T}$ by the sum

$$\sum_{t \in \mathcal{T}} \frac{C(t)}{P(t)} \cdot p_t$$

following (2.13). Applying Corollary 2.20, a necessary condition is that the utilization is bounded from above by one, expressed by

$$\sum_{t \in \mathcal{T}} \frac{C(t)}{P(t)} \cdot p_t \leqslant 1. \tag{5.12}$$

A sufficient, but not necessary schedulability test on the processor utilization $U(T)$ for a set of tasks $T \subseteq \mathcal{T}$ is stated in Theorem 2.23 by

$$U(T) \leqslant U^* \text{ with } U^* = |T| \cdot \left(2^{\frac{1}{|T|}} - 1\right),$$

where U^* tends to be $\ln(2) \simeq 0.69$ for large sets T. The number of scheduled tasks $|\{p_t \mid p_t = 1\}|$ can be expressed by the integral variable $\text{size}(p) \in \mathbb{N}$ by the linear constraint

$$\text{size}(p) = \sum_{t \in \mathcal{T}} p_t. \tag{5.13}$$

The computation of the upper bound U^* must be done in advance up to a predefined accuracy depending on the number of scheduled tasks $\text{size}(p)$. Setting the accuracy to 10^{-2} we can build ten intervals on the number of scheduled tasks, see Table 5.1.

The variables $I_j \in \mathbb{B}$ for $j = 1, \ldots, 10$ express if the number of scheduled tasks is greater than or equal to the left interval boundary $lb_j \in \mathbb{N}$ of interval j by the constraints

$$\frac{\text{size}(p) + |\mathcal{T}| + 1}{|\mathcal{T}| + lb_j} - 1 \leqslant I_j \leqslant \frac{\text{size}(p) + |\mathcal{T}|}{|\mathcal{T}| + lb_j} \qquad \forall\ j = 1, \ldots, 10. \tag{5.14}$$

We prove that the binary variables I_j for $j = 1, \ldots, 10$ are determined correctly by (5.14). If $|\mathcal{T}| > 1$, the left-hand side is always smaller than the right-hand side. First, consider $\text{size}(p) < lb_j$, then the left-hand side is non-positive, the right-hand side is smaller than one, thus $I_j = 0$. Second, consider $\text{size}(p) \geqslant lb_j$, then the left-hand side is greater than zero, the right-hand side is greater than or equal to one, thus $I_j = 1$.

As the number of scheduled tasks is not known in advance, we have to formulate the constraint for every task interval of Table 5.1 and ensure that the correct upper bound is active. We remark that independently of the active upper bound,

Table 5.1: Intervals and upper bounds on the processor utilization.
If the number of scheduled tasks $|T|$ is in the given intervals, we get a sufficient, but not necessary schedulability test of accuracy 10^{-2}, if we bound the processor utilization $U(T)$ from above by $\underline{U}^* = 0.01 \cdot \lfloor 100 \cdot U^* \rfloor$.

j	$\lvert T\rvert$	$\underline{U}^*$
1	$[1, 1]$	1.00
2	$[2, 2]$	0.82
3	$[3, 3]$	0.77
4	$[4, 4]$	0.75
5	$[5, 5]$	0.74
6	$[6, 6]$	0.73
7	$[7, 9]$	0.72
8	$[10, 14]$	0.71
9	$[15, 35]$	0.70
10	$[36, \infty)$	0.69

all larger upper bounds remain valid, i.e., $I_j = 1 \Rightarrow I_{j'} = 1$ for every $j' > j$. The following constraint uses the differences between consecutive upper bounds to tighten the upper bound of the processor utilization up to the correct one, in detail

$$\sum_{t \in \mathcal{T}} \frac{C(t)}{P(t)} \cdot p_t \leqslant I_1 - 0.18 \cdot I_2 - 0.05 \cdot I_3 - 0.02 \cdot I_4 - 0.01 \cdot \sum_{i=5}^{10} I_j. \tag{5.15}$$

Although this schedulability test is not necessary, it can be used to fast computing subsets of tasks that can be scheduled for sure – keeping in mind that we lose overall optimality by this approach.

5.1.5.2 Response Time Analysis

In this section we examine the response time analysis of Theorem 2.24 to formulate the schedulability test (2.15). Therefore, we have to determine the WCRT $R(t)$ for every $t \in \mathcal{T}$ with the recursive formula (2.19)

$$R(t) = C(t) + \sum_{t' \in \mathrm{hp}(t)} \left\lceil \frac{R(t)}{P(t')} \right\rceil \cdot C(t').$$

We follow the approach of Eisenbrand et al. [31] to state linear constraints for (2.19), where we introduce integral optimization variables for the WCRT $R(t) \in \mathbb{N}$ for every task $t \in \mathcal{T}$ with $R(t) = 0$ if and only if the task is not scheduled. Therefore, we can formulate the schedulability test by

$$R(t) \leqslant D(t) \cdot p_t \qquad \forall\, t \in \mathcal{T} \tag{5.16}$$
$$R(t) \in \mathbb{N} \qquad \forall\, t \in \mathcal{T}. \tag{5.17}$$

The crucial step in the work of Eisenbrand et al. [31] is to linearize the recursive formula for the WCRT computation (2.19). We introduce binary variables $\varphi(t, t') \in \mathbb{B}$ that express the preemption of tasks, i.e., $\varphi(t, t') = 1$ if and only if task t' may preempt task t and they are both assigned. Additional conditions[2] to the preemption variables are the transitivity rule, i.e., if t'' may preempt t' and t' may preempt t, then t'' may also preempt t, see (5.22), and anti-symmetry for tasks that are assigned, see (5.23). We achieve this by stating

$$\varphi(t, t) = p_t \qquad \forall\, t \in \mathcal{T} \tag{5.18}$$
$$\varphi(t, t') \leqslant p'_t \qquad \forall\, t, t' \in \mathcal{T} \tag{5.19}$$
$$\varphi(t, t') \leqslant p_t \qquad \forall\, t, t' \in \mathcal{T} \tag{5.20}$$
$$\varphi(t, t') + \varphi(t', t) \geqslant p_t + p'_t - 1 \qquad \forall\, t, t' \in \mathcal{T}, t \neq t' \tag{5.21}$$
$$\varphi(t', t'') + \varphi(t, t') - \varphi(t, t'') \leqslant 1 \qquad \forall\, t, t', t'' \in \mathcal{T} \tag{5.22}$$
$$\varphi(t, t') + \varphi(t', t) \leqslant 1 \qquad \forall\, t, t' \in \mathcal{T}, t \neq t' \tag{5.23}$$
$$\varphi(t, t') \in \mathbb{B} \qquad \forall\, t, t' \in \mathcal{T}. \tag{5.24}$$

We remark that (5.23) for $t = t'$ would force $\varphi(t, t) = 0$ for every $t \in \mathcal{T}$ that is not consistent with the following definitions, so we define $\varphi(t, t) = p_t$ for $t \in \mathcal{T}$, see (5.18).

With the help of the above construction, we model the sum in (2.19) over all tasks by introducing integral optimization variables $h(t, t') \in \mathbb{N}$ that express

$$h(t, t') = \left\lceil \frac{R(t)}{P(t')} \right\rceil \cdot \varphi(t, t')$$

to obtain linear constraints for the WCRT computation of the form

$$R(t) = \sum_{t' \in \mathcal{T}} C(t') \cdot h(t, t') \qquad \forall\, t \in \mathcal{T}. \tag{5.25}$$

Including tasks t' with lower priority than t is correct, since $\varphi(t, t') = 0$ for all tasks $t' \in \mathrm{lp}(t)$, and tasks with priority equal to t have to break the tie according to the preemption variables. Also including the task itself is correct,

[2] A good overview to formulating logical implications in combinatorial optimization can be found in Plastria [51].

since $h(t,t) = \varphi(t,t) = p_t$ is obtained by (5.27) and (5.28). We formulate the bounds on $h(t,t')$ for $t,t' \in \mathcal{T}$ by the linear constraints

$$h(t,t') \leqslant \frac{R(t)}{P(t')} + 1 \qquad \forall\ t,t' \in \mathcal{T} \tag{5.26}$$

$$h(t,t') \leqslant \left\lceil \frac{R^{(ub)}(t)}{P(t')} \right\rceil \cdot \varphi(t,t') \qquad \forall\ t,t' \in \mathcal{T} \tag{5.27}$$

$$h(t,t') \geqslant \frac{R(t)}{P(t')} - \frac{R^{(ub)}(t)}{P(t')} \cdot (1 - \varphi(t,t')) \qquad \forall\ t,t' \in \mathcal{T} \tag{5.28}$$

$$h(t,t') \geqslant \left\lceil \frac{C(t) + C(t')}{P(t')} \right\rceil \cdot \varphi(t,t') \qquad \forall\ t,t' \in \mathcal{T}, t \neq t' \tag{5.29}$$

$$h(t,t') \in \mathbb{N} \qquad \forall\ t,t' \in \mathcal{T}, \tag{5.30}$$

where $R^{(ub)}(t)$ denotes an upper bound on the WCRT $R(t)$, e.g., $R^{(ub)}(t) = D(t)$. Constraints (5.26) and (5.28) are obtained by estimations of the ceiling function, constraint (5.27) is obtained by $R(t) \leqslant R^{(ub)}(t)$, and constraint (5.29) is obtained by $R(t) \geqslant C(t) + C(t')$ if t' preempts t.

Summarizing, we obtain the conditions $h(t,t') = 0$ if $\varphi(t,t') = 0$, and

$$\max\left\{ \frac{R(t)}{P(t')}, \left\lceil \frac{C(t) + C(t')}{P(t')} \right\rceil \right\} \leqslant h(t,t') \leqslant \min\left\{ \left\lceil \frac{R^{(ub)}(t)}{P(t')} \right\rceil, \frac{R(t)}{P(t')} + 1 \right\}$$

if $\varphi(t,t') = 1$ for $t \neq t'$, i.e., task t' may preempt the execution of task t, with the special case $h(t,t) = \varphi(t,t) = p_t$.

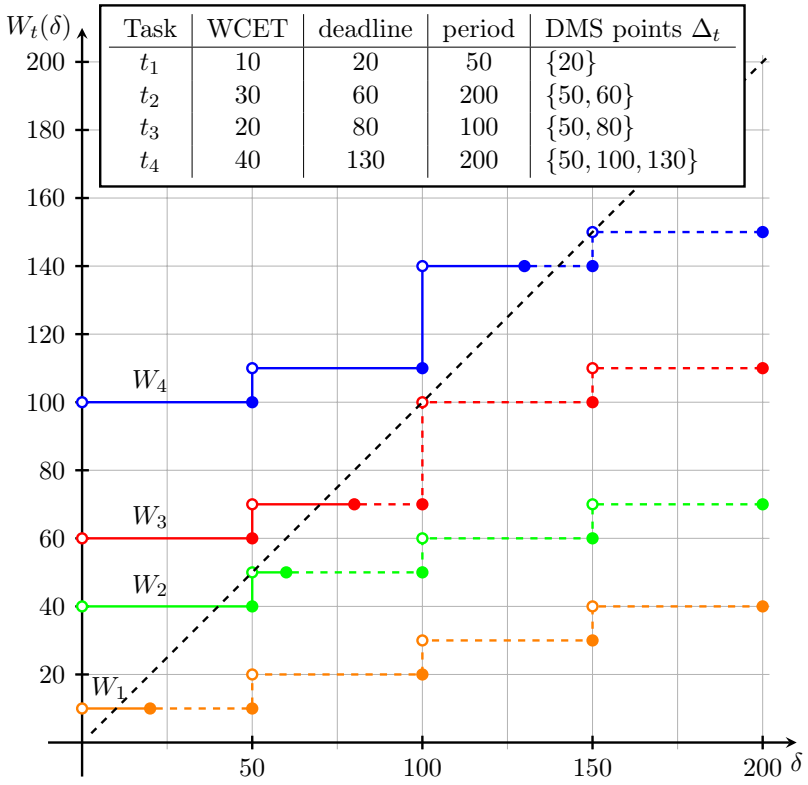
Since we are considering a FPS policy, we apply the DMS policy if we know the deadlines for specific tasks, i.e., $D(t) \neq \perp$. In case two tasks $t,t' \in \mathcal{T}$ with $D(t) \neq \perp$, respectively $D(t') \neq \perp$, are assigned to the same subsystem, we can predetermine the preemption variables by the following implication

$$\varphi(t,t') = 1 \Leftrightarrow (D(t) > D(t'))\ \vee\ ((D(t) = D(t'))\ \wedge\ (\mathrm{ID}_t \geqslant \mathrm{ID}_{t'})),$$

where ties are broken according to the task ordering by a task ID.

5.1.5.3 Workload Analysis

In this section we examine the workload analysis of Theorem 2.25, respectively Theorem 2.26. Both theorems differ only in the testing set $\Delta_t \subseteq (0, D(t)]$ for all $t \in \mathcal{T}$. A task $t \in \mathcal{T}$ is schedulable if there exists a point of time in the testing set $\delta \in \Delta_t$, so that the cumulative workload demand until δ is less than or equal to δ. An example is given in Figure 5.1.



Task	WCET	deadline	period	DMS points Δ_t
t_1	10	20	50	$\{20\}$
t_2	30	60	200	$\{50, 60\}$
t_3	20	80	100	$\{50, 80\}$
t_4	40	130	200	$\{50, 100, 130\}$

Figure 5.1: Cumulative workload demand.

An example of the cumulative workload demand function $W_t(\delta)$ for some tasks in the interval $[0, \delta]$. It is obvious that the schedulability condition has to be checked only in the DMS points Δ_t for a schedulability analysis to see that task t_4 cannot be added if the other three tasks are assigned.

Since the workload analysis is used for the DMS policy, the priority of a task is reciprocal to its deadline. We remark that we can group tasks together in *deadline classes*, i.e., $DC_d = \{t \in \mathcal{T} \mid D(t) = d\}$ for several $d \in \mathbb{N}$. Obviously, the priority of tasks in DC_d is higher than for tasks in $DC_{d'}$ with $d' < d$, i.e.,

$$t' \in \mathrm{hp}(t) \Leftrightarrow t' \in \bigcup_{\substack{d' < d: \\ t \in DC_d}} DC_{d'}.$$

Consequently, as the testing sets depend on the deadlines only, they are equal for all tasks belonging to the same deadline class, thus we define $\Delta_d = \Delta_t$ for all $t \in DC_d$ in the context of both theorems.

The crucial step in this analysis is to model the existential quantifier, i.e., several conditions can be violated for schedulability if there is at least one valid condition. Therefore, we introduce binary auxiliary variables $\alpha_{d,\delta} \in \mathbb{B}$ for every deadline

class DC_d and point of the testing set $\delta \in \Delta_d$ with the linear constraints

$$\sum_{\delta \in \Delta_d} \alpha_{d,\delta} \leqslant |\Delta_d| - 1 \qquad \forall\, d : DC_d. \tag{5.31}$$

By (5.31) we ensure that there is at least one variable $\alpha_{d,\delta} = 0$, and at most $|\Delta_d| - 1$ variables equal to one that are used to artificially satisfy a constraint of the following workload analysis.

The next step is to compute the cumulative workload. Following Theorem 2.25, respectively Theorem 2.26, we obtain the cumulative workload $W_d(\delta)$ of all tasks in deadline class DC_d at a point of the testing set $\delta \in \Delta_d$ by

$$\begin{aligned}
W_d(\delta) &= \sum_{t \in DC_d} C(t) + \sum_{t' \in \mathrm{hp}(t)} \left\lceil \frac{\delta}{P(t')} \right\rceil \cdot C(t') \\
&= \sum_{t \in DC_d} C(t) + \sum_{\substack{t' \in DC_{d'}: \\ d' < d}} \left\lceil \frac{\delta}{P(t')} \right\rceil \cdot C(t') \\
&= \sum_{t \in DC_d} \left\lceil \frac{\delta}{P(t)} \right\rceil \cdot C(t) + \sum_{\substack{t' \in DC_{d'}: \\ d' < d}} \left\lceil \frac{\delta}{P(t')} \right\rceil \cdot C(t') \\
&= \sum_{\substack{t \in DC_{d'}: \\ d' \leqslant d}} \left\lceil \frac{\delta}{P(t)} \right\rceil \cdot C(t).
\end{aligned}$$

For the schedulability test we only have to consider currently scheduled tasks that fulfill $p_t = 1$. For every deadline class DC_d, there has to be at least one point $\delta \in \Delta_d$ where the cumulative workload demand is less than or equal to δ. Thus we can artificially satisfy at most $|\Delta_d| - 1$ of the following constraints with the help of the variables $\alpha_{d,\delta}$ with a big constant $M \in \mathbb{N}$ by

$$\sum_{\substack{t \in DC_{d'}: \\ d' \leqslant d}} \left\lceil \frac{\delta}{P(t)} \right\rceil \cdot C(t) \cdot p_t \leqslant \delta + M \cdot \alpha_{d,\delta} \qquad \forall\, \delta \in \Delta_d, d : DC_d. \tag{5.32}$$

We remark that the feasibility of constraints (5.32) for DC_d does not restrict the feasibility of constraints for $DC_{d'}$ with $d' < d$. Consider a task $t' \in DC_{d'}$ with $p_{t'} = 1$, then there is a $\delta' \in \Delta_{d'}$ with $\alpha_{d',\delta'} = 0$. Especially, we have $\delta' \leqslant d'$ and there exists at least one point $\delta \in \Delta_d$ with $\delta' \leqslant \delta$, thus there is also one constraint of DC_d that will be fulfilled with $\alpha_{d,\delta} = 0$.

5.2 Message Pricing Formulations

Considering the message pricing problem (4.41) – (4.43), the conditions for schedulability of the global bus are hidden in $q \in \mathcal{Q}$. In this section, we derive linear formulations that realize schedulability tests for the global bus depending on different message scheduling policies, see Section 2.6.3.

5.2.1 Analysis for TAN buses

The TAN bus works time-triggered with a token moving between the subsystems in a round robin fashion. The conditions of the schedulability test of Section 2.6.3.1 are easily expressible in linear constraints by

$$\sum_{m' \in \mathcal{M}} C(m') \cdot q_{m'} \leqslant D(m) \cdot q_m + (1 - q_m) \cdot \sum_{m' \in \mathcal{M}} C(m') \qquad \forall\, m \in \mathcal{M} \quad (5.33)$$

$$R(m) \geqslant \sum_{m' \in \mathcal{M}} C(m') \cdot q_{m'} + C(m) \cdot q_m \qquad \forall\, m \in \mathcal{M} \quad (5.34)$$

$$R(m) \leqslant D(m) \cdot q_m \qquad \forall\, m \in \mathcal{M} \quad (5.35)$$

$$R(m) \in \mathbb{N} \qquad \forall\, m \in \mathcal{M}. \quad (5.36)$$

A necessary and sufficient schedulability test is already given by (5.33), because we state that the TRT on the left-hand side is less than or equal to the deadline of message m if $q_m = 1$, and if $q_m = 0$ the constraint is fulfilled by default. By (5.34), respectively (5.35), we obtain a lower bound, respectively upper bound, on the WCRT, however, we do not obtain the minimal WCRT.

5.2.2 Response Time Analysis for CAN buses

A CAN bus works priority-based and event-triggered, thus the response time analysis for schedulability resembles the analysis of tasks with the only difference that messages are transmitted without preemption, see Section 2.6.3.2. For this reason the transmission of a message can additionally be blocked by a lower priority message that is currently on transmission.

The WCRT of a message $m \in \mathcal{M}$ is composed of its WCTT $C(m)$, its blocking time $B(m)$ by lower or equal priority messages, and its interference time $I(m)$ by higher priority messages also waiting for transmission. A necessary and sufficient schedulability test is given by (2.27)

$$R(m) = C(m) + B(m) + I(m) \leqslant D(m) \qquad \forall\, m \in \mathcal{M},$$

where the blocking time is given by

$$B(m) = \max_{m' \in \mathrm{lep}(m)} \{0, C(m')\} \qquad \forall\, m \in \mathcal{M},$$

and the interference time is given by

$$I(m) = \sum_{m' \in \mathrm{hp}(m)} \left\lceil \frac{R(m)}{P(m')} \right\rceil \cdot C(m') \qquad \forall\, m \in \mathcal{M},$$

according to (2.26).

There is no efficient scheduling policy with fixed priorities for non-preemptive messages known, however, Davis [29] states two simple sufficient but not necessary schedulability tests. For this reason, we formulate the response time analysis for CAN buses, where we have to introduce binary optimization variables $\varphi(m, m') \in \mathbb{B}$ for all $m, m' \in \mathcal{M}$ that express the priority ordering of two messages[3]. We formulate that $\varphi(m, m') = 1$ if and only if messages m and m' are transmitted and m' has a higher priority than m, by the constraints

$$\varphi(m, m) = q_m \qquad \forall\, m \in \mathcal{M} \tag{5.37}$$

$$\varphi(m, m') \leqslant q'_m \qquad \forall\, m, m' \in \mathcal{M} \tag{5.38}$$

$$\varphi(m, m') \leqslant q_m \qquad \forall\, m, m' \in \mathcal{M} \tag{5.39}$$

$$\varphi(m, m') + \varphi(m', m) \geqslant q_m + q'_m - 1 \qquad \forall\, m, m' \in \mathcal{M}, m \neq m' \tag{5.40}$$

$$\varphi(m', m'') + \varphi(m, m') - \varphi(m, m'') \leqslant 1 \qquad \forall\, m, m', m'' \in \mathcal{M} \tag{5.41}$$

$$\varphi(m, m') + \varphi(m', m) \leqslant 1 \qquad \forall\, m, m' \in \mathcal{M}, m \neq m' \tag{5.42}$$

$$\varphi(m, m') \in \mathbb{B} \qquad \forall\, m, m' \in \mathcal{M}, \tag{5.43}$$

where constraint (5.41) expresses the transitivity rule, and (5.42) expresses the anti-symmetry. We remark that (5.23) for $m = m'$ would force $\varphi(m, m) = 0$ for every $m \in \mathcal{M}$ that is not consistent with the following definitions, so we define $\varphi(m, m) = q_m$ for $m \in \mathcal{M}$ in (5.37).

To model the blocking time $B(m)$ we introduce integral optimization variables $B(m) \in \mathbb{N}$ for $m \in \mathcal{M}$ with lower and upper bounds by

$$B(m) \geqslant C(m') \cdot (1 - \varphi(m, m')) \qquad \forall\, m, m' \in \mathcal{M} \tag{5.44}$$

$$B(m) \leqslant q_m \cdot \max_{m' \in \mathcal{M}} C(m') \qquad \forall\, m \in \mathcal{M} \tag{5.45}$$

$$B(m) \in \mathbb{N} \qquad \forall\, m \in \mathcal{M}, \tag{5.46}$$

[3] In Section 5.1.5.2 we use $\varphi(t, t') \in \mathbb{B}$ to express the preemptive behaviour of tasks $t, t' \in \mathcal{T}$.

where $B(m) = 0$, if message m is not transmitted over the global bus or there is no message with lower priority than m. The lower bound on $B(m)$ is obtained by constraints over all lower or equal priority messages, see (5.44).

To model the recursive formula (2.27) we rewrite the formula for the WCRT $R(m) \in \mathbb{N}$ by summing over all messages $m' \in \mathcal{M}$ with the use of the priority variables $\varphi(m, m')$ to obtain

$$R(m) = B(m) + \sum_{m' \in \mathcal{M}} \left\lceil \frac{R(m)}{P(m')} \right\rceil \cdot C(m') \cdot \varphi(m, m') \qquad \forall\, m \in \mathcal{M},$$

where $R(m) = 0$ if and only if message m is not transmitted over the global bus.

Similar to the approach in Section 5.1.5.2, we have to introduce integral optimization variables $h(m, m') \in \mathbb{N}$ for $m, m' \in \mathcal{M}$ to model the terms in the sum, i.e.,

$$h(m, m') = \left\lceil \frac{R(m)}{P(m')} \right\rceil \cdot \varphi(m, m').$$

We formulate the bounds on $h(m, m')$ for $m, m' \in \mathcal{M}$ by the linear constraints

$$h(m, m') \leqslant \frac{R(m)}{P(m')} + 1 \qquad \forall\, m, m' \in \mathcal{M} \tag{5.47}$$

$$h(m, m') \leqslant \left\lceil \frac{R^{(ub)}(m)}{P(m')} \right\rceil \cdot \varphi(m, m') \qquad \forall\, m, m' \in \mathcal{M} \tag{5.48}$$

$$h(m, m') \geqslant \frac{R(m)}{P(m')} - \frac{R^{(ub)}(m)}{P(m')} \cdot (1 - \varphi(m, m')) \qquad \forall\, m, m' \in \mathcal{M} \tag{5.49}$$

$$h(m, m') \geqslant \left\lceil \frac{C(m) + C(m')}{P(m')} \right\rceil \cdot \varphi(m, m') \qquad \forall\, m, m' \in \mathcal{M}, m \neq m' \tag{5.50}$$

$$h(m, m') \in \mathbb{N} \qquad \forall\, m, m' \in \mathcal{M}, \tag{5.51}$$

where $R^{(ub)}(m)$ denotes an upper bound on the WCRT $R(m)$, e.g., $R^{(ub)}(m) = D(m)$. Summarizing, we obtain the conditions $h(m, m') = 0$ if $\varphi(m, m') = 0$, and

$$\max\left\{ \frac{R(m)}{P(m')}, \left\lceil \frac{C(m) + C(m')}{P(m')} \right\rceil \right\} \leqslant h(m, m') \leqslant \min\left\{ \left\lceil \frac{R^{(ub)}(m)}{P(m')} \right\rceil, \frac{R(m)}{P(m')} + 1 \right\}$$

if $\varphi(m, m') = 1$ for $m \neq m'$, i.e., message m' has a higher priority than message m, with the special case $h(m, m) = \varphi(m, m) = q_m$.

Finally, the WCRT is obtained by

$$R(m) = B(m) + \sum_{m' \in \mathcal{M}} C(m') \cdot h(m, m') \qquad \forall\, m \in \mathcal{M} \tag{5.52}$$

$$R(m) \leqslant D(m) \cdot q_m \qquad \forall\, m \in \mathcal{M} \tag{5.53}$$

$$R(m) \in \mathbb{N} \qquad \forall\, m \in \mathcal{M}. \tag{5.54}$$

5.2.3 Analysis for a simple TDMA bus

We consider a TDMA bus, see Section 2.6.3.3, with one TDMA cycle, $N \in \mathbb{N}$ equally sized time slots $\mathcal{B} = \{b_1, \ldots, b_N\}$ of length $size \in \mathbb{N}$, and messages with zero release jitter that fit exclusively in these slots. A message $m \in \mathcal{M}$ fits into a time slot if and only if $C(m) \leqslant size$.

An upper bound to the WCRT $R(m)$ is given by the minimum of the messages deadline, the sender tasks period, and the length of the TDMA round, i.e., $R^{(ub)}(m) = \min\{D(m), P(\text{send}(m)), N \cdot size + C(m)\}$. By $\left\lceil \frac{R^{(ub)}(m)}{size} \right\rceil$ we obtain the minimal number of time slots before message m needs to occupy an additional slot. The minimal number of slots that a message needs for on time transmission is then obtained by

$$\left\lceil \frac{N}{\left\lceil \frac{R^{(ub)}(m)}{size} \right\rceil} \right\rceil,$$

where the inner ceiling function denotes the maximal number of slots before the message occupies the next time slot. We formulate a necessary schedulability condition by the above discussion in the number of available time slots by

$$C(m) \cdot q_m \leqslant size \cdot q_m \qquad \forall\, m \in \mathcal{M} \tag{5.55}$$

$$\sum_{m \in \mathcal{M}} \left\lceil \frac{N}{\left\lceil \frac{R^{(ub)}(m)}{size} \right\rceil} \right\rceil \cdot q_m \leqslant N. \tag{5.56}$$

To ensure the schedulability of the TDMA round we introduce auxiliary Boolean assignment variables $\alpha(m, b) \in \mathbb{B}$ for $b \in \mathcal{B}$ and $m \in \mathcal{M}$ that express that message m occupies time slot b if and only if $\alpha(m, b) = 1$. Consequently, every time slot is occupied by at most one message, see constraints (5.57), and a message does not occupy a time slot if it is not transmitted, see (5.58).

We introduce optimization variables $R(m) \in \mathbb{N}$ to determine the WCRT for every message $m \in \mathcal{M}$. The crucial step in determining the WCRT is to control the maximal gap between two time slots occupied by the same message.

Therefore, we introduce auxiliary Boolean variables $\beta(m, b_i, b_j) \in \mathbb{B}$ which express

that slots b_i and b_j are both occupied by message m and that there is no time slot between b_i and b_j that is also occupied by m, also considering that the time slots are periodical. By $\sqcup(b_i, b_j) \subseteq \mathcal{B}$ we denote the set of time slots b_k that are between b_i and b_j exclusively, taking into account that the TDMA bus operates cyclically, i.e.,

$$\sqcup(b_i, b_j) = \begin{cases} \{b_k \in \mathcal{B} \mid i < k < j\} & , i < j \\ \{b_k \in \mathcal{B} \mid k < j \ \vee \ i < k\} & , i \geqslant j \end{cases},$$

and we remark that this is convenient even if $b_i = b_j$. Constraint (5.59) forces $\beta(m, b_i, b_j)$ to be zero if not both time slots b_i and b_j are assigned to message m, and constraint (5.60) interferes if there is a slot between b_i and b_j that is assigned to message m. Vice versa, constraint (5.61) forces $\beta(m, b_i, b_j)$ to be one, if b_i and b_j are assigned to message m and there is not at least one time slot b_k in between that is assigned to m, too.

The maximal consecutive gap between two time slots assigned to message m determines a lower bound on the WCRT $R(m)$ by (5.63), where $\text{dist}(b_i, b_j) \in \mathbb{N}$ counts the time slots between b_i and b_j exclusively, also considering the cyclic operation of the TDMA bus and the case if $b_i = b_j$, and (5.62) formulates the upper bound from above.

Except for (5.57), the following constraints and variables are necessary for all message $m \in \mathcal{M}$:

$$\sum_{m \in \mathcal{M}} \alpha(m, b) \leqslant 1 \qquad \forall\, b \in \mathcal{B} \tag{5.57}$$

$$\sum_{b \in \mathcal{B}} \alpha(m, b) \leqslant N \cdot q_m \tag{5.58}$$

$$\beta(m, b_i, b_j) \leqslant \frac{1}{2}\left(\alpha(m, b_i) + \alpha(m, b_j)\right) \qquad \forall\, b_i, b_j \in \mathcal{B} \tag{5.59}$$

$$\beta(m, b_i, b_j) \leqslant 1 - \alpha(m, b_k) \qquad \forall\, b_i, b_j \in \mathcal{B},\ b_k \in \sqcup(b_i, b_j) \tag{5.60}$$

$$\beta(m, b_i, b_j) \geqslant \alpha(m, b_i) + \alpha(m, b_j) - 1 - \sum_{b_k \in \sqcup(b_i, b_j)} \alpha(m, b_k) \qquad \forall\, b_i, b_j \in \mathcal{B} \tag{5.61}$$

$$R(m) \leqslant R^{(ub)}(m) \cdot q_m \tag{5.62}$$

$$R(m) \geqslant size \cdot \text{dist}(b_i, b_j) \cdot \beta(m, b_i, b_j) + C(m) \cdot q_m \tag{5.63}$$

$$R(m) \in \mathbb{N} \tag{5.64}$$

$$\alpha(m, b_i) \in \mathbb{B} \qquad \forall\, b_i \in \mathcal{B} \tag{5.65}$$

$$\beta(m, b_i, b_j) \in \mathbb{B} \qquad \forall\, b_i, b_j \in \mathcal{B}. \tag{5.66}$$

Formulations for the Additional Constraints In Definition 3.12 we defined additional message allocation constraints that can be realized with the help of the

optimization variables $\alpha(m, b) \in \mathbb{B}$ for TDMA buses from the above discussion. The predeployment of a message to a set of bus slots results in a change of the lower bound, respectively the upper bound is changed if the message is forbidden to be assigned to specific bus slots. We fix the optimization variables according to

$$\alpha(m,b) = \begin{cases} 1, & b \in \text{pre}_{\text{msg}}(m) \\ 0, & b \in \text{for}_{\text{msg}}(m) \end{cases}, \tag{5.67}$$

where it is satisfied that the cases are disjoint.

5.3 Combined Formulations

In this section we discuss formulations that affect both the master problem and the pricing problems. The necessity of these combined formulations are conditions that concern both the tasks and messages synchronously.

The leading example of this section is the chain, see Section 5.3.1, with its scheduling analysis. Taking these into account leads to modifications of the master problem that are discussed in Section 5.3.2, and modifications in the selection of schedulability analyses for the pricing problems, see Section 5.3.3.

Our new ILP formulation for computing response times with the help of the dual variables of the master LP is presented in Section 5.3.4.

5.3.1 Analysis for Chains

Consider a chain $c \in \mathcal{C}$ of Definition 3.7 with an overall end-to-end deadline $D(c) \in \mathbb{N}$ for the execution of its sequence

$$\text{seq}(c) = \langle t_1, m_1, t_2, m_2, t_3, \ldots, t_k, m_k, t_{k+1} \rangle,$$

where $\text{send}(m_i) = t_i$, $t_i \notin \text{dest}(m_i)$, and $t_{i+1} \in \text{dest}(m_i)$ for $i = 1, \ldots, k$.

Following Zhu et al. [63], the *worst-case latency* of the chain $L(c) \in \mathbb{N}$ is the worst-case response time of the consecutive execution of the sequence $\text{seq}(c)$ that must be less than or equal to its end-to-end deadline $D(c)$ to be schedulable, where the worst-case end-to-end latency is given by

$$L(c) = R(t_1) + \sum_{i=2,\ldots,k+1} (R(t_i) + P(t_i)) + \sum_{\substack{m_i \in \text{seq}(c): \\ t_i \text{ is distributed} \\ \text{from } t_{i+1}}} (R(m_i) + P(m_i)), \tag{5.68}$$

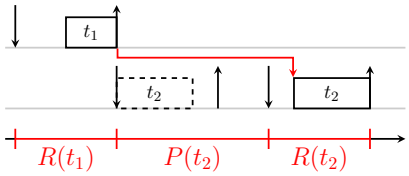


Figure 5.2: Worst-case latency of a chain with tasks of the same subsystem.
An example of the worst-case end-to-end latency occurring if there is no need for task t_1 to send a message to task t_2 over the global bus, because the later is assigned to the same subsystem. Nevertheless t_1 finishes its execution shortly after the start of task t_2, thus the information will first arrive at the second instance of t_2. The down arrows indicate the activation times, the up arrows the WCRTs of tasks and messages.

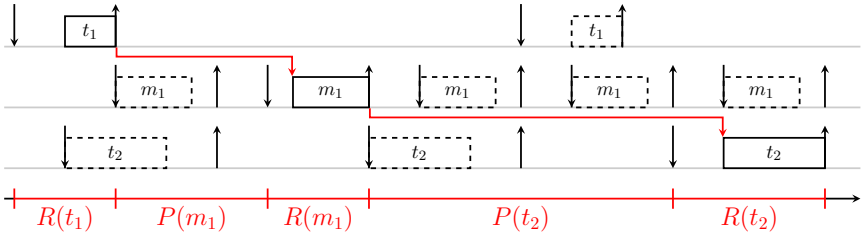


Figure 5.3: Worst-case latency of a chain with distributed tasks.
An example of the worst-case end-to-end latency occurring in the transmission of a message m_1 from task t_1 to the distributed task t_2 over the global bus. Task t_1 misses the first instance of message m_1, and the second instance of m_1 finishes shortly after the start of task t_2. The down arrows indicate the activation times, the up arrows the WCRTs of tasks and messages.

where $R(t_i)$, respectively $R(m_i)$, denotes the WCRT of task t_i, respectively message m_i.

We remark that the latency (5.68) only uses the WCRT of message m_i if the sender t_i and the destination task in the chain t_{i+1} are distributed. This covers the case when the WCRT of message m_i is greater than zero, although the next task in the sequence of the chain is assigned to the same subsystem, because another destination task is distributed from the sender. Both situations are shown in Figures 5.2 and 5.3.

5.3.2 Modifications of the Master Problem

The definition of the worst-case end-to-end latency of a chain $L(c)$ in (5.68) shows the connection between the tasks' and messages' WCRTs that depend on the schedule of the according subsystems and the global bus. For this reason the

scheduling of tasks is no longer independent of the scheduling of messages, thus the decisions have to be made with the combined knowledge in the master problem (4.26) – (4.35).

We now derive the linear formulation of $L(c) \leqslant D(c)$ for every chain $c \in \mathcal{C}$ to expand the master problem. To model the WCRTs of tasks and messages in the LP we do not use variables since this leads to a huge increase in complexity.

We consider the WCRTs to be parameters of the pattern variables $X_{s,p}$, respectively Z_q, but we have to remark that the pattern has to be expanded. Up until now, a task pattern was equal to another task pattern if it considered the same subsystem and contained the same set of tasks. As from now, the same set of tasks that runs on the same subsystem can have different WCRTs for its containing tasks, and therefore the pattern has to be included multiply into the master problem, because they differ by the WCRTs of their tasks.

By the above argument, we expand the meaning of the WCRTs and pattern variables $X_{s,p}$ and Z_q by adding auxiliary variables for every pattern with different WCRTs by an index $j \in J(s,p)$, respectively, $j \in J(q)$,

$$X_{s,p} = \sum_{j \in J(s,p)} X_{s,p}^{(j)} \qquad \forall\ s \in \mathcal{S}, p \in \mathcal{P}^s \tag{5.69}$$

$$Z_q = \sum_{j \in J(q)} Z_q^{(j)} \qquad \forall\ q \in \mathcal{Q} \tag{5.70}$$

$$X_{s,p}^{(j)} \geqslant 0 \qquad \forall\ s \in \mathcal{S}, p \in \mathcal{P}^s, j \in J(s,p) \tag{5.71}$$

$$Z_q^{(j)} \geqslant 0 \qquad \forall\ q \in \mathcal{Q}, j \in J(q) \tag{5.72}$$

to the master problem (4.26) – (4.35). Accordingly, the correct WCRT of a task $t \in \mathcal{T}$ depends on the subsystem $s \in \mathcal{S}$, pattern $p \in \mathcal{P}^s$ with $t \in p$, and index $j \in J(s,p)$, thus it is given by

$$R(t) = \sum_{s \in \mathcal{S}} \sum_{p \in \mathcal{P}^s : t \in p} \sum_{j \in J(s,p)} R^{(s,p,j)}(t) \cdot X_{s,p}^{(j)}.$$

The WCRT of a message $m \in \mathcal{M}$ depends on the selected bus pattern $q \in \mathcal{Q}$ with $m \in q$ and the index $j \in J(q)$, thus it is given by

$$R(m) = \sum_{q \in \mathcal{Q} : m \in q} \sum_{j \in J(q)} R^{(q,j)}(m) \cdot Z_q^{(j)}.$$

We remind that the number of pattern variables is expected to stay small by using column generation, as well as the index sets J that are first considered if the corresponding pattern variable exists in the master problem.

With the variables at hand, we consider a chain $c \in \mathcal{C}$ with its sequence

$$\text{seq}(c) = \langle t_1, m_1, t_2, m_2, t_3, \ldots, t_k, m_k, t_{k+1} \rangle,$$

and introduce the linear constraint to (5.68). To keep the constraint readable we neglect the index sets J and just indicate the dependence of the WCRTs of the patterns. The constraint for a chain $c \in \mathcal{C}$ is given by

$$\begin{aligned}
&\sum_{s \in \mathcal{S}} \sum_{p \in \mathcal{P}^s : t_1 \in p} R^{(s,p)}(t_1) \cdot X_{s,p} \\
&\quad + \sum_{s \in \mathcal{S}} \sum_{\substack{p \in \mathcal{P}^s : t_i \in p \\ i=2,\ldots,k+1}} \Big(R^{(s,p)}(t_i) + P(t_i) \Big) \cdot X_{s,p} \\
&\quad + \sum_{\substack{q \in \mathcal{Q} : m_i \in q \\ i=1,\ldots,k}} \sum_{s \in \mathcal{S}} \sum_{\substack{p \in \mathcal{P}^s : \\ t_i \in p \,\wedge \\ t_{i+1} \notin p}} \Big(R^{(q)}(m_i) + P(m_i) \Big) \cdot X_{s,p} \leqslant D(c). \qquad (5.73)
\end{aligned}$$

The first and second term translate straightforward from (5.68), however, we show the validity of (5.73) for a message m_i with $i = 1, \ldots, k$ with the sender task $t_i = \text{send}(m_i)$ and the destination task $t_{i+1} \in \text{dest}(m_i)$ in the sequence of the chain.

First, we consider that tasks t_i and t_{i+1} are scheduled in the same subsystem by task pattern $p \in \mathcal{P}^s$. In this case, there is no need to transmit m_i over the global bus, however, m_i can be sent over the bus because of another distributed destination task $t' \in \text{dest}(m_i)$ with $t' \notin p$. Therefore, it would be wrong to add $R(m_i) + P(m_i)$ to the latency according to the value of Z_q with $m_i \in q$.

In the second case, if tasks t_i and t_{i+1} are distributed, we have to add $R(m_i) + P(m_i)$ to the latency. By (4.31) it is ensured that m_i is transmitted over the global bus.

To satisfy both cases, we add $R(m_i) + P(m_i)$ only in the case when t_i and t_{i+1} are distributed according to the regarded task pattern p, as given in the last term of (5.73) by the condition $t_i \in p \ \wedge \ t_{i+1} \notin p$.

To conclude the modifications of the master problem, we have to introduce the dual variables for the pricing problems. Since we link both types of pattern variables, we have to consider the dual values in both types of pricing problems. We call constraints of the form (5.73) chain constraints and denote the dual variables by $\pi_c \leqslant 0$ for every $c \in \mathcal{C}$, additionally to Table 4.1. The dual value of $\pi_c \leqslant 0$ works as a penalty in the objective functions of the pricing problems if a task, respectively a message, of the chain is scheduled, see Section 5.3.3.

5.3.3 Modifications of the Pricing Problems

In Section 5.3.2 we examined the modifications of the master problem to handle chains that introduce new chain constraints with dual variables $\pi_c \leqslant 0$ for every $c \in \mathcal{C}$. These dual variables affect the objective functions of the pricing problems, so that they penalize the objective value by the amount of response time of a task in the task pricing problems, respectively the amount of response time of a message in the message pricing problem.

We first focus on the task pricing problems, where the WCRT of a task is computed by the integral optimization variables $R(t) \in \mathbb{N}$ for every $t \in \mathcal{T}$. The additional term of the objective function (4.36) of a task pricing problem is

$$-\sum_{t \in \mathcal{T}} \sum_{c \in \mathcal{C}: t \in \text{seq}(c)} \pi_c \cdot R(t). \tag{5.74}$$

With non-positive dual variables π_c the linear term (5.74) spells out that a higher WCRT cannot decrease the negative reduced cost with all other variables held fix.

In the message pricing problem, we similarly introduce integral optimization variables $R(m) \in \mathbb{N}$ for every message $m \in \mathcal{M}$ that leads to the additional term of the objective function (4.41)

$$-\sum_{m \in \mathcal{M}} \sum_{c \in \mathcal{C}: m \in \text{seq}(c)} \pi_c \cdot R(m). \tag{5.75}$$

5.3.4 New ILP formulation for Computing Response Times

In this section we examine our new approach for computing response times in the pricing problems with the help of the workload analysis for FPS policies, see Section 2.6.2.6, in the case of chain constraints with dual variables $\pi_c \leqslant 0$ for every $c \in \mathcal{C}$, as introduced above.

To obtain a FPS policy, we have to determine a priority for every task $\text{pr}(t)$ to handle preemption of tasks that are assigned to the same ECU. Therefore, we use the penalty on the objective function expressed in (5.74).

The Testing Set We start in analyzing the testing set of the workload demand analysis for FPS stated in Theorem 2.25. A set of tasks $T \subseteq \mathcal{T}$ is schedulable if

and only if there exists a $\delta \in (0, D(t)]$ with

$$W_t(\delta) = C(t) + \sum_{t' \in \mathrm{hp}(t)} \left\lceil \frac{\delta}{P(t')} \right\rceil \cdot C(t') \leqslant \delta \qquad \forall\, t \in T.$$

We remark that it is sufficient to evaluate the inequality for the testing set of t

$$\Delta_t = \{D(t)\} \cup \{\ell \cdot P(t') \leqslant D(t) \mid \ell \in \mathbb{N}, t' \in \mathrm{hp}(t)\},$$

which is the set of multiples of periods of higher priority tasks up to the tasks deadline inclusively. A refinement of the testing set is given in Theorem 2.26.

Since the priority of the tasks is still unknown we expand the definition to the testing set of the entire set of tasks $\mathcal{T}$ in ascending order by

$$\Delta = \{\delta_0, \ldots, \delta_k\} = \{D(t) \mid t \in \mathcal{T}\} \cup \{\ell \cdot P(t) \leqslant \max_{t' \in \mathcal{T}} D(t') \mid \ell \in \mathbb{N}\} \tag{5.76}$$

for some $k \in \mathbb{N}$, $\delta_0 = 0$, and $\delta_j < \delta_{j+1}$ for $j = 1, \ldots, k-1$. Additionally, we define k left half-open intervals $I_j = (\delta_{j-1}, \delta_j]$ for $j = 1, \ldots, k$.

The idea is to assign WCRTs to intervals, because the WCRT of a task is influenced by tasks with WCRTs in earlier intervals and the same interval. Hereafter, we show that we can use a ratio of dual variables and WCETs to allocate priorities to the tasks.

The Ratio We are considering two schedules with two tasks t and t', and compare the WCRTs in the cases where t is scheduled before, respectively after t'.

In the case where t is executed before t', the WCRTs are given by $R(t) = C(t)$, and $R(t') = C(t) + C(t')$. The additional contribution to the computation of reduced cost by (5.74) is given by

$$\begin{aligned} c_{t \to t'} &= - \sum_{c \in \mathcal{C}: t \in \mathrm{seq}(c)} \pi_c \cdot R(t) - \sum_{c \in \mathcal{C}: t' \in \mathrm{seq}(c)} \pi_c \cdot R(t') \\ &= - \sum_{c \in \mathcal{C}: t \in \mathrm{seq}(c)} \pi_c \cdot C(t) - \sum_{c \in \mathcal{C}: t' \in \mathrm{seq}(c)} \pi_c \cdot \big(C(t) + C(t')\big)\,. \end{aligned}$$

In the other case where t' is executed before t, we get an additional contribution of

$$c_{t' \to t} = - \sum_{c \in \mathcal{C}: t \in \mathrm{seq}(c)} \pi_c \cdot \big(C(t') + C(t)\big) - \sum_{c \in \mathcal{C}: t' \in \mathrm{seq}(c)} \pi_c \cdot C(t').$$

The first case leads to reduced cost less than or equal to the second case if and

only if the difference $c_{t\to t'} - c_{t'\to t} \leqslant 0$ which can be stated for better readability as

$$\frac{\Sigma_t}{C(t)} \geqslant \frac{\Sigma_{t'}}{C(t')}.$$

with $\Sigma_t = \sum_{c\in\mathcal{C}:t\in\text{seq}(c)} -\pi_c$ and $\Sigma_{t'} = \sum_{c\in\mathcal{C}:t'\in\text{seq}(c)} -\pi_c$. Finally, we observe that the priority of tasks is given by the ratio $\frac{\Sigma_t}{C(t)} \geqslant 0$. In case of equal ratios, an arbitrary order is taken, e.g., by a task ID, thus we obtain

$$\text{hp}(t) = \left\{ t' \in \mathcal{T} \;\middle|\; \frac{\Sigma_{t'}}{C(t')} > \frac{\Sigma_t}{C(t)} \right\} \cup \left\{ t' \in \mathcal{T} \;\middle|\; \frac{\Sigma_{t'}}{C(t')} = \frac{\Sigma_t}{C(t)} \text{ with } \text{ID}_{t'} > \text{ID}_t \right\}.$$

The Computation of WCRTs For every task $t \in \mathcal{T}$ we adopt an integral optimization variable for the WCRT $R(t)$ and an upper bound by $D(t) \cdot p_t$ from (5.17) and (5.16).

$$\begin{aligned}
R(t) \geqslant & \sum_{i<j} \sum_{t'\in\mathcal{T}} C(t') \cdot \left\lceil \frac{\delta_j}{P(t')} \right\rceil \cdot \ell(t', i) \\
& + \sum_{t'\in\text{hp}(t)} C(t') \cdot \left\lceil \frac{\delta_j}{P(t')} \right\rceil \cdot \ell(t', j) \\
& + C(t) \cdot \ell(t, j) - \text{M} \cdot (1 - \ell(t, j)) && \forall\, t \in \mathcal{T}, j = 1, \ldots, k && (5.77) \\
R(t) \leqslant & D(t) \cdot p_t && \forall\, t \in \mathcal{T} && (5.78) \\
\delta_j \geqslant & \sum_{i\leqslant j} \sum_{t\in\mathcal{T}} C(t) \cdot \left\lceil \frac{\delta_j}{P(t)} \right\rceil \cdot \ell(t, i) \\
& - \text{M}' \cdot (1 - \ell(j)) && j = 1, \ldots, k && (5.79) \\
\ell(j) \geqslant & \ell(t, j) && \forall\, t \in \mathcal{T}, j = 1, \ldots, k && (5.80) \\
p_t = & \sum_{j=1,\ldots,k} \ell(t, j) && \forall\, t \in \mathcal{T} && (5.81) \\
\ell(t, j) \in & \mathbb{B} && \forall\, t \in \mathcal{T}, j = 1, \ldots, k && (5.82) \\
\ell(j) \in & \mathbb{B} && j = 1, \ldots, k && (5.83) \\
R(t) \in & \mathbb{N} && \forall\, t \in \mathcal{T}. && (5.84)
\end{aligned}$$

To compute the WCRT of a task we introduce binary variables $\ell(t, j) \in \mathbb{B}$ for $t \in \mathcal{T}$, $j = 0, \ldots, k$ with $\ell(t, j) = 1$ if and only if the WCRT $R(t)$ is in interval I_j, see (5.82). Since the intervals are a disjoint partition of $[0, \delta_k]$, at most one variable $\ell(t, j)$ can be equal to zero for scheduled tasks. This condition is implicitly given by (5.81), stating that the sum over all intervals of $\ell(t, j)$ is equal to p_t for every task $t \in \mathcal{T}$.

From Theorem 2.25 we know that the cumulative workload only increases if a scheduled task is initiated again, i.e., after its period. For this reason we introduce additional binary variables $\ell(j) \in \mathbb{B}$ for $j = 0, \ldots, k$ with $\ell(j) = 1$ if and only if there is a task with WCRT in the interval I_j, see (5.80) and (5.83).

The necessary and sufficient conditions of the workload demand analysis from Theorem 2.25 translate to (5.79) and (5.81). For the right bound δ_j of interval I_j, we sum over all tasks in earlier and the same interval, i.e., $i \leqslant j$, to obtain the cumulative workload up to δ_j that must be less than or equal to δ_j, if a task has its WCRT in interval I_j, otherwise the constraint is artificially satisfied with M' sufficiently large.

A lower bound to the WCRT for a fixed $j = 1, \ldots, k$ is obtained by summing the corresponding WCETs, see (5.77). The first term sums all WCETs according to their quantity until δ_j if the tasks have WCRTs in intervals $i < j$. The second term catches all tasks with WCRTs in interval j according to a higher priority than t. The last two terms consist of the tasks WCET in the case the task is selected, otherwise the constraint is artificially satisfied with M sufficiently large.

We remark that the constraints (5.77) have to be updated after every solving process of the master LP, since the ratio that affects the tasks' priority is modeled as a fixed parameter, but depends on the current value of the dual variables π_c for all chains $c \in \mathcal{C}$ with $t \in \mathrm{seq}(c)$.

Chapter 6

Experiments and Evaluation

This chapter presents the experiments executed with our column generation approach for DSP in several scenarios. Comparing the scheduling analyses, parameter settings, and realized strategies we show the performance of our implementation.

For comparison we introduce other approaches that we also use for generating start solutions in Section 6.2. Section 6.3 presents details about the set of problem instances we work on. The objectives of this work are evaluated by the experiments in Section 6.4 with progress and comparison charts explained in Section 6.1.

A particular conclusion is given after each experiment, whereas a complete summary of the results is given in Chapter 7.

6.1 Progress and Comparison Charts

The progress chart in Figure 6.1 shows the development of lower and upper bounds on the objective function over the running time in seconds.

The x-axis is not linearly scaled, because we show the development of the bounds in every node of the branch and bound tree of our algorithm. The single nodes are alternately colored in light red and white ( ) and separated by an offset for clarity.

A preceding heuristic phase is separated by a black vertical line ( ) from the exact phase. In the heuristic phase the nodes are processed in a depth first search

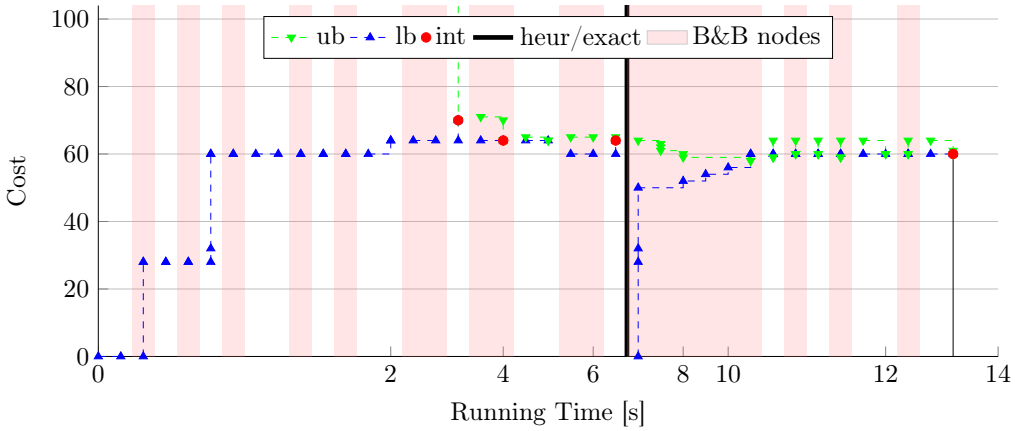


Figure 6.1: Exemplary progress chart.

This figure exemplary shows a progress chart of our algorithm with the development of lower and upper bounds on the objective function over the running time in seconds. It consists of a first heuristic phase and a subsequent exact phase.

order, in the exact phase the execution order is given by a best first search strategy by the lower bounds, see Section 4.4.5. The first root node of each phase generates global lower bounds, every other node of the branch and bound tree provides only local information that are also valid in its child nodes. The computed fractional upper bounds are still valid in the child nodes, whereas the detected integral solutions are globally valid.

Inside each branch and bound node, the improvement of the lower bound is plotted by a blue filled triangle pointing upwards (▲) connected by blue dashed line segments. The initial lower bound is 0.

The improvement of upper bounds inside a branch and bound node is plotted by a green filled triangle pointing downwards (▼) connected by green dashed line segments. The initial upper bound is given by the highest possible cost $\text{cost}(\hat{s}, \mathcal{T})$ of the artificial solution, see Section 4.5.2.

Figure 6.1 shows that nodes at the end of the heuristic phase can still have a weak lower bound, because they were pushed early into the priority queue with a small recursive depth. On the contrary in the exact phase, the lower bound behaves non-decreasingly due to the processing order.

If the lower bound is raised above the upper bound, the node starts branching with the new lower bound, unless there already is an integral solution with this objective value. This happens if the Lagrangian lower bound, see Section 4.5.1, provides a better lower bound after solving all pricing problems.

If the upper bound corresponds to an integral solution of the system, the point

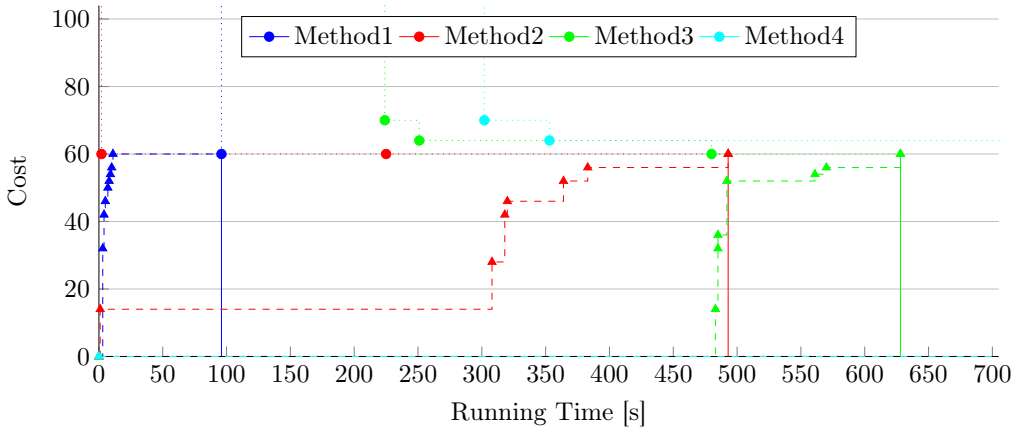


Figure 6.2: Exemplary comparison chart.
This figure exemplary shows a comparison chart of our algorithm with different analyses with the development of the global lower and upper bounds on the objective function over the running time in seconds. The fourth method exceeds the time limit with an improved upper bound, but an initial lower bound of zero.

is marked by a filled red dot (•).

In the heuristic phase a solution is marked as such whenever it is integral, whereas in the exact phase the upper bound is immediately reduced below the best integral solution to cut the search space, since we are only interested in one optimal solution. In some cases the result of the LP is already integral, then the solution is marked as well although it does not improve the objective function.

The comparison chart in Figure 6.2 shows the development of the global lower and upper bounds on the objective function over the running time in seconds for different analyses.

The x-axis is linearly scaled and the results inside the branch and bound nodes are only visible if their results are of global scope. A separation of the heuristic and exact phase of the analyses is not shown, since the point of time differs significantly for the different analyses.

Every analysis is plotted in a separate color, e.g., red, and the total running time is marked by a vertical line (|) pointing to the x-axis. Global lower bounds are plotted by a filled triangle pointing upwards (- -▲- -) connected by dashed line segments. Every global upper bound corresponds to an integral solution of the system, thus they are plotted by a filled dot (···●···) connected by dotted line segments.

6.2 Experimental Setup

Büker et al. [15, 16] and Thaden [55] describe a modular two-tier approach when solving variants of the DSP. In the first heuristic global phase the pre-allocation of tasks to processing subsystems is done mainly by utilization aspects. In underlying exact local phases schedulability and memory analyses are performed for subsystems and their processing ECUs to ensure that the allocation is executable. If there are remaining non-allocated tasks, called *odd set*, they are returned to the global phase and will be reallocated in subsequent iterations.

The local analyses are exchangeable respecting the schedulability policies and strategies. Althaus et al. [2] proposed a branch and bound algorithm solving LP relaxations by column generation that can be extended to a local analysis that provides odd sets for the two-tier approach. Moreover, the extension can minimize the overall cost of the computation in a subsystem if the tasks are given virtual cost values that conflict with the required modifications of the subsystem's hardware to keep the odd set small, respectively the cost minimal.

Thaden [55] introduced a *spare-time/MaxWCET* analysis that ensures the schedulability of the tasks and provides an odd set if non-allocated tasks remain due to a given cost limit for a subsystem. Due to preemption and the fact that pre-allocated tasks already received a priority for the execution order, the process of scheduling a new task has to consider two different aspects. On the one hand, the task receives a priority that causes all lower priority tasks to check their schedulability conditions since now they can be interrupted by this new task. This scenario is tackled by the spare-time analysis that computes the remaining computational capacity for a priority on each ECU so that the pre-allocated tasks remain schedulable. On the other hand, the task itself will be interrupted by all higher priority tasks that increase its WCRT. The MaxWCET analysis estimates the tasks maximal WCET with respect to the deadline.

In the subsequent experiments we examine variants of our column generation approach and often use the approach by Thaden with the following notations:

TwoTier Two-tier global/local approach with local spare-time/MaxWCET analysis, see Thaden [55].

ColGen-l Our column generation approach applying the workload analysis by Lehoczky, see Theorem 2.25, that performs DMS which is optimal in DSP-GF and non-sufficient in DSP-CF.

ColGen-e Our column generation approach applying the response time analysis by Eisenbrand et al., see Section 5.1.5.2, that performs optimal FPS in DSP-GF and DSP-CF.

ColGen-n Our column generation approach applying our new ILP formulation for computing WCRTs, see Section 5.3.4, that performs optimal FPS in DSP-GF and DSP-CF.

The experiments are performed on one of two six-core processors (Intel® Core™ i7–970 at 3,2 GHz) with 12 GB of physical memory running Linux with kernel version 3.8.0. The implementation of our algorithm was done in C++ and was compiled with GNU g++ 4.8.1 with the optimization flag -O3 turned on. For solving the generated LPs and ILPs we use the commercial solver Gurobi Optimizer 5.6.2 [36] working on all 6 available cores. In general, the time for solving an instance was limited to 3600s = 60min.

6.3 Problem Instances

The problem instances arise from a student project at the Carl von Ossietzky-University, Oldenburg, one of the research centers of AVACS [9], where the task was to develop a *Virtual Driver Assistance system (ViDAs)* [6] capable of automatically joining a two-lane motorway. Cars changing the lane use sensors to identify a sufficiently large gap between cars on the adjacent lane. This lane change assistant leads to an adaptive cruise control system that additionally checks the distances to traffic in front of the car.

The hardware architecture in the ViDAs context is given by two subsystems, each of them with a capacity of two ECUs. They are connected by a global FlexRay bus, whereas the local communication is handled via CAN buses of sufficient speed. Four ECU types of different memory capacity and cost are given, see Table 6.1. Initially, the two ECUs in every subsystem are preset to the cheapest ECU type by a requirement of the TwoTier approach, however empty hardware

Table 6.1: Hardware architecture of the instances.

This table shows the available (avail) and pre-set hardware configuration of a single subsystem in the ViDAs instances. Each subsystem can hold two ECUs which are preset to the ECU Type3, leading to initial costs of 28 per subsystem.

ECU type	Type0	Type1	Type2	Type3
Memory	30000	20000	15000	10000
Cost	38	28	18	14
ECU0	avail	avail	avail	preset
ECU1	avail	avail	avail	preset

Table 6.2: Initial task network of the instances.

This table shows the details of the initial task network of the ViDAs instances. One task of the first block is predeployed to each preset ECU, however the tasks of the second block can be allocated freely. The memory consumption of every task was set to 450 equally for every ECU type, so that the memory restrictions have minor impact.

Task	WCET				Deadline	Period
	Type0	Type1	Type2	Type3		
TaskI0	240	320	480	680	4000	4000
TaskI1 – I3	240	320	480	1190	6000	6000
TaskI4	240	320	480	850	6000	6000
TaskI5	240	320	480	900	6000	6000
TaskI6 – I7	240	320	480	1000	6000	6000
TaskI8 – I19	240	320	480	1190	6000	6000

architectures are also manageable by our algorithm.

The task network in the ViDAs context consists of 20 artificial base load tasks, because in practical applications most systems come with predefined tasks already allocated to some computing units. Due to requirements of the TwoTier approach, each of the preset ECUs has exactly one task predeployed to it. These 20 initial base load tasks are given in detail in Table 6.2.

The specification model in the ViDAs case study was developed in MATLAB® Simulink and has to be partitioned into software parts defining tasks and messages with information about WCRTs, deadlines, periods, and memory consumption. This task creation process is part of the design flow of developing embedded systems and is out of focus of this work, see Büker et al. [14–16] for more information on this topic. The resulting task networks build the instances for the experiments in this thesis.

Five problem instances were generated consisting of the initial base load tasks, see Table 6.2, and different compilations of the additional tasks from the task creation process, see Table 6.3.

ProblemA is given by 7 additional tasks, ProblemB consists of two copies of the additional tasks of ProblemA. By task creation processes with different parameters both ProblemC is built with additional 13 tasks and ProblemD is generated with additional 10 tasks. Taking together all different additional tasks from ProblemA, ProblemC, and ProblemD, the largest ProblemE consists of additional $7 + 13 + 10 = 30$ tasks.

Table 6.3: Additional task networks of the instances.

This table shows the details of the additional task networks of the ViDAs instances. The first block forms ProblemA and doubled ProblemB, the second block builds ProblemC, and the third block generates ProblemD. ProblemE is generated by all three blocks together. The memory consumption of every task was set to 0 equally for every ECU type, so that the memory restrictions have no impact.

Task	WCET				Deadline	Period
	Type0	Type1	Type2	Type3		
TaskA0	611	844	925	1003	5000	5000
TaskA1	98	123	112	124	5000	5000
TaskA2	29	30	18	22	5000	5000
TaskA3	403	455	175	205	5000	5000
TaskA4	261	284	481	510	5000	5000
TaskA5	505	706	751	813	5000	5000
TaskA6	7	7	4	5	5000	5000
TaskC0	382	562	626	677	5000	5000
TaskC1	14	14	7	9	5000	5000
TaskC2	56	64	58	64	5000	5000
TaskC3	181	211	216	237	5000	5000
TaskC4	80	98	99	108	5000	5000
TaskC5	95	111	109	119	5000	5000
TaskC6	29	30	18	22	5000	5000
TaskC7	385	437	171	201	5000	5000
TaskC8	212	233	470	498	5000	5000
TaskC9	227	313	298	322	5000	5000
TaskC10	340	458	472	512	5000	5000
TaskC11	18	20	9	11	5000	5000
TaskC12	7	7	4	5	5000	5000
TaskD0	606	839	923	1001	5000	5000
TaskD1	113	138	168	186	5000	5000
TaskD2	10	10	5	6	5000	5000
TaskD3	380	435	167	197	5000	5000
TaskD4	188	205	445	471	5000	5000
TaskD5	22	27	10	12	5000	5000
TaskD6	21	26	11	13	5000	5000
TaskD7	252	340	343	372	5000	5000
TaskD8	273	380	413	446	5000	5000
TaskD9	7	7	4	5	5000	5000

Table 6.4: Sizes of the task networks of the instances.
This table shows the sizes of the examined instances consisting of the initial and additional tasks of the ViDAs instances.

Instance	Tasks		
	initial	additional	total
ProblemA	I0 – I19	A0 – A6	27
ProblemB	I0 – I19	A0 – A6 A0 – A6	34
ProblemC	I0 – I19	C0 – C12	33
ProblemD	I0 – I19	D0 – D9	30
ProblemE	I0 – I19	A0 – A6 C0 – C12 D0 – D9	50

6.4 Objectives and Evaluation

The three major objectives introduced in Section 1.3 are examined in Sections 6.4.1 – 6.4.3. We show the performance and advantages of our algorithm by performing relevant experiments and revealing differences to existing approaches.

6.4.1 Assessing Quality of Solutions by Providing Lower Bounds

Providing good lower bounds on the objective value allows for assess the quality of heuristic solutions that come with no measurement of quality.

We follow the attempt to apply necessary but non-sufficient scheduling policies to estimate a lower bound with our ColGen-l approach. Nevertheless we also apply the more complex scheduling policies ColGen-e and ColGen-n, since they also provide good lower bounds during the optimization process due to the LP relaxation that is solved to optimality in every branch and bound node.

In this experiment we compare the results of the heuristic TwoTier approach with our algorithm. We use our implementation to provide a lower bound on the objective value to assess the quality of externally computed solutions. Therefore, we apply ColGen-l with the workload analysis by Lehoczky which performs the DMS policy, see Theorem 2.25.

The allocation of priorities by DMS is only optimal in DSP-GF, see Section 3.7,

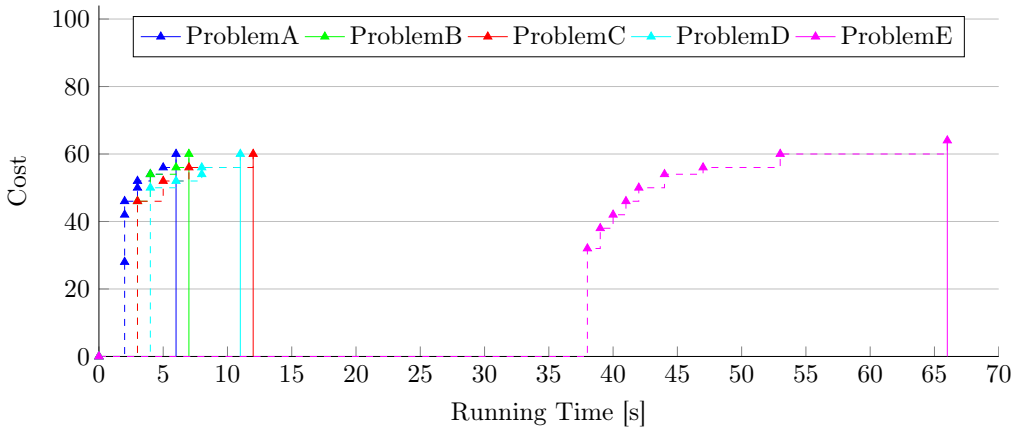


Figure 6.3: Evaluation of lower bounds with ColGen-l.
The chart shows the evaluation of the lower bound when applying ColGen-l to the different problem instances.

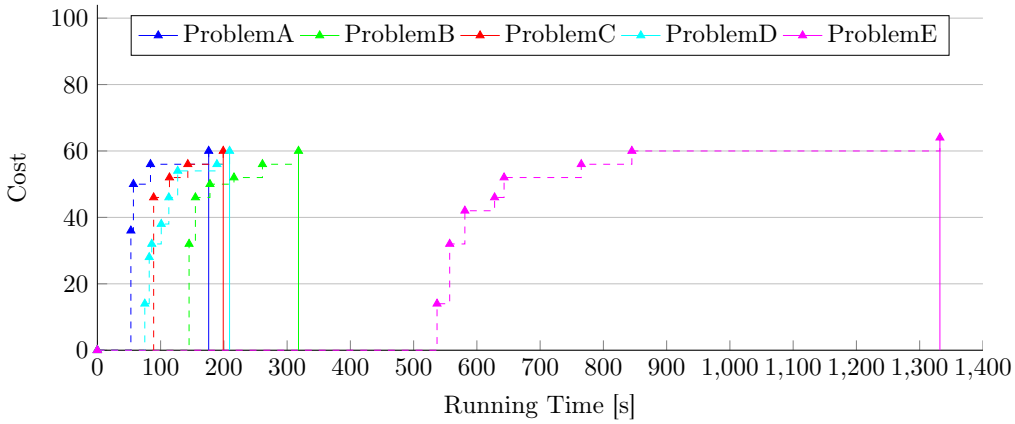


Figure 6.4: Evaluation of lower bounds with ColGen-n.
The chart shows the evaluation of the lower bound when applying ColGen-n to the different problem instances.

since chains with end-to-end deadlines influence the value of the WCRT and the DMS policy does not necessarily remain optimal. However, even in DSP-CF the DMS policy provides safe lower bounds with the benefit of smaller running times and the same quality of lower bounds in the test cases compared to the approaches ColGen-e and ColGen-n which compute the WCRTs.

This method is also applied when we compute optimal allocations as a presolve process, since it provides safe lower bounds in small running times as opposed to starting without a good lower bound, see Section 6.4.3.2.

Table 6.5: Results in providing lower bounds only.
We show the result serving as upper bound (ub) and running time in seconds for the TwoTier approach, compared to the lower bounds (lb) of ColGen-l, ColGen-e, and ColGen-n. The time limit was 3600s.

Instance	TwoTier		ColGen-l		ColGen-e		ColGen-n	
	ub	time	lb	time	lb	time	lb	time
ProblemA	60	20s	60	6s	0	–	60	176s
ProblemB	64	21s	60	7s	0	–	60	318s
ProblemC	60	75s	60	12s	0	–	60	199s
ProblemD	60	37s	60	11s	0	–	60	209s
ProblemE	–	–	64	66s	0	–	64	1332s

In Table 6.5 we show the results in providing lower bounds only. The TwoTier approach provides a not necessarily optimal solution to the problem, while our algorithm only solves the root node to optimality. We present the computed lower bounds on the objective function and running times with DMS policy in ColGen-l and WCRT computation with the two formalizations ColGen-e and ColGen-n. All ColGen approaches use multiple columns generation, so that up to 20 columns can enter the master problem per subsystem, see Section 4.6.2.

The results in Table 6.5 show that ColGen-l is able to provide safe lower bounds in similar time as the heuristic TwoTier approach provides upper bounds. Also the WCRT approach ColGen-n does provide the same lower bounds in all instances, but consumes up to the 45-fold running time. However, the approach ColGen-e turns out to be even more time consuming and does not improve the initial lower bound of 0 before reaching the given time limit of 3600s. Especially for ProblemE the heuristic approach only produced partial solutions that did not allocate all tasks.

The evaluation of lower bounds is shown in Figure 6.3 when applying ColGen-l and in Figure 6.4 for the ColGen-n implementation. Since ColGen-e does not provide lower bounds within the time limit, the results are not shown.

Conclusion The results show that ColGen-l is capable of providing lower bounds faster than TwoTier provides heuristic solutions. This property is used again in Section 6.4.3.2 to develop a hybrid approach for DSP. Comparing the approaches ColGen-e and ColGen-n to compute WCRTs, we see that the formulation in ColGen-n turns out to be faster computable than in ColGen-e. Summarizing, both approaches ColGen-l and ColGen-n compute the correct lower bounds in reasonable running times. All results are shown in Table 6.5.

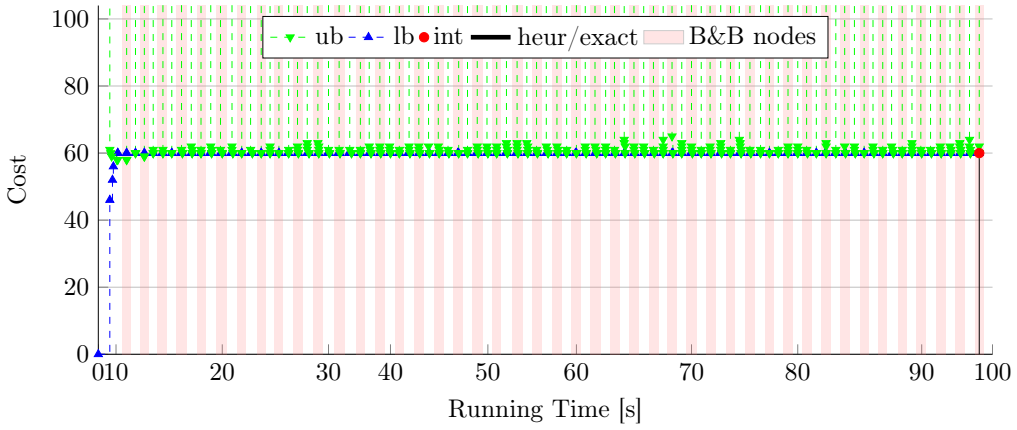


Figure 6.5: Progress chart with exact phase only. ColGen-l applied to ProblemC without a heuristic phase.

6.4.2 Providing Global Optimal Allocations

The results of Section 6.4.1 can be seen as a byproduct of the optimization process when searching for an optimal solution that we examine in this section.

We have to distinguish between the two scenarios DSP-GF, where the DMS policy is sufficient, see Section 6.4.2.1, and DSP-CF with constraints on chains where the WCRTs have to be computed, see Section 6.4.2.3. Nevertheless we present the results if we apply the more complex scheduling policies with WCRT computation on DSP-GF in Section 6.4.2.2. We deal with the expected larger running times by applying several strategies in the branch and bound process as well as in the column generation process.

All following experiments use multiple columns generation with up to 20 columns per subsystem, see Section 4.6.2. If a heuristic phase is applied to a ColGen approach, it is either aborted after providing three integral solutions or after 40 processed branch and bound nodes at the latest, see Section 4.6.3.

6.4.2.1 DSP-GF with the DMS policy ColGen-l

When applying the DMS policy by Lehoczky et al. we expect small running times due to the ILP formulations described in Section 5.1.5.3 that are similar to multi-dimensional knapsack constraints. Exemplary we show the progress charts of our algorithm ColGen-l with several strategies applied to ProblemC, as described in Section 4.6. The scheduling of the messages is neglected to emphasize the performance of the scheduling strategies.

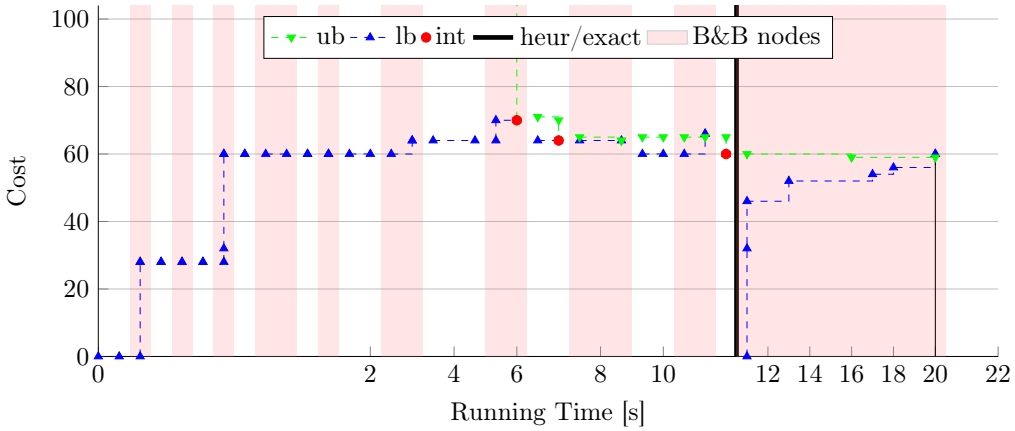


Figure 6.6: Progress chart with heuristic and exact phase.
ColGen-l applied to ProblemC with a first heuristic phase.

In Figure 6.5 we see the progress of ColGen-l if we only apply the exact phase of our algorithm ColGen-l. Due to a missing integral solution the algorithm traverses a lot of branch and bound nodes, although the lower bound is already optimal in the first root node. An optimal allocation with objective value 60 is not found until after 95s.

In Figure 6.6 we show the result if we slot a heuristic phase, see Section 4.6.3, ahead of the exact one, separated by the black vertical line (|). We see that a first integral solution of value 70 is found in a branch and bound node after 6s, a second of value 64 after 7s, and a last one of objective value 60 around 11s. In the root node of the exact phase the lower bound raises up to 60 and the algorithm terminates with an optimal allocation of value 60 after 20s.

The observed behaviour is that the integral solutions found by the heuristic phase can reduce the upper bound and cut the search space for the exact phase. In this case aborting the heuristic phase after three encountered integral solutions turns out to be a good parameter choice. In Table 6.6 we present the results for all instances. Unfortunately, we see a doubling in running time for ProblemE when applying a previous heuristic phase.

In Figure 6.6 we show what benefit is reachable by applying several heuristic strategies. In the heuristic phase on the left we see that the lower bound does not change in the first heuristic root node, whereas in Figure 6.5 it increases up to 60 by the price of a larger resting time in the first branch and bound node.

The strategy to terminate the solving process of a branch and bound node if the absolute error κ_{abs} remains constant for several iterations (here five) is also applied in the heuristic phase. It is clearly visible that there is no improvement

Table 6.6: Results in providing optimal allocations in DSP-GF. We show the objective value (val) and running time in seconds for ColGen-l without (w/o) and with (w) a first heuristic phase for DSP-GF. The time limit was 3600s.

Instance	ColGen-l w/o		ColGen-l w	
	val	time	val	time
ProblemA	60	28s	60	35s
ProblemB	60	168s	60	207s
ProblemC	60	95s	60	20s
ProblemD	60	55s	60	50s
ProblemE	64	1707s	64	3414s

in the lower nor the upper bound in the first nodes, thus the branching is pushed on in Figure 6.6. In the subsequent exact phase, only the root node is solved to raise the global valid lower bound to the already known integral solution of value 60.

Conclusion The results show that ColGen-l is capable of quickly providing optimal solutions in the scenario of DSP-GF. The application of a heuristic phase with the chosen parameters does not pay off in every instance, especially not for ProblemE, since ColGen-l without the heuristic phase turns out to solve the instances to optimality even faster or equally fast. All results are shown in Table 6.6.

6.4.2.2 DSP-GF with WCRT computation ColGen-e and ColGen-n

If we apply more complex scheduling strategies than in ColGen-l, the pricing problems become more difficult to solve. We compare the two described formulations ColGen-e with the WCRT computation by Eisenbrand et al., see Section 5.1.5.2 and our new ILP formulation ColGen-n, see Section 5.3.4. Both approaches apply a heuristic phase before the exact computation.

In Figure 6.7 we see the progress of ColGen-e applied to ProblemC, exemplary. In the heuristic phase a first integral solution of value 64 was found after 2243s, a second and third of value 60 were found after 2813s, respectively 2899s, the latter started the subsequent exact phase. The lower bound was raised up to the value of 42 in the root node before the time limit of 3600s. Summarizing, optimal solutions have been detected, but the proof of optimality is missing, however the

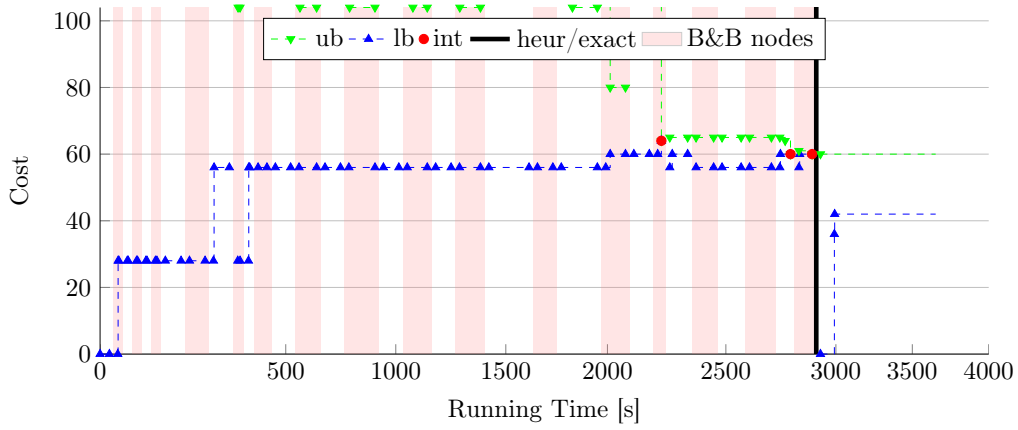


Figure 6.7: Progress chart with WCRT computation. ColGen-e applied to ProblemC with a first heuristic phase.

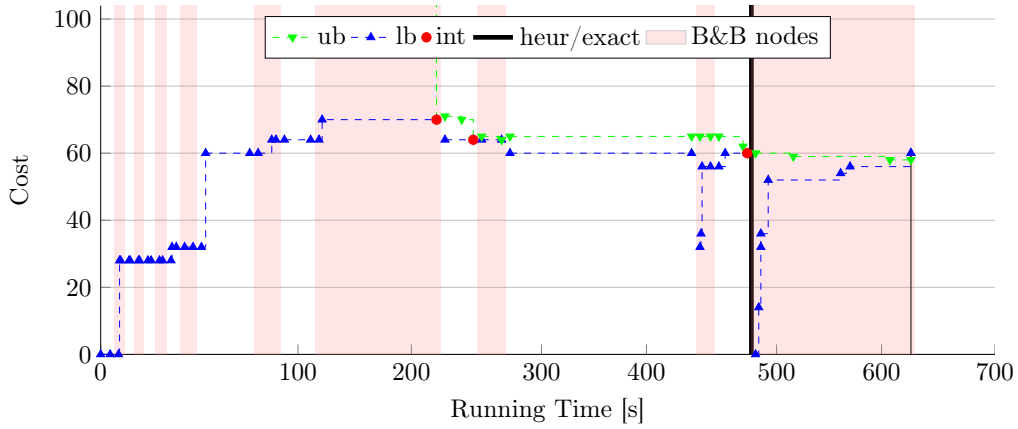


Figure 6.8: Progress chart with WCRT computation. ColGen-n applied to ProblemC with a first heuristic phase.

solution comes with an improved lower bound.

The drawback of the approach of Eisenbrand et al. is the complex formulation leading to large running times in the pricing ILPs. Figure 6.8 shows the progress of ColGen-n applied to the same ProblemC. Obviously the first integral solutions in the heuristic phase have values of 70 and 64, but the third one reaches the optimum value of 60. Due to faster computations the heuristic phase aborts after 480s and the root node of the subsequent exact phase proves the optimality after 628s.

Table 6.7: Results in providing optimal allocations in DSP-GF with WCRTs. We show the objective value (val), respectively the corresponding lower and upper bound, and running time in seconds for ColGen-e and ColGen-n, both apply a first heuristic phase and the described strategies of Section 4.6 for DSP-GF with the additional computation of WCRTs. The time limit was 3600s.

Instance	ColGen-e		ColGen-n	
	val	time	val	time
ProblemA	[0,60]	–	60	295s
ProblemB	[42,64]	–	[60,74]	–
ProblemC	[42,60]	–	60	628s
ProblemD	[0,60]	–	60	1109s
ProblemE	[0,152]	–	[0,104]	–

Conclusion The results show that our new ILP formulation is advantageous over the formulation of Eisenbrand et al. in these instances, even if it does not solve all instances to the optimum in the time limit of 3600s. In the instances where both methods cannot provide the optimum within the time limit, ColGen-n provides strictly tighter bounds on the objective value than ColGen-e. All results are shown in Table 6.7.

6.4.2.3 DSP-CF with WCRT computation ColGen-e and ColGen-n

We consider DSP-CF with the complex linear constraints in the master problem, see Section 5.3.2, that links the results of several pricing problems together. These combining constraints carry the values of the WCRTs of tasks and messages as linear coefficients, thus the pricing problems need to report on these data by application of ColGen-e or ColGen-n.

In the ViDAs context, the instances also bring chains with end-to-end deadlines consisting of sequences with up to eight tasks. We neglect both the initial tasks TaskI0 – TaskI19 of Table 6.2 as well as signals, and halve every end-to-end deadline to increase the pressure on the scheduling requirements by the chains. The preset ECU assignment necessary for the TwoTier approach is also discarded.

Both compared formulations ColGen-e with the WCRT computation by Eisenbrand et al., see Section 5.1.5.2 and our new ILP formulation ColGen-n, see Section 5.3.4, apply the heuristic phase before the exact phase.

In Figure 6.9 we show the progress results for lower and upper bounds if we

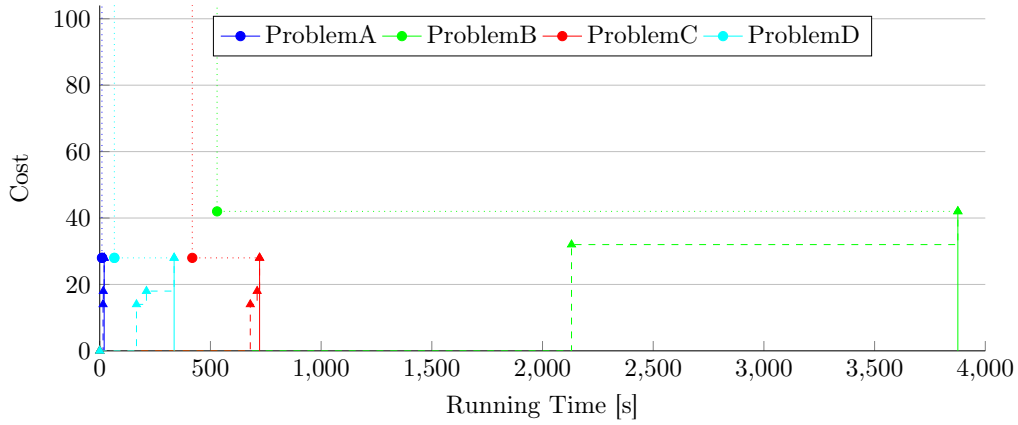


Figure 6.9: Comparison chart for ColGen-e for chains in DSP-CF. Lower and upper bounds for ColGen-e applied to ProblemA – ProblemD with a heuristic phase for DSP-CF with chains.

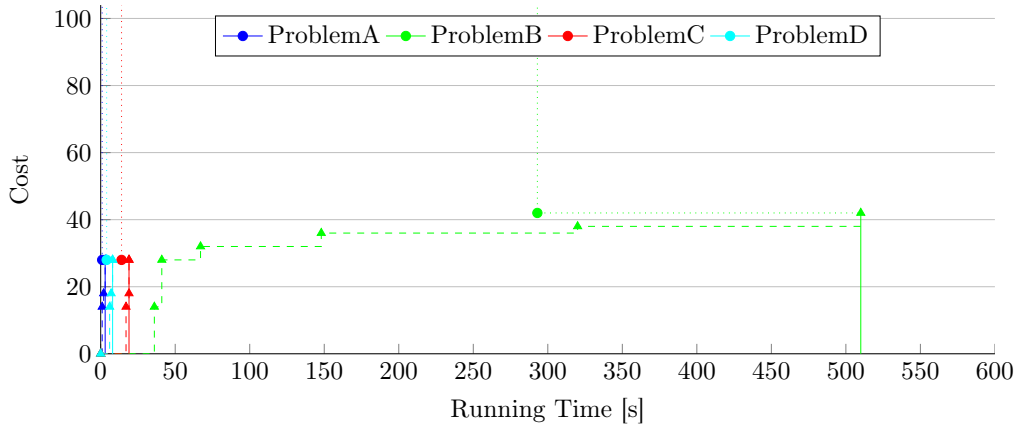


Figure 6.10: Comparison chart for ColGen-n for chains in DSP-CF. Lower and upper bounds for ColGen-n applied to ProblemA – ProblemD with a heuristic phase for DSP-CF with chains.

apply ColGen-e to ProblemA – ProblemD. For comparison, Figure 6.10 shows the progress results with our new ILP formulation in the approach ColGen-n. We see a similar performance of the bounds in both approaches with shorter running times for ColGen-n.

Exemplary, we also present the progress charts for both approaches applied to ProblemA. In Figure 6.11 we see that ColGen-e spends more time in processing the branch and bound nodes, compared to ColGen-n, see Figure 6.12. As the time for solving the master LP relaxations is almost negligible, the major part of the running time is spend in solving the pricing ILPs.

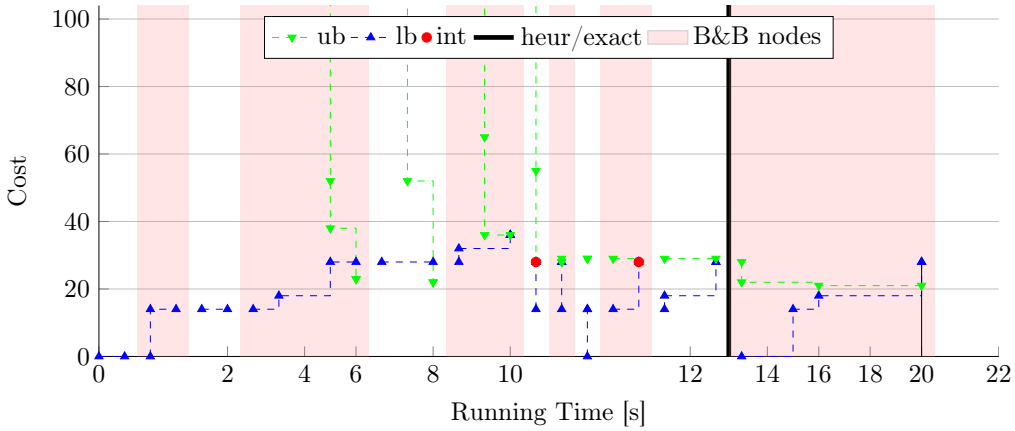


Figure 6.11: Progress chart for ColGen-e in DSP-CF.

ColGen-e applied to ProblemA in the scenario of DSP-CF with a first heuristic phase before the exact phase terminates after the computation of the root node.

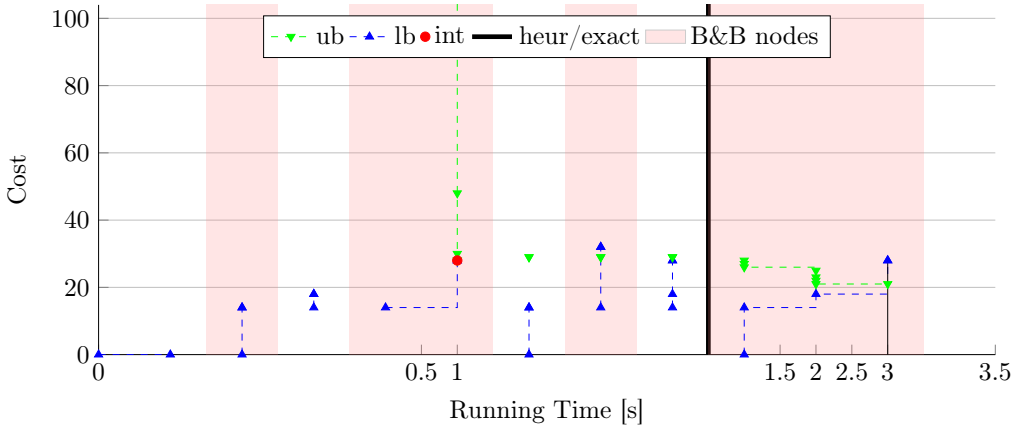


Figure 6.12: Progress chart for ColGen-n in DSP-CF.

ColGen-n applied to ProblemA in the scenario of DSP-CF with a first heuristic phase before the exact phase terminates after the computation of the root node.

We remark that in both Figures 6.11 and 6.12 we see that the heuristic phase runs out of branching possibilities due to the pursued heuristic branching strategy. Since we try to predeploy as many tasks as possible to the same subsystem in one branching decision we do not generate the complete branching tree, see Section 4.6.3 for details.

Table 6.8: Results in providing optimal allocations in DSP-CF.
We show the objective value (val), respectively the corresponding lower and upper bound, and running time in seconds for ColGen-e and ColGen-n, both apply first heuristic phases and the described strategies of Section 4.6 for DSP-CF containing chains with an end-to-end deadline. The initial tasks of Table 6.2 are neglected and the time limit was 7200s.

Instance	ColGen-e		ColGen-n	
	val	time	val	time
ProblemA	28	20s	28	3s
ProblemB	42	3876s	42	510s
ProblemC	28	721s	28	19s
ProblemD	28	336s	28	8s
ProblemE	[0,152]	–	[0,152]	–

Conclusion In Table 6.8 we present all results and observe that our new ILP formulation is advantageous over the formulation of Eisenbrand et al. in these instances. Due to more complex formulations in the pricing ILPs, ColGen-e consumes more running time in solving a branch and bound node.

6.4.3 Development of Hybrid Approaches

The existing two-tier approaches with heuristic algorithms and our global optimal approach point in different directions concerning the trade-off between running time and optimality.

The observations is that lower bounds can assess the quality of heuristic approaches, whereas heuristic solutions can reduce the search space of LP approaches by providing upper bounds. To achieve a compound benefit, we combine start solutions of heuristic approaches with our algorithm seeking for an optimal solution in DSP-GF.

All following experiments use multiple columns generation with up to 20 columns per subsystem, see Section 4.6.2. If a heuristic phase is applied to a ColGen approach, then it is aborted after providing three integral solutions or after 40 processed branch and bound nodes at the latest, see Section 4.6.3.

The heuristic start solutions are generated by TwoTier with the details given in Table 6.9. All five heuristic solutions function as input for the ColGen approaches, even if TwoTier does not provide a complete allocation of all tasks for ProblemE.

Table 6.9: Results of TwoTier in providing start solutions.

We show the objective value (val) and running time in seconds for the TwoTier approach. For ProblemE the TwoTier approach only provides incomplete, partial allocations marked with a prime symbol (′). The time limit was 3600s.

Instance	TwoTier		
	values	times	avg. time
ProblemA	60, 60, 60, 64, 64	15s, 26s, 3s, 10s, 10s	13s
ProblemB	64, 64, 64, 64, 64	26s, 25s, 21s, 32s, 35s	28s
ProblemC	60, 60, 60, 60, 60	69s, 112s, 69s, 126s, 25s	80s
ProblemD	60, 60, 60, 60, 60	43s, 38s, 39s, 46s, 47s	43s
ProblemE	56′, 56′, 56′, 56′, 56′	761s, 47s, 13s, 54s, 7s	177s

6.4.3.1 Including heuristic start solutions

The following experiments show the advantage of including the heuristic start solutions by TwoTier from Table 6.9 into our column generation approaches.

In Table 6.10 we show the results of ColGen-l, ColGen-e and ColGen-n both without start solutions and with start solutions. Except for the execution of ColGen-l for ProblemB and ColGen-e that did not terminate within the time limit of 3600s, the provision of heuristic start solutions to the ColGen approaches is advantageous in terms of shorter running times.

We emphasize that the execution of ColGen-l for ProblemE even benefits from the partial heuristic start solutions by TwoTier. The ColGen approach can use these incomplete assignments as long as the subsets of tasks are schedulable, since they represent a valid pattern variable for the master LP.

Exemplary, we show the results when applying ColGen-n to ProblemC in comparison to the previous results without heuristic start solutions, see Figure 6.13. We see that the integral heuristic start solution is detected at the start of the heuristic phase and interestingly leads to a direct raise of the lower bound to the value 14. ColGen-n without heuristic start solutions detects the first integral solutions of values 70, 64, and 60 after 224s, 251s, and 480s, whereas the lower bound closes the gap to the upper bound after 628s.

Additionally, we show ProblemC if ColGen-e is applied, see Figure 6.14, where both approaches do not reach the optimal objective value of 60 before the time limit of 3600s. We remark that ColGen-e without heuristic start solutions provides a better lower bound of 42, compared to the lower bound of 28 from ColGen-e

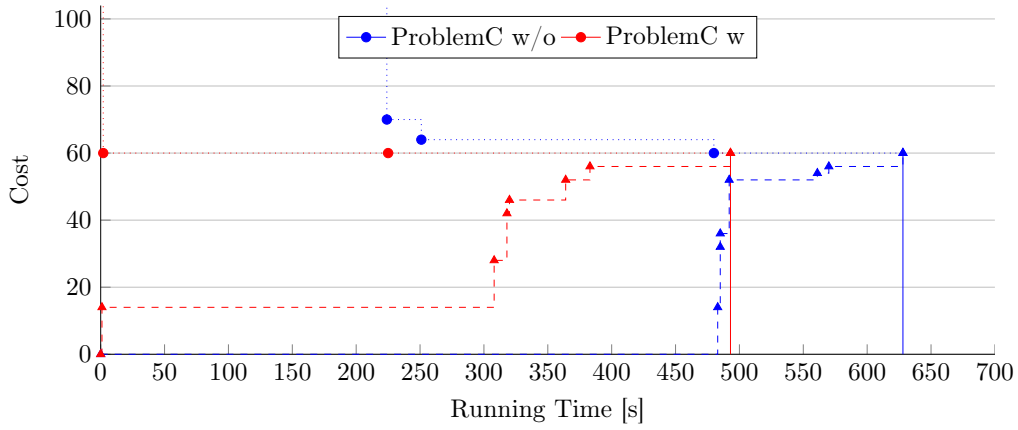


Figure 6.13: Comparison chart for ColGen-n with heuristic start solutions. ColGen-n applied to ProblemC with heuristic start solutions (red) provided by TwoTier and without (blue).

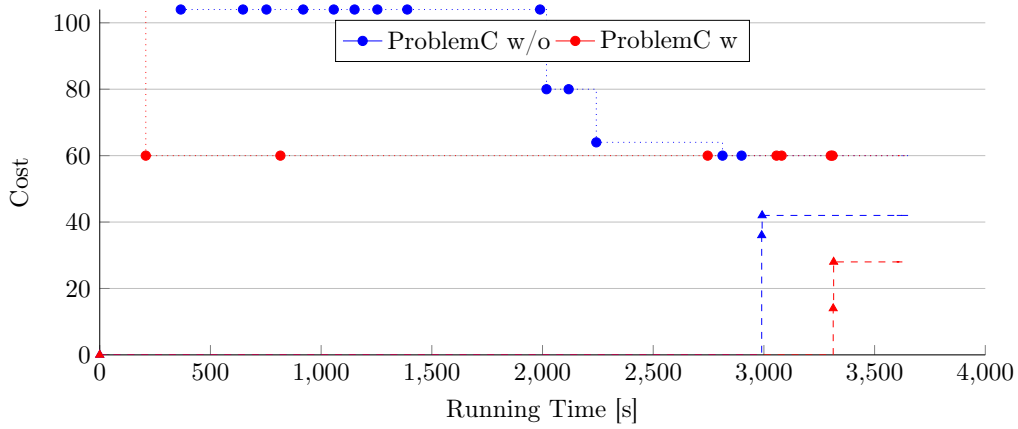


Figure 6.14: Comparison chart for ColGen-e with heuristic start solutions. ColGen-e applied to ProblemC with heuristic start solutions (red) provided by TwoTier and without (blue).

with heuristic start solutions. This happens, because ColGen-e without heuristic start solutions finishes its heuristic phase after 2899s and starts to improve the lower bound in the root node of the exact phase. ColGen-e with heuristic start solutions reaches the exact phase after 3310s and runs out of time.

Table 6.10: Results of ColGen with heuristic start solutions in DSP-GF.
We show the average running time in seconds for the TwoTier approach, and the running time in seconds for the approaches ColGen-l, ColGen-e and ColGen-n both without (w/o) heuristic start solutions and with (w) heuristic start solutions. If an approach does not terminate with the optimal objective value before the time limit of 3600s, we present the corresponding lower and upper bound.

Instance	TwoTier	ColGen-l		ColGen-e		ColGen-n	
	time	time		time		time	
	avg.	w/o	w	w/o	w	w/o	w
ProblemA	13s	35s	13s	[0,60]	[0,60]	295s	521s
ProblemB	28s	207s	180s	[42,64]	[0,64]	[60,74]	[60,74]
ProblemC	80s	20s	24s	[42,60]	[28,60]	628s	493s
ProblemD	43s	50s	13s	[0,60]	[14,60]	1109s	396s
ProblemE	177s	3414s	1129s	[0,152]	[0,152]	[0,104]	[60,84]

Conclusion The results show that almost every approach benefits from provided heuristic start solutions. Even if partial heuristic start solutions are provided to the column generation approach, it can recombine these in the master LP to valid full solutions. This property can be observed by the improved bounds in the application of ColGen-n with heuristic start solutions for ProblemE. All results are shown in Table 6.10.

6.4.3.2 Additional presolve phase

We discussed the provision of lower bounds from necessary but non-sufficient scheduling policies, like DMS, in Section 6.4.1. Since the quickly computed lower bound of ColGen-l is still valid if we apply ColGen approaches for WCRT computation, we make use of these lower bounds in a first additional presolve phase.

To illustrate our algorithmic strategy, we exemplary examine ProblemA, see Figure 6.15. We start our algorithm with a first presolve phase that computes a lower bound by applying ColGen-l with DMS policy. Here we increase the initial lower bound up to the value of 60 after 6s.

Separated by the left black vertical line (|) in Figure 6.15, the next step is the heuristic phase which already uses the lower bound of the presolving. The exploration of the heuristic branch and bound tree provides an integral solution of value 60 after 169s. Due to the tight lower bound of the presolving our algorithm terminates with an optimal solution without performing an exact phase.

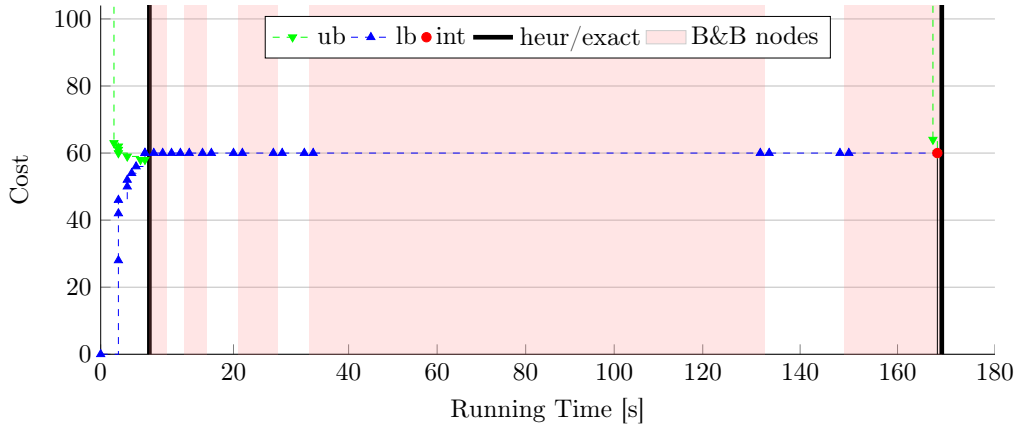


Figure 6.15: Progress chart of the presolve and heuristic phase.

The first presolve phase performs ColGen-l with DMS policy, then the heuristic phase performs ColGen-n, here applied to ProblemA.

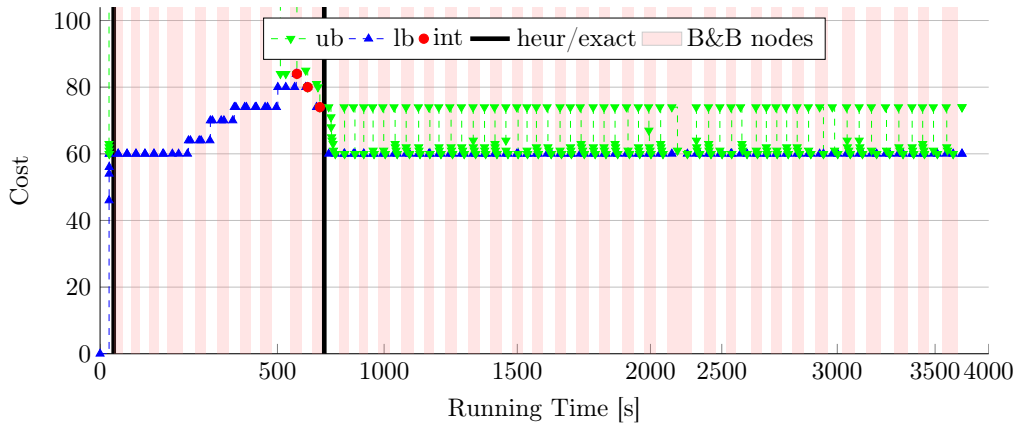


Figure 6.16: Progress chart of the presolve, heuristic and exact phase.

Applied to ProblemB, the first presolve phase performs ColGen-l with DMS policy, then the heuristic phase performs ColGen-n and reduces the upper bound, but the last exact phase does not close the gap within the time limit of 3600s.

Unfortunately, this behaviour is not guaranteed. If the heuristic phase does not close the gap from integral solutions to the lower bound, we have to apply a last exact phase to find the optimal solution. Exemplary, Figure 6.16 shows ColGen-n applied to ProblemB with a weaker heuristic phase that provides strictly better upper bounds, but the optimal solution of value 60 is not found within the time limit.

In Table 6.11 we see that almost every instance was solved to optimality even by ColGen-e with the additional presolving. Obviously, the presolve phase makes no

Table 6.11: Results of ColGen with presolve phase in DSP-GF.
We show the average running time in seconds for the TwoTier approach, and the running time in seconds for the approaches ColGen-l, ColGen-e and ColGen-n both without (w/o) presolve phase and with (w) presolve phase. If an approach does not terminate with the optimal objective value before the time limit of 3600s, we present the corresponding lower and upper bound.

Instance	TwoTier	ColGen-l		ColGen-e		ColGen-n	
	time	time		time		time	
	avg.	w/o	w	w/o	w	w/o	w
ProblemA	13s	35s	35s	[0,60]	1947s	295s	169s
ProblemB	28s	207s	207s	[42,64]	3099s	[60,74]	[60,74]
ProblemC	80s	20s	20s	[42,60]	2071s	628s	499s
ProblemD	43s	50s	50s	[0,60]	3495s	1109s	879s
ProblemE	177s	3414s	3414s	[0,152]	[64,152]	[0,104]	[64,104]

difference in ColGen-l since it applies DMS policy anyway.

Conclusion The presolve phase with the necessary but non-sufficient DMS policy applied by ColGen-l helps tremendously in reducing the running time. All approaches gain from the good lower bound provided by the presolving. The results for all instances are shown in Table 6.11.

6.4.3.3 Heuristic start solutions and presolve phase

The advantages of heuristic start solutions from Section 6.4.3.1 and of the presolve phase from Section 6.4.3.2 are mergeable. If we pursue this strategy, the remaining task is to close the gap by raising the lower bound on the one hand, or computing cost minimal integral solutions on the other hand. Combining both strategies leads to a major decrease in running time.

In Figure 6.17 we exemplary show the progress of ColGen-n applied to ProblemA, where the heuristic start solutions of TwoTier are provided to our algorithm and a presolve phase is applied. After the presolve phase of about 7s the lower bound already reached the objective value of 60. With the heuristic start solutions of value 60, the optimal value is reached already after 7s.

We also present the progress chart of ProblemB if ColGen-n is applied. After 8s the presolving computed the lower bound of value 60, then the heuristic phase detects integral solutions of the values 84, 80, and 74 until the exact phase starts

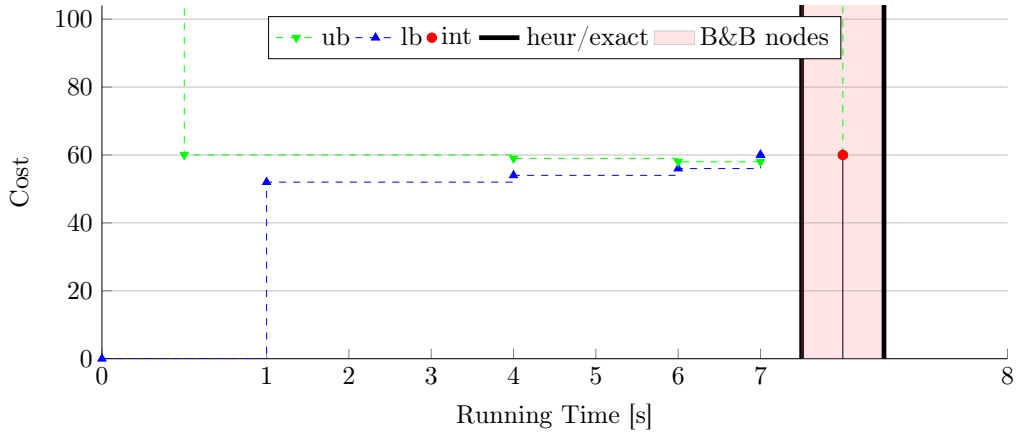


Figure 6.17:
Progress chart of the presolve and heuristic phase with heuristic start solutions. The first presolve phase performs ColGen-l with DMS policy, then the heuristic phase performs ColGen-n, here applied to ProblemA. Both phases make use of the provided heuristic start solutions of TwoTier.

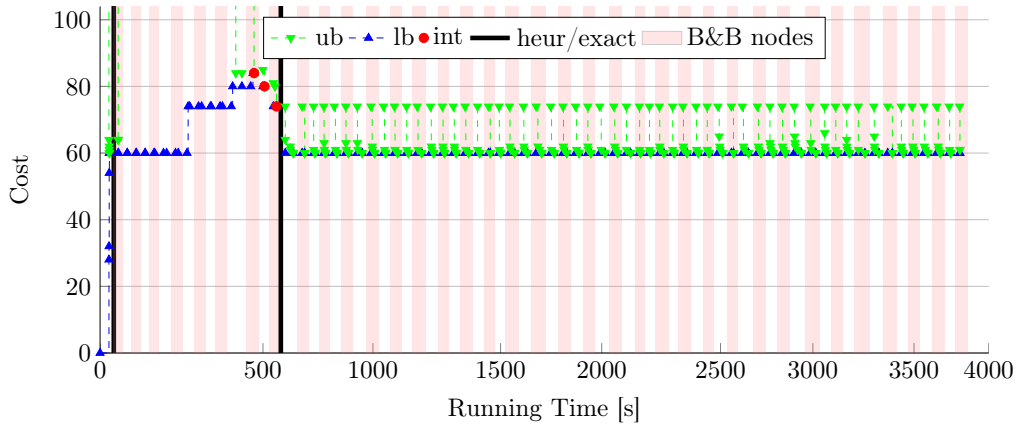


Figure 6.18:
Progress chart of the presolve and heuristic phase with heuristic start solutions. The first presolve phase performs ColGen-l with DMS policy, then the heuristic phase performs ColGen-n, here applied to ProblemB. Both phases make use of the provided heuristic start solutions of TwoTier.

after 576s. The processing of the branch and bound tree of the exact phase does not lead to an improvement of the bounds $[60, 74]$ for ProblemB.

Table 6.12:
Results of ColGen with presolve phase and heuristic start solutions in DSP-GF. We show the average running time in seconds for the TwoTier approach, and the running time in seconds for the approaches ColGen-l, ColGen-e and ColGen-n both without (w/o) heuristic start solutions and presolving, and with (w+w) heuristic start solutions and presolving. If an approach does not terminate with the optimal objective value before the time limit of 3600s, we present the corresponding lower and upper bound.

Instance	TwoTier	ColGen-l		ColGen-e		ColGen-n	
	time	time		time		time	
	avg.	w/o	w+w	w/o	w+w	w/o	w+w
ProblemA	13s	35s	13s	[0,60]	5s	295s	7s
ProblemB	28s	207s	180s	[42,64]	[60,64]	[60,74]	[60,74]
ProblemC	80s	20s	24s	[42,60]	16s	628s	8s
ProblemD	43s	50s	13s	[0,60]	8s	1109s	8s
ProblemE	177s	3414s	1129s	[0,152]	[64,152]	[0,104]	[64,84]

Conclusion In the cases an optimal solution was found before the time limit of 3600s, all ColGen approaches reach a major speed-up with this hybrid approach. Interestingly, the approaches that compute WCRTs ColGen-e and ColGen-n gain more than ColGen-l that performs DMS. The results for all instances are given in Table 6.12.

Chapter 7

Conclusion

This chapter begins with a short summary of the results of the examined experiments from Chapter 6 in Section 7.1. The results are then discussed in Section 7.2 where we also identify ideas for further directions of research.

7.1 Summary of the Results

To summarize, our developed algorithm is capable of solving the DSP in its both variants. Due to the safety-critical origin of our application, the main focus in the development was on optimality, rather than on competing with fast heuristic approaches. We showed that all necessary real-time requirements are expressible with linear formulations, including the hardware architecture, the periodic schedulability of tasks, and the transmission of messages via the global bus. Our new ILP formulation for computing WCRTs turns out to be advantageous in the considered scenario of chains with end-to-end deadlines.

The subsequent summary of our results is divided into the three major objectives introduced in Section 1.3 and examined in detail in Section 6.4.

7.1.1 Assessing Quality of Solutions by Providing Lower Bounds

The experiments of providing lower bounds on the objective value to assess the quality of heuristic solutions are given in Section 6.4.1.

The results show that our algorithm is capable of determining a good lower bound

on the objective value of the DSP in every scenario. Furthermore, the exact lower bound is detected in all instances by our algorithm both with the necessary but non-sufficient scheduling analysis of Lehoczky et al. [45] and with our new ILP formulation. Especially the formulation with the DMS policy by Lehoczky et al. [45] leads to smaller running times for providing lower bounds than the tested heuristic approach of Thaden [55] computes solutions.

Since our algorithm detects the exact lower bounds in all tested ViDAs instances, we are able to prove the optimality of the heuristic approaches within the same time horizon.

7.1.2 Providing Global Optimal Allocations

The experiments of providing global optimal allocations are presented in Section 6.4.2.

In the scenario of DSP-GF the DMS policy is optimal and with the formulation of Lehoczky et al. [45] the optimal solution is detected quickly. The application of a heuristic phase prior to the exact computation was not beneficial in every instance in DSP-GF.

With the more complex scheduling policies to additionally compute WCRTs, the heuristic phase is necessary to prune the branch and bound tree. The results show that our new ILP formulation is more efficient than the formulation of Eisenbrand et al. [31], in terms of smaller running times and tighter bounds if the time limit was reached.

In the scenario of DSP-CF with the combining constraints in the master problem we obtained strictly smaller running times with our new ILP formulation than with the formulation of Eisenbrand et al. [31] which is more complex.

7.1.3 Development of Hybrid Approaches

The development of a hybrid approach is based upon the idea of combining generated heuristic start solutions, a non-sufficient presolve phase, and a subsequent exact phase to determine global optimal allocations in the end. The detailed experiments are given in Section 6.4.3.

If we only use start solutions generated by external heuristic approaches, we observe benefits in smaller running times and tighter bounds for almost every approach and instance. A noteworthy ability of our approach is that even partial heuristic solutions can be used in the master LP to be recombined to valid full solutions.

The presolve phase with the DMS policy applied prior to the exact phase also pays off for every approach and instance, because the branch and bound tree is decreased in size by good lower bounds.

Combining both ideas leads to the hybrid approach with a presolve phase and the use of heuristic start solutions. Every approach gains from this combination and optimal solutions can be obtained in the same time horizon as heuristic solutions.

7.2 Discussion

The following points of discussion outlines possibilities and options for further improvements, and highlights various areas of future work.

We start by discussing our algorithm design in Section 7.2.1 and then commenting on scalability in Section 7.2.2. The high similarity within the tested instances provide possibilities for our algorithm to use these circumstances by symmetry, see Section 7.2.3, and parallelization, see Section 7.2.4. Additionally, we show some options for further parameter tuning of our algorithm with the help of the used ILP solver in Section 7.2.5.

7.2.1 Algorithm Design

The previous work of Althaus et al. [2] already showed that it is useful to apply a Dantzig-Wolfe decomposition with a column generation approach to the LP formulation of the problem of scheduling a single subsystem with periodic tasks. In this thesis, we extended the previously mentioned idea to schedule several subsystems with more complex bus systems and requirements that connect WCRTs of tasks and messages, but the concept of decomposing the initial LP is conserved.

The results show that this algorithm design is capable of solving most instances in reasonable time. When providing safe lower bounds our algorithm keeps up with heuristic approaches that provide valid solutions. The price of optimality is the investment in larger running times which is necessary if the optimal solution is provided by our algorithm alone – especially if we handle chains with combining constraints in DSP-CF.

In the just mentioned scenario with chains and when the computation of WCRTs is required, our new ILP formulation turns out to be advantageous over the formulation of Eisenbrand et al. [31], see Sections 5.3.4 and 5.1.5.2. This can be explained by the clustered formulation of the priorities of tasks. Eisenbrand et al. [31] model the full information on preemption in their ILP formulation with binary variables, whereas our new ILP formulation only uses binaries to allocate

WCRTs to intervals. Since real-world instances contain only a small number of these intervals, which occur as multiples of periods, our new ILP formulation promises to be faster solvable.

If we drop the goal of obtaining optimal solutions and only provide safe lower bounds on the objective function, we could try to disregard several hard constraints for faster computations. Specifically, we could try to decrease the complexity of the pricing ILPs in a way that they are solvable with combinatorial approaches in polynomial or pseudo-polynomial running times. The challenging task is to find safe underapproximations of the objective function to provide reliable lower bounds.

The hybrid approach, using heuristics and optimal strategies in combination, is very sustainable. The benefits of heuristics are quickly available integral solutions that can be used directly by our algorithm, and furthermore they can be recombined if they are only partial. Several strategies to speed-up the processing of the branch and bound tree are applicable and the most promising ones are presented in Section 4.6. In the end, the optimal solution is provided in the same time horizon as heuristic approaches compute solutions. The advantage of our algorithm design is that heuristic solutions can be integrated directly and the overall process remains the same.

7.2.2 Scalability

The tested ViDAs instances in Section 6.3 cover a range from 27 to 50 tasks with a hardware architecture consisting of two subsystems with two ECUs each and four different ECU types. The experiments showed that our algorithm is able to handle these instances in reasonable time. As we focused on schedulability analyses, neither the memory consumption nor the signal transmission was the bottleneck of computation.

The literature provides diffuse numbers on other real-world instances from industry. In the earlier works of Tindell [57, 59, 60] we find up to seven subsystems, six groups of tasks, and 22 signals with a periodicity of multiples of 5ms in the automotive section. In more current works, Schmidt and Schmidt [52] quote numbers around 70 ECUs and 2500 signals between tasks in luxury cars, and Zhu et al. [63] quote nine ECUs, 41 tasks, 83 signals, and ten chains with end-to-end deadlines.

A reliable statement about the behaviour of our algorithm when applied to larger real-world instances cannot be given, due to the fact that larger instances may become easier to solve if they include several restrictions. The possibilities to drastically change the hardness of the instances by the parameters are numerous. For example, real-world instances can have tighter gaps between the WCETs and

deadlines of tasks, which increase the utilization of the ECUs; periods can be multiples of a fixed time, which makes other schedulability analyses applicable; or larger transmission times of the signals lead to harder schedulability analyses of the global bus.

Our algorithm is designed in a way to cover almost all of the known restrictions and to make only minor assumptions. As we are still solving an $\mathcal{NP}$-hard problem to the optimum, we have to expect increasing running times if the instances grow – probably with an exponential behaviour. Further real-world instances and a detailed sense of which parameters to scale on are necessary to judge about scalability in the end.

7.2.3 Symmetry

The algorithm is designed to partition the given problem into independent subproblems, which reflect the scheduling of the subsystems, as we expect differences in the architecture of the subproblems as well as restrictions in the allocation of the tasks as described in Section 3.4.

However, the evaluated ViDAs instances show a high similarity as they come with copies of sets of tasks, no additional constraints, and equal hardware architecture in every subsystem. For this reason, the subproblems are highly symmetric in the sense that a solution of one subproblem is also feasible for all other subproblems. Therefore, the maintenance of several subproblems boils down to solving only a few or one subproblem and including several variables into the master LP. Consequently, a dramatic speed-up is highly likely and the possibility to solve large-scaled instances is achieved.

7.2.4 Parallelization

In Section 7.2.3 we already mentioned how to use the symmetry of instances and how to restrict the number of subproblems to the case of solving only a few. The opposite scenario is that we are faced with a large amount of independent but non-similar subproblems.

Our current algorithm design solves the nodes of the branch and bound tree in sequence, allowing the LP and ILP solver to use all available cores when solving a node. On the one hand, this is useful as the ILP solver itself starts branch and bound processes whenever it computes the integer optimum to a pricing problem. On the other hand, we can manually allocate the available cores to solve several pricing problems in parallel. As some pricing problems are hard to solve, especially if the formulation gets more and more complex, we decided to

select the first option and leave the second for further developments.

On a higher level, we can also parallelize the processing of the branch and bound tree to obtain a speed-up. Despite the price of increased running times for every node, this strategy can be of benefit, as branching decisions can be made based on more detailed knowledge from several evaluated child nodes. We propose this option for further developments with more realistic instances to test on.

7.2.5 Parameter Tuning

For solving the generated LPs and ILPs we used the commercial solver Gurobi Optimizer 5.6.2 [36], which includes a parameter tuning tool. If the scenario of the DSP and the applied scheduling policies are fixed, we can start a parameter tuning on the generated LPs and ILPs, where the ILPs are more interesting as they consume a multiple of running time compared to the LPs.

Several tests showed that a speed-up of up to a factor of ten is possible, if the parameters are tuned according to the generated ILPs. We decided to develop a generally good performing algorithm and solely experimented with the tuning results, as they differed widely depending on the scenarios and scheduling policies. To include this parameter tuning in the overall process, we could start the tuning of parameters on several pricing ILPs in the beginning to speed-up the solving process of subsequent ILPs. As solving the pricing ILPs is the bottleneck of computation, the tuning possibilities seem helpful, but additional instances are necessary in order to be more precise.

Bibliography

[1] S. Agmon. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6(3):382, 1954.

[2] E. Althaus, R. Naujoks, and E. Thaden. A column generation approach to scheduling of periodic tasks. In *Experimental Algorithms – 10th International Symposium, SEA 2011, Proceedings*, volume 1 of *LNCS 6630*, pages 340–351. Springer Berlin, 2011.

[3] E. Althaus, S. Hoffmann, J. Kupilas, and E. Thaden. A column generation approach to scheduling of real-time networks. In *Proceedings of the World Congress on Engineering and Computer Science (WCECS)*, volume 1, pages 224–229. IAENG, 2012.

[4] E. Althaus, S. Hoffmann, J. Kupilas, and E. Thaden. Scheduling of real-time networks with a column generation approach. In H. K. Kim, S.-I. Ao, M. A. Amouzegar, and B. B. Rieger, editors, *IAENG Transactions on Engineering Technologies*, volume 247 of *Lecture Notes in Electrical Engineering*, pages 397–412. Springer Netherlands, 2014.

[5] N. C. Audsley. Deadline monotonic scheduling, 1990.

[6] J. Bao, P. Battram, A. Enkelmann, A. Gabel, J. Heyen, T. Koepke, C. Läsche, and S. Sieverding. Projektgruppe Vidas – Endbericht. Technical report, Carl von Ossietzky-University, Oldenburg, 2010.

[7] S. Baruah, L. Rosier, and R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2(4):301–324, 1990.

[8] E. M. L. Beale. An alternative method for linear programming. *Mathematical Proceedings of the Cambridge Philosophical Society*, 50:513–523, 9 1954.

[9] B. Becker, W. Damm, M. Fränzle, E.-R. Olderog, A. Podelski, and R. Wilhelm. SFB/TR 14 AVACS – Automatic Verification and Analysis of Complex Systems, 2004. URL `http://www.avacs.org`.

[10] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1st edition, 1997.

[11] A. Beutelspacher. *Lineare Algebra*. Vieweg, 6th edition, 2003.

[12] E. Bini and G. C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *Computers, IEEE Transactions on*, 53(11):1462–1473, 2004.

[13] R. G. Bland. New Finite Pivoting Rules for the Simplex Method. *Mathematics of Operations Research*, 2(2):103–107, May 1977.

[14] M. Büker. *An Automated Semantic-based Approach for Creating Task Structures*. PhD thesis, Carl von Ossietzky Universität Oldenburg, 2012.

[15] M. Büker, W. Damm, G. Ehmen, A. Metzner, I. Stierand, and E. Thaden. Automating the design flow for distributed embedded automotive applications: keeping your time promises, and optimizing costs, too. Technical Report 69, SFB/TR 14 AVACS, 2011.

[16] M. Büker, W. Damm, G. Ehmen, A. Metzner, I. Stierand, and E. Thaden. Automating the design flow for distributed embedded automotive applications: Keeping your time promises, and optimizing costs, too. In *Proc. International Symposium on Industrial Embedded Systems (SIES'11)*, 2011.

[17] A. Burns and A. Wellings. *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*. International computer science series. Addison-Wesley, 2001.

[18] G. C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2004.

[19] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., 1996.

[20] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

[22] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965.

[23] G. B. Dantzig. Programming in a linear structure. *Washington, DC*, 1948.

[24] G. B. Dantzig. Programming of interdependent activities: II mathematical model. *Econometrica*, 17(3/4):200–211, 1949.

[25] G. B. Dantzig. *Maximization of a Linear Function of Variables Subject to Linear Inequalities, in Activity Analysis of Production and Allocation*, chapter XXI. Wiley, New York, 1951.

[26] G. B. Dantzig. *Computational algorithm of the revised simplex method*. Rand Corporation, 1953.

[27] G. B. Dantzig. *Linear programming and extensions*. Princeton University Press Princeton, N.J, 1963.

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

[29] R. Davis, A. Burns, R. Bril, and J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[30] R. Devillers and J. Goossens. Liu and Layland's schedulability test revisited. *Information Processing Letters*, 73(5):157–161, 2000.

[31] F. Eisenbrand, W. Damm, A. Metzner, G. Shmonin, R. Wilhelm, and S. Winkel. Mapping task-graphs on distributed ECU networks: Efficient algorithms for feasibility and optimality. In *Proceedings of the 12th IEEE Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE Computer Society, 2006.

[32] J. J. Forrest and D. Goldfarb. Steepest-edge simplex algorithms for linear programming. *Math. Program.*, 57:341–374, 1992.

[33] M. R. Garey and D. S. Johnson. Strong NP-completeness results: Motivation, examples, and implications. *Journal of the ACM (JACM)*, 25(3):499–508, 1978.

[34] D. Goldfarb and J. Reid. A practicable steepest-edge simplex algorithm. *Mathematical Programming*, 12(1):361–371, 1977.

[35] R. E. Gomory. Solving linear programming problems in integers. In *Combinatorial Analysis, Symposia in Applied Mathematics X*, pages 211–215, 1960.

[36] Gurobi Optimization, Inc. Gurobi Optimizer Reference Manual, 2014. URL `http://www.gurobi.com`.

[37] P. M. Harris. Pivot selection methods of the devex LP code. *Mathematical Programming*, 5(1):1–28, 1973.

[38] K. Jeffay and D. Stone. Accounting for interrupt handling costs in dynamic priority task systems. In *Real-Time Systems Symposium, 1993., Proceedings.*, pages 212–221. IEEE, 1993.

[39] M. Johnson. Proof that timing requirements of the FDDI token ring protocol are satisfied. *Communications, IEEE Transactions on*, 35(6):620–625, 1987.

[40] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[41] R. M. Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[42] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1096, 1979.

[43] V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities*, volume III, pages 159–175. Academic Press, New York, 1972.

[44] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

[45] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium*, pages 166–171, 1989.

[46] C. E. Lemke. The dual method of solving the linear programming problem. *Naval Research Logistics Quarterly*, 1:36–47, 1954.

[47] J. Y. T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[48] L. A. Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973.

[49] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20:46–61, 1973.

[50] M. Lukasiewycz, M. Glaß, J. Teich, and P. Milbredt. FlexRay schedule optimization of the static segment. In *Proceedings of the 7th IEEE/ACM international conference on hardware/software codesign and system synthesis*, CODES+ISSS '09, pages 363–372, New York, NY, USA, 2009. ACM.

[51] F. Plastria. Formulating logical implications in combinatorial optimisation. *European Journal of Operational Research*, 140(2):338–353, 2002.

[52] K. Schmidt and E. G. Schmidt. Message scheduling for the FlexRay protocol: The static segment. *Vehicular Technology, IEEE Transactions on*, 58(5): 2170–2179, 2009.

[53] A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986.

[54] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer, 2004.

[55] E. M. Thaden. *Semi-Automatic Optimization of Hardware Architectures in Embedded Systems.* PhD thesis, Carl von Ossietzky Universität Oldenburg, 06 2013.

[56] K. Tindell and A. Burns. Guaranteeing message latencies on control area network (CAN). In *Proceedings of the 1st International CAN Conference.* Citeseer, 1994.

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

[58] K. Tindell, H. Hansson, and A. J. Wellings. Analysing real-time communications: controller area network (CAN). In *Real-Time Systems Symposium, 1994., Proceedings.*, pages 259–263. IEEE, 1994.

[59] K. Tindell, A. Burns, and A. J. Wellings. Calculating controller area network (CAN) message response times. *Control Engineering Practice*, 3(8):1163–1169, 1995.

[60] K. W. Tindell, A. Burns, and A. J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.

[61] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution time problem – overview of methods and survey of tools. Reports of SFB/TR 14 AVACS 14, SFB/TR 14 AVACS, April 2007.

[62] S. Zhang, A. Burns, A. Mehaoua, E. S. Lee, and H. Yang. Testing the schedulability of synchronous traffic for the timed token medium access control protocol. *Real-Time Systems*, 22(3):251–280, 2002.

[63] Q. Zhu, Y. Yang, M. D. Natale, E. Scholte, and A. L. Sangiovanni-Vincentelli. Optimizing the software architecture for extensibility in hard real-time distributed systems. *IEEE Trans. Industrial Informatics*, 6(4):621–636, 2010.