Optimization of Schedulability and Quality of Service in Real-Time Mixed-Criticality Systems

Pavić, Ivan

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Ivan Pavić

OPTIMIZATION OF SCHEDULABILITY AND QUALITY OF SERVICE IN REAL-TIME MIXED-CRITICALITY SYSTEMS

DOCTORAL THESIS

Zagreb, 2021

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Ivan Pavić

OPTIMIZATION OF SCHEDULABILITY AND QUALITY OF SERVICE IN REAL-TIME MIXED-CRITICALITY SYSTEMS

DOCTORAL THESIS

Supervisor: Associate Professor Hrvoje Džapo, PhD

Zagreb, 2021

Sveučilište u Zagrebu
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Ivan Pavić

OPTIMIZACIJA RASPOREDIVOSTI I KVALITETE USLUGE U SUSTAVIMA ZA RAD U STVARNOM VREMENU S MJEŠOVITOM KRITIČNOŠĆU

DOKTORSKI RAD

Mentor: Izv. prof. dr. sc. Hrvoje Džapo

Zagreb, 2021.

This doctoral thesis has been prepared at the University of Zagreb, Faculty of Electrical Engineering and Computing, Department of Electronic Systems and Information Processing.

Supervisor: Associate Professor Hrvoje Džapo, PhD

The doctoral thesis has 206 pages.

Doctoral thesis number: _____

About the Supervisor

Hrvoje Džapo was born on 15th December 1975 in Zagreb, Croatia. He received his Ph.D. degree from the University of Zagreb, Faculty of Electrical Engineering and Computing (FER), Zagreb, Croatia, in 2007 in the field of Technical Sciences, Electrical Engineering. He received his Dipl. ing. degree from FER in 1999, study of Electrical Engineering, module Industrial Electronics.

From December 1999 until now he has been working at the Department of Electronic Systems and Information Processing at FER. He was promoted to Associate Professor in January 2017. He participated in 12 scientific projects financed by the Ministry of Science, Education and Sports of the Republic of Croatia and other institutions of Republic of Croatia, and 2 international projects. He is a project leader of a project "New generation telemetry technology for measurement on rotational clutch components", and actively participates in projects "Development of a new generation of numerical protection devices (KONPRO 2)", "Development of Greyp Micromobility Platform - GMP", "A-Unit - Research and development of advanced unit for autonomous control of mobile vehicles in logistics" and "Crossing the Gap: Startup education and support for PhD students, researchers and scientists – COGSTEPS". He published more than 30 papers in journals and conference proceedings in the area of measurement instrumentation, sensor technologies, electromagnetic modeling, embedded systems and signal processing.

Associate Professor Džapo is a senior member of IEEE (Institute of Electrical and Electronics Engineers), IFMBE (International Federation for Medical and Biological Engineering), and CROBEMPS (Croatian Biomedical Engineering and Medical Physics Society). Has was a chair of IEEE Instrumentation and Measurement Society Chapter of IEEE Croatia Section from 2009 to 2013. He has been the head of the Career Center at FER from its foundation at 2015 until now. He participated in 2 conference international program committees and he serves as a reviewer for 2 international journals. In 2019 he was awarded IEEE Croatia Section Outstanding Educator Award for outstanding contribution to engineering education through connecting the academic sector and industry in the fields of student internships, career development and promotion of student entrepreneurial activities. He received award „Roberto Giannini" from FER for academic year 2009/2010 for exellence in teaching and work with students. He received silver medal "Josip Lončar" from FER for outstanding Ph.D. dissertation for academic year 2006/2007. For his innovations "Biomechanical scale for dynamic posture analysis" and "System for continuous measurement of power on ship motor shaft " he received golden and silver medal award on International innovation exhibition ARCA 2016 and ARCA 2011, respectively. He is one of inventors on an internationally recognized patent.

O mentoru

Hrvoje Džapo rođen je 15. prosinca 1975. u Zagrebu, Republika Hrvatska. Doktorat znanosti stekao je na Sveučilištu u Zagrebu, Fakultet elektrotehnike i računarstva (FER), Zagreb, Republika Hrvatska, 2007. godine, u znanstvenom području Tehničkih znanosti, polje Elektrotehnika. Titulu diplomiranog inženjera elektrotehnike stekao je na FER-u 1999. godine, na studiju Elektrotehnike, smjeru Industrijska elektronika.

Od prosinca 1999. do sada zaposlen je u Zavodu za elektroničke sustave i obradbu informacija na FER-u. U siječnju 2017. postao je izvanredni profesor. Sudjelovao je u realizaciji 12 znanstvenih projekata financiranih od strane Ministarstva znanosti, obrazovanja i športa te drugih institucija Republike Hrvatske i u realizaciji 2 međunarodna projekta. Trenutno vodi projekt "Nova generacija telemetrijske tehnologije za mjerenja na rotacijskim komponentama spojke" te aktivno sudjeluje u realizaciji i vođenju projekata "Razvoj nove generacije uređaja numeričke zaštite (KONPRO 2)", "Razvoj Greyp platforme za mikromobilnost - GMP", "A-Unit - Istraživanje i razvoj napredne jedinice za autonomno upravljanje mobilnim vozilima u logistici" i "Crossing the Gap: Startup education and support for PhD students, researchers and scientists – COGSTEPS". Autor je više od 30 članaka objavljenih u časopisima i na međunarodnim znanstvenim konferencijama u području mjerne instrumentacije, senzorskih tehnologija, elektromagnetskog modeliranja, ugradbenih računalnih sustava i obradbe signala.

Izvanredni profesor Džapo stariji je član IEEE (Institute of Electrical and Electronics Engineers), IFMBE (International Federation for Medical and Biological Engineering), i CROBEMPS (Hrvatsko društvo za biomedicinsko inženjerstvo i medicinsku fiziku). Bio je voditelj Odjela za instrumentaciju i mjerenja Hrvatske sekcije IEEE od 2009. do 2013. Voditelj je Centra karijera FER-a od osnutka 2015. godine do danas. Sudjelovao je u 2 međunarodna programska odbora konferencija i bio recenzent u 2 međunarodna časopisa. Dobitnik je Nagrade za izniman doprinos u inženjerskom obrazovanju za 2019. godinu Hrvatske sekcije IEEE, radi doprinosa u inženjerskom obrazovanju kroz povezivanje akademske zajednice i gospodarstva u području stručnih praksi, razvoja karijera i poticanja studentskih poduzetničkih aktivnosti. Dobitnik je FER-ove nagrade „Roberto Giannini" za akademsku godinu 2009./2010. za izvrsnost u nastavi i radu sa studentima. Dobitnik je FER-ove srebne plakete "Josip Lončar" za istaknutu doktorsku disertaciju za akademsku godinu 2006./2007. Za inovacije "Biomehanička vaga za dinamičku analizu posture" i "Sustav za kontinuirano mjerenje snage na brodskoj osovini" dobio je zlatnu i srebrnu medalju na Međunarodnoj izložbi inovacija ARCA 2016 and ARCA 2011. Jedan je od izumitelja na međunarodno priznatom patentu.

I dedicate this work to my family.

Acknowledgment

Writing a proper acknowledgement that would fairly address the contributions and the tremendous help that I received during this research and writing of the thesis, is not an easy task. Hence, I thank all the people that gave me a valuable input over the past four years.

To my family

Writing this thesis would not be possible without my family. Therefore, special thanks goes to my family that kept me alive during this tiring endeavor. Firstly, to my parents Mirko and Mirjana, and grandparents Vlado and Marica, who encouraged me to pursue the PhD degree. Secondly, to my loving wife Magdalena who helped me and motivated me throughout the doctoral studies. Nobody knows the trouble I have seen as much as her, and she was always there to pick me up before I fall. Special thanks goes to my grandmother Marica because she made sure that a hot meal was waiting for me whenever I came home. I want to thank my grandfather Vlado for giving me enough free time for studying. On the other hand, I want to thank my father Mirko (Puri) for giving me enough work to put my mind at ease from the studies. I want to thank my mother Mirjana for all the good advice and countless conversations that we had during hard times. I want to thank my brother Nikola for all the conversations and fun that we had, and for the music and ideas that we shared, and especially for visiting him in Ireland.

To my friends

I want to thank all my friends for help and support during the writing of the thesis. I want to thank Jurica and Dane for all the hikings to Sljeme. Moreover, for all the good games we played on Play Station during the lockdown and earthquakes, which kept us calm and cheerful during these hard times. I want to thank my band Tvoja Mama for all the gigs and rehearsals we had during these years. Special thanks goes to my best man Antun for countless discussions about metaphysics, ethics, religion and our doctoral studies during the extended coffee breaks.

To my supervisor

I want to thank my supervisor Hrvoje Džapo for the opportunity for enrolling in PhD studies and the employment at the University of Zagreb, Faculty of Electrical Engineering and Computing. His mentorship, guidance and valuable advice lead me through the doctoral research. His advice was helpful during master's degree and bachelor's degree as well, which helped me to grow as an engineer.

To my colleagues

I want to thank my colleague Hrvoje Mihaldinec for sparking the research interest for the topic of the dissertation. I want to thank all the professors from courses I was enrolled in, especially professor Mladen Vučić who was always available for conversation about any topic even due to his overloaded schedule. I want to thank my colleagues Tin, Dorijan and Kristijan for numerous discussions during lunch that helped to put my mind at ease and improve the research. I want to thank my colleagues Ivana, Krešimir and Dominik for all the good times and discussions in the office that helped us all to grow and finish our doctoral dissertations. I want to thank my colleague Karla for cooperation on all the projects which improved my research significantly. In the end, special thanks goes to Marko Đurasević who helped me in the critical parts of the doctoral studies by reviewing my ideas, giving insightful comments, and great help in the revising the doctoral thesis.

Abstract

This thesis investigates three different methods for optimization of schedulability and quality of service in real-time mixed-criticality systems. The research is motivated by requirements of safety-critical systems, typically encountered in transportation and industrial systems, which are often represented with various real-time mixed-criticality system models.

The first method is focused on the schedulability testing in the adaptive mixed-criticality system model. A novel sufficient schedulability test for adaptive mixed-criticality task systems is devised, which improves the schedulability in comparison with the existing sufficient schedulability tests. Moreover, the devised test requires significantly less computing power in comparison with exhaustive exact methods. In addition, a framework for schedulability testing was devised, which ensures comprehensive and systematic experimental evaluation of the devised test as well as the validation of the existing schedulability tests. Using this framework, errors and inconsistencies in the existing schedulability tests were corrected. The results presented in the thesis consist of extensive experimental evaluation on a large number of different synthetically generated task sets as well as on small numerical examples.

The second method deals with the harmonic period assignment from period ranges, which is of great importance for maintaining schedulability and quality of service in the safety-critical real-time systems. Unlike the existing harmonic period assignment methods from period ranges, the method for harmonic period assignment devised in this research is optimal, and it enables optimization of the number of different period values in the system, which is often of practical interest in real-world applications. Moreover, the devised harmonic period assignment method enables the optimization of arbitrary utilization values. The method is validated and compared to existing methods using an extensive experimental evaluation. It was shown that the usage of this method can significantly increase schedulability and quality of service in sense of utilization in systems of interest.

The third method addresses the optimization of quality of service of non-critical or low-criticality tasks. The method is based on the genetic programming method, which is often used in solving various scheduling problems. In this research, the genetic programming is exploited to generate dynamic priority assignment functions for scheduling of low-criticality tasks in the adaptive mixed-criticality environment, which is typically overloaded. The extensive experimental evaluation on synthetically generated task sets demonstrates that the proposed method can generate heuristics for various system configurations, which dominate single-variable based heuristics that can be found in the literature.

Keywords: real-time, scheduling, mixed-criticality, period assignment, genetic programming

Prošireni sažetak

Optimizacija rasporedivosti i kvalitete usluge u sustavima za rad u stvarnom vremenu s mješovitom kritičnošću

Uvod

U današnje vrijeme, pojam stvarnog vremena je od velike važnosti u svakodnevnom životu odnosno u cijelom spektru ljudskih aktivnosti koje uključuju interakciju s računalnim sustavima. Nadalje, koncept stvarnog vremena jako je važan u ugradbenim računalnim sustavima koji su dio sigurnosno kritičnih sustava koji se koriste u industrijskim računalnim sustavima, automobilima, željezničkim sustavima te biomedicinskim aplikacijama. Grubo govoreći, sigurnosno kritični sustavi su sustavi u kojima kvar tijekom rada sustava može uzrokovati veliku financijsku štetu ili gubitak ljudskog života. Dizajn i proizvodnja takvih sustava prolazi rigorozni proces utvrđivanja sigurnosti koji osigurava nisku vjerojatnost kvara te povećava pouzdanost sustava. Fokus ovog istraživanja je elektronički dio sigurnosno kritičnih sustava odnosno računalni sustav koji izvršava različite funkcije bitne za rad sustava. Štoviše, fokus je na vremenskoj analizi ovakvih sustava. Inherentni strogi zahtjevi za rad u stvarnom vremenu u sigurnosno kritičnim sustavima kao i rigorozna procedura provjere tih sustava koja je nametnuta standardima za funkcionalnu sigurnost motivirali su inženjere i znanstvenu zajednicu za razvitak metoda za učinkovitu analizu i dizajn takvih sustava. U tom kontekstu, cilj ove disertacije razviti nove metode za dizajn i analizu sigurnosno kritičnih sustava koje u obzir uzimaju trenutne trendove i potrebe koje se javljaju u njihovoj implementaciji. Preciznije, metode koje su razvijene u ovom radu adresiraju optimizaciju rada sustava u smislu rasporedivosti zadataka i kvalitete usluge izvođenja zadataka.

Motivacija za istraživanje

Motivacija za istraživanje proizašla je iz stvarnih problema u dizajnu i potreba utvrđivanja ispravnosti sigurnosno kritičnih sustava u industriji. Iako je praksa certifikacije takvih sustava dobro poznata i određena standardima za funkcionalnu sigurnost, mnogi današnji trendovi vezani uz izradu ugradbenih računalnih sustava utjecali su značajno na taj proces. Jedan od aktualnih trendova u kontekstu ugradbenih računalnih sustava je povećanje procesorske moći što je omogućilo implementaciju više različitih funkcija na istoj računalnoj platformi. Štoviše, to je omogućilo da se funkcije koje nisu kritične za rad sustava izvode uz sigurnosno kritične funkcije na jednoj računalnoj platformi. Sustavi koji izvode funkcije različite kritičnosti nazivaju se sustavima s mješovitom kritičnošću. Zajednička karakteristika takvih sustava je da potencijalno mogu ući u preopterećeno stanje što znači da se neke nekritične funkcije neće moći izvesti na vrijeme. U klasičnim sustavima za rad u stvarnom vremenu ovakva pojava nije prihvatljiva dok

je u različitim varijantama sustava s mješovitom kritičnošću dozvoljeno da se pojedine funkcije ne stignu izvesti do krajnjeg roka završetka. Isto tako, u tradicionalnim implementacijama sustava s mješovitom kritičnošću, funkcije različite kritičnosti izvode se na različitim računalima što omogućuje prostornu i vremensku odvojenost tih funkcija. Ipak, time je povećana složenost sustava u smislu broja računala, vremena za razvoj sustava, a povećana je i potrošnja. Pristup u kojem se funkcije različite kritičnosti izvode na istoj računalnoj platformi smanjuje složenost sustava smanjenjem broj računala što uzrokuje smanjenje vremena potrebnog za razvoj te smanjenje potrošnje i cijene sustava.

U ovom istraživanju, razvijene su metode za optimizaciju dviju važnih mjera u sustavima za rad u stvarnom vremenu s mješovitom kritičnošću, a to su rasporedivost i kvaliteta usluge. Rasporedivost je svojstvo sustava koje kazuje je li moguće rasporediti sustav. Na primjer, opišemo li sustav kao skup poslova, on je rasporediv ako se svi poslovi mogu izvršiti do krajnjeg roka završetka. Rasporedivost je minimalni uvjet koji mora biti zadovoljen za sigurnosno kritične funkcije u sustavima s mješovitom kritičnošću. S druge strane kvaliteta usluge, u ovom kontekstu, je mjera koja kazuje koliko su performanse sustava pogoršane u odnosu na zadana ograničenja. Drugim riječima, u okolnostima u kojima nije moguće garantirati rasporedivost svih zadataka, kvaliteta usluge kazuje koliko dobro ili loše određeni algoritam za raspoređivanje raspoređuje poslove u sustavu. Iako su metode koje su razvijene u ovom radu predviđene za poboljšanje rasporedivosti i kvalitete usluge, one imaju utjecaj na ostala svojstva koja su bitna u sustavima za rad u stvarnom vremenu kao što su robusnost, predvidljivost i stabilnost.

Pregled disertacije

Disertacija sadrži šest poglavlja. U prvom i drugom poglavlju opisani su kontekst i teoretska pozadina istraživanja. U trećem, četvrtom i petom poglavlju opisane su tri metode za optimizaciju rasporedivosti i kvalitete usluge u sustavima za rad u stvarnom vremenu s mješovitom kritičnošću koje čine doprinos ove disertacije. U šestom poglavlju dan je zaključak.

U prvom poglavlju disertacije opisan je kontekst istraživanja te je dana motivacija za istraživanje. Naglašena je važnost istraživanja u kontekstu trendova koji se pojavljuju u izradi ugradbenih računalnih sustava te je objašnjena paradigma sigurnosno kritičnih sustava i sustava s mješovitom kritičnošću. Definirani su cilj i hipoteze istraživanja te su naglašeni znanstveni doprinosi istraživanja. Cilj predloženog istraživanja je razviti metode za poboljšanje rasporedivosti i kvalitete usluge u sustavima za rad u stvarnom vremenu s mješovitom kritičnošću. Glavne hipoteze istraživanja su:

1. Testovi rasporedivosti za adaptivne sustave s mješovitom kritičnošću s fiksnim prioritetima mogu biti poboljšani.
2. Postojeći pristupi za harmonijsku dodjelu perioda mogu se poboljšati uvođenjem dodatnih ograničenja na dizajn sustava.

3. Heuristike generirane korištenjem genetičkog programiranja mogu se koristiti u sustavima s mješovitom kritičnošću za povećanje kvalitete usluge zadataka koji nisu kritični.

U drugom poglavlju disertacije opisana je teoretska pozadina sustava za rad u stvarnom vremenu. Dan je pregled modela sustava za rad u stvarnom vremenu, a detaljnije su opisani modeli periodičkih i sporadičkih skupova zadataka. Detaljnije je objašnjeno jednoprocesorsko raspoređivanje te su opisane metode za računanje vremena odziva sporadičkih kao i periodičkih skupova zadataka u kontekstu određivanja rasporedivosti sustava. Detaljnije je opisan i industrijski kontekst sustava s mješovitom kritičnošću s aspekta pouzdanosti računalnih sustava. Ukratko je opisan proces utvrđivanja sigurnosti sustava. Na kraju je dan pregled različitih konfiguracija sustava s mješovitom kritičnošću koje se susreću u praksi.

U trećem poglavlju opisana je metoda za ispitivanje rasporedivosti za sustave s adaptivnom mješovitom kritičnošću i nepromjenjivim prioritetima. Opisana je motivacija za uvođenje koncepta adaptivne mješovite kritičnosti koji uz diskriminaciju zadataka u sustavima po kritičnosti uvodi i različita stanja sustava. Dan je pregled različitih modela sustava s mješovitom kritičnošću te su uspoređeni statički i adaptivni modeli. Isto tako, dan je pregled postojećih testova rasporedivosti za statičke i adaptivne sustave koji mogu biti nužni, dovoljni i precizni. Uveden je novi dovoljni test rasporedivosti koji daje bolje rezultate u odnosu na postojeće, a temelji se na analizi vremena odziva zadataka u sustavu. U odnosu na postojeće dovoljne testove rasporedivosti, novi test se temelji na preciznijoj analizi vremena odziva koja se može dobiti dodatnom diskriminacijom kritičnih i nekritičnih zadataka u sustavu prema prioritetima. Pokazano je da u odnosu na postojeće dovoljne testove rasporedivosti, novi test ima veću vremensku složenost. Teoretske razlike između testova rasporedivosti demonstrirane su na malim numeričkim primjerima. Nadalje, u radu je pokazano da kod postojećeg preciznog testa rasporedivosti postoje pogreške u formulaciji algoritma i uvedeni su ispravci istih grešaka. U radu je za potrebe konzistentne evaluacije različitih testova rasporedivosti opisan radni okvir razvijen u sklopu istraživanja koji omogućava usporedbu različitih testova na sintetski generiranim skupovima zadataka. Korištenjem radnog okvira napravljena je eksperimentalna evaluacija razvijenog i postojećih testova rasporedivosti na velikom broju sintetski generiranih skupova zadataka.

U četvrtom poglavlju opisana je metoda za dodjelu harmonijskih perioda za poboljšanje rasporedivosti sigurnosno kritičnih zadataka u sustavima za rad u stvarnom vremenu s mješovitom kritičnošću. Metoda je stavljena u kontekst istraživanja odnosno u kontekst sigurnosno kritičnih i mješovito kritičnih sustava. Objašnjena je važnost harmonijskih perioda zadataka, odnosno perioda kod kojih je svaka viša vrijednost perioda višekratnik niže vrijednosti perioda, u kontrolnim industrijskim sustavima. Dan je pregled postojećih istraživanja i metoda za dodjelu harmonijskih perioda. Objašnjeni su nedostaci postojećih pristupa kao što njihova suboptimalnost i nemogućnost optimizacije broja različitih vrijednosti perioda koja postoji kao zahtjev u mnogim industrijskim kontrolnim aplikacijama. Pomoću malih numeričkih primjera koji oslikavaju

stvarne industrijske probleme, objašnjena je motivacija za uvođenje novih metoda te su formulirani novi problemi dodjele harmonijskih perioda. Analizirana je složenost novih problema dodjele harmonijskih perioda koji omogućuju optimizaciju broja različitih vrijednosti perioda u rješenju i utvrđeno je da problemi spadaju u klasu NP-teških problema. Opisani su heuristički i optimalni algoritmi dodjele perioda zadacima te su određene njihove složenosti. Pokazano je da nove formulacije problema omogućavaju fleksibilan pristup dizajnu sustava. Eksperimentalnom evaluacijom na velikom broj sintetski generiranih skupova zadataka uspoređeni su novi i postojeći algoritmi i pokazano je da novi algoritmi donose poboljšanja u kontekstu rasporedivosti i kvalitete usluge.

U petom poglavlju opisana je metoda za raspoređivanje zadataka za poboljšanje kvalitete usluge nekritičnih zadataka u sustavima za rad u stvarnom vremenu s mješovitom kritičnošću temeljena na genetičkom programiranju. U poglavlju je dan pregled literature i pristupa koji se koriste za raspoređivanje nekritičnih zadataka i zadataka niske kritičnosti u sustavima s mješovitom kritičnošću. Isto tako, objašnjene su metode raspoređivanja u preopterećenim sustavima. Nadalje, dan je pregled pristupa temeljenih na genetičkom programiranju koji se koriste za generiranje heuristika odnosno funkcija prioriteta za raspoređivanje poslova u kontekstu operacijskih istraživanja. Objašnjena je sličnost u ograničenjima postupaka raspoređivanja u sustavima s mješovitom kritičnošću i raspoređivanja u kontekstu operacijskih istraživanja i pretpostavljeno je da bi se postupak genetičkog programiranja mogao koristiti i u sustavima s mješovitom kritičnošću za generiranje pravila raspoređivanja odnosno funkcija prioriteta. Motivacija za uvođenje genetičkog programiranja za generiranje heuristika za raspoređivanje objašnjena je na malim numeričkim primjerima. Nadalje, detaljno je objašnjen radni okvir temeljen na genetičkom programiranju koji služi za optimizaciju funkcija prioriteta koje omogućuju poboljšanje kvalitete usluge nekritičnih zadataka i zadataka niske kritičnosti. Heuristike dobivene predloženom metodom temeljito su evaluirane na velikom broju sintetski generiranih skupova zadataka i uspoređene su po performansama s postojećim heuristikama. Pokazano je da generirane heuristike donose poboljšanje kvalitete usluge u usporedbi s postojećim heuristikama. Metoda je proširena tako da se uz kvalitetu usluge može optimirati i pravednost u raspoređivanju. Isto tako, objašnjeno je kako se metoda može koristiti tako da se koristi više od jednog pravila raspoređivanja pomoću algoritma kooperativne koevolucije. Na kraju su evaluirani i uspoređeni svi pristupi. Na numeričkom primjeru je objašnjeno kako se metoda za dodjelu harmonijskih perioda može koristiti uz metodu genetičkog programiranja za dizajn sustava.

U šestom poglavlju izloženi su najbitniji zaključci istraživanja u kontekstu cilja te hipoteza istraživanja te znanstvenih doprinosa. Opisani su mogući daljnji pravci istraživanja.

Zaključak

U ovoj doktorskoj disertaciji istražene su tri različite metode za optimizaciju rasporedivosti i kvalitete usluge u sustavima za rad u stvarnom vremenu s mješovitom kritičnošću. Istraživanje je motivirano zahtjevima sigurnosno kritičnih sustava koji se tipično susreću u transportnim i industrijskim sustavima, a uz to se modeliraju različitim modelima sustava za rad u stvarnom vremenu s mješovitom kritičnošću. U disertaciji su opisane metode koje čine ostvareni znanstveni doprinos istraživanja:

1. Metoda za ispitivanje rasporedivosti za sustave s adaptivnom mješovitom kritičnošću i nepromjenjivim prioritetima.
2. Metoda za dodjelu harmonijskih perioda za poboljšanje rasporedivosti sigurnosno kritičnih zadataka u sustavima za rad u stvarnom vremenu s mješovitom kritičnošću.
3. Metoda za raspoređivanje zadataka za poboljšanje kvalitete usluge nekritičnih zadataka u sustavima za rad u stvarnom vremenu s mješovitom kritičnošću temeljena na genetičkom programiranju.

Fokus prve metode je ispitivanje rasporedivosti u adaptivnim sustavima s mješovitom kritičnošću. U radu je razvijen novi dovoljni test rasporedivosti za adaptivne sustave s mješovitom kritičnošću koji donosi poboljšanje rasporedivosti u usporedbi s postojećim dovoljnim testovima rasporedivosti. U radu su opisani ispravci potrebni za ispravan rad postojećih preciznih testova rasporedivosti. Razvijen je radni okvir za evaluaciju testova kojim su potvrđena poboljšanja i ispravci testova rasporedivosti.

Druga metoda koja je razvijena je metoda za dodjelu harmonijskih perioda iz intervala perioda. Razvijena metoda za dodjelu harmonijskih perioda je optimalna i omogućuje optimizaciju broja različitih vrijednosti perioda u konačnom rješenju što je od praktičnog interesa u stvarnim aplikacijama. Razvijena metoda je validirana i uspoređena s postojećim metodama temeljem ekstenzivne eksperimentalne evaluacije. Pokazano je da korištenje ove metode može značajno povećati rasporedivost i kvalitetu usluge u kontekstu faktora zauzeća u sustavima od interesa.

Treća metoda adresira optimizaciju kvalitete usluge nekritičnih zadataka ili zadataka s niskom kritičnošću. Razvijena metoda temelji se na genetičkom programiranju koje se koristi za generiranje dinamičkih funkcija za dodjelu prioriteta za raspoređivanje zadataka s niskom kritičnošću u sustavima s adaptivnom mješovitom kritičnošću koji su tipično preopterećeni. Pomoću ekstenzivne eksperimentalne evaluacije pokazano je da se predloženom metodom može dobiti poboljšanje u odnosu na postojeće heuristike.

Nove metode razvijene u ovom radu mogu se koristiti u analizi i dizajnu sustava s mješovitom kritičnošću u industriji. Ipak, treba uzeti u obzir da se u dizajnu industrijskih sigurnosno kritičnih sustava inženjeri okreću korištenju metoda koje dokazano imaju visoku učinkovitost u ciljanim aplikacijama. Time je teoretska pozadina razvijenih metoda koja je razrađena u ovom radu još bitnija jer jasno definira mogućnosti i ograničenja onog što se može postići u dizajnu

sustava, a to je od iznimne važnosti u sigurnosno kritičnim sustavima.

Ključne riječi: stvarno vrijeme, raspoređivanje, mješovita kritičnost, dodjela perioda, genetičko programiranje

Contents

Chapter 1

Introduction

Nowadays, the notion of *real-time* is of high importance in everyday life, i.e., the spectrum of different human activities which involve interaction with computer systems. Moreover, the concept of *real-time* has been especially important in the embedded computer systems which are a part of safety-critical systems that are typically encountered in automotive, railway, industrial and biomedical applications. Roughly speaking, safety-critical systems are systems in which an error or a failure in system operation can cause severe financial damage or loss of human life. The design and production of such systems undergoes rigorous *safety assessment process* which ensures low failure rate and increases reliability of the system. In this research, the focus is on the electronic part of safety-criticality systems, i.e., computer systems that execute various functions that are crucial for system performance. Moreover, the timing properties and timing phenomena in these systems are studied. Inherent hard real-time constraints in safety-critical systems, and rigorous safety constraints imposed by safety standards have been motivating engineers and the academia to provide efficient means for design and analysis of safety-critical real-time systems. In that regard, this thesis aims to provide novel techniques for design and analysis of these systems which take into account current trends and needs in the design of embedded systems.

1.1 Motivation for the research

The motivation for this research has arisen from the collaboration of the Faculty of Electrical Engineering and Computing, University of Zagreb and Končar Electrical Engineering Institute, Inc on *System for increased driving safety in public urban rail traffic (SafeTRAM)* project [1]. Therefore, the primary motivation for this research is the study of safety-critical embedded systems present in the railway applications as critical components that affect the overall system reliability and safety. Recent trend in design of safety-critical systems includes combined implementation of safety-critical functions alongside non-critical functions on a single com-

puting platform. Systems which execute functions with different criticality are referred to as *mixed-criticality* systems. The emergence of modern high-performance embedded computing platforms enabled the viability of such an approach in many practical solutions. The common characteristic of mixed-criticality systems is that they have a potential to enter an overloaded state, meaning that some non-critical tasks might not be able to meet their deadlines, which is not considered to be a system malfunction. On the other hand, this is considered to be a malfunction in classical hard real-time systems, and therefore it is not acceptable. It is worth noting that in the classical approach to the safety-critical system design, i.e., partitioned or federated approach, functions of different criticality are executed on separate computing platforms. The aim of the *mixed-criticality* approach, in general sense, is to reduce the number of computing platforms in the system, which consequently should reduce the time for design, the cost of the system as well as the power consumption. Therefore, in last ten years many projects were funded by the European Union for research of mixed-criticality systems, namely:

- MultiPARTES,
- DREAMS,
- CONTREX,
- SAFURE, etc.

These projects aim to solve many problems which emerged by introduction of mixed-criticality concept in the design of safety-critical systems. These problems include:

- temporal and spatial isolation of functions with different criticality,
- implementation of mixed-criticality aware schedulers,
- techniques for testing, verification and validation of software design,
- mixed-criticality aware techniques for safety assessment process of safety-critical systems.

In this research, the focus is on two important metrics of real-time mixed-criticality systems: schedulability and quality of service. *Schedulability* is the ability of the system of being schedulable, i.e., at least one algorithm exists which can produce a feasible schedule of system functions. Schedulability *per se* is a minimal requirement which has to be ensured in safety-critical systems in the context of critical system functions. On the other hand, the *quality of service* in mixed-criticality systems depends on the quality of schedule of non-critical tasks in the system with regard to a certain performance metric. In other words, quality of service measures the degradation of system performance with regard to imposed constraints. Although the methods developed in this research target schedulability and quality of service, they have effect on other real-time system properties such as runtime robustness, predictability and stability.

1.2 Summary of basic definitions

For sake of completeness and clarity, the basic definitions of some important terms are provided as follows:

- **Schedulability** is the ability of being schedulable. System is schedulable if there exists at least one algorithm that can produce a feasible schedule.
- **Quality of service** is a metric of degradation of system performance with regard to imposed constraints.
- **Real-time** systems are systems in which the validity of a system function depends on timeliness.
- **Mixed-criticality** systems are systems in which a single computing platform executes functions of different criticality.
- **Safety-critical** systems are systems in which an error can cause significant financial damage or loss of life.
- **Overloaded** systems are systems in which there is a lack of resources required by system functions to meet a set of imposed constraints.

Note that, by definition, mixed-criticality systems are not safety-critical unless at least one function in the system is critical. In this work, the focus is on mixed-criticality systems with at least one safety-critical function. Moreover, note that an error in safety-critical system in the context of real-time systems corresponds to delay, overload, overrun, or any violation of timing constraints.

1.3 Thesis hypotheses and contributions

The goal of the proposed research is to provide methods for improving schedulability and quality of service in real-time mixed-criticality systems.

The main hypotheses are the following:

1. Schedulability tests for fixed-priority adaptive mixed-criticality system model can be improved.
2. Existing harmonic period assignment approaches can be improved by imposing additional constraints on a system design.
3. Heuristics generated using genetic programming can be used in mixed-criticality systems to increase the quality of service of non-critical tasks.

To show that each of the latter hypotheses holds, novel methods are devised and they correspond to scientific contributions of the thesis. The main contributions of this research are:

1. Method for schedulability testing for adaptive mixed-criticality systems with fixed prior-

ities.

2. Method for assignment of harmonic periods for improving the schedulability of safety-critical tasks in real-time mixed-criticality systems.
3. Method for task scheduling for improving the quality of service of non-critical tasks in real-time mixed-criticality systems based on genetic programming.

1.4 Organization

Contributions of this thesis are introduced in three separate chapters, which address different aspects of mixed-criticality scheduling. Each chapter brings a unique method which correspond to each of the three contributions and hypotheses stated in the previous section. Additionally, chapter 2 explains preliminaries for real-time and safety-critical systems, which are highly important for the rest of the thesis.

In chapter 3, a **method for schedulability testing in adaptive mixed-criticality system with fixed priorities is introduced**. Moreover, in this chapter following topics are discussed:

- Summary of existing schedulability tests and response-time analysis for fixed-priority mixed-criticality systems (section 3.5).
- Novel schedulability test for fixed-priority adaptive mixed-criticality systems (section 3.6).
- Validation and correction of an existing exact schedulability test (section 3.8).
- A framework for evaluation of schedulability tests (section 3.9).

In chapter 4, a **method for assignment of harmonic periods for improving the schedulability of safety-critical tasks in real-time mixed-criticality system**. Moreover, in this chapter following topics are discussed:

- Summary of existing work on harmonic period assignment in control real-time systems (section 4.2.1).
- Introduction of novel harmonic period assignment method for real-time systems (sections 4.3-4.8.2).
- Evaluation of the devised harmonic period assignment and comparison with the existing approaches (section4.9).

In chapter 5, a **method for task scheduling for improving the quality of service of non-critical tasks in real-time mixed-criticality systems based on genetic programming**. Moreover, in this chapter following topics are discussed:

- Summary of existing work in overloaded real-time systems, overloaded mixed-criticality systems and scheduling using genetic programming (section 5.2).
- Introduction of novel method for improvement of the quality of service in real-time systems (sections 5.3.1-5.4).

- Evaluation of the devised method using in the single-objective optimizing configuration (section 5.5).
- Extending the method using multi-objective optimization and acceptance tests (sections 5.6-5.7).

In chapter 6, concluding remarks are stated and discussed.

Chapter 2

Preliminaries

The aim of this chapter is to highlight distinct research areas related to this research. Additionally, it provides a brief description of basic models and tools for analysis used throughout the thesis. Moreover, it depicts the industrial aspects of mixed-criticality systems with the aim of providing the better context for the rest of the thesis.

2.1 A brief introduction to real-time scheduling

Real-time scheduling theory serves as a theoretical basis for design and development of safety-critical and mixed-criticality embedded systems with timing constraints. In this chapter, a brief overview of basic real-time system task models, which are used in this research, will be provided. The choice of the appropriate system model is crucial for efficient system analysis. Nowadays, there are plenty of different approaches to analysis of real-time systems, each having specific properties applicable to a group of real-time systems. Naturally, more general real-time system models exist as well. Intuitively, more general system models increase the complexity of system analysis. The relation between expressiveness of a model and the difficulty of analysis is depicted in Fig. 2.1. As it can be seen in the figure, lower expressiveness enables a more efficient system analysis. Since in the safety-critical real-time systems arguments for safety of the system have to be made as clear as possible, it is more appropriate to choose the model with lower expressiveness. Therefore, both, the industry and the academia, have been using sporadic and periodic task models for modeling real-time systems since their introduction in the early real-time research [2, 3]. In the remainder of the section, these models will be explained in detail.

2.1.1 Sporadic and periodic task model

In sporadic and periodic task models, the system is described as a set $\mathscr{T} = \{\tau_1, \tau_2, ..., \tau_n\}$ of n independent tasks which generate infinite series of jobs. Each task τ_i in the system is typically

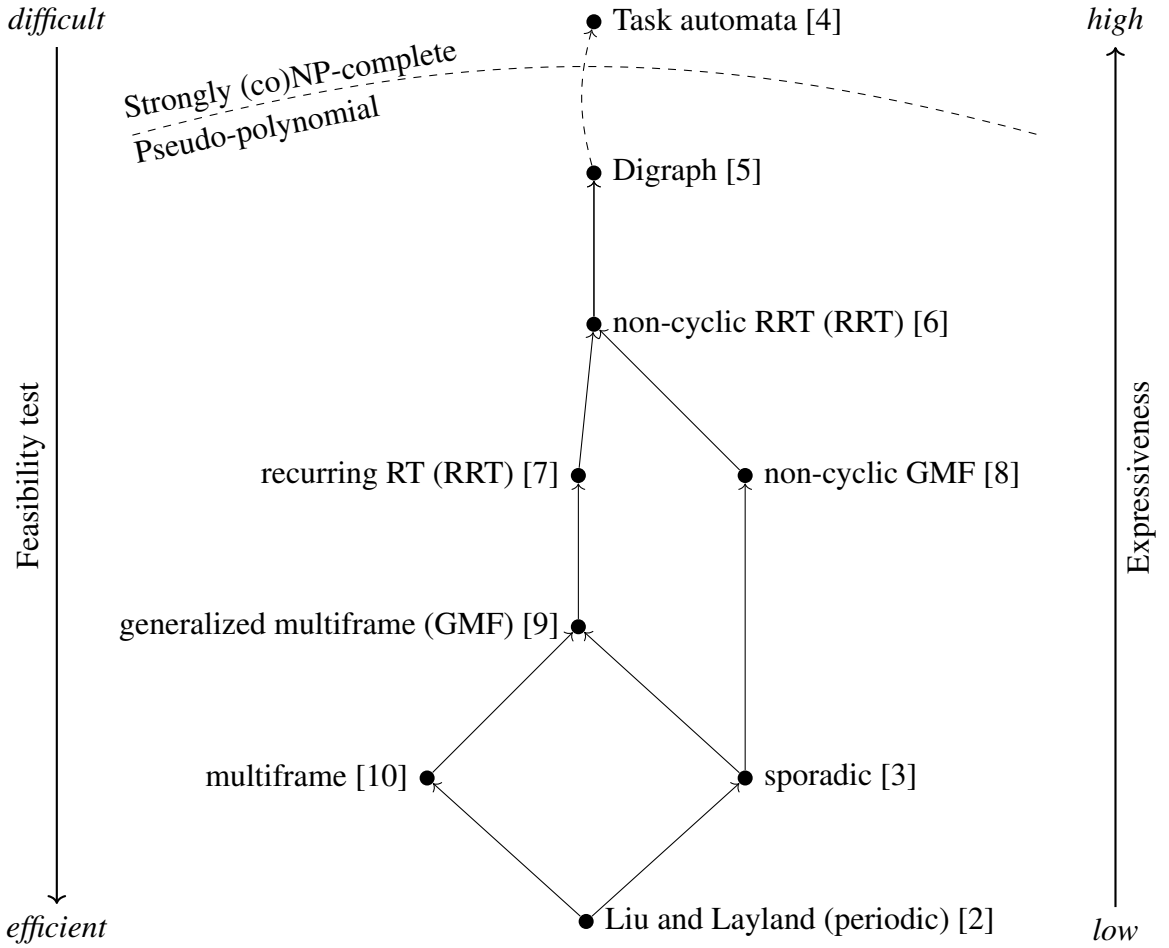


Figure 2.1: Comparison of different real-time system models with regard to the difficulty of analysis and expressiveness of the model (based on the illustration from [5]). Arrows show the direction of the generalization.

described as a set with three elements:

$$\tau_i = \{C_i, T_i, D_i\} \tag{2.1}$$

where C_i is the worst-case execution time (WCET) and $D_i \leq T_i$ is the relative deadline. In case of periodic systems parameter T_i denotes the period, and in sporadic systems this parameter corresponds to the minimum interarrival time of two consecutive instances, i.e., jobs, of a task τ_i. In certain cases, deadline is implicit, i.e., $D_i = T_i$. The j-th job of the i-th task in the system is denoted with τ_{ij} and can be represented as a set:

$$\tau_{ij} = \{c_{ij}, d_{ij}\} \tag{2.2}$$

where c_{ij} denotes the execution time, and d_{ij} corresponds to the absolute deadline of a job. The basic metric used for comparison of task sets is *processor utilization factor* U which is defined as:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2.3}$$

Intuitively, processor utilization factor corresponds to the difficulty of finding a feasible schedule for a task set. Moreover, it is well known that for a task set with utilization such that $U > 1$ a schedule does not exist upon a uniprocessor computing platform. In this research, the focus is primarily on the scheduling problems on preemptive uniprocessor platforms since safety-critical systems of interest typically employ single core microprocessor units in the implementation.

2.1.2 Uniprocessor real-time scheduling

Finding a feasible schedule or a scheduling policy that will generate a feasible schedule is one of the most important problems in the design or real-time systems. In real-time systems, jobs that are generated by sporadic or periodic tasks are scheduled according to a job priority assignment. For clarity, more formal definitions are provided below.

Definition 1. ***Schedule.*** *Schedule $\mathscr{S}$ is a function which assigns a value to every job τ_{ij} in the system, at each discrete instant of time t:*

$$\mathscr{S}(\tau_{ij}, t) : \mathbb{N}^2 \rightarrow \mathbb{N} \tag{2.4}$$

The assigned value corresponds to the resource assigned to a job. *

For uniprocessor real-time scheduling, schedule is a function $\mathbb{N}^2 \rightarrow \{0, 1\}$ since the only resource in the system is processing time upon a single processor.

*Note that each job can have an unique index. Thus, the function maps from $\mathbb{N}^2$ to $\mathbb{N}$.

Definition 2. ***Priority assignment function.*** *Priority assignment function is a mapping that assigns a priority to each job in the system at any given discrete time instant t:*

$$\Pi(\tau_{ij}, t) : \mathbb{N}^2 \rightarrow \mathbb{N} \tag{2.5}$$

Definition 3. ***Scheduler.*** *Scheduler $\mathscr{S}$ is an algorithm which generates a schedule by assigning a resource to a job according to priority assignment function* Π.

Generally, classes of schedulers, i.e., scheduling algorithms, can be distinguished based on the type of priority assignment which is employed. There are two distinct groups of schedulers with regard to the type of priority assignment:

- **Fixed-priority** or **static** scheduling algorithms assign priority to tasks and their corresponding jobs only once, prior to the system runtime.
- **Dynamic** scheduling algorithms can assign a priority to jobs generated by sporadic or periodic tasks at any point in time during the system runtime.

In uniprocessor systems, the scheduler assigns processing time to a job of a task with the highest priority.

2.1.3 Schedulability analysis

The primary goal of the analysis of real-time systems comprised of sporadic or periodic tasks is to determine whether a given task set can be scheduled with an algorithm in a manner that each generated job completes its execution by its corresponding deadline. More precisely, the goal is to determine if a feasible schedule exists for a given scheduling algorithm and a task set. The secondary, but also very important problem, is to determine an algorithm which can generate a feasible schedule.

Definition 4. ***Schedulability test.*** *Schedulability test for a scheduling algorithm $\mathscr{A}$ is a condition which is used to verify if a feasible schedule exists for a given task set.*

Based on the type of the schedulability condition, there are three distinguished classes of schedulability tests:

1. Necessary schedulability test yields **NO** if a task set is not schedulable.
2. Sufficient schedulability test yields **YES** if a task set is schedulable.
3. Necessary and sufficient, i.e, exact, schedulablity test yields **YES** if a task set is schedulable and **NO** if a task set is not schedulable.

Intuitively, a more precise schedulability test requires more computational power in comparison with a less precise one. Therefore, for an exact schedulability test, one would have to allocate the larger amount of computational power than for the corresponding necessary or sufficient

schedulability test. Note that a simple utilization-based necessary schedulability test was introduced in the previous section, i.e., a task set is not schedulable if $U > 1$. However, such a test cannot discover infeasible task sets with $U \leq 1$.

2.1.4 Response-time analysis

In fixed-priority preemptive uniprocessor real-time systems comprised of periodic or sporadic tasks, schedulability tests are often based on the response-time analysis. Response-time analysis is a technique of determining the *worst-case response time* (WCRT) of each task in a task set. If the WCRT is known for each task in the system, schedulability is determined by comparing WCRTs to the corresponding relative deadlines. Generally, the response time R_i of a job of task τ_i in the system can be expressed as:

$$R_i = C_i + I_i \tag{2.6}$$

where I_i is the interference, i.e., time spent for executing, of jobs of tasks with the priority higher than τ_i. To determine the WCRT of a task in the system, the worst-case interference from tasks with higher priority in the system has to be taken into account when the WCRT is calculated. It is well known that the worst-case interference of tasks with priority higher than the observed task occurs when all jobs are released simultaneously, i.e., synchronously. This event is known as *synchronous arrival sequence* or *critical instant*. In sporadic systems and periodic systems without the initial offset, a critical instant occurs at the time instant 0 and at the every other simultaneous arrival of jobs of all tasks in the system. These simultaneous arrivals occur every hyperperiod which is generally the least common multiple of periods of all tasks. Therefore, the worst-case interference can be expressed as:

$$I_i = \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \tag{2.7}$$

where set $hp(\tau_i)$ contains tasks with priority higher than τ_i. Term $\left\lceil \frac{R_i}{T_j} \right\rceil$ corresponds to the number of releases of a task with higher priority τ_j up to the R_i time units. By multiplying the number of releases with the worst-case execution time of higher-priority task, i.e., C_j, the interference caused by τ_j is taken into account.

To calculate the WCRT of task τ_i, the following recurrence relation has to be solved in an iterative or a recursive manner using C_i as the initial value for R_i in the calculation:

$$R_i = C_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \tag{2.8}$$

Using the latter response-time analysis the exact schedulability test for periodic and sporadic systems can be expressed as follows.

Theorem 1. ***Schedulability of a sporadic task set.*** *A sporadic task set $\mathscr{T}$ is schedulable with fixed-priority scheduler if R_i given with equation (2.8) is such that $R_i \leq D_i$ for each task $\tau_i \in \mathscr{T}$.*

In this research, especially in the chapter 3, response-time analysis techniques are heavily exploited to extend and improve existing adaptive mixed-criticality schedulability tests.

2.2 Industrial context of safety-critical systems and introduction of criticality levels

In the design of safety-critical systems, non-functional characteristics such as safety, security and performance must be taken into account as well as the operative functions of the system. The process for assessing the safety characteristics of the system is called *system safety assessment process* [11]. In the next subsections, definitions of basic concepts in dependable and secure computing are introduced and connected with *system safety assessment process*. Moreover, industrial aspects of mixed-criticality systems are explained as well.

2.2.1 Concepts of dependable and secure computing

Definitions of service, error and failure which are introduced here can be found in Aviženis et al. [12]. These dependable and secure computing concepts are defined similarly in different dependable computer systems literature [13, 14] and they are used extensively in safety standards.

Definition 5. ***Service.*** *A service is behaviour of a system as it is perceived by a user. A service is* ***correct*** *when it implements its system function.*

Definition 6. ***Service failure.*** *A service failure or just failure, is an event that occurs when the delivered service deviates from correct service. It is a transition from correct service to incorrect service. There can be different forms of failure, which are referred to as* ***failure modes****. Each failure mode has its* ***failure severity****.*

Definition 7. ***Error.*** *An error is deviation of a system service from correct service.*

Definition 8. ***Fault.*** *Fault is the adjudged or hypothesized cause of an error.*

Based on the latter definitions, causal connection between these concepts is self-evident. When a fault in a system is active, it produces an error. An error causes a transition from a correct service to an incorrect service (service failure). This is illustrated in Fig. 2.2.

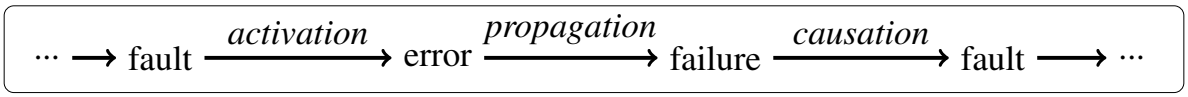


Figure 2.2: Causal connection between fault, error and failure. Based on illustration from [12].

2.2.2 Critical services and safety assessment process

Based on the system or application requirements, a certain function, i.e., system service, can be deemed critical in the sense of a financial loss or danger for human life. In other words, failure of such a service causes unacceptable damage. In that regard, all components of electronic systems which are necessary for the correct execution of the critical service have to be properly designed. These includes both the hardware and software part of an electronic system. In this research, the focus is mainly on the software aspect of safety-critical systems.

Software hazard analysis is a process that should identify the parts of the software which could interfere with the correct operation of the system. Moreover, the hazard analysis should classify the interference in terms of severity while taking into account techniques that are employed for the mitigation of interference. Software hazard analysis is a requirement, and it is often used as a base for fault and failure mode analyses [15]. There are several types of fault analysis and failure mode analysis that can be found in safety standards:

- *Fault Tree Analysis* (FTA) [16],
- *Failure Modes and Effects Analysis (FMEA)* [17],
- *Failure Modes and Effect Criticality Analysis (FMECA)* [17].

These techniques are used for analysis of possible software failures in the system. They are performed after the hazard analysis and can discover different failures that are undiscoverable by the hazard analysis. The aim of the failure modes analysis is to assign a corresponding failure mode to every failure of a safety-critical system function. In the context of embedded computing platforms, the failure mode can be:

- non-execution,
- late execution,
- incorrect execution [11].

To every failure, severity is assigned based on the effects that failure causes to the corresponding critical service. The last step in failure mode analysis, such as FMEA, is the identification of the existing compensating provisions that can mitigate the effects of failure. The final result of these processes is the assignment of a ”criticality“ level to a failure mode based on the effect analysis and discovered compensating provisions, i.e., mitigation techniques. The assigned ”criticality“ level is referred to as the *development assurance level* (DAL) [11]. This terminology is specific to DO-178C [18] (avionics domain). The terminology in other standards is different, but the basic concept remains preserved (e.g., IEC 61508 and EN-50128 use SIL - *Safety Integrity Level*

Table 2.1: Comparison of nomenclature of development assurance levels across different industrial domains. Mapping of safety integrity levels is approximate. Table is based on the table found in [22].

Domain	**Domain-specific nomenclature**					
Automotive (ISO 26262)	QM	ASIL-A	ASIL-B	ASIL-C	ASIL-D	-
General (IEC-61508)	SIL-0	SIL-1	SIL-2		SIL-3	SIL-4
Railway (EN 50128)	SIL-0	SIL-1	SIL-2		SIL-3	SIL-4
Space (ECSS-Q-ST-80)	Category E	Category D	Category C		Category B	Category A
Aviation: airborne (DO-178)	DAL-E	DAL-D	DAL-C		DAL-B	DAL-A
Aviation: ground (DO-278)	AL6	AL5	AL4	AL3	AL2	AL1

[19, 20], ISO 26262 uses ASIL - *Automotive Safety Integrity Level* [21]). Table 2.1 contains safety integrity levels as defined in the aforementioned safety standards sorted with regard to the safety assessment severity. It can be seen that the SIL-4 level requires the most rigorous safety assessment process, which is reasonable since the failure of SIL-4 system services can cause significant damage.

Note that the notion of criticality in mixed-criticality systems that is used throughout the academic and industrial research, originates from different safety integrity levels, which are typically found in safety-critical systems. Almost any safety-critical electronic system today contains at least two services, which are different with regard to safety integrity level. In chapter 3, it is clarified how the raise in the safety assessment rigor affects the mixed-criticality system from the timing, i.e., real-time, perspective.

2.2.3 Mixed-criticality system design

The choice of the proper computer system architecture for mixed-criticality applications is crucial for: firstly, safety of the system, and secondly, efficiency of the system. In general, system designers find these requirements to be in a contradiction, which poses a significant issue in the design of safety-critical systems. In other words, the problem is to design an efficient and safe system. Often, efficiency of the system is disregarded in favor of safety of the system, which may be reasonable from the safety aspect of the system, but this can be very expensive from the system production and development aspect. As stated in chapter 1, the aim of ongoing mixed-criticality system research is to increase the efficiency of systems. To further clarify this issue, the common approaches to mixed-criticality system design are reviewed and the advantages and drawbacks of each approach are discussed in the remainder of the section. Figs. 2.4-2.6 depict abstract architectures of mixed-criticality systems, which are commonly found in real-world applications nowadays. Refer to Fig. 2.3 for differences in borders in the figures.

The traditional federated approach to mixed-criticality system design that follows the “one

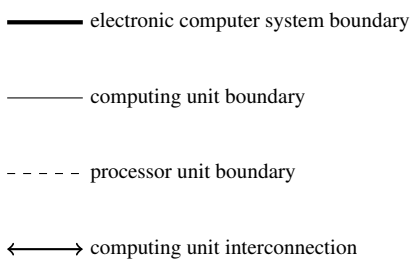


Figure 2.3: Legend highlights the differences between different boundaries in the system figures.

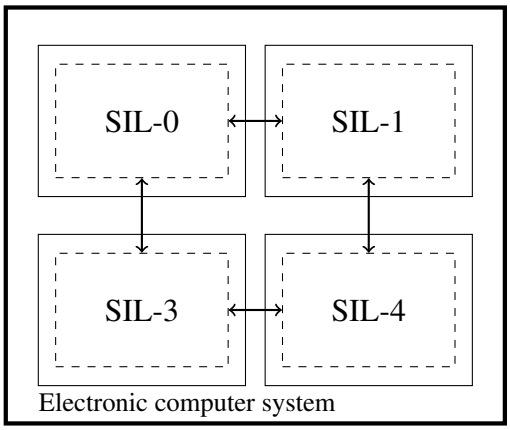
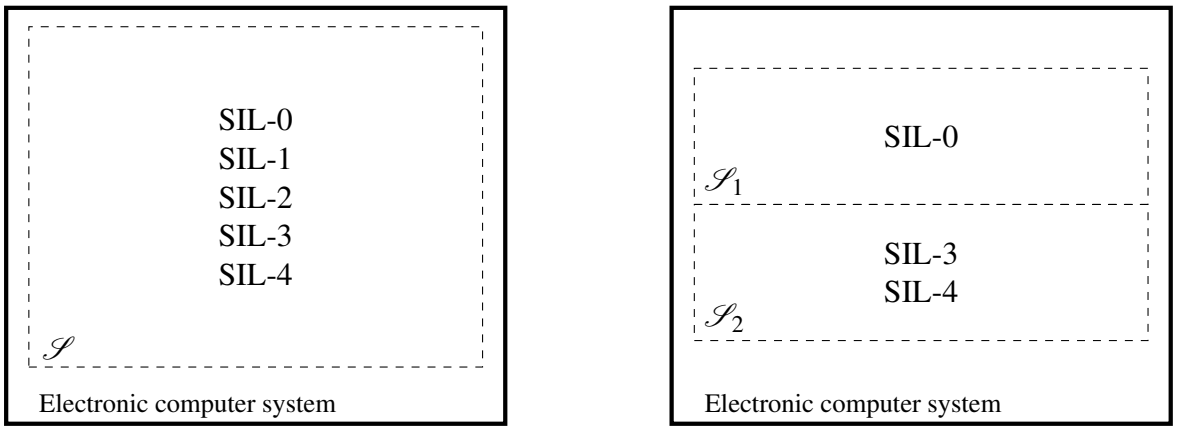


Figure 2.4: Functions with different safety integrity levels distributed across computing platforms in the system.

computer - one function" paradigm is shown in Fig. 2.4. In such an approach, functions of different importance are distributed across different computing platforms, which are typically uniprocessor platforms connected with a robust external communication bus such as CAN (*Controller Area Network*). The main advantage of such an approach is that each system can have independent system design and corresponding safety assessment process required for the targeted safety integrity level. Moreover, safety of the overall system is increased as the possible interference between functions of different criticality is reduced due to the usage of independent computing platforms, i.e., systems are completely spatially isolated. An obvious drawback of such an approach is the potentially high number of separate computing units in the system, which reduces the overall efficiency of the system due to the higher power consumption. Moreover, connection with the external communication bus may be limiting in some applications, which further diminishes system performance efficiency-wise.

Today, the efficiency of the system can be increased by reducing the number of computing units in the system. This was not an option twenty years ago since the processing power of embedded computers was significantly lower than today. Figs. 2.5 and 2.6 show the approaches in which multiple functions of different criticality can be allocated on the same uniprocessor or multiprocessor computing platform. An immediate drawback of such an approach is the loss of the full spatial isolation. However, development of software which follows the recommendations of safety standards can significantly mitigate the lack of spatial independence. Moreover,



(a) Functions of different criticality level are scheduled with single scheduler $\mathscr{S}$ on a single processor.

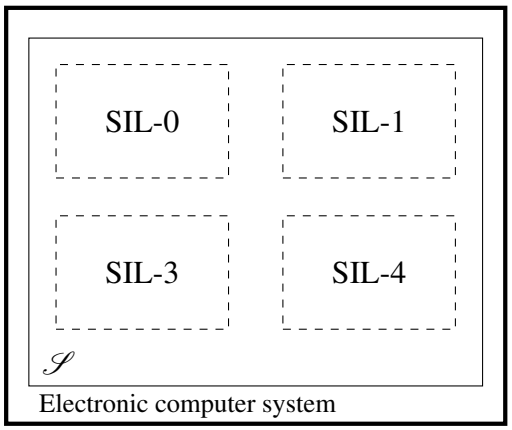
(b) Functions of different criticality level are scheduled with partitioned scheduler, i.e., with scheduler $\mathscr{S}_1$ and $\mathscr{S}_2$, on a single processor.

Figure 2.5: Mixed-criticality system design approaches on uniprocessor platforms.

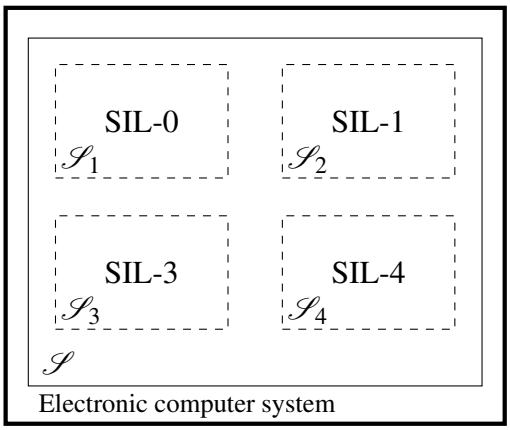
the efficiency can be increased.

The system architectures on uniprocessor platforms presented in Fig. 2.5 are relatively simple and probably are used the most among the single processor approaches to the mixed-criticality system design. In the first approach, depicted in Fig. 2.5a, a single scheduler is used to determine a schedule for functions of higher criticality (from SIL-1 to SIL-4) and lower criticality (SIL-0). Scheduler $\mathscr{S}$ in such a configuration has to take into account different timing properties of functions with lower and higher criticality. This will be further explained in chapter 3, where scheduling techniques in such systems are presented, and schedulability in such systems is investigated. In the second uniprocessor approach, depicted in 2.5b, a partitioned scheduler is used, i.e., scheduling of critical functions is temporally isolated from scheduling of non-critical functions in the system. Such an approach is efficient from the aspect of reducing the number of different computing units in the systems. However, enforcing temporal isolation decreases processing unit efficiency since non-critical functions can be executed only when all the critical functions are executed correctly. More precisely, processor time is inefficiently exploited. Nevertheless, an approach with partitioned scheduler has an advantage from the safety aspect. Note that enforcing temporal isolation has an analogous effect on efficiency in uniprocessor systems as enforcing spatial isolation has in the federated systems. Scheduling of critical functions in the mixed-criticality system with partitioned scheduler is further investigated in chapter 4, where period assignment techniques for task scheduling are presented. In chapter 5, both uniprocessor configurations are considered.

Although in this thesis multiprocessor configurations are not in the main focus, in the continuation two more configurations are presented, which are often considered in the design of mixed-criticality systems. These approaches are depicted in Fig. 2.6. The first multiprocessor configuration shown in Fig. 2.6a, is analogous to single scheduler approach shown in Fig. 2.5a.



(a) Functions of different criticality level are scheduled with single scheduler $\mathscr{S}$ on multiple processors.



(b) Functions of different criticality level are scheduled with local schedulers S_1, S_2, S_3, and S_4 on dedicated processors based on criticality. Scheduler S determines the global schedule.

Figure 2.6: Mixed-criticality system design approaches on multiprocessor platforms.

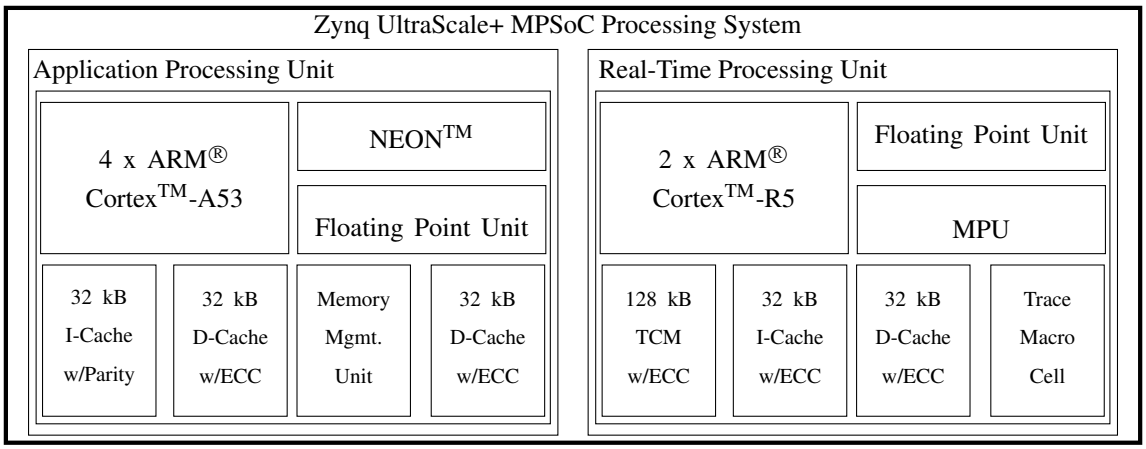


Figure 2.7: Architecture of Zynq UltraScale+ embedded computer systems. Based on illustration from [23].

However, a significant benefit in the multiprocessor approach is increase in temporal isolation, which is the consequence of increase in the processing power. A drawback of such an approach is the lack of spatial isolation since a single scheduler is used for all functions. The second multiprocessor configuration shown in Fig. 2.6b increases both temporal and spatial isolation, as the functions of the same criticality are always executed on the dedicated processor core and scheduled with an independent scheduler ($\mathscr{S}_{1-4}$). The global scheduler $\mathscr{S}$ determines the criticality of each core. Generally, the number of local schedulers $\mathscr{S}$ can be higher than the number of physical processing units.

From the implementation point of view, approaches shown in Figs. 2.4-2.5 are implemented using simple single-core microcontroller computing units that were improved significantly in a sense of processing power in recent years. Moreover, multiprocessor configurations in Fig. 2.6 are implemented on multiprocessor embedded platforms, which are designed to address

the already mentioned issues of safety and efficiency in the system. An example of such an embedded computing platform is depicted in Fig. 2.7. Fig. 2.7 illustrates Xilinx Zynq UltraScale+ system on chip that contains ARM Cortex-A53 processors. The architecture of these and similar embedded processors encouraged engineers to develop frameworks which use more advanced software abstractions for temporal and spatial isolation [24]. These approaches include hypervisors, i.e., virtual machine monitors, based on ARM virtualization extensions, and secure monitors based on ARM TrustZone extensions. In Fig. 2.6b scheduler $\mathscr{S}$ corresponds to the scheduler of a virtual machine monitor, and schedulers $\mathscr{S}_{1-4}$ correspond to the schedulers of guest operating systems.

Chapter 3

Method for schedulability testing for adaptive mixed-criticality systems with fixed priorities

3.1 Context of the research

The focus in this chapter is on uniprocessor fixed-priority systems, which execute functions of different criticality. These systems are depicted in Fig. 2.5a. Existing methods for schedulability testing in adaptive mixed-criticality systems are reviewed and discussed. Moreover, novel methods for schedulability testing and corrections to existing schedulability tests are presented. The formal definition of the adaptive mixed-criticality system behavior is provided in the subsequent sections. Some of the results presented in this chapter were published in [25] and [26].

3.2 The WCET estimation problem and mixed-criticality conjecture

In chapter 2, it was stated that efficiency of real-time systems is significantly impacted by introduction of spatial and temporal isolation in the architecture of systems. Intuitively, isolation degrades efficiency, but increases the safety-related characteristics. On the lower level of analysis, allocation of an appropriate time slot for execution of a task, i.e., function or system service, is equally important for efficient system design as an appropriate choice of system architecture. For instance, if allocated time for execution of a certain system service is too long, the number of different functions which can be executed on the processor will be reduced. Therefore, to efficiently design a system, allocated execution times, which typically correspond to WCETs, have to be carefully chosen. However, this is not a particularly easy task. Moreover, if this re-

quirement is not met, the efficiency of system can be deteriorated regardless of the architecture that is chosen.

Nowadays, a plethora of different techniques for estimation of the WCET exists [27]. Often, the WCET estimation techniques have to be combined in order to increase assurance in the WCET estimation. Usage of a combination of the WCET estimation techniques becomes a strict requirement as the complexity of observed embedded computer platform increases. Today, we witness the increase in architectural complexity even in relatively simple RISC (*Reduced Instruction Set Computer*) processor architectures due to the introduction of architectural modifications such as cache memories, branch predictions, multi-stage pipelines, etc. Although the processing time of a task can be decreased in the best-case scenario, these features significantly increase the uncertainty in the knowledge of the execution time in the worst case, i.e., they raise the uncertainty in estimation of the WCET. Uncertainty in the knowledge of the WCET in this context is primarily epistemic, rather then aleatory. This means that the uncertainty in the knowledge of the WCET is caused by our lack of knowledge about the system rather than the system itself [28].

Therefore, in safety-critical systems, system designers are often pressured by requirements of safety standards to run extensive component and integration testing of a system function to discover the actual WCET. Intuitively, with more exhaustive and comprehensive testing and measurements, larger WCET values will be discovered for a certain task. Since the requirements of safety standards are more strict for functions with higher safety integrity level (SIL), larger WCET values will have to be allocated for functions with higher SIL. In addition, in many cases, it is mandatory to overestimate the actual value of the WCET for safety precautions. Again, this will degrade the efficiency of the system. These assumptions in a similar form were stated in the seminal mixed-criticality scheduling paper by Vestal [29] and resulted in a significant publishing and research trend. In last ten years, a large number of different techniques for scheduling mixed-criticality tasks emerged which mitigate efficiency degradation in mixed-criticality systems. To better illustrate this trend and mixed-criticality scheduling paradigm, a simple example, which is often found in the mixed-criticality scheduling literature, is discussed in the next section.

3.3 Motivational example for mixed-criticality scheduling

In this section, a motivational example that is often found in the literature [30, 31], which illustrates a basic mixed-criticality scheduling problem, is presented.

Example 1. *Consider a system that consists of three jobs J_1, J_2, J_3, which are executed on a fixed-priority uniprocessor platform. These jobs differ in criticality. More precisely, job J_1 does not execute safety-critical functions, and J_2 and J_3 execute safety-critical functions. Therefore,*

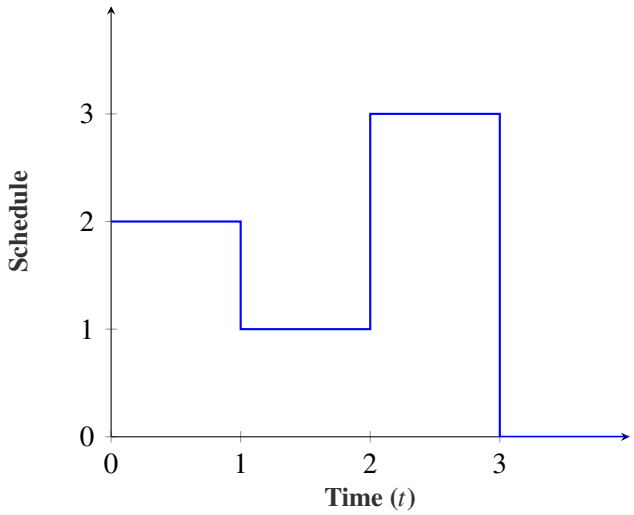


Figure 3.1: A feasible schedule when J_1 has the "medium" priority. Note that priority assignment of jobs J_2 and J_3 can be interchanged.

the execution of J_2 and J_3 requires certification and rigorous examination in safety assessment process. In this process, the system designer has to present an appropriate justification for choice of timing parameters and scheduling technique to an external accredited safety assessor. The parameters of jobs as specified by the system designer are:

$$\begin{aligned} J_i &= \{C_i, D_i\} \\ J_1 &= \{1, 2\} \\ J_2 &= \{1, 3.5\} \\ J_3 &= \{1, 3.5\} \end{aligned} \tag{3.1}$$

Additionally, the system designer finds that there are three possible priority assignments for which the system is schedulable, i.e., the system is schedulable as long as the job J_1 is not assigned the lowest priority. Respective schedules, which show activity of jobs over time, are illustrated on Figs. 3.1 and 3.2.

From the perspective of the system designer, the problem is solved and adequate safety properties are acquired with any choice of particular schedule with the given job parameters.

During the safety assessment process of the system, the safety assessor determines that additional time may be needed for the execution of critical jobs J_2 and J_3. Therefore, the safety assessor demands that the system designer increases the WCET of critical jobs by 0.5 time units since the execution of these jobs is safety-critical. After the latter modification, new job

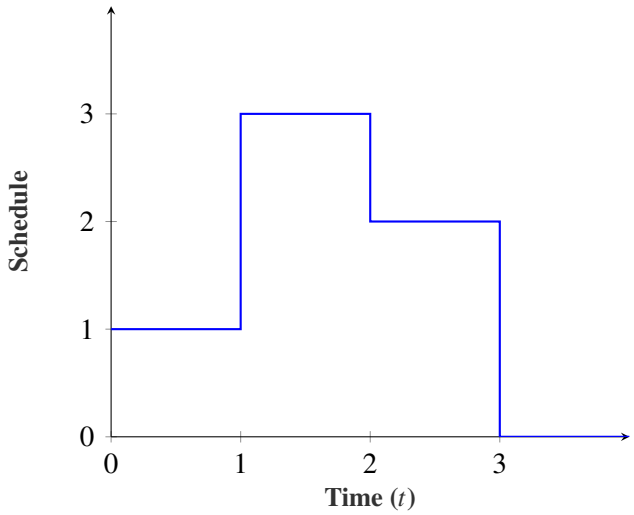


Figure 3.2: A feasible schedule when J_1 has the "highest" priority. Note that priority assignment of jobs J_2 and J_3 can be interchanged.

parameters are:

$$\begin{aligned} J_i &= \{C_i, D_i\} \\ J_1 &= \{1, 2\} \\ J_2 &= \{1.5, 3.5\} \\ J_3 &= \{1.5, 3.5\} \end{aligned} \tag{3.2}$$

At this point, the system designer concludes that there is no priority assignment in the system that would guarantee that all jobs will be executed. This is illustrated in Figs. 3.3-3.4. From the perspective of the safety assessor, the system designer can generate a feasible schedule by discarding the non-critical job J_1. However, the system designer is concerned with the efficiency of the system, i.e., dropping the job J_1 will decrease the overall efficiency. A solution which would satisfy both the assessor and the system designer would have the following properties:

1. *efficient resource usage has to be preserved,*
2. *J_2 and J_3 have to be safely executed.*

Again, from the aspect of the system designer, such a schedule exists, i.e., J_2 and J_3 spend at most one time unit for execution, but from the perspective of the safety assessor there is no firm guarantee that J_2 and J_3 will be safely executed.

An example of a solution to this problem is illustrated in Fig. 3.5. Note that the blue line in the figure corresponds to a system behavior that the system designer expects. On the other hand, the red line corresponds to the expectations and requirements of the safety assessor, i.e., J_2 and J_3 are executed although J_1 is discarded. For the system to be safe and efficient it has to enable that both behaviors are possible, but not at the same time. The fixed-priority scheduler which ensures that both scenarios are possible yields the following priority ordering J_2, J_1, J_3,

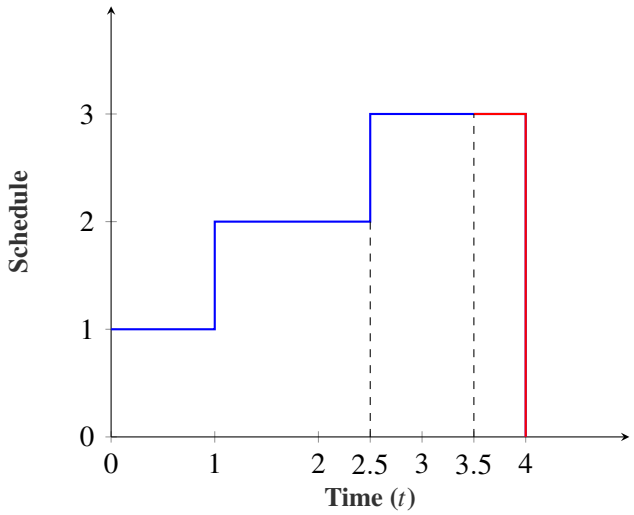


Figure 3.3: A priority order J_1, J_2, J_3 is not feasible since J_3 misses deadline.

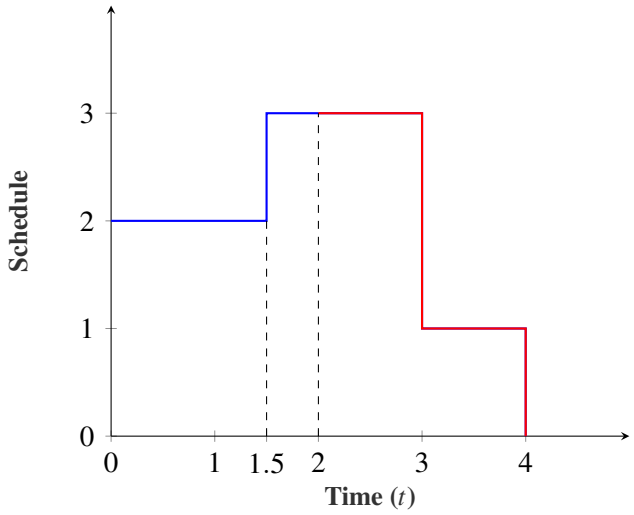


Figure 3.4: A priority order J_2, J_3, J_1 is not feasible since J_1 misses deadline.

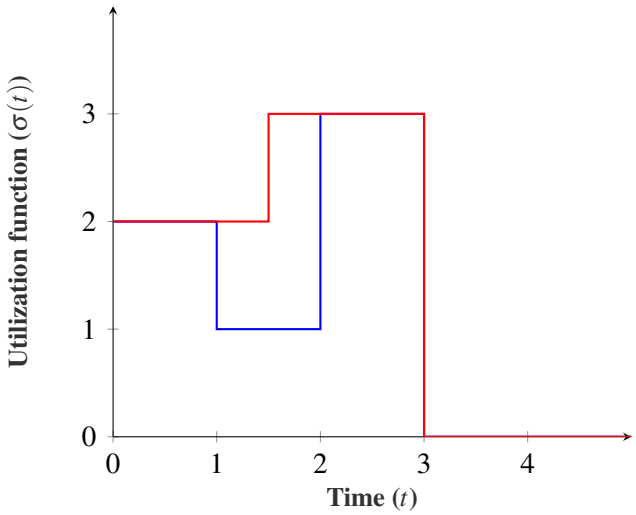


Figure 3.5: A priority order J_2, J_1, J_3 can produce a feasible schedule with appropriate runtime monitoring.

i.e., J_2 is the highest priority task. At the time instant 1 *bifurcation in Fig. 3.5 occurs as there are two possible outcomes:*

1. *Job J_2 has finished execution at time instant* 1.
2. *Job J_2 did not finish the execution at time instant* 1.

If the first statement is true, the system designer's expectations are satisfied even if J_3 executes for 1.5 *time units, and the schedule is feasible. If the second contention is true, the safety assessor is correct and at time instant* 1 *job J_1 has to be discarded and only J_2 and J_3 are going to be executed. This way, in the worst-case, the expectations of both system designer and safety assessor are satisfied since system maintains efficiency and safety properties.*

The latter example illustrates mechanisms which can be used in scheduling of jobs with mixed-criticality levels, i.e., runtime monitoring and job discarding. More formal analysis of these mechanisms and runtime behaviors is presented in the subsequent sections.

3.4 Mixed-criticality sporadic and periodic task models

In order to analyze different aspects of mixed-criticality scheduling, sporadic and periodic task system models introduced in section 2.1.1 have to be extended to take different task WCET estimations into consideration. Therefore, system with m criticality levels is described with task set $\mathscr{T}$ that consists of n tasks described as sets:

$$\tau_i = \{\vec{C}_i, T_i, D_i, L_i\} \tag{3.3}$$

where $\vec{C}_i$ is the monotonic non-decreasing vector of m WCET values, and L_i is one out of m criticality levels. Although it is possible to have multiple criticality levels in a system, i.e., up to m

levels, the state-of-the-art research is typically focused on systems with two different criticality levels [28]. The main reason for this is that the analysis of dual-criticality models is the most applicable in practical use-cases. More precisely, in real-world scenarios, system designers deal with non-critical functions, which do not require certification, and critical functions, which require certification, as illustrated in Example 1. Therefore, in the analysis of dual-criticality task systems, two types of tasks exists: *low-criticality* (LO), and *high-criticality* (HI) tasks. The most generic schedulability definition for mixed-criticality task systems is as follows:

Definition 9. *Mixed-criticality task system is schedulable if each job released by high-criticality tasks finishes execution before its respective deadline.*

Note that the latter statement is not biconditional, and therefore if high-criticality jobs indeed finish execution before their respective deadline, mixed-criticality task system still may not be schedulable. More precisely, Definition 9 formulates a necessary condition for schedulability of mixed-criticality system. More specific schedulability definitions shall depend on the properties of runtime behavior, which will be discussed in the next section.

3.5 Runtime behaviors in mixed-criticality task systems

Example 1 illustrates that in order to produce a feasible schedule, a system has to handle additional events which occur during runtime, i.e., it should utilize some sort of runtime monitoring. However, this may not be a strict requirement for scheduling on a fixed-priority uniprocessor. In this section, different approaches that can be used for fixed-priority mixed-criticality task scheduling on uniprocessor platforms are discussed. First off, mixed-criticality task schedulers can be classified according to the ability of system to monitor execution time, i.e., runtime monitoring, into two different classes:

1. schedulers that do not employ runtime monitoring,
2. schedulers that employ runtime monitoring.

The main difference between these two classes is in the ability of system to detect overrun of a task. This is a precondition for any mitigating action which can be employed to ensure adequate scheduling of tasks. Secondly, mixed-criticality schedulers can be classified into two different classes with regard to the number of different system states:

1. scheduler with only one system state,
2. schedulers with multiple systems states.

System state, in this context, designates the number of global scheduling policy changes with regard to the initial, i.e., default state. In the literature, systems with only one state are known as *static mixed-criticality* (SMC) systems. On the other hand, systems with more than one system state are referred to as *adaptive mixed-criticality* (AMC) systems. AMC systems must implement runtime monitoring for operation, and SMC systems may implement runtime monitoring,

i.e., there are SMC systems with or without runtime monitoring. In the following subsections, schedulability analyses for the most common variants of static and adaptive mixed-criticality behavior are discussed.

3.5.1 Naive scheduling approaches

To better illustrate AMC and SMC approaches, it is useful to consider an intuitive approach that one would in employ in scheduling tasks with multiple criticality levels. This approach is known as *partitioned criticality* approach [30]. In such an approach, tasks with higher criticality have higher priority than tasks with lower criticality. Moreover, priority of tasks with the same criticality is assigned according to the *deadline-monotonic priority assignment* (DMPA) that is optimal in classical, i.e., non mixed-criticality, sporadic or periodic task systems. This approach is referred to as *criticality-monotonic priority ordering* (CrMPO) in the context of mixed-criticality priority assignments. This approach will most certainly yield a feasible schedule of high-criticality tasks if it exists, but it will perform very poorly with regard to low-criticality tasks, i.e., the approach is not optimal for a whole task set. To visualize this, consider a task set in which task with the lowest period is a low-criticality task. The interference of high-priority tasks with higher criticality, prior to the execution of the task with the lowest period will cause a deadline miss even if optimistic WCET estimation of tasks, i.e., $C_i(LO)$, is taken into account. The pitfalls of this approach are depicted in Example 2.

Example 2. *Consider a fixed-priority uniprocessor computing platform that executes the task set* $\mathscr{T} = \{\tau_1, \tau_2, \tau_3\}$. *Parameters of tasks are:*

$$\begin{aligned} \tau_i &= \{[C_i(LO), C_i(HI)], T_i, D_i, L_i\} \\ \tau_1 &= \{[1,2], 4, 4, LO\} \\ \tau_2 &= \{[1,2], 10, 10, HI\} \\ \tau_3 &= \{[1,2], 11, 11, HI\} \end{aligned} \tag{3.4}$$

The goal is to determine priority ordering which would ensure successful execution of all high-criticality tasks in the system, and if possible ensure successful execution of low-criticality tasks. If criticality and deadline-monotonic priority assignment are employed as described above, the following priority ordering τ_2, τ_3, τ_1 *is obtained. In such a configuration, high-criticality tasks* τ_2 *and* τ_3 *are always executed prior to the low-criticality task* τ_1. *To test schedulability, the schedulability test given with Theorem 1 can be employed. The corresponding response times*

are calculated in the following manner:

$$\begin{aligned} R_1 &= 2 + \left\lceil \frac{R_1}{10} \right\rceil \cdot 2 + \left\lceil \frac{R_1}{11} \right\rceil \cdot 2 = 6 \rightarrow \mathbf{R_1} \not\leq \mathbf{D_1} \\ R_2 &= 2 \rightarrow \mathbf{R_2} \leq \mathbf{D_2} \\ R_3 &= 2 + \left\lceil \frac{R_3}{10} \right\rceil \cdot 2 = 4 \rightarrow \mathbf{R_3} \leq \mathbf{D_3} \end{aligned} \tag{3.5}$$

As it can be seen, low-criticality task τ_1 misses its deadline and the task set is not schedulable. The task set is not schedulable even if the low-criticality estimation for WCET of task τ_1 is used, i.e., response time is $R_1 = 5 \not\leq 4$.

On the other hand, if the deadline-monotonic scheduling approach, which is known to be optimal for sporadic task system, is used, a feasible schedule is obtained. Priority ordering in that case is τ_1, τ_2, τ_3 and corresponding response times are:

$$\begin{aligned} R_1 &= 2 \rightarrow \mathbf{R_1} \leq \mathbf{D_1} \\ R_2 &= 2 + \left\lceil \frac{R_2}{4} \right\rceil \cdot 2 = 4 \rightarrow \mathbf{R_2} \leq \mathbf{D_2} \\ R_3 &= 2 + \left\lceil \frac{R_3}{4} \right\rceil \cdot 2 + \left\lceil \frac{R_3}{10} \right\rceil \cdot 2 = 8 \rightarrow \mathbf{R_3} \leq \mathbf{D_3} \end{aligned} \tag{3.6}$$

The obvious drawback of deadline-monotonic priority ordering is its incapability to distinguish low-criticality from high-criticality tasks. Note that when using the partitioned criticality, the low-criticality WCET could be used in analysis. In contrast, when using the deadline-monotonic priority ordering, high-criticality WCET has to be used in analysis since execution time of low-criticality task contributes to the interference of high-criticality tasks. This will be further addressed in the next subsection.

3.5.2 Static mixed-criticality systems without runtime monitoring

In Example 2, the provided schedulability analysis in case of the deadline-monotonic priority ordering is insensitive with regard to different criticality levels. This has not been obvious at first since deadline-monotonic priority ordering has yielded a feasible schedule. However, consider a case in which low-criticality task is the lowest priority task in the system. The allocation of high-criticality WCET for execution of low-priority low-criticality task is a waste of system resources. To address this issue, a simple mixed-criticality scheduling approach was introduced by Vestal in [29]. This approach is known as the static mixed-criticality without runtime monitoring and it is often referred to as **SMC-NO**. Schedulability for **SMC-NO** approach is defined with Definition 10.

Definition 10. ***Schedulability of SMC-NO. SMC-NO*** *system is schedulable iff two conditions are satisfied:*

1. *Tasks with high-criticality are schedulable with regard to their respective high-criticality WCET and high-critically WCET of tasks with higher priority.*
2. *Tasks with low-criticality are schedulable with regard to their low-criticality WCET and low-criticality WCET of tasks with higher priority.*

The first condition in the latter definition ensures the schedulability of high-criticality tasks even if there are low-criticality tasks with higher priority in the system. The second condition ensures that low-criticality tasks are schedulable if all tasks are executed within their low-criticality, i.e., optimistic, WCET estimation. However, if a higher-priority task executes longer than its allocated low-criticality execution time, jobs of low-criticality tasks are allowed to miss deadline, and in case of such an event they are immediately discarded. The main problem of this approach is that if low-criticality tasks are allowed to have a priority higher than high-criticality tasks, low-criticality tasks have to be certified to the same level of criticality as high-criticality tasks. This is a strict requirement since the correct execution of higher-priority low-criticality tasks is a necessary precondition for the correct execution of high-criticality tasks. From the practical perspective, this is expensive in a sense that an additional effort is needed for the certification of low-criticality tasks, which by themselves are not safety-critical. Formally, runtime behavior of **SMC-NO** system is defined as follows.

Definition 11. ***Runtime behavior of SMC-NO.*** *Tasks are scheduled according to priority assignment* Π*. Job of a low-criticality task is discarded if it misses its deadline.*

Schedulability of a task set in **SMC-NO** system can be determined using slightly modified recurrence relation (2.8) used for response-time analysis of classical sporadic task sets. Response-time of task τ_i in **SMC-NO** mixed-criticality systems is given with:

$$R_i = C_i(L_i) + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(L_i) \tag{3.7}$$

It is worth noting that the latter equation will yield lower response times than (2.8). Moreover, testing response times given with (3.7) against corresponding deadlines constitutes an exact schedulability test in **SMC-NO** systems. In the next example, **SMC-NO** systems are further illustrated.

Example 3. *Consider a fixed-priority uniprocessor computing platform that executes the task*

set $\mathscr{T} = \{\tau_1, \tau_2, \tau_3\}$. *Parameters of tasks are:*

$$\begin{aligned} \tau_i &= \{[C_i(LO), C_i(HI)], T_i, D_i, L_i\} \\ \tau_1 &= \{[2,4], 8, 8, HI\} \\ \tau_2 &= \{[1,2], 14, 14, HI\} \\ \tau_3 &= \{[2,4], 9, 9, LO\} \end{aligned} \tag{3.8}$$

The goal is to determine priority ordering such that conditions in Definition 10 are satisfied, i.e., response times given with equation (3.7) are such that $R_i \leq D_i, \forall i$. *If the deadline-monotonic priority ordering is applied, the order of tasks is* τ_1, τ_3, τ_2 *and the corresponding response times are:*

$$\begin{aligned} R_1 &= 4 \rightarrow \mathbf{R_1} \leq \mathbf{D_1} \\ R_2 &= 2 + \left\lceil \frac{R_2}{8} \right\rceil \cdot 4 + \left\lceil \frac{R_2}{9} \right\rceil \cdot 4 = 18 \rightarrow \mathbf{R_2} \not\leq \mathbf{D_2} \\ R_3 &= 2 + \left\lceil \frac{R_3}{8} \right\rceil \cdot 2 = 4 \rightarrow \mathbf{R_2} \leq \mathbf{D_3} \end{aligned} \tag{3.9}$$

As it can be seen, the deadline-monotonic priority assignment does not yield a feasible schedule. However, the priority ordering τ_2, τ_1, τ_3 *will yield a feasible schedule, i.e., the worst-case response times are:*

$$\begin{aligned} R_1 &= 4 + \left\lceil \frac{R_1}{14} \right\rceil \cdot 2 = 6 \rightarrow \mathbf{R_1} \leq \mathbf{D_1} \\ R_2 &= 2 \rightarrow \mathbf{R_2} \leq \mathbf{D_2} \\ R_3 &= 2 + \left\lceil \frac{R_3}{14} \right\rceil \cdot 1 + \left\lceil \frac{R_3}{8} \right\rceil \cdot 2 = 5 \rightarrow \mathbf{R_3} \leq \mathbf{D_3} \end{aligned} \tag{3.10}$$

Since τ_3 *has the lowest priority, the interference of high-criticality tasks is reduced due to the usage of optimistic, i.e., low-criticality, WCET values in the calculation of response time for* τ_3, *which makes the system schedulable. However, such a reduction is not available in the situation in which low-criticality tasks are not low-priority. Another important point to consider is that this example is in itself a proof that deadline-monotonic priority ordering is not optimal in* ***SMC-NO*** *systems. These two issues are discussed hereafter.*

It is known from Vestal's initial work [29] that priority assignment for **SMC-NO** can be obtained using a modified version of Audsley's optimal priority assignment (OPA) algorithm, which was originally used for priority assignment in periodic task systems with initial offset [32]. This algorithm was found to be optimal for SMC systems somewhat later by Dorin et al. [33]. The OPA algorithm is depicted in Alg. 1. The algorithm assigns priority levels to tasks in the system starting with the lowest priority, i.e., $j \leftarrow n$. At each step algorithm assigns priority

level j to task τ if τ is schedulable as the lowest priority task in a task set $\mathscr{T}'$. Task set $\mathscr{T}'$ consists of tasks to which priority level is not yet assigned, i.e., tasks to which a higher priority level will be assigned in the subsequent steps. Steps of the algorithm are further illustrated with the following example.

Example 4. *Consider the task set from Example 3. The priority ordering τ_2, τ_1, τ_3 can be obtained using the OPA algorithm as follows. In the first step, i.e., $j = 3$, the following values are obtained:*

$$\begin{aligned} i=1: &\quad R_1 = 4 + \left\lceil \frac{R_1}{9} \right\rceil \cdot 4 + \left\lceil \frac{R_1}{14} \right\rceil \cdot 2 = 10 \rightarrow \mathbf{R_1} \not\leq \mathbf{D_1} \\ i=2: &\quad R_2 = 2 + \left\lceil \frac{R_2}{8} \right\rceil \cdot 4 + \left\lceil \frac{R_2}{9} \right\rceil \cdot 4 = 18 \rightarrow \mathbf{R_2} \not\leq \mathbf{D_2} \\ i=3: &\quad R_3 = 2 + \left\lceil \frac{R_3}{8} \right\rceil \cdot 2 + \left\lceil \frac{R_3}{14} \right\rceil \cdot 1 = 5 \rightarrow \mathbf{R_3} \leq \mathbf{D_3} \rightarrow \textit{add } \tau_3 \textit{ to } \Psi \end{aligned} \tag{3.11}$$

and τ_3 is assigned the lowest priority level. In the second step, i.e., $j = 2$, the following value is obtained:

$$i=1: \quad R_1 = 4 + \left\lceil \frac{R_1}{14} \right\rceil \cdot 2 = 6 \rightarrow \mathbf{R_1} \leq \mathbf{D_1} \rightarrow \textit{add } \tau_1 \textit{ to } \Psi \tag{3.12}$$

and τ_1 is assigned the "medium" priority level. Lastly, in the third step, i.e., $j = 1$:

$$i=0: \quad R_2 = 2 \rightarrow \mathbf{R_2} \leq \mathbf{D_2} \rightarrow \textit{add } \tau_2 \textit{ to } \Psi \tag{3.13}$$

and τ_2 is assigned the highest priority. The obtained priority ordering produces a feasible schedule since each task is schedulable on its respective priority level.

3.5.3 Static mixed-criticality systems with runtime monitoring

As mentioned before, in **SMC-NO** systems, additional testing efforts are needed for low-criticality tasks with priority higher than high-criticality tasks. The solution to this issue of additional certification is in employing runtime monitoring, which enables the system to discard low-criticality tasks in case of an overrun of its allocated execution time. This way, additional certification efforts are not needed. Static mixed-criticality scheduling approach that employs runtime monitoring is referred to as the **SMC** scheduling approach, and its schedulability is defined as follows.

Definition 12. ***Schedulability of SMC.*** ***SMC*** *system is schedulable iff two conditions are satisfied:*

Algorithm 1 Audsley's algorithm

Input: $\mathscr{T} = \{\tau_1, \tau_2, ..., \tau_n\}$
Output: Ψ - priority ordered task set

```
1:  function PRIORITYASSIGNMENT(Δ)
2:      Ψ ← ∅
3:      n ← |T|
4:      unassigned← true
5:      T' ← T
6:      j ← n
7:      while j ≥ 1 do
8:          unassigned← true
9:          for each task τ in T do
10:             if isFeasible(τ, T'/τ) ∧ unassigned then
11:                 T' ← T'/τ
12:                 Ψ ← Ψ ∪ {τ}
13:                 unassigned ← false
14:             end if
15:         end for
16:         if unassigned then
17:             return ∅                          ▷ feasible schedule does not exist
18:         end if
19:         j ← j − 1
20:     end while
21:     return Ψ                                  ▷ contains priority ordered set
22: end function
```

1. *Tasks with high-criticality are schedulable with regard to their respective high-criticality WCET and high-criticality WCET of high-criticality tasks with higher priority and low-criticality WCET of low-criticality tasks with higher-priority.*
2. *Tasks with low-criticality are schedulable with regard to their low-criticality WCET and low-criticality WCET of tasks with higher priority.*

Runtime behavior in **SMC** systems with runtime monitoring is defined similarly as in case of **SMC-NO** systems.

Definition 13. ***Runtime behavior of SMC.*** *Tasks are scheduled according to priority assignment* Π*. Job of a low-criticality task is discarded if it executes for longer than it is allocated for its low-criticality execution.*

Response-time analysis in **SMC** systems is similar as in case of **SMC-NO** systems and it is given with equation (3.14).

$$R_i = C_i(L_i) + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(\min(L_i, L_j)) \tag{3.14}$$

Example 5 illustrates the benefit that runtime monitoring introduces in **SMC** systems.

Example 5. *Consider a fixed-priority uniprocessor computing platform that executes the task set* $\mathscr{T} = \{\tau_1, \tau_2, \tau_3\}$. *Parameters of tasks are:*

$$\begin{aligned} \tau_i &= \{[C_i(LO), C_i(HI)], T_i, D_i, L_i\} \\ \tau_1 &= \{[2,4], 13, 13, HI\} \\ \tau_2 &= \{[1,2], 4, 4, LO\} \\ \tau_3 &= \{[2,4], 14, 14, HI\} \end{aligned} \tag{3.15}$$

To goal is determine a feasible priority ordering for both ***SMC-NO*** *and* ***SMC*** *system. Again, the OPA algorithm can be used for priority assignment in both configurations. The first step of the algorithm in case of* ***SMC-NO****, i.e.,* $j = 3$*, is:*

$$\begin{aligned} i = 0 : R_1 &= 4 + \left\lceil \frac{R_1}{14} \right\rceil \cdot 4 + \left\lceil \frac{R_1}{4} \right\rceil \cdot 2 = 14 \rightarrow \mathbf{R_1} \nleq \mathbf{D_1} \\ i = 1 : R_2 &= 1 + \left\lceil \frac{R_2}{13} \right\rceil \cdot 2 + \left\lceil \frac{R_2}{14} \right\rceil \cdot 2 = 5 \rightarrow \mathbf{R_2} \nleq \mathbf{D_2} \\ i = 2 : R_3 &= 4 + \left\lceil \frac{R_3}{13} \right\rceil \cdot 4 + \left\lceil \frac{R_3}{4} \right\rceil \cdot 2 = 20 \rightarrow \mathbf{R_3} \nleq \mathbf{D_3} \end{aligned} \tag{3.16}$$

As it can be seen, in the first step, the OPA algorithm fails to assign the lowest priority level to any task in the system, and therefore, ***SMC-NO*** *system is not schedulable.*

On the other hand, in the first step of the algorithm for the ***SMC*** *system, the lowest priority is assigned to* τ_1*:*

$$i = 0 : R_1 = 4 + \left\lceil \frac{R_1}{14} \right\rceil \cdot 4 + \left\lceil \frac{R_1}{4} \right\rceil \cdot 1 = 11 \rightarrow \mathbf{R_1} \leq \mathbf{D_1} \rightarrow \textit{add } \tau_1 \textit{ to } \Psi \tag{3.17}$$

The latter equation illustrates the main advantage of runtime monitoring. The interference of low-criticality task τ_3 *is reduced due to the fact that* τ_3 *can be aborted if it does not signal completion after one time unit of execution. In contrast, allocation of high-criticality execution time for low-criticality task in* ***SMC-NO*** *system directly causes infeasibility of task* τ_1*. For the second priority level, i.e.,* $j = 2$*, the following value is obtained:*

$$i = 0 : R_3 = 4 + \left\lceil \frac{R_3}{4} \right\rceil \cdot 1 = 6 \rightarrow \mathbf{R_3} \leq \mathbf{D_3} \rightarrow \textit{add } \tau_3 \textit{ to } \Psi \tag{3.18}$$

and for the third priority level, i.e., $j = 3$*:*

$$i = 0 : R_2 = 1 \rightarrow \mathbf{R_2} \leq \mathbf{D_2} \rightarrow \textit{add } \tau_2 \textit{ to } \Psi \tag{3.19}$$

Therefore, ***SMC*** *system is schedulable with the priority ordering* τ_2, τ_3, τ_1.

3.5.4 Adaptive mixed-criticality systems

In **SMC** systems, low-criticality tasks are discarded, i.e., descheduled upon an overrun. It is important to note that an overrun in real-time systems is often the indicator that system itself executes beyond its design parameters. Therefore, it is reasonable to reduce computing load of the system to the essential, i.e., safety-critical, functionalities only. This is in accordance with safety standard IEC 61508 (see section 7.2.2. in the part III. of [19]), where system is required to enter the *safe state* upon activation of a fault. In this context, the error produced by an activated fault causes an overrun in the system. Moreover, reducing the computing load will consequently increase the schedulability of safety-critical tasks. An adaptive fixed-priority scheduling policy which takes the latter assumptions into consideration is defined with the following runtime behavior.

Definition 14. ***Runtime behavior of AMC systems.*** *System has two different operating modes. Namely, low-criticality execution mode, and high-criticality execution mode, denoted as LO and HI respectively. The current operating mode is denoted with* Γ*. The system starts its operation in LO mode (*$\Gamma = LO$*), and executes both low-criticality and high-criticality tasks. If a job of low-criticality task in the system executes for its low-criticality execution time, i.e.,* $C_i(LO)$*, without signaling completion, it is prevented from further execution. If a job of high-criticality task in the system executes for its low-criticality execution time, i.e.,* $C_i(LO)$*, the system switches to HI mode (*$\Gamma \leftarrow HI$*). In HI mode (*$\Gamma = HI$*) low-criticality jobs are not executed.*

The latter definition applies to runtime behavior of **AMC** systems and it is found in the most of the state-of-the-art research [28]. Note that a criticality switch is specified only in one direction, i.e., from LO mode to HI mode, since this is of interest in this research. Moreover, there are papers that deal with the return to the normal, i.e., LO state [34]. The adaptive runtime behavior was already illustrated in the Example 1, in which a criticality switch occurs when job J_2 executes for its low-criticality execution time without signaling completion. The following definition determines the schedulability of **AMC** systems.

Definition 15. ***Schedulability of AMC systems.*** ***AMC*** *system is schedulable iff two conditions are satisfied:*

1. *All tasks in the system are schedulable in LO mode with regard to their low-criticality WCETs.*
2. *High-criticality tasks are schedulable in HI mode with regard to their high-criticality WCETs.*

Determining schedulability of AMC systems is known to be a hard problem. It is known that determining schedulability of a set of independent jobs is NP-hard in the strong sense even if the dual-criticality synchronous jobs are considered [31]. Determining schedulability of period

and sporadic implicit-deadline task sets has been proven to be NP-hard in strong sense in [35]. For completeness, the findings from [35] are stated with the following theorem.

Theorem 2. *Determining schedulability of periodic and sporadic implicit deadline dual-criticality task systems is NP-hard in the strong sense.*

In the rest of this section, schedulability tests for fixed-priority **AMC** systems are discussed. Firstly, sufficient schedulability tests based on response-time analysis are explained. Secondly, more complex exact schedulability tests that are based on state space enumeration are analyzed.

Schedulability tests for fixed-priority AMC systems based on response-time analysis

There are two main response-time analyses for **AMC** systems that are found in the literature [28]. These methods were devised in [30]. Schedulability tests based on these analyses can be summarized as a list of three independent conditions:

1. Tasks are schedulable in LO mode.
2. High-criticality tasks are schedulable in HI mode.
3. High-criticality tasks are schedulable during any criticality switch from LO to HI mode.

The first contention can be verified exactly by comparing the deadline and corresponding response time that is given with the following equation:

$$R_i(LO) = C_i(LO) + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO) \tag{3.20}$$

Similarly, response time can be calculated for HI mode, i.e., the second contention:

$$R_i(HI) = C_i(HI) + \sum_{j \in hpH(\tau_i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) \tag{3.21}$$

where $hpH(\tau_i)$ is a set that contains high-criticality tasks with priority higher than τ_i. In case of a criticality switch, response time of a high-criticality task can be expressed as follows:

$$R_i(MC) = C_i(HI) + \sum_{j \in hpH(\tau_i)} \left\lceil \frac{R_i(MC)}{T_j} \right\rceil C_j(HI) + \sum_{k \in hpL(\tau_i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO) \tag{3.22}$$

where $hpL(\tau_i)$ is a set that contains low-criticality tasks with priority higher than τ_i. Unlike the equations (3.20) and (3.21) which yield the exact worst-case response time of task τ_i in corresponding operating modes, the last equation provides a bound on response time during a criticality switch. The first sum in the equation corresponds to the interference of high-criticality tasks which can be released at any time instant between the release of τ_i and its completion $R_i(MC)$. It is obvious that this is an upper bound since $C_i(HI)$ is used in calculation despite the fact that before the instant of a criticality switch there are jobs that finished execution spending

at most $C_i(LO)$ time units. The second sum in the equation corresponds to the interference of low-criticality tasks that are released between the release of τ_i and its low-criticality response-time $R_i(LO)$. Note that expression $\left\lceil \frac{R_i(LO)}{T_k} \right\rceil$ corresponds to the maximal number of releases of low-criticality high-priority tasks, which corresponds to a case when τ_i is a task which overruns its execution budget and causes a criticality switch. Equations (3.20), (3.21), and (3.22) constitute the sufficient schedulability test that is referred to as **AMC-rtb**.

Example 6. *Consider a fixed-priority uniprocessor computing platform that executes the task set* $\mathscr{T} = \{\tau_1, \tau_2, \tau_3\}$. *Parameters of tasks are:*

$$\begin{aligned} \tau_i &= \{[C_i(LO), C_i(HI)], T_i, D_i, L_i\} \\ \tau_1 &= \{[3,6], 12, 12, HI\} \\ \tau_2 &= \{[1,2], 8, 8, HI\} \\ \tau_3 &= \{[1,2], 4, 4, LO\} \end{aligned} \tag{3.23}$$

The goal is to determine the schedulability and priority ordering for the task set in ***SMC*** *system, and in* ***AMC*** *system. Again, the OPA algorithm can be used to determine both schedulability and priority ordering. In the first step (*$j = 3$*), the OPA algorithm fails to assign the lowest priority to any task in* ***SMC*** *system:*

$$\begin{aligned} i = 0 : R_1 &= 6 + \left\lceil \frac{R_1}{4} \right\rceil \cdot 1 + \left\lceil \frac{R_1}{8} \right\rceil \cdot 2 = 13 \rightarrow \mathbf{R_1} \not\leq \mathbf{D_1} \\ i = 1 : R_2 &= 2 + \left\lceil \frac{R_2}{12} \right\rceil \cdot 6 + \left\lceil \frac{R_2}{4} \right\rceil \cdot 1 = 9 \rightarrow \mathbf{R_2} \not\leq \mathbf{D_2} \\ i = 2 : R_3 &= 1 + \left\lceil \frac{R_3}{12} \right\rceil \cdot 3 + \left\lceil \frac{R_3}{8} \right\rceil \cdot 1 = 5 \rightarrow \mathbf{R_3} \not\leq \mathbf{D_3} \end{aligned} \tag{3.24}$$

Therefore, there is no feasible priority ordering for ***SMC*** *system. By applying the OPA algorithm and using* ***AMC-rtb*** *response-time analysis as the feasibility test, a feasible priority ordering is obtained. Steps of the algorithm are as follows. In the first step (*$j = 3$*), the lowest priority is assigned to* τ_1*:*

$$\begin{aligned} R_1^{LO} &= 3 + \left\lceil \frac{R_1}{8} \right\rceil \cdot 1 + \left\lceil \frac{R_1}{12} \right\rceil \cdot 3 = 7 \\ R_1^{HI} &= 6 + \left\lceil \frac{R_1}{8} \right\rceil \cdot 2 = 8 \\ R_1^{MC} &= 6 + \left\lceil \frac{R_1^{LO}}{4} \right\rceil \cdot 1 + \left\lceil \frac{R_1^{MC}}{8} \right\rceil \cdot 2 = 12 \\ \mathbf{R_1^{LO,HI,MC}} &\leq \mathbf{D_1} \rightarrow \textit{add } \tau_1 \textit{ to } \Psi \end{aligned} \tag{3.25}$$

In the second step, priority is assigned to τ_3*:*

$$R_3^{LO} = 1 + \left\lceil \frac{R_3}{8} \right\rceil \cdot 1 = 2 \rightarrow \mathbf{R_1^{LO,HI,MC}} \leq \mathbf{D_1} \rightarrow \textit{add } \tau_3 \textit{ to } \Psi \tag{3.26}$$

In the third step, priority is assigned to τ_2*:*

$$\begin{aligned} R_2^{LO} &= 1 \\ R_2^{HI} &= 2 \\ R_2^{MC} &= 2 \\ \mathbf{R_1^{LO,HI,MC}} &\leq \mathbf{D_1} \rightarrow \textit{add } \tau_1 \textit{ to } \Psi \end{aligned} \tag{3.27}$$

Tasks cannot be scheduled in ***SMC*** *system due to the large amount of interference introduced by jobs of low-criticality task* τ_3 *which will continue to execute even though jobs of high-criticality tasks executed beyond their allocated execution budget. This interference is reduced in* ***AMC*** *system since low-criticality jobs are discarded upon a criticality switch. This corresponds to reducing the amount of interference of low-criticality tasks to* $\left\lceil \frac{R_1^{LO}}{4} \right\rceil \cdot 1$ *in* ***AMC-rtb*** *analysis.*

Interferences of both low and high-criticality tasks in 3.22 can be reduced with a more precise analysis. First off, interferences in equation 3.22 can be separated:

$$R_i^{MC} = C_i(HI) + I_L(i,s) + I_H(i,s,R_i^{MC}) \tag{3.28}$$

where s is an instant of a criticality switch. A bound on interference from low-criticality tasks can be expressed as:

$$I_L(i,s) = \sum_{j \in hpL(\tau_i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) \cdot C_j(LO) \tag{3.29}$$

The latter equation takes into account the interference from low-criticality tasks released before or exactly at time instant of a criticality switch. The interference from high-criticality tasks can be bounded with the following expression:

$$I_H(i,s,t) = \sum_{k \in hpH(\tau_i)} \left\{ M(k,s,t) \cdot C_k(HI) + \left(\left\lceil \frac{t}{T_k} \right\rceil - M(k,s,t) \right) \cdot C_k(LO) \right\} \tag{3.30}$$

where $M(k,s,t)$ is the number of releases of high-criticality task τ_k between the time instant t and the instant of criticality switch s. It can be expressed as follows:

$$M(k,s,t) = \min \left(\left\lceil \frac{t - s - (T_k - D_k)}{T_k} \right\rceil + 1, \left\lceil \frac{t}{T_k} \right\rceil \right) \tag{3.31}$$

In comparison with the high-criticality interference given with **AMC-rtb** analysis, this analysis separates the interference of high-criticality tasks before and after a criticality switch since

the high-criticality WCET is used only for task releases that occur after a criticality switch. Equations (3.28), (3.29), and (3.30) constitute the sufficient schedulability test which is known as **AMC-max**. In order to find maximum R_i, $I_L(i,s)$ and $I_H(i,s,t)$ have to be evaluated for different values of s and the maximal value obtained is the worst-case response time. Variable s is chosen as a multiple of period, i.e., interarrival time, of low-criticality tasks since expression $\left\lceil \frac{s}{T_j} \right\rceil$ changes its value when s is exactly equal to a multiple of any such period.

Example 7. *Consider a fixed-priority uniprocessor computing platform that executes the task set* $\mathscr{T} = \{\tau_1, \tau_2, \tau_3\}$. *Parameters of tasks are:*

$$\begin{aligned} \tau_i &= \{[C_i(LO), C_i(HI)], T_i, D_i, L_i\} \\ \tau_1 &= \{[3,6], 18, 18, HI\} \\ \tau_2 &= \{[1,2], 4, 4, HI\} \\ \tau_3 &= \{[1,2], 3, 3, LO\} \end{aligned} \tag{3.32}$$

The goal is to determine schedulability and priority ordering for the task set in ***AMC*** *system using* ***AMC-rtb*** *and* ***AMC-max*** *analysis with the OPA algorithm. When using the* ***AMC-rtb*** *analysis, the algorithm fails to assign the lowest priority to any task in the system:*

$$\begin{aligned} &i = 0: \\ &\quad R_1^{LO} = 3 + \left\lceil \frac{R_1}{3} \right\rceil \cdot 1 + \left\lceil \frac{R_1}{4} \right\rceil \cdot 1 = 8 \\ &\quad R_1^{HI} = 6 + \left\lceil \frac{R_1}{4} \right\rceil \cdot 2 = 12 \\ &\quad R_1^{MC} = 6 + \left\lceil \frac{R_1^{LO}}{3} \right\rceil \cdot 1 + \left\lceil \frac{R_1^{MC}}{4} \right\rceil \cdot 2 = 19 \rightarrow \mathbf{R_1^{MC} \not\leq D_1} \\ &i = 1: \\ &\quad R_2^{LO} = 1 + \left\lceil \frac{R_2}{18} \right\rceil \cdot 3 + \left\lceil \frac{R_2}{3} \right\rceil \cdot 1 = 5 \rightarrow \mathbf{R_2^{LO} \not\leq D_2} \\ &i = 2: \\ &\quad R_3^{LO} = 1 + \left\lceil \frac{R_3}{18} \right\rceil \cdot 3 + \left\lceil \frac{R_3}{4} \right\rceil \cdot 1 = 5 \rightarrow \mathbf{R_3^{LO} \not\leq D_2} \end{aligned} \tag{3.33}$$

*In contrast, the lowest priority can be assigned using **AMC-max** analysis:*

$$\begin{aligned} R_1^{LO} &= 3 + \left\lceil \frac{R_1}{3} \right\rceil \cdot 1 + \left\lceil \frac{R_1}{4} \right\rceil \cdot 1 = 8 \\ R_1^{HI} &= 6 + \left\lceil \frac{R_1}{4} \right\rceil \cdot 2 = 12 \\ & M(R_1^{MC}, s_{max}, k) = 4, s_{max} = 0 \\ R_1^{MC} &= 6 + \left(\left\lfloor \frac{6}{3} \right\rfloor + 1 \right) \cdot 1 + 4 \cdot 2 + \left(\left\lceil \frac{R_1^{MC}}{4} \right\rceil - 4 \right) \cdot 1 = 18 \\ & \mathbf{R_1^{LO,HI,MC}} \leq \mathbf{D_1} \rightarrow \textit{add } \tau_1 \textit{ to } \Psi \end{aligned} \tag{3.34}$$

Similarly, the medium priority is assigned to τ_3:

$$R_3^{LO} = 1 + \left\lceil \frac{R_3}{4} \right\rceil \cdot 1 = 2 \rightarrow \mathbf{R_3^{LO}} \leq \mathbf{D_3} \rightarrow \textit{add } \tau_3 \textit{ to } \Psi \tag{3.35}$$

Finally, the highest priority is assigned to τ_2:

$$\begin{aligned} R_2^{LO} &= 1 \\ R_2^{HI} &= 2 \\ R_2^{MC} &= 2 \\ & \mathbf{R_2^{LO,HI,MC}} \leq \mathbf{D_2} \rightarrow \textit{add } \tau_2 \textit{ to } \Psi \end{aligned} \tag{3.36}$$

The part of the contribution of this thesis is providing improvements over the existing **AMC-max** analysis.

Schedulability tests for fixed-priority AMC systems based on state space enumeration

As mentioned before, determining schedulability of sporadic and periodic **AMC** systems is **NP-hard** in the strong sense. Therefore, the problem does not admit a polynomial-time algorithm unless $P = NP$. Note that the sufficient schedulability tests presented in the latter section were of pseudo-polynomial time complexity since the complexity of response-time calculation is exponential in the number of bits that are required for representation of the input. However, this is not an exact test. Since the problem is **NP-hard**, to exactly determine the schedulability of a task set, one has to perform an exhaustive search of the state space. Such a method was proposed in determining schedulability of multiprocessor real-time systems in [36]. A similar approach was used in devising an exact schedulability test for fixed-priority adaptive mixed-criticality systems in paper [37] by Asyaban and Kargahi. In the remainder of this section, their exact test is discussed in more detail and compared to the existing sufficient analyses. Later in this thesis, it will be shown that there are some incorrect statements in their work which can

reduce reproducibility of their results.

Firstly, the notation which is used for the state space exploration is introduced. In [37], state is defined as a tuple:

$$s_t := \langle \Gamma, (c_i, q_i, p_i, \varepsilon_i, \phi_i)_{i=1}^{k} \rangle \tag{3.37}$$

where:

- $\Gamma \in \{LO, HI\}$ is the criticality mode of the system in the state,
- $c_i \in \{0, 1, ..., C_i(HI)\}$ denotes the remaining execution time of the unique pending job of task τ_i,
- $q_i \in \{0, 1, ..., D_i\}$ is the remaining time to the deadline of the latest released job of task τ_i,
- $p_i \in \{0, 1, ..., T_i\}$ is the minimum remaining time until the next release of task τ_i; if L_i= LO $\wedge\, \Gamma =$ HI, then $p_i = 0$,
- $\varepsilon_i \in \{0, 1, ..., C_i(HI)\}$ is the actual execution-time of the unique pending job of task τ_i,
- $\phi_i \in \{0, ..., T_i\}$ is an offset which indicates how much later than the minimum inter-release time the most recent job of task τ_i is released.

Moreover, $\sigma(t) = \{\sigma_1(t), ..., \sigma_N(t)\}$ denotes a job sequence of a task set at time instant t, e.g., if $\sigma_1(0) = 4$ task τ_1 is released at time instant 0 with the execution time 4. Two additional variables are used for state transitions:

- $\alpha_i \in \{1, ..., n\}$, $\alpha_i = 1$ if τ_i is the highest priority task with a pending job a time t, and $\alpha_i = 0$, otherwise.
- $\beta_i \in \{1, ..., n\}$, $\beta_i = 1$ if τ_i releases a new job at time $t+1$, i.e., $\sigma_i(t+1) > 0$, and $\beta_i = 0$, otherwise.

There are two rules that are used to determine new states and state transitions. The first rule is Rule 1*, which is the state transition rule and it is defined as follows:

Definition 16. ***Rule 1**.*** *Given state $s_t := \langle \Gamma, (c_i, q_i, p_i, \varepsilon_i, \phi_i)_{i=1}^{k} \rangle$, job sequence σ, and priority ordering Π, the next state $s_{t+1} := \langle \Gamma, (c'_i, q'_i, p'_i, \varepsilon'_i, \phi'_i)_{i=1}^{k} \rangle$ is obtained as:*

- $c'_i = c_i - \alpha_i + \sigma_i(t+1)$.
- $q'_i = \max(q_i - 1, 0) + \beta_i D_i$.
- *If ($\beta_i = 0 \wedge c_i - \alpha_i = 0$), then $\varepsilon'_i = 0$; else if ($\beta_i = 0 \wedge c_i - \alpha_i > 0$), then $\varepsilon'_i = \varepsilon_i$, otherwise $\varepsilon'_i = \sigma_i(t+1)$.*
- *if $\Gamma = LO$ and there exists a HI-criticality task τ_i such that $\varepsilon_i > C_i(LO)$ and $\varepsilon_i - (c_i - \alpha_i) = C_i(LO)$, i.e., the pending job of task τ_i spent its execution budget without signalling completion, then $\Gamma' = HI$, otherwise $\Gamma' = \Gamma$.*
- *If $\sigma_i(t+1) = 0$, then $\phi'_i = \phi_i$; else if $s_t = s_{-1}$, then $\phi'_i = 0$; otherwise $\phi'_i = |p_i - 1|$.*
- *If $\sigma_i(t+1) \geq 1$ *, then $p'_i = T_i$; else if $s_t = s_{-1}$, then $p'_i = 0$; otherwise $p'_i = p_i - 1$.*

*Original statement from [37] specified condition $\sigma_i(t+1) = 1$ that was shown to be incorrect in [25], and correction is further elaborated in this work.

The second rule is referred to as Rule 2*, which is the job sequence generation rule defined as follows.

Definition 17. ***Rule 2**.*** *Given state* $s_t := \langle \Gamma, (c_i, q_i, p_i, \varepsilon_i, \phi_i)_{i=1}^{k} \rangle$*, job sequence* $\sigma(t+1)$ *is obtained as:*

- *If* $p_i - 1 > 0$ *or* $(\Gamma = HI \wedge L_i = LO)$*, then* $\sigma_i(t+1) = 0$*, where the first condition comes from the constraint on the interarrival time, and the second condition comes from the definition of* ***AMC*** *behavior, i.e., suspension of low-criticality tasks in HI mode.*
- *If* $(p_i - 1 \leq 0 \wedge C_i(LO) = C_i(HI) \wedge \Gamma = LO)$*, then* $\sigma_i(t+1)$ *gets value from set* $\{0, C_i(LO)\}$.
- *If* $(p_i - 1 \leq 0 \wedge C_i(LO) < C_i(HI) \wedge \Gamma = LO)$*, then* $\sigma_i(t+1)$ *gets value from set* $\{0, C_i(LO), C_i(HI)\}$.
- *If* $(p_i - 1 \leq 0 \wedge \Gamma = HI)$*, then* $\sigma_i(t+1) = C_i(HI)$.
- *If* $(s_t = s_{-1})$*, i.e.,* s_t *is pre-initial state, then* $\sigma_k(0) = C_i(HI)$.

Based on job sequences generated according to Rule 2* the current state s_t transitions to successor states s_{t+1} according to Rule 1*. By exploring all possible system states, the approach discovers the exact worst-case response time for each task in the system. The algorithm is depicted in Alg. 2. In lines 2 and 4, two special cases are handled using response-time

Algorithm 2 Efficient Exact Schedulability Test

Input: $\mathscr{T} = \{\tau_1, \tau_2, ..., \tau_n\}$, where tasks are priority ordered from the lowest to the highest priority

Output: The worst-case response time R_k for all tasks $\tau_k \in \mathscr{T}$, if $\mathscr{T}$ is schedulable; *null*, otherwise;

```
1:  for τ_k ∈ 𝒯 do
2:      if (L_k == LO)||(∀τ_j ∈ {τ_1,...,τ_k}, C_j(LO) == C_j(HI)) then
3:          R_k = C_k(LO) + Σ_{j<k} ⌈R_k/T_j⌉ C_j(LO)
4:      else if (∀τ_j ∈ {τ_1,...,τ_k}, L_j == HI) then
5:          R_k = C_k(L_k) + Σ_{j<k} ⌈R_k/T_j⌉ C_j(L_k)
6:      else
7:          R_k = SB-RTA(𝒯, k)
8:      end if
9:      if R_k > D_k then
10:         return null
11:     end if
12: end for
13: return {R_1,...,R_n}
```

analyses based on recurrence relations. The state-based response-time analysis is invoked in line 7. More detailed explanation of this algorithm can be found in [37].

This state-based response-time analysis yields the exact worst-case response time, which is always lower or equal to the worst-case response time given with any sufficient analysis. Consequently, this increases the schedulability. The improvements of the exact schedulability test over the sufficient **AMC-max** test are shown in the following example.

Example 8. *Consider a fixed-priority uniprocessor computing platform that executes the task set* $\mathscr{T} = \{\tau_1, \tau_2, \tau_3\}$. *Parameters of tasks are:*

$$\begin{aligned} \tau_i &= \{[C_i(LO), C_i(HI)], T_i, D_i, L_i\} \\ \tau_1 &= \{[1,2], 5, 5, HI\} \\ \tau_2 &= \{[1,2], 2, 2, LO\} \\ \tau_3 &= \{[1,2], 7, 7, HI\} \end{aligned} \tag{3.38}$$

The goal is to determine if a feasible schedule for the task set exists with regard to ***AMC-max*** *and the exact analysis. To find a feasible schedule with regard to* ***AMC-max*** *analysis, the OPA algorithm can be employed. Again, the algorithm fails at the first step since it cannot assign the lowest priority to any task in the system:*

$$\begin{aligned} &i = 0: \\ &\quad R_1^{LO} = 1 + \left\lceil \frac{R_1}{7} \right\rceil \cdot 1 + \left\lceil \frac{R_1}{2} \right\rceil \cdot 1 = 4 \\ &\quad R_1^{HI} = 2 + \left\lceil \frac{R_1}{7} \right\rceil \cdot 2 = 4 \\ &\quad R_1^{MC} = 2 + 1 \cdot 2 + \left(\left\lceil \frac{R_1^{MC}}{7} \right\rceil - 1 \right) \cdot 1 + \left(\left\lfloor \frac{2}{2} \right\rfloor + 1 \right) \cdot 1 = 6 \rightarrow \mathbf{R_1^{MC} \not\leq D_1} \\ &i = 1: \\ &\quad R_2^{LO} = 1 + \left\lceil \frac{R_2}{5} \right\rceil \cdot 1 + \left\lceil \frac{R_2}{7} \right\rceil \cdot 1 = 3 \rightarrow \mathbf{R_2^{LO} \not\leq D_2} \\ &i = 2: \\ &\quad R_3^{LO} = 1 + \left\lceil \frac{R_3}{5} \right\rceil \cdot 1 + \left\lceil \frac{R_3}{2} \right\rceil \cdot 1 = 4 \\ &\quad R_3^{HI} = 2 + \left\lceil \frac{R_3}{5} \right\rceil \cdot 2 = 4 \\ &\quad R_3^{MC} = 2 + 2 \cdot 2 + \left(\left\lceil \frac{R_3^{MC}}{5} \right\rceil - 2 \right) \cdot 1 + \left(\left\lfloor \frac{2}{2} \right\rfloor + 1 \right) \cdot 1 = 8 \rightarrow \mathbf{R_3^{MC} \not\leq D_3} \end{aligned} \tag{3.39}$$

However, this task set is schedulable with priority ordering τ_1, τ_2, τ_3. *This can be verified easily using Alg. 2. Since* τ_1 *is the highest priority task, its response time is* $R_1 = 2$ *(line 4 in Alg. 2). Similarly, response time of low-criticality task* τ_2 *can be obtained as specified in line 2 in Alg. 2:*

$$R_2 = 1 + \left\lceil \frac{R_2}{5} \right\rceil \cdot 1 = 2 \tag{3.40}$$

The worst-case response time of high-criticality task τ_3 *obtained using the state space exploration according to Rule 1* and Rule 2* is depicted in Fig. 3.7. Legend for the state space exploration is shown in Fig. 3.6. Since all three tasks were found to be schedulable,* ***AMC***

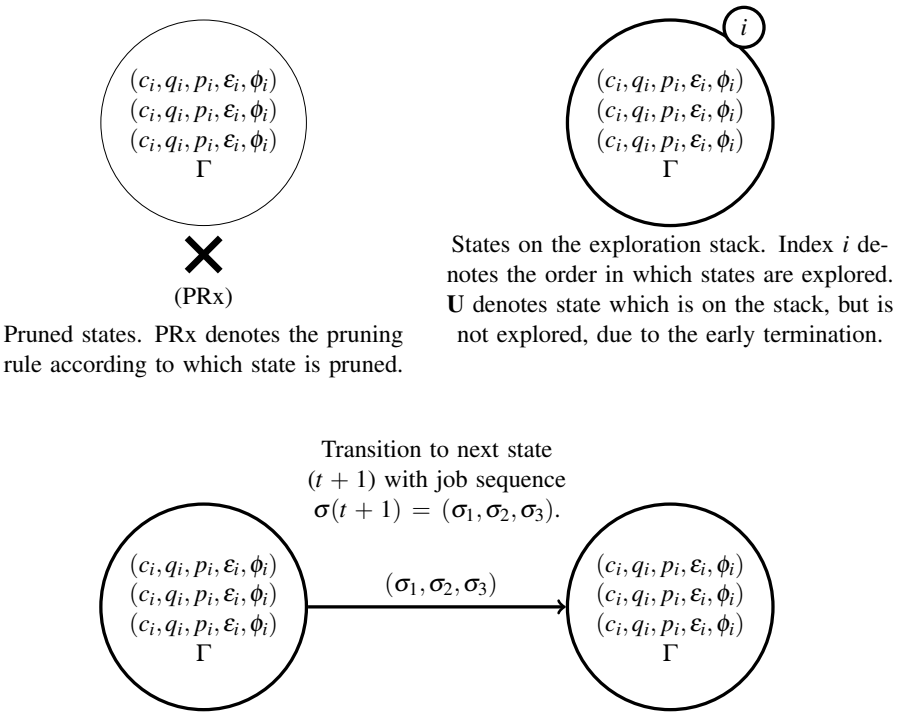


Figure 3.6: Legend for state space exploration diagrams.

system with the task set $\mathscr{T}$ is schedulable with priority ordering τ_1, τ_2, τ_3.

An issue that is not discussed in the latter example is the priority assignment in the second case, i.e., when the exact analysis is used. It can be seen that the OPA algorithm has not been used. As it can be seen in Alg. 1, which depicts the OPA algorithm, the performance of the OPA algorithm relies on usage of an adequate schedulability test. An important property of schedulability tests is the *OPA compatibility*. The OPA compatibility property ensures the optimality of the OPA algorithm for a given schedulability test and it is a necessary precondition for usage of the schedulability test with the OPA algorithm. The conditions for OPA compatibility were devised in [38] and can be expressed as follows.

Condition 1. *The schedulability of a task τ_k may, according to test S, depend on any independent properties of task with priorities higher than k, but not on any properties of those tasks that depend on their relative priority ordering.*

Condition 2. *The schedulability of a task τ_k may, according to test S, depend on any independent properties of task with priorities lower than k, but not on any properties of those tasks that depend on their relative priority ordering.*

Condition 3. *When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to test S, if it was previously schedulable at the lower priority. (As a corollary, the task being assigned the lower priority cannot become schedulable according to test S,if it was previously unschedulable at the higher priority.)*

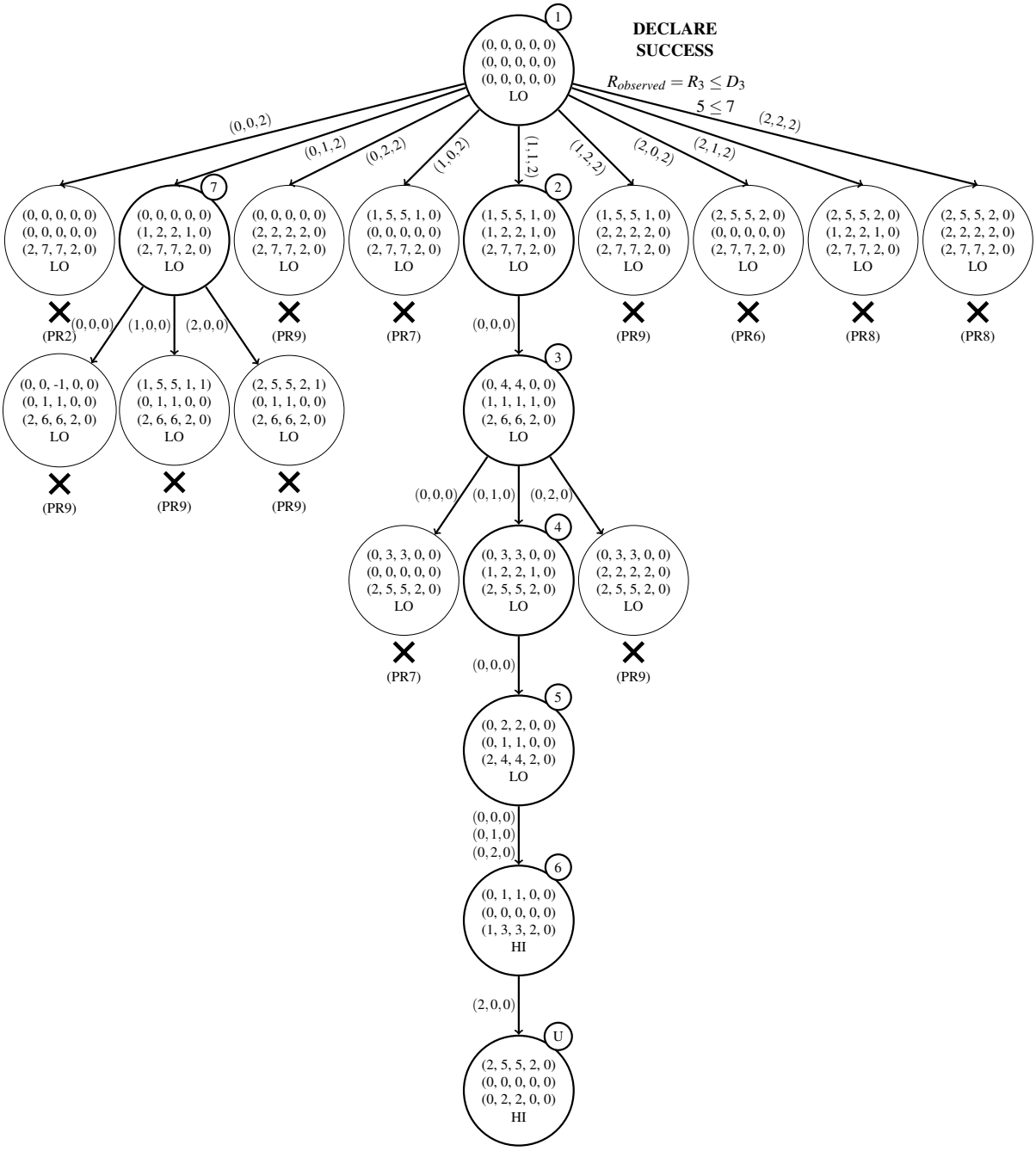


Figure 3.7: State space exploration for τ_3 in Example 8.

Algorithm 3 NOPA algorithm

Input: $\mathscr{T} = \{\tau_1, \tau_2, ..., \tau_n\}$
Output: Ψ - priority assignment function

function NONOPTIMALPRIORITYASSIGNMENT($\mathscr{T}$)
  $n \leftarrow |\mathscr{T}|$, $\mathscr{T}' \leftarrow \mathscr{T}$, $j \leftarrow n$
  while $\exists \tau_i \in \mathscr{T}' | L_i = LO \wedge \exists \tau_j \in \mathscr{T}' | L_i = HI$ **do**
    $\tau_i \leftarrow \max_{\tau_i \in \mathscr{T}' | L_i = LO}(D_i)$
    $R_i = C_i(LO) + \sum_{\tau_j \in \mathscr{T}' \setminus \tau_i} \lceil \frac{R_i}{T_j} \rceil \cdot C_j(LO)$
    if $R_i \leq D_i$ **then**
      $\Psi(\tau_i) \leftarrow j$
      $\mathscr{T}' \leftarrow \mathscr{T}' \setminus \tau_i$
    else
      $\tau_i \leftarrow \max_{\tau_i \in \mathscr{T}' | L_i = HI}(D_i)$
      $\Psi(\tau_i) \leftarrow j$
      $\mathscr{T}' \leftarrow \mathscr{T}' \setminus \tau_i$
    end if
    $j \leftarrow j - 1$
  end while
  while $\exists \tau_i \in \mathscr{T}'$ **do**
    $\tau_i \leftarrow \max_{\tau_i \in \mathscr{T}'}(D_i)$
    $\Psi(\tau_i) \leftarrow j$
    $\mathscr{T}' \leftarrow \mathscr{T}' \setminus \tau_i$
    $j \leftarrow j - 1$
  end while
  return Ψ
end function

The exact approach is not OPA compatible as it is in the strict violation of the first condition. More precisely, the schedulability depends on the relative priority ordering of tasks with higher priority, e.g., different worst-case response times can be discovered if priority of a task that causes criticality switch is changed. To overcome this issue, the authors in [37] proposed using a heuristic priority assignment algorithm, which is not optimal, but highly effective in terms or resulting schedulability. This algorithm is known as *non-optimal priority assignment* (NOPA) algorithm and it is depicted in Alg. 3.

3.6 Improvement of existing schedulability test for periodic adaptive mixed-criticality systems

3.6.1 Refinement of AMC-max schedulability test

Improvements of existing **AMC-max** schedulability test are based on more precise response-time analysis of low-criticality and high-criticality interference of tasks with higher priority

than an observed task. Response time of a high-criticality task τ_i in case of a criticality switch at time instant s caused by a high-criticality task τ_ξ[†] can be represented as:

$$R_i^{MC}(s) = C_i(HI) + I_L(i,s) + I_H(i,s,R_i^{MC}(s)) \tag{3.41}$$

where $I_L(s)$ and $I_H(s)$ are interferences caused by low and high-criticality tasks, respectively. As stated in [30]: *In this formulation we could differentiate between those tasks that have a priority greater than τ_ξ, and those that have a lower priority. Those with priority greater than τ_ξ must have completed this 'current' job (so only executed for $C(LO)$), while those with priority equal or less may not yet have completed and hence their current job must be assumed to need $C(HI)$.* However, Baruah et al. in [30] did not pursue this path due to the problem of OPA compatibility which will be discussed later. Additionally, in the statement above they do not take low-criticality tasks into consideration and it will be showed that this more precise response-time analysis for low-criticality tasks is crucial for improvement of schedulability test. In the rest of this section several propositions accompanied with proofs for enhancement of **AMC-max** schedulability test are provided. Additionally, an example is presented, which shows how propositions improve response time of some task systems which are not feasible if **AMC-max** schedulability test is used. Firstly, the improvement of interference for low criticality tasks ($I_L(s)$ term) is addressed and the interference from high-criticality tasks ($I_H(s)$ term) is addressed afterwards. Improvements are made by introducing separate analysis for different subsets of sets of tasks with priority higher than task τ_i, i.e., $hpL(\tau_i)$ and $hpH(\tau_i)$.

Reducing the interference of low-criticality tasks.

The set of low-criticality tasks with priority higher than τ_i, i.e., $\mathbf{hpL}(\tau_i)$ can be separated into two disjunct sets:

- set $\mathbf{hpL}(\tau_\xi)$, which contains low-criticality tasks with a priority higher than τ_ξ,
- set $\mathbf{hpL}(\tau_i) \cap \mathbf{lpL}(\tau_\xi)$, which contains low-criticality tasks with a priority lower than τ_ξ, but higher than τ_i.

Interference of low-criticality tasks as represented in [30] is:

$$I_L(i,s) = \sum_{j \in \mathbf{hpL}(\tau_i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j(LO) \tag{3.42}$$

Equation (3.42) overestimates the execution time of low-criticality tasks with the priority lower than the priority of task which caused a criticality switch τ_ξ. This is due to the fact that unfinished jobs of tasks with a priority lower than τ_ξ will not spend their entire $C_j(LO)$ budget in LO mode. The latter statement is stated in the form of proposition that follows.

[†]Notation τ_ξ is used instead of τ_s to evade overload of variable s.

Proposition 1. *Unfinished job of low-criticality task τ_j with priority lower than the priority of high-criticality task τ_ξ (which causes a criticality switch) and priority higher than τ_i will execute for time c_{ju} in LO mode, such that $c_{ju} < C_j(LO)$, where u is the index of the unfinished job.*

Proof. Consider a scenario in which jobs of tasks τ_i, τ_j and τ_ξ are released. Since high-criticality task τ_ξ has the highest priority, its job will execute for its entire low-execution budget without signaling completion and consequently cause a criticality switch. Since the low-criticality tasks are abandoned in **AMC** systems, job of task τ_j will be discarded before it spends its execution budget, i.e., $C_i(LO)$. □

Based on Proposition 1, it can be instantly inferred that the maximum execution time for the last unfinished job of a low-criticality task $\tau_j \in \mathbf{hpL}(\tau_i) \cap \mathbf{lpL}(\tau_\xi)$ corresponds to the largest integer for which $c_{ju} < C_j(LO)$ since the discrete time model is considered. Therefore, $c_{ju}^{max} = C_j(LO) - 1$ and the execution budget of such low-criticality tasks is effectively reduced in case of a criticality switch.

Corollary 1. *For execution time of last unfinished jobs of low-criticality tasks with a priority higher than τ_i and lower than τ_ξ the following condition applies:*

$$\sum_{j \in \kappa} c_{ju} \leq \sum_{j \in \kappa} (C_j(LO) - 1) \tag{3.43}$$

where $\kappa = \{\mathbf{hpL}(\tau_i) \cap \mathbf{lpL}(\tau_\xi)\}$.

The problem with the stated proposition and its immediate corollary is to determine if job is indeed unfinished at some time instant. In essence, to determine if job is unfinished one has to simulate the system behavior, i.e., sporadic or periodic release and execution of tasks, up to the instant of a criticality switch. Naturally, such an approach would significantly increase the computation time and in a lot of cases may not be acceptable. This is extremely difficult, i.e., computationally expensive, to determine in case of sporadic systems. However, in case of periodic systems with no initial offset, these observations can be exploited using the information about time instants of release of τ_j and τ_ξ, and time instant of a criticality switch s. There are two different cases that are considered, which depend on relative phasing of jobs of tasks τ_j and τ_ξ. These cases are depicted in Figs. 3.8-3.9.

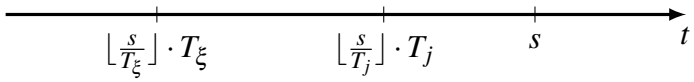


Figure 3.8: Relative phasing of τ_ξ and τ_j prior to the criticality switch when the last job of τ_j is released after the job of τ_ξ that caused the criticality switch.

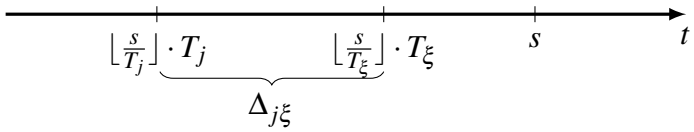


Figure 3.9: Relative phasing of τ_ξ and τ_j prior to the criticality switch when the last job of τ_j is released before the job of τ_ξ that caused the criticality switch.

Proposition 2. *Interference of low-criticality task τ_j with priority lower than τ_ξ, which is released after job of τ_ξ that caused a criticality switch is given with:*

$$I_L(i,s,j) = \left\lfloor \frac{s}{T_j} \right\rfloor \cdot C_j(LO) \tag{3.44}$$

Note that this corresponds to expression given with (3.42) when $C_j(LO)$ is subtracted.

Proof. In the case shown in Fig. 3.8, the last job of τ_ξ is released before τ_j. Since τ_ξ has a higher priority than τ_j, the last job of τ_j will never be executed if τ_ξ causes a criticality switch at time instant s. Therefore, interference of last job can be discarded, which proves the proposition. □

Proposition 3. *Interference of low-criticality task τ_j with priority lower than τ_ξ, which is released before job τ_ξ that caused a criticality switch is given with:*

$$I_L(i,s,j) = \left\lfloor \frac{s}{T_j} \right\rfloor \cdot C_j(LO) + \min(\Delta_{j\xi}, C_j(LO)) \tag{3.45}$$

where $\Delta_{j\xi}$ is given with:

$$\Delta_{j\xi} = \left\lfloor \frac{s}{T_\xi} \right\rfloor \cdot T_\xi - \left\lfloor \frac{s}{T_j} \right\rfloor \cdot T_j \tag{3.46}$$

Proof. In the case shown in Fig. 3.9, the last job of τ_ξ is released after τ_j. It is known that the earliest point in time when job of τ_j is finished is given with:

$$f_{ju} = \left\lfloor \frac{s}{T_j} \right\rfloor \cdot T_j + C_j(LO) \tag{3.47}$$

There are two different cases with regard to relative phasing f_{ju} and $\lfloor \frac{s}{T_\xi} \rfloor \cdot T_\xi$:

1. $f_{ju} \leq \lfloor \frac{s}{T_\xi} \rfloor \cdot T_\xi$,
2. $f_{ju} > \lfloor \frac{s}{T_\xi} \rfloor \cdot T_\xi$.

In the first case, the maximum execution time of the last job of task τ_j corresponds to the WCET of task since in the worst case the job will spend its entire execution time prior to release of τ_ξ. In the second case, the maximum execution time of the last job τ_j corresponds to the difference of release times $\Delta_{j\xi}$ because once τ_ξ is released, τ_j will not be executed again. Therefore, the

job of τ_j executes for maximally $C_i(LO)$ or $\Delta_{j\xi}$ time units if τ_ξ is released before its earliest finish time, which proves this proposition. □

The total interference of low-criticality tasks with the latter modifications can be expressed as:

$$\begin{aligned} I_L(i,s,\tau_\xi) = & \sum_{j\in \mathbf{hpL}(\tau_\xi)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) \cdot C_j(LO) \\ & + \sum_{j\in \mathbf{hpL}(\tau_i)\cap \mathbf{lpL}(\tau_\xi)} \left\lfloor \frac{s}{T_j} \right\rfloor \cdot C_j(LO) \\ & + \sum_{j\in \mathbf{hpL}(\tau_i)\cap \mathbf{lpL}(\tau_\xi) \mid \lfloor \frac{s}{T_j} \rfloor T_j \leq \lfloor \frac{s}{\tau_\xi} \rfloor T_\xi} \min(C_j(LO), \Delta_{j\xi}) \end{aligned} \tag{3.48}$$

Reducing the interference of high-criticality tasks.

The interference of high-criticality tasks at some time instant t can be expressed as:

$$I_H(s,t) = \sum_{k\in \mathbf{hpH}(\tau_i)} \left\{ M(k,s,t)\, C_k(HI) + \left(\left\lceil \frac{t}{T_k} \right\rceil - M(k,s,t) \right) C_k(LO) \right\} \tag{3.49}$$

where $M(k,s,t)$ is the number of job releases of task τ_k at time instant t after a criticality switch. Similarly as before, set $\mathbf{hpH}(\tau_i)$ can be split into three disjunct sets:

- set $\mathbf{hpH}(\tau_\xi)$, which contains high-criticality tasks with priority higher than τ_ξ,
- set $\mathbf{hpH}(\tau_i) \cap \mathbf{lpH}(\tau_\xi)$, which contains high-criticality tasks with priority lower than τ_ξ but higher than τ_i,
- set $\{\tau_\xi\}$, which contains τ_ξ, which causes a criticality switch.

Tasks in set $\mathbf{hpH}(\tau_i) \cap \mathbf{lpH}(\tau_\xi)$ are assumed to need high-criticality execution time after time instant s. Therefore, the $M(k,s,t)$ term is the same as in **AMC-max** analysis:

$$M(k,s,t) = \min \left(\left\lceil \frac{t - s - (T_k - D_k)}{T_k} \right\rceil + 1, \left\lceil \frac{t}{T_k} \right\rceil \right) \tag{3.50}$$

The latter expression applies to τ_ξ as well. Last jobs of tasks from set $\mathbf{hpH}(\tau_\xi)$ must have completed their execution before time instant s. Therefore, equation (3.50) overestimates the interference from tasks in set $\mathbf{hpH}(\tau_\xi)$.

Proposition 4. *Maximum number of job releases of high-criticality tasks with priority higher than task which causes a criticality switch (τ_ξ) after a criticality switch ($t > s$) is:*

$$M(k,s,t) = \left\lceil \frac{t - s - (T_k - D_k)}{T_k} \right\rceil \tag{3.51}$$

Proof. Consider interference $I_H(k,s,t)$ from high-criticality task τ_k at time instant s in interval of length t in **AMC-max** analysis:

$$I_H(k,s,t) = M_{max}(k,s,t)C_k(HI) + \left(\left\lceil \frac{t}{T_k} \right\rceil - M_{max}(k,s,t) \right) C_k(LO) \tag{3.52}$$

where $M_{max}(k,s,t)$ is the number of job releases given with equation (3.50). If task τ_k has priority higher than τ_ξ, then the last job of task τ_k executed for its low-criticality execution time. Hence, $C(HI)$ needs to be subtracted and $C(LO)$ needs to be added to **AMC-max** interference. After these modifications, interference can be expressed as:

$$I_H(k,s,t) = (M_{max}(k,s,t)-1)C_k(HI) + \left(\left\lceil \frac{t}{T_k} \right\rceil - (M_{max}(k,s,t)-1) \right) C_k(LO) \tag{3.53}$$

Therefore, $M^*(k,s,t) = M_{max}(k,s,t) - 1$, which proves the proposition ($M^*(k,s,t)$ is the number of releases given with equation (3.51)). Term $\lceil \frac{t}{T_k} \rceil$ is dropped from (3.51) because $\lceil \frac{t-s-(T_k-D_k)}{T_k} \rceil \leq \lceil \frac{t}{T_k} \rceil$. □

By combining equations (3.41), (3.48), (3.49), (3.50), and (3.51), response time in case of a criticality switch, i.e., during the mode change, can be expressed as follows:

$$\begin{aligned}
R_i = {} & C_i(HI) \\
& + \sum_{j \in \mathbf{hpL}(\tau_\xi)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) \cdot C_j(LO) \\
& + \sum_{j \in \mathbf{hpL}(\tau_i) \cap \mathbf{lpL}(\tau_\xi)} \left\lfloor \frac{s}{T_j} \right\rfloor \cdot C_j(LO) \\
& + \sum_{j \in \mathbf{hpL}(\tau_i) \cap \mathbf{lpL}(\tau_\xi) | \lfloor \frac{s}{T_j} \rfloor T_j \leq \lfloor \frac{s}{\tau_\xi} \rfloor T_\xi} \min(C_j(LO), \Delta_{j\xi}) \\
& + \sum_{k \in \mathbf{hpH}(\tau_\xi)} \left\{ M^*(k,s,R_i) C_k(HI) + \left(\left\lceil \frac{R_i}{T_k} \right\rceil - M^*(k,s,R_i) \right) C_k(LO) \right\} \\
& + \sum_{k \in \mathbf{hpH}(\tau_i) \cap \mathbf{lpH}(\tau_\xi)} \left\{ M_{max}(k,s,R_i) C_k(HI) + \left(\left\lceil \frac{R_i}{T_k} \right\rceil - M_{max}(k,s,R_i) \right) C_k(LO) \right\}
\end{aligned} \tag{3.54}$$

As well as in **AMC-max** approach, the instant of a criticality switch s is chosen from the interval $[0, R_i^{LO}\rangle$ as a multiple of minimal interarrival times, i.e., periods, of low-criticality tasks with priority higher than or equal to τ_i. This is sufficient since the term in the low-criticality interference $\left\lfloor \frac{s}{T_j} \right\rfloor$ changes when s is such that $s = k \cdot T_j, k \in \mathbb{N}$. Note that this captures the worst-case of interference if a criticality switch occurs at any time instant in interval determined with two consecutive multiples of low-criticality tasks, i.e., $s \in [k \cdot T_j, (k+1) \cdot T_j], k \in \mathbb{N}$. Task τ_ξ is chosen from tasks with priority higher than τ_i. Finally, for response time in case of a criticality

switch, the maximum time among all choices of s and τ_ξ is chosen:

$$R_i^{MC} = \max(R_i(s, \tau_\xi)) \tag{3.55}$$

An illustrative example of the schedulability test refinement

With the following example, cases when the devised test outperforms **AMC-max** is illustrated and explanations are provided.

Example 9. *Consider a fixed-priority uniprocessor computing platform that executes the task set* $\mathscr{T} = \{\tau_1, \tau_2, \tau_3\}$. *Parameters of tasks are:*

$$\begin{aligned} \tau_i &= \{[C_i(LO), C_i(HI)], T_i, D_i, L_i\} \\ \tau_1 &= \{[1,2], 10, 10, HI\} \\ \tau_2 &= \{[1,1], 5, 5, LO\} \\ \tau_3 &= \{[4,8], 13, 13, HI\} \end{aligned} \tag{3.56}$$

The goal is to determine if a feasible schedule exists with regard to ***AMC-max*** *and the devised analysis. To find a feasible priority assignment with regard to* ***AMC-max*** *analysis, the OPA algorithm can be employed. Similarly as in the latter examples, the algorithm fails at the first step since it cannot assign the lowest priority to any task in the system:*

$$\begin{aligned} &i = 0: \\ &\quad R_1^{LO} = 1 + \left\lceil \frac{R_1}{13} \right\rceil \cdot 4 + \left\lceil \frac{R_1}{5} \right\rceil \cdot 1 = 7 \\ &\quad R_1^{HI} = 2 + \left\lceil \frac{R_1}{13} \right\rceil \cdot 8 = 10 \\ &\quad R_1^{MC} = 2 + 1 \cdot 4 + \left(\left\lceil \frac{R_1^{MC}}{13} \right\rceil - 1 \right) \cdot 4 + \left(\left\lfloor \frac{0}{5} \right\rfloor + 1 \right) \cdot 1 = 12 \rightarrow \mathbf{R_1^{MC} \not\leq D_1} \\ &i = 1: \\ &\quad R_2^{LO} = 1 + \left\lceil \frac{R_2}{10} \right\rceil \cdot 1 + \left\lceil \frac{R_2}{13} \right\rceil \cdot 4 = 6 \rightarrow \mathbf{R_2^{LO} \not\leq D_2} \\ &i = 2: \\ &\quad R_3^{LO} = 4 + \left\lceil \frac{R_3}{10} \right\rceil \cdot 1 + \left\lceil \frac{R_3}{5} \right\rceil \cdot 1 = 7 \\ &\quad R_3^{HI} = 8 + \left\lceil \frac{R_3}{10} \right\rceil \cdot 2 = 10 \\ &\quad R_3^{MC} = 8 + 2 \cdot 2 + \left(\left\lceil \frac{R_3^{MC}}{10} \right\rceil - 2 \right) \cdot 1 + \left(\left\lfloor \frac{5}{5} \right\rfloor + 1 \right) \cdot 1 = 14 \rightarrow \mathbf{R_3^{MC} \not\leq D_3} \end{aligned} \tag{3.57}$$

However, this task set is schedulable with priority ordering τ_1, τ_2, τ_3. *This can be verified using*

*the devised test. The worst-case response times for tasks τ_1 and τ_2 are obtained in the same way as in the **AMC-max** analysis:*

$$\begin{aligned} R_1^{LO} &= 1, R_1^{HI} = R_1^{MC} = 2 \\ R_2^{LO} &= 1 + \left\lceil \frac{R_2^{LO}}{10} \right\rceil \cdot 1 = 2 \end{aligned} \tag{3.58}$$

*The difference between the **AMC-max** analysis and the devised approach is in response time for τ_3. The worst-case response time for τ_3 according **AMC-max** analysis is given with:*

$$R_3^{MC} = 8 + 2 \cdot 2 + \left(\left\lceil \frac{R_3^{MC}}{10} \right\rceil - 2 \right) \cdot 1 + \left(\left\lfloor \frac{5}{5} \right\rfloor + 1 \right) \cdot 1 = 14 \rightarrow \mathbf{R_3^{MC} \not\leq D_3} \tag{3.59}$$

As it can be seen, the worst-case response time for τ_3 is obtained for $s = 5$. Using the devised analysis, the worst-case response time is effectively reduced and can be obtained using recurrence relation (3.54) when τ_1 is a task which causes the criticality switch:

$$\begin{aligned} R_3 &= 8 \\ &+ \left\lfloor \frac{5}{5} \right\rfloor \cdot 1 \\ &+ 2 \cdot 2 + \left(\left\lceil \frac{R_i}{10} \right\rceil - 2 \right) \cdot 1 = 13 \end{aligned} \tag{3.60}$$

The interference of low-criticality task τ_2 is reduced as τ_2 is in the set of tasks with priority lower than τ_ξ, i.e., τ_1, and higher than the observed task τ_3. The reduction is justified since it is known that in periodic release pattern the last instance of τ_1 which caused the criticality switch at time instant 5 is released before τ_2. Similarly, the reduction of high-criticality interference lowers the worst-case response time when τ_3 is assumed to cause a criticality switch:

$$\begin{aligned} R_i &= 8 \\ &+ \left(\left\lfloor \frac{5}{5} \right\rfloor + 1 \right) \cdot 1 \\ &+ 1 \cdot 2 + \left(\left\lceil \frac{R_i}{10} \right\rceil - 1 \right) \cdot 1 = 13 \end{aligned} \tag{3.61}$$

This, in fact, demonstrates that both reductions, i.e., reductions of low-criticality and high-criticality interferences, are needed for the task set to be schedulable according to the devised analysis.

3.6.2 Schedulability test properties

Property 1. ***Dominance over AMC-max.*** *The devised analysis dominates the* ***AMC-max*** *analysis in a sense that it will yield the worst-case response time for any task in a task set, which is lower than or equal to the worst-case response time given with* ***AMC-max*** *analysis.*

Proof. It is clear that the devised analysis dominates **AMC-max** analysis because at minimum the following conditions apply to low and high-criticality interferences of analyses:

$$\begin{aligned} I_L^*(i,s) &\leq I_L^{max}(i,s) \\ I_H^*(i,s,R_i^{MC}) &\leq I_H^{max}(i,s,R_i^{MC}) \end{aligned} \tag{3.62}$$

where I_L^{max}, I_H^{max}, I_L^*, and I_H^* are given with (3.29), (3.30), (3.48), and (3.30) with M^* given with (3.51), respectively. The proof that conditions hold is contained in corresponding proofs of interference reduction propositions 2, 3, and 4. □

Property 2. ***OPA compatibility***. *The devised analysis is not OPA compatible, i.e., usage of the devised analysis with the OPA algorithm may not yield a feasible priority assignment if one exists.*

Proof. As it is stated before, in order for an analysis to be OPA compatible, conditions 1, 2, and 3 have to be satisfied. In the case of devised analysis, it is clear that Condition 1 is not satisfied since the worst-case response time depends on relative priority ordering of task with higher priority than τ_i, i.e., it depends on choice of τ_ξ. □

Property 3. ***Computational complexity***. *In the devised analysis, response time of a task is calculated in $O(n^2)$ complexity.*

Proof. For every task τ_i in a system, one task from a set of tasks with a higher priority must be chosen as τ_ξ (task which causes a criticality switch). Complexity of evaluating schedulability of a task set with given priority order is $O(n^3)$. Therefore, it involves more computation than **AMC-max**. □

Property 4. ***Applicable priority assignment.*** *Due to the OPA incompatibility it is not possible to use this test with the OPA algorithm as an entire priority assignment must be known to evaluate worst-case response time of a task. Therefore, the heuristic NOPA priority assignment devised in [37] is used. In essence, any heuristic method can be used to assign the priorities, but the optimality is not guaranteed.*

3.7 Evaluation of the devised schedulability test

So far in the thesis, the focus has been on the theoretical aspects of schedulability tests and response-time analysis. Various schedulability tests have been compared using examples and counterexamples to show the pitfalls and drawbacks of certain schedulability tests. Such an approach is very useful to highlight the fundamental differences between approaches. However, the performance of schedulability tests on a larger number of task sets, i.e., classes of task sets, is important as well. Therefore, in this section, the results of evaluation of the existing schedulability tests as well as the devised schedulability test are presented. The results were generated using the framework for schedulability testing of adaptive mixed-criticality systems, which is discussed in detail later in section 3.9.

3.7.1 Task set generation

Task sets for experiments were generated using the UUnifast algorithm [39], which is often used in measuring the performance of real-time systems. A problem with the algorithm is that for small input range of minimal interarrival times or small number of tasks, it is difficult to generate a large amount of task sets with targeted utilization factor. Therefore, the feature of the schedulability testing framework (see section 3.9) is exploited, which enables discarding of task sets that have utilization factor that does not fit into interval $[u-\delta, u+\delta\rangle$ where u is the target utilization factor in low criticality mode and δ is the utilization increment. Input parameters for the task set generation are:

- n - number of tasks in a task set,
- $[T_{min}, T_{max}]$ - interval of interarrival times,
- u_{LO} - target utilization of a task set in low criticality execution mode,
- δ - utilization increment,
- N - number of tasks per utilization factor,
- L - number of criticality levels in a system,
- DF - deadline scaling factor,
- CF - criticality factor,
- CP - fraction of high criticality tasks in a task set.

Interarrival times are generated according to the uniform distribution from interval $[T_{min}, T_{max}]$. The worst-case execution time for low criticality mode of execution can be calculated as $C_i(LO) = \lceil u_i \cdot T_i \rceil$ (u_i is the fraction of processor time used by task τ_i in LO mode, i.e., $u_{LO} = \sum_{i=1}^{n} u_i$). The high-criticality WCETs can be calculated as $C_i(HI) = CF \cdot C_i(LO)$, where CF is the criticality factor. Deadline scaling factor DF determines interval from which deadline is chosen. Deadline is chosen according to the uniform distribution or as a fixed value from the interval $[\frac{T}{DF}, T]$.

3.7.2 Results

The performance of schedulability test and their corresponding priority assignment algorithm is evaluated since a combination of schedulability test and priority assignment algorithm is used in practice for the system design. In evaluation, schedulability, i.e., the fraction of task sets that are schedulable, and average execution time are observed. Moreover, in the evaluation, utilization (U), the number of task sets (n), and the maximum period (T_{max}) were independently varied. The default parameters for all experiments are the following:

- $n = 6$,
- $T_{min} = 2$, $T_{max} = 100$,
- $L = 2$,
- $DF = 1$,
- $CF = 2$,
- $CP = 0.5$,
- $\delta = 0.025$.

Firstly, the performance of scheduling approaches with regard to low-criticality processor utilization is evaluated. Results are shown in Figs. 3.10-3.13. For evaluation, 2000 task sets per utilization factor were generated for each utilization factor in interval $[0.1, 0.9]$ with 0.025 increment, i.e., the total of $33 \cdot 2000 = 66000$ task sets. Fig 3.10 shows the overall schedulability of all static and adaptive mixed-criticality approaches as well as non-mixed criticality and partitioned criticality approaches, which were discussed in the previous sections. The figure clearly shows that the non mixed-criticality approaches, i.e., **DMPA** and **CrMPO**, perform poorly in terms of schedulable task sets in comparison with static and adaptive mixed-criticality approaches. Furthermore, the figure shows the performance of the following static and adaptive mixed-criticality scheduling approaches:

- **SMC-NO**, **SMC**, **AMC-rtb**, and **AMC-max** schedulability tests that are used with the OPA algorithm as proposed in [29, 30] and discussed in section 3.5.
- the schedulability test devised in section 3.6, which is referred to as **AMC-tight**.
- The **Exact** test is the exact schedulability test devised in [37].
- The **ExactPeriodic** test designates the exact schedulability test for periodic adaptive mixed-criticality system based on exhaustive state space exploration, i.e., simulation of periodic system. It can be seen as the special case of the approach in [37] when arrivals of task instances are strictly periodic.
- **UBH&L** test is a schedulability test which checks response times for low-criticality and high-criticality mode. A task set is schedulable if response times for high and low-criticality mode for each task in a task set are lower than their respective deadlines. In essence, this is a necessary schedulability test for fixed-priority adaptive mixed-criticality systems. Therefore, **UBH&L+DMPA** graph represents a theoretical schedula-

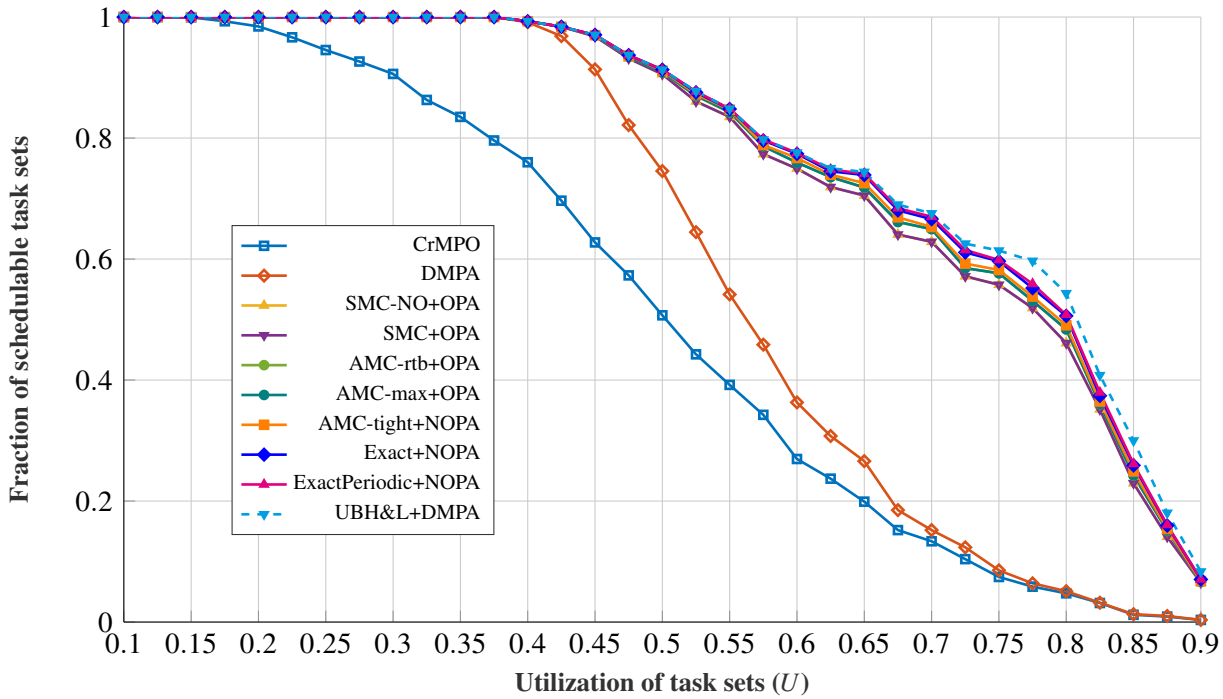


Figure 3.10: Fraction of schedulable task sets for different scheduling approaches and corresponding schedulability tests with regard to processor utilization factor U.

bility bound for fixed-priority mixed-criticality systems.

More importantly, Fig. 3.10 shows that the devised test, i.e., **AMC-tight**, outperforms **AMC-max** schedulability test in terms of schedulable task sets. The difference is better illustrated in Fig. 3.11. Fig. 3.12 shows the timing performance of the approaches with regard to the utilization. As expected, exact approaches have the longest execution time, which is at least one order of magnitude larger than the sufficient tests. The distinction between execution times of sufficient schedulability tests is better depicted in Fig. 3.13. In addition, it can be seen that there are no significant differences in execution time of sufficient tests with regard to the utilization.

Figs. 3.14-3.17 show the performance of schedulability tests with regard to the number of tasks in the system. Firstly, the case when n is in interval $[3,8]$ is analyzed, i.e., cases in which there is relatively low number of tasks in the system. 2000 task sets were generated for each n, i.e., the total of $6 \cdot 2000 = 12000$ task sets. Again, in Fig. 3.14, it can be seen that the devised test, i.e., **AMC-tight**, performs better than other sufficient test in sense of schedulable task sets. Moreover, Fig. 3.15 shows that the average execution time is at least one order of magnitude higher in case of the exact schedulability tests. Moreover, the exponential growth in the execution time for exact tests can be noticed. For a higher number of tasks in a set, i.e, interval $[8,100]$, the total of $93 \cdot 1000 = 93000$ tasks sets were generated. Moreover, the **Exact** test is omitted in evaluation due to the exponential growth in its execution time. In Fig 3.17, it can be seen that for a larger number of tasks in the system, execution time of **AMC-tight** becomes significantly higher than the other sufficient tests. Again, this is to be expected since

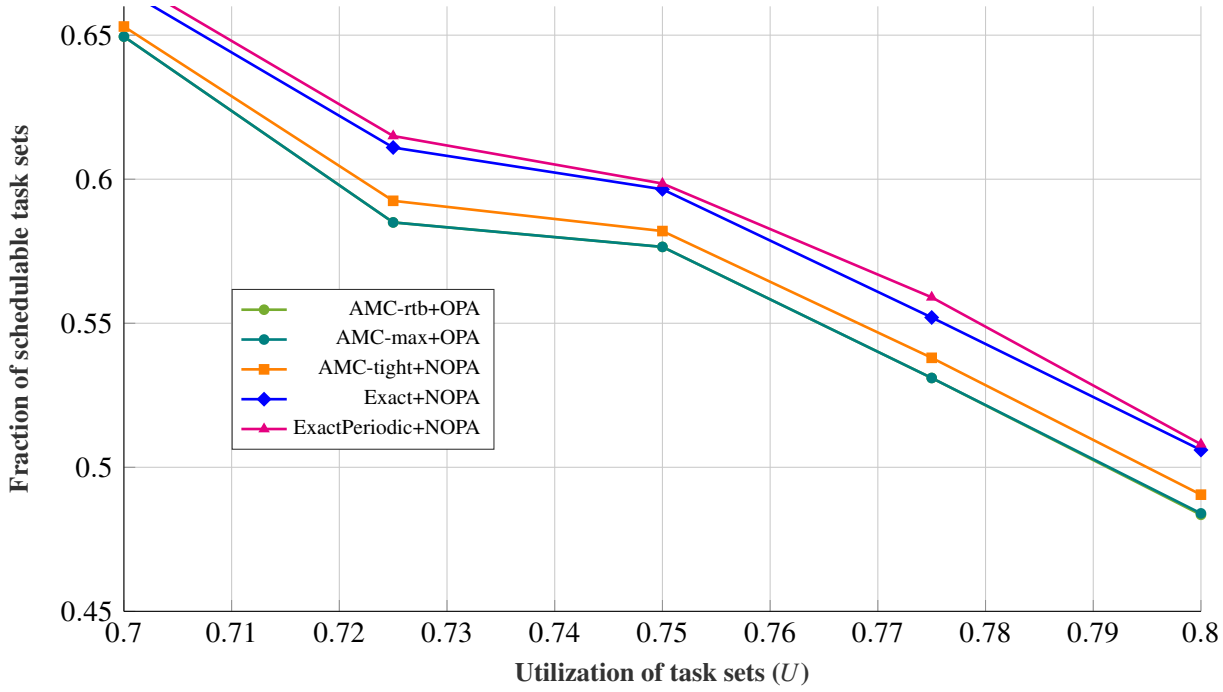


Figure 3.11: Fraction of schedulable task sets for different AMC scheduling approaches and corresponding schedulability tests with regard to processor utilization factor U.

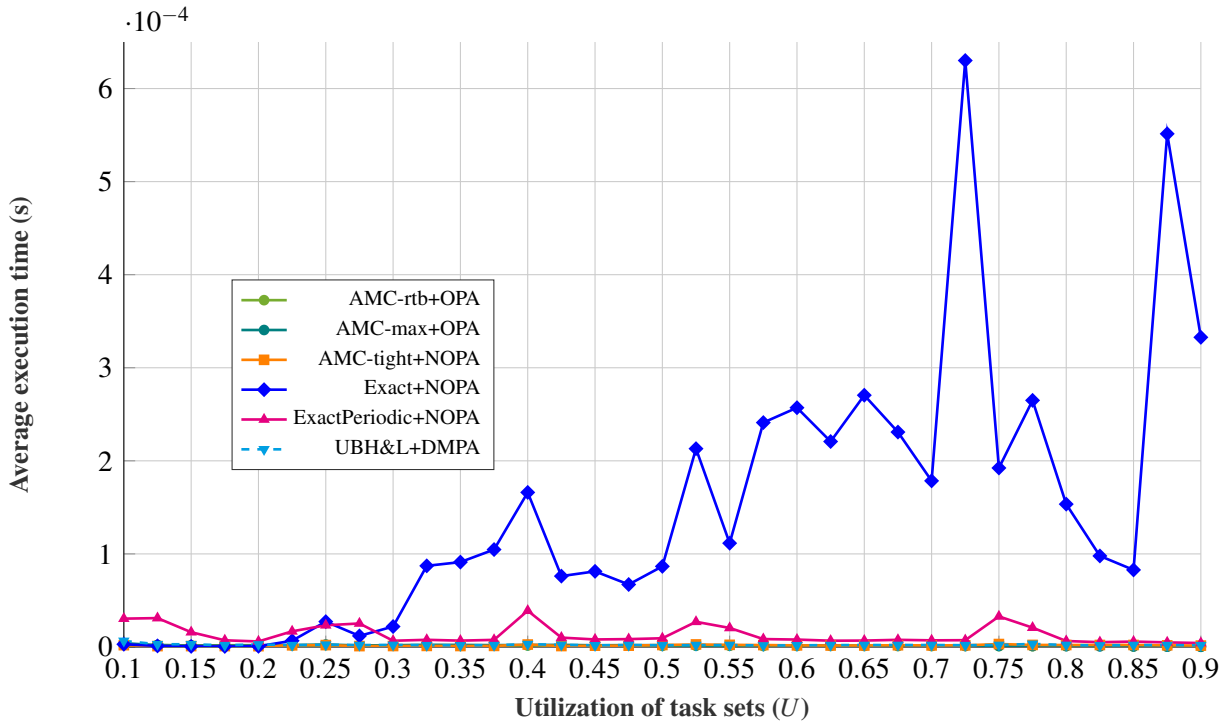


Figure 3.12: Average execution time of priority assignments with corresponding schedulability tests with regard to processor utilization factor U.

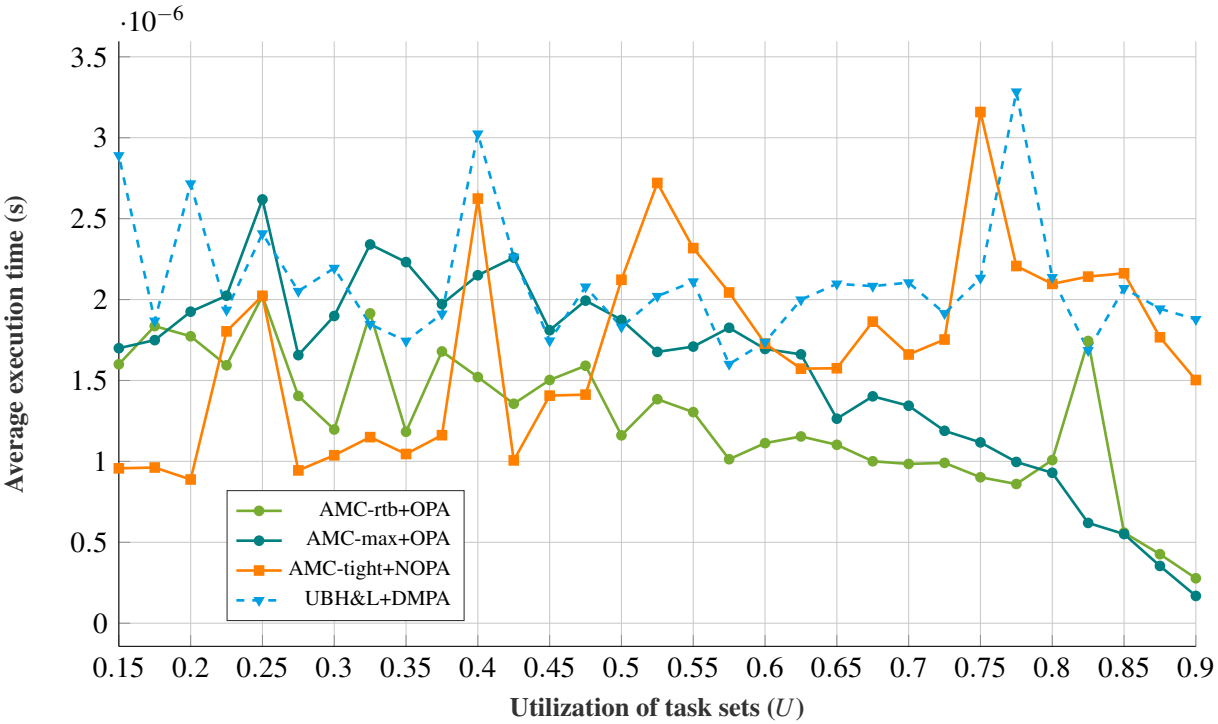


Figure 3.13: Average execution time of priority assignments with corresponding sufficient schedulability tests with regard to processor utilization factor U.

the devised test involves more computation than **AMC-max** test as stated in Property 3. Fig. 3.16 shows that even for a higher number of tasks in system, a fraction of task sets that are schedulable with the devised test is higher than in case of existing sufficient schedulability tests.

Figs. 3.18-3.20 show the performance of schedulability tests with regard to the maximum period in the system T_{max}. 1000 task sets were generated for each value of T_{max} in interval $[100, 500]$ with increment of 50, i.e., the total of $9 \cdot 1000 = 9000$ task sets. Again, the dominance properties are preserved, and in Fig. 3.18 it can be seen that a higher fraction of task sets are schedulable with regard to the devised test than in case of existing schedulability tests. Moreover, exact schedulability tests dominate sufficient tests. On the other hand, Figs. 3.19-3.20 show that in terms of execution time exact schedulability tests have significantly higher execution time than sufficient tests.

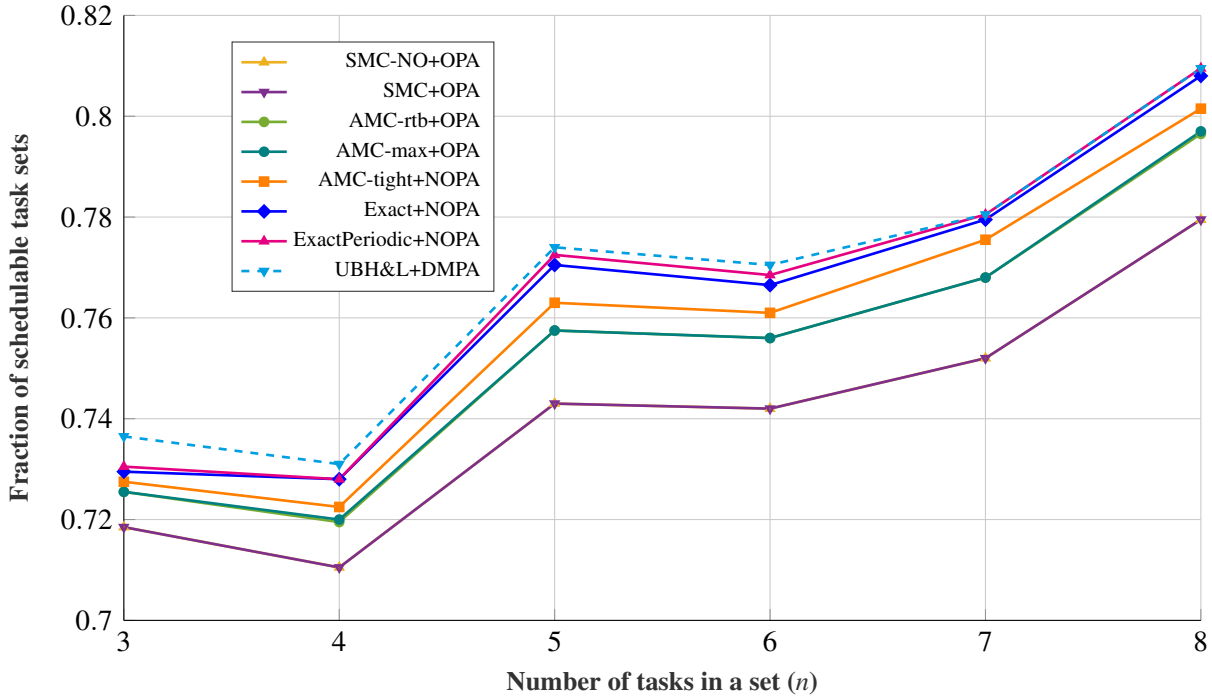


Figure 3.14: Fraction of schedulable task sets for different scheduling approaches and corresponding schedulability tests with regard to the number of task in a set $n \in [3,8]$.

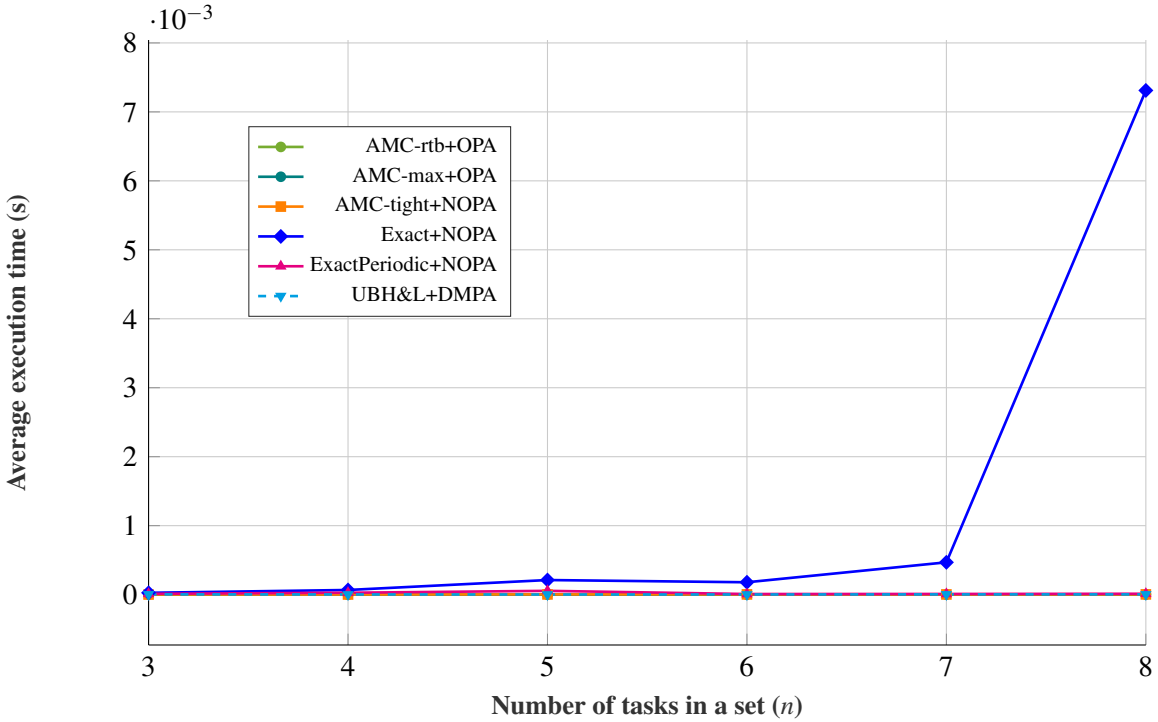


Figure 3.15: Average execution time of priority assignments with corresponding schedulability tests with regard to the number of task in a set $n \in [3,8]$.

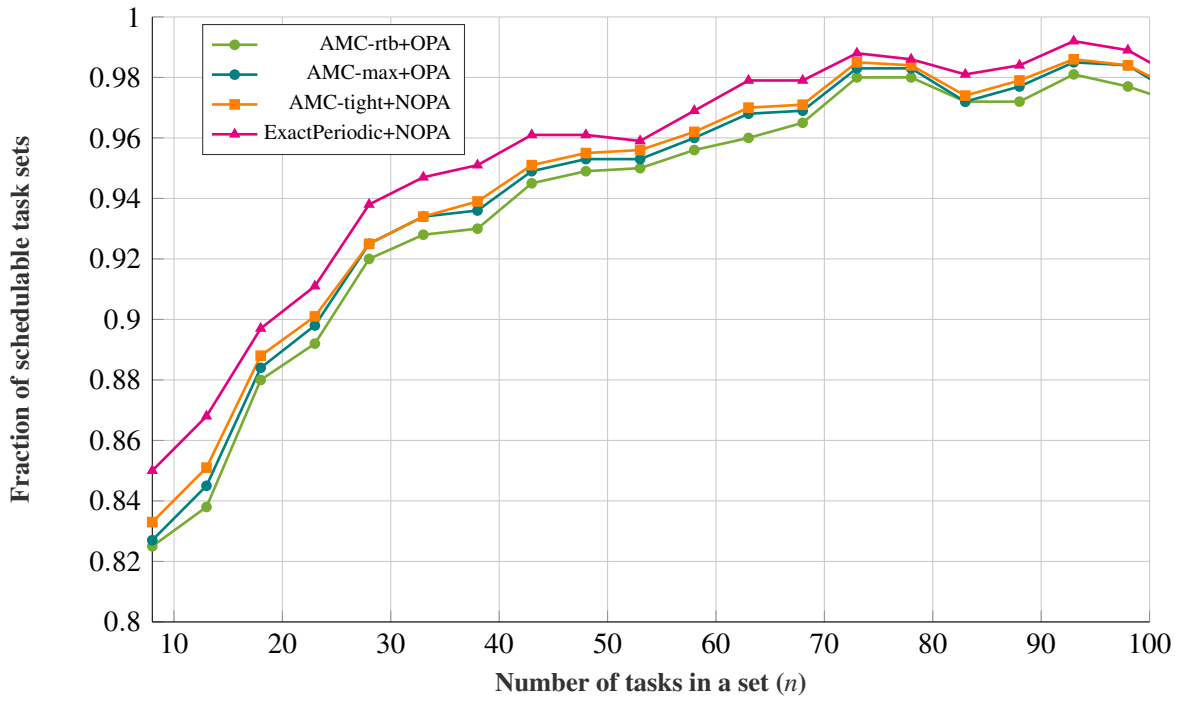


Figure 3.16: Fraction of schedulable task sets for different scheduling approaches and corresponding schedulability tests with regard to the number of task in a set $n \in [8, 100]$.

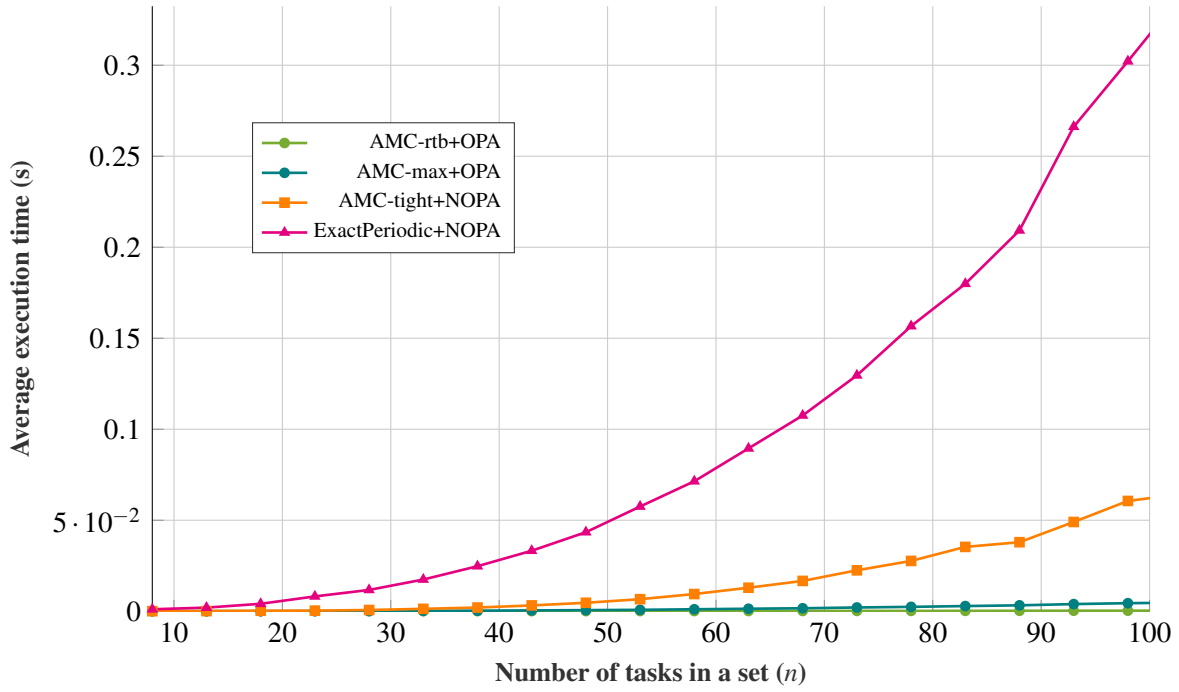


Figure 3.17: Average execution time of priority assignments with corresponding schedulability tests with regard to the number of task in a set $n \in [8, 100]$.

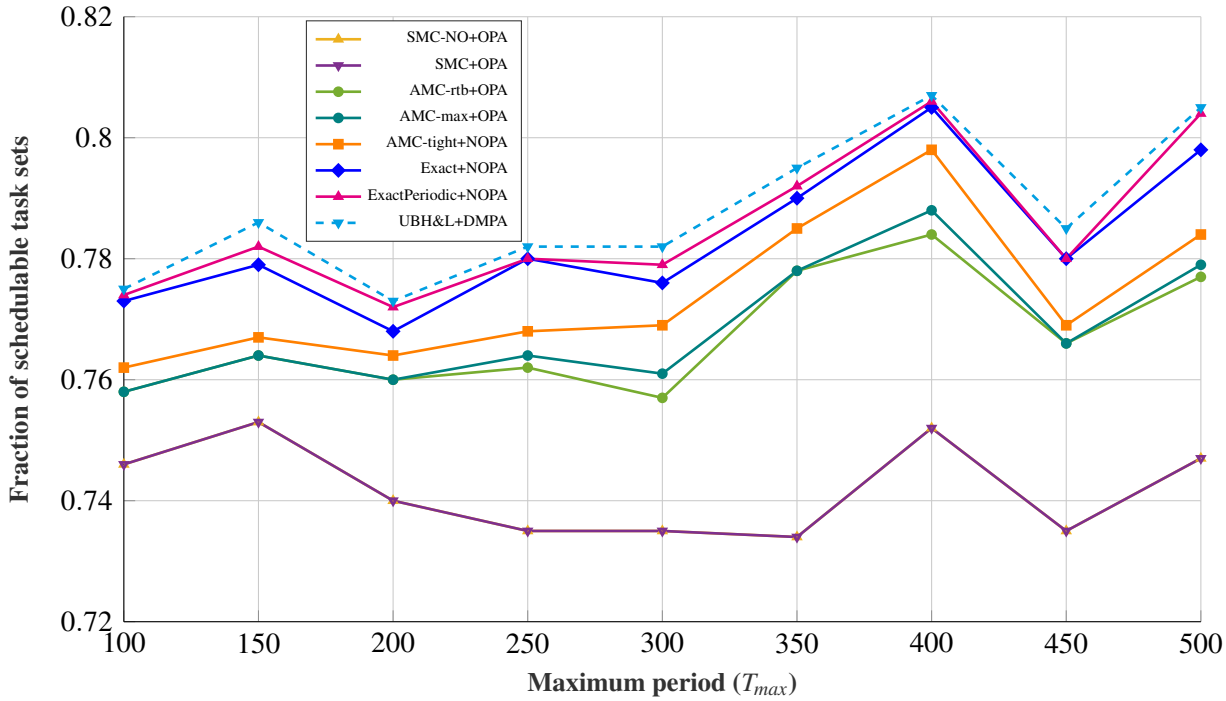


Figure 3.18: Fraction of schedulable task sets for different scheduling approaches and corresponding schedulability tests with regard to the maximum period of task in a set $T_{max} \in [100, 500]$.

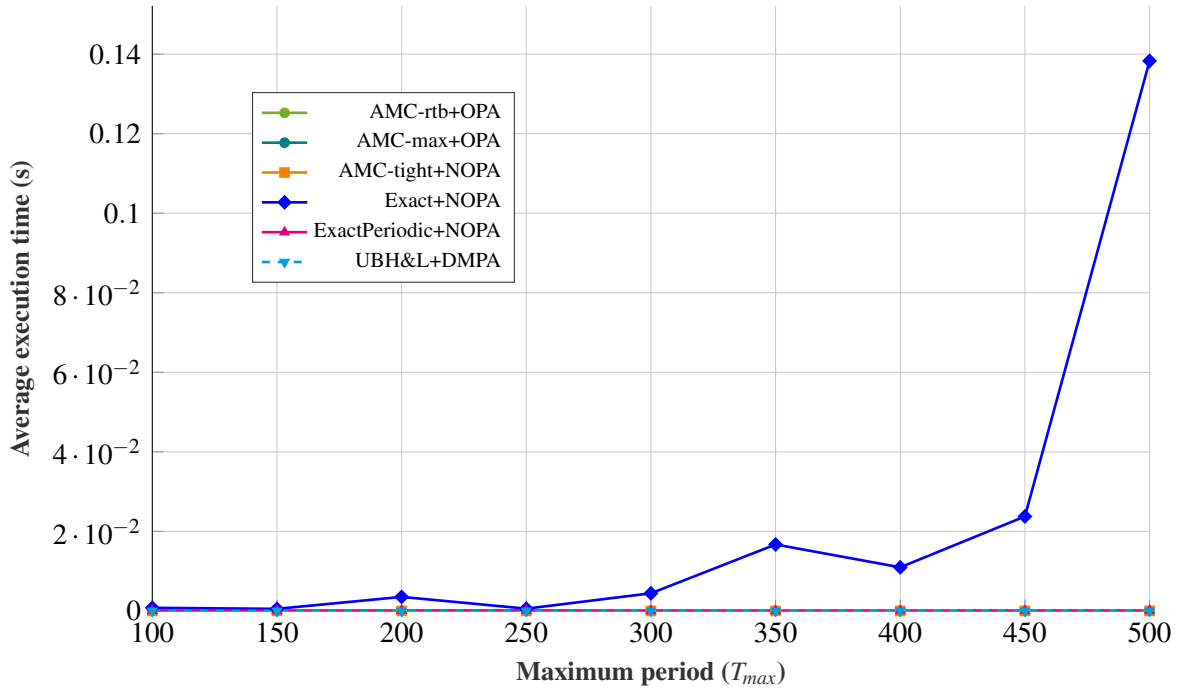


Figure 3.19: Average execution time of priority assignments with corresponding schedulability tests with regard to the maximum period of task in a set $T_{max} \in [100, 500]$.

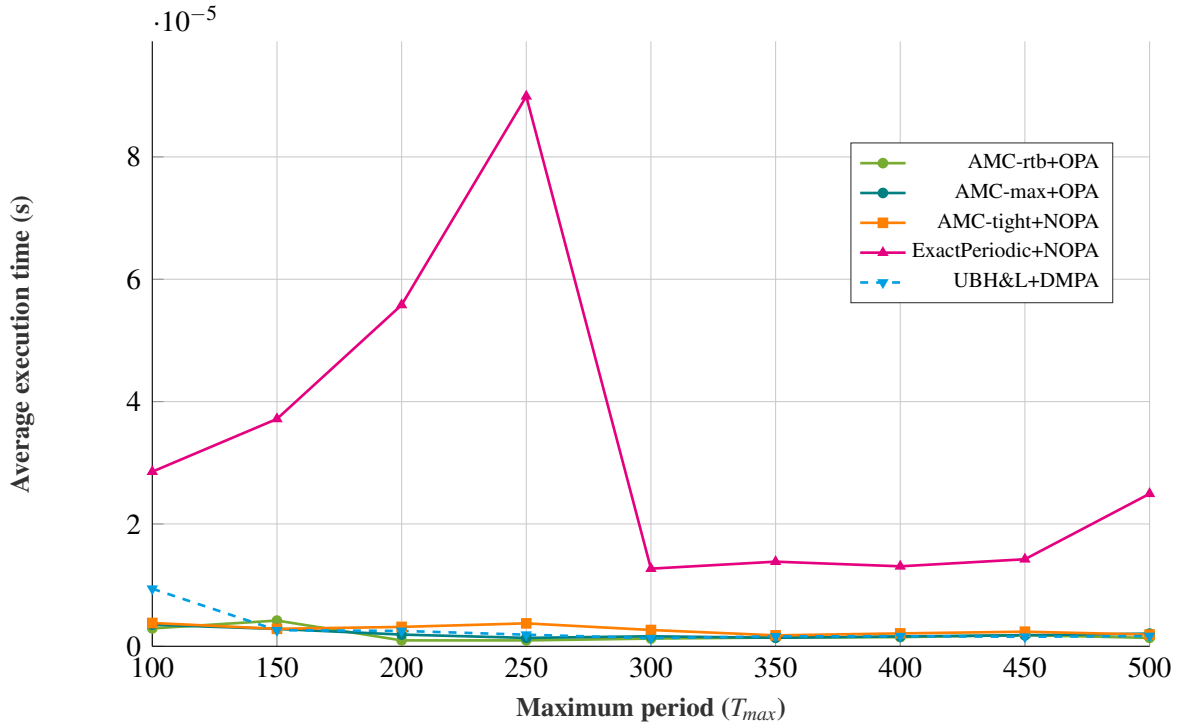


Figure 3.20: Average execution time of priority assignments with corresponding schedulability tests with regard to the maximum period of task in a set $T_{max} \in [100, 500]$. The exact schedulability test is excluded.

3.8 Validating existing schedulability tests

The important part of validation of the newly devised schedulability tests is testing and systematic comparison to different existing schedulability tests. Through systematic evaluation of schedulability tests during the research using the framework for schedulability testing discussed in section 3.9, errors and faults in existing response-time analyses were discovered. In this section, errors and inconsistencies in the efficient formulation of the exact schedulability test described in chapter 4.2 of [37] are stated and discussed. The algorithm is depicted in section 3.5.4 of this thesis as well.

3.8.1 The first error: Rule 1*

The first error is in Rule 1* which describes assignment of the minimum remaining time until the next release of task τ_i, i.e., p'_i. In the paper, it is stated that if $\sigma_i(t+1) = 1$, then $p'_i = T_i$. This is not correct as it discriminates against other release patterns in which job of task τ_i is released with execution time larger than 1. Therefore, the condition should be $\sigma_i(t+1) \geq 1$.

The first error causes invalid state transitions. Consequently, the schedulability test discovers incorrect task response times, which leads to an incorrect schedulability decision. To illustrate the consequences of the first error, the following example can be observed.

Example 10. *Consider task set with 3 tasks* $\tau_i = \{[C_i(LO), C_i(HI)], T_i, D_i, L_i\}$*:*

- $\tau_1 = \{[1,2],5,5,HI\}$
- $\tau_2 = \{[1,1],3,3,LO\}$
- $\tau_3 = \{[2,4],7,7,HI\}$

Tasks are ordered according to priority, i.e., task τ_1 *has the highest priority and task* τ_3 *has the lowest priority. Consider the transition from the pre-initial state* $s_{-1} = \langle LO, (0,0,0,0,0)_{i=1}^{3} \rangle$*. According to Rule 2* there are six different job sequences at time instant* 0*:* $\sigma(0)^1 = \{0,0,4\}$*,* $\sigma(0)^2 = \{0,1,4\}$*,* $\sigma(0)^3 = \{1,0,4\}$*,* $\sigma(0)^4 = \{1,1,4\}$*,* $\sigma(0)^5 = \{2,0,4\}$*,* $\sigma(0)^6 = \{2,1,4\}$*. For instance, we can take the sixth job sequence* $\sigma(0)^6$ *and apply Rule 1* to get the next state* s_0^6*. According to Rule 1* task state* $(c_i, q_i, p_i, \varepsilon_i, \phi_i)$ *for* τ_2 *is:*

- $(c_2 = 1, q_2 = 3, p_2 = 3, \varepsilon_2 = 1, \phi_2 = 0)$

As $\sigma_2(0)^6 = 1$*, the minimum remaining time until the next release of a job* p_2 *is set to* 3 *what is correct. However, this is not the case for tasks* τ_1 *and* τ_3*:*

- $(c_1 = 2, q_1 = 5, p_1 = 0, \varepsilon_1 = 2, \phi_1 = 0)$
- $(c_3 = 4, q_3 = 7, p_3 = 0, \varepsilon_3 = 4, \phi_3 = 0)$

where p_1 *and* p_3 *are set to* 0 *what is incorrect because the remaining time until the next release of a job corresponds to task periods* 5 *and* 7*, respectively. Therefore, the part of Rule 1* which states: "if* $\sigma_i(t+1) = 1$ *then* $p_i' = T_i$*" is incorrect and should be corrected to: "if* $\sigma_i(t+1) \geq 1$ *then* $p_i' = T_i$*".*

As pointed out earlier an incorrect state transition causes the algorithm to produce an incorrect response time and consequently an incorrect schedulability decision. To better illustrate this, Fig. 3.21 and Fig. 3.22 that depict the state space exploration for τ_3 *for the corrected and the incorrect Rule 1* respectively are provided. In Fig. 3.21 it can be easily seen that the task set is not schedulable. Failure is declared after transition from the 7-th state to the final state in which* $c_3 > q_3$*, i.e., the remaining execution time is greater than the remaining time to deadline. On the other hand, the algorithm with incorrect Rule 1* declares success as it discards all states except the initial state. As the observed worst-case response time (*$R_{observed}$*) of* τ_3 *is set to LO-criticality mode response time (*$R_3^{LO} \leq D_3$*) prior to the state space exploration, the algorithm produces an incorrect response time and schedulability decision.*

3.8.2 The second error: pruning rule PR8

The second error is in the pruning rule PR8. As stated in the paper:

PR8 If $p_m = T_m$ *and there exists a LO-criticality task* $\tau_i, i \in \{m+1, ..., k\}$*, such that* $(c_i > 0 \wedge \forall \tau_j, j \in \{1, ..., m-1, m+1, ..., i\}, c_j = 0)$.

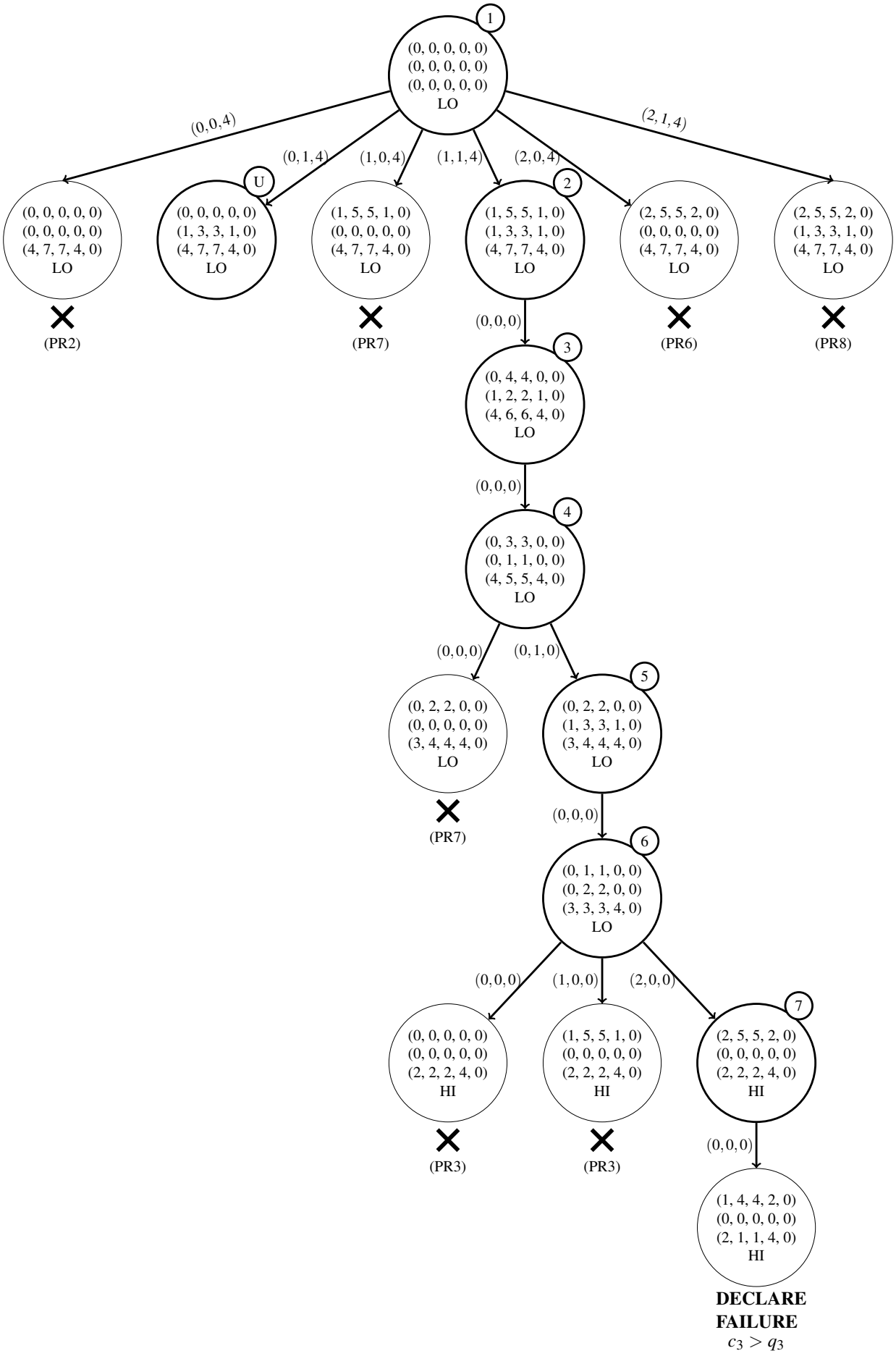


Figure 3.21: State space exploration with corrected Rule 1*.

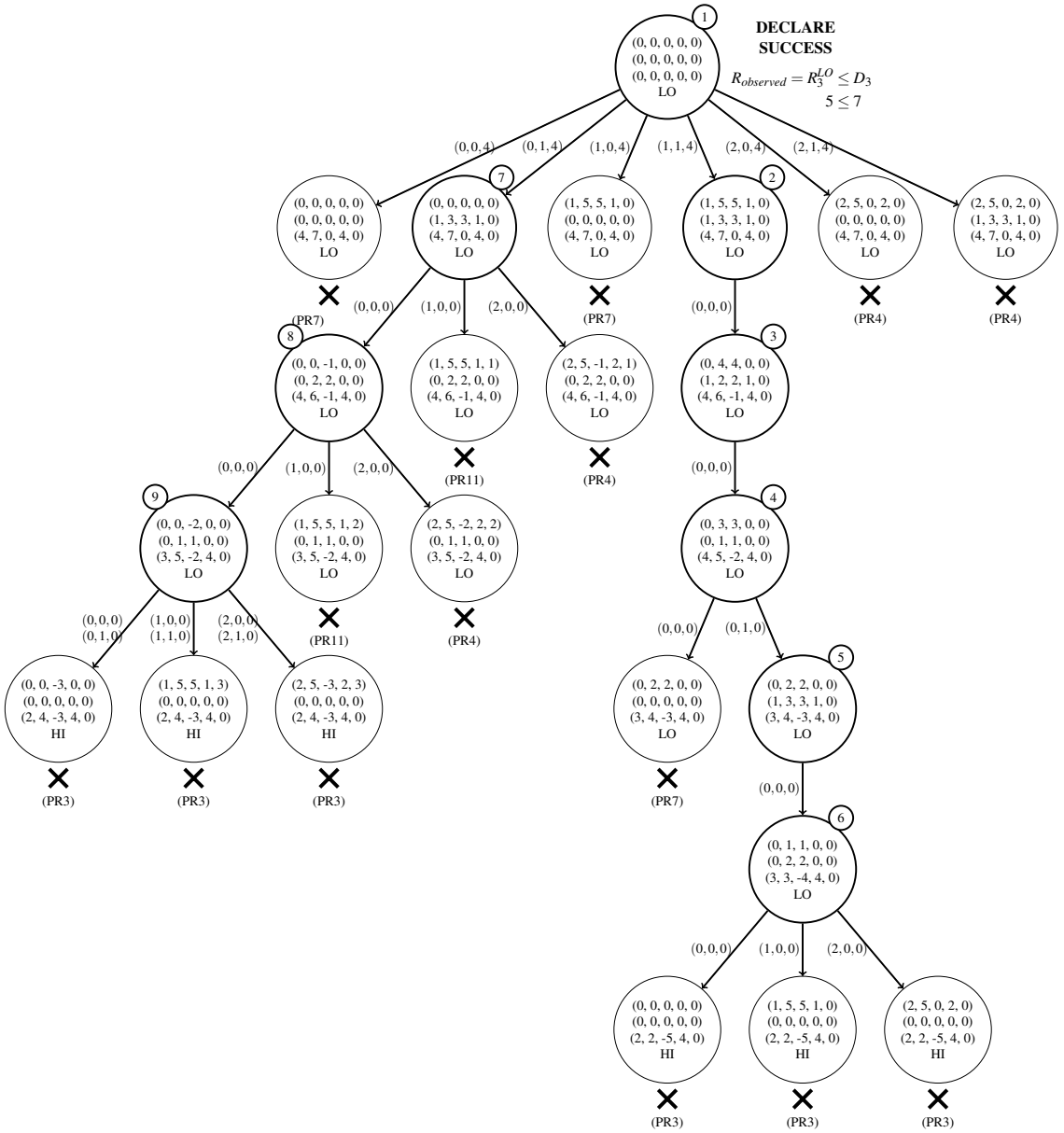


Figure 3.22: State space exploration with incorrect Rule 1*.

The problem with the rule is that if task τ_i, i.e., index i, is included in set $\{1,...,m-1,m+1,...,i\}$ then for the condition to be valid c_i has to be equal zero ($c_j = 0$). However, the first part of the condition states that $c_i > 0$. Therefore, the *PR8* is incorrect as the condition is always false due to the contradiction. The latter set in the pruning rule should be corrected to $\{1,...,m-1,m+1,...,i-1\}$ and then task τ_i is excluded from the former part of the condition. Additionally, index k should be excluded from set $\{m+1,...,k\}$ as τ_k is a high-criticality task.

3.8.3 An inconsistency: schedulability test algorithm

There is an inconsistency in the efficient schedulability test formulation depicted with Alg. 4 from section 4.2 regarding the Proposition 4 from the section 3.3 of the paper. For completeness and clarity, Alg. 4 and Proposition 4 are reproduced here as defined in [37]. The corrected version of the algorithm was already stated (see Alg. 2).

Proposition 4. *If criticality levels of task τ_i and all tasks with priorities higher than it are the same (i.e., $\forall \tau_j \in hp(\tau_i), Lj = L_i$), the critical instant for task τ_i corresponds to what SAS offers. Hence, the standard RTA can be used as follows:*

$$R_i = C_i(L_i) + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(L_i)$$

An inconsistency which may cause confusion is in the lines 4-5 of Algorithm 4 which depicts the schedulability test. The condition specified in `else if` statement (line 4) is: $\forall \tau_j \in \{\tau_1,...,\tau_k\}, L_j == L_k$. Although, this represents Proposition 4 accurately, at this point it is obvious that $L_k = HI$ as the condition in line 2 is not satisfied, i.e., task τ_k is undoubtedly a high-criticality task.

3.9 A framework for comparison of different schedulability tests and response-time analyses

As mentioned before, the common technique for validating and evaluating scheduling techniques, response-time analyses and schedulability tests in real-time scheduling theory, is evaluation on synthetically generated task sets. This testing technique enables researchers to test validity, measure performance and further investigate properties of an observed system. Although frameworks for evaluation of theoretical real-time system models exists [40, 41, 42, 43, 44], they were found to be either too generic or inflexible to enable appropriate validation for schedulability tests in mixed-criticality systems, which is needed in this research. Therefore, as a part of this research, a framework for evaluation of schedulability tests was devised. The framework is maintained in GitHub repository [45].

Algorithm 4 Efficient Exact Schedulability Test (original version)

Input: $\Pi = \{\tau_1, \tau_2, ..., \tau_N\}$, where for each task $\tau_i \in \Pi$, $Pr(\tau_i) = i$
Output: The worst-case response time R_k for all tasks $\tau_k \in \Pi$, if Π is schedulable; *null*, otherwise;

1: **for** $\tau_k \in \Pi$ **do**
2: **if** $(L_k == LO) || (\forall \tau_j \in \{\tau_1, ..., \tau_k\}, C_j(LO) == C_j(HI))$ **then**
3: $R_k = C_k(LO) + \sum_{j<k} \lceil \frac{R_k}{T_j} \rceil C_j(LO)$
4: **else if** $(\forall \tau_j \in \{\tau_1, ..., \tau_k\}, L_j == L_k)$ **then**
5: $R_k = C_k(L_k) + \sum_{j<k} \lceil \frac{R_k}{T_j} \rceil C_j(L_k)$
6: **else**
7: $R_k = \text{SB-RTA}(\Pi, k)$
8: **end if**
9: **if** $R_k > D_k$ **then**
10: **return** *null*
11: **end if**
12: **end for**
13: **return** $\{R_1, ..., R_N\}$

3.9.1 Framework features

The devised framework provides classes and API (*Application Programming Interface*) which allows an user to model a particular sporadic or periodic mixed-criticality system. Using testing API the user can easily evaluate existing schedulability tests. This includes validation of schedulability for tests, validation of response-time, and measuring runtime for each particular test. Moreover, the framework enables the user:

- To define a custom sporadic or periodic mixed-criticality system with arbitrary number of criticality levels.
- To define a custom priority assignment function.
- To define a custom response-time analysis.
- To define a custom feasibility test.
- To use API for testing and comparing user-defined priority assignments, response-time analyses and feasibility tests and comparison with existing tests.
- To find counter examples and differences between different response-time or feasibility analyses.
- To use highly flexible API for generation of synthetic task sets.

Most of these features were used in evaluation of the devised test (section 3.7.2), for generation of examples (throughout chapter 3), and validating existing schedulability tests and detecting errors as well as correcting the existing schedulability tests 3.8. In the following subsection, a brief overview of code organization of the framework is provided.

3.9.2 Code organization

The framework is implemented in Java programming language. The source code of the framework is organized in packages, which contain classes that implement different functionalities required for the schedulability testing. Framework packages are listed in Table 3.1. The contents of crucial packages are shown in Tables 3.2-3.7.

Since the framework enables introduction of new schedulability tests and extension of the existing functionalities, the features that are related to the system model are implemented on a higher level of abstraction, i.e., as abstract classes and interfaces. The list of abstract classes and interfaces, which constitute the core of the framework are shown in Table 3.5. Three basic classes and interfaces that are specified in `interfaces` package define priority assignment, feasibility test, and response time interfaces. Their respective implementations are in separate packages: package with priority assignments implementation shown in Table 3.2, package with feasibility tests is shown in Table 3.4, and package with response times is shown in Table 3.6. Another generic item in the framework is class `FeasibilityTestResponseTime`, which compares the response time of each task to its respective deadline. Note that this is in line with the theoretical observation about the classes of schedulability tests, i.e., schedulability test can be either boolean test or response-time based test [46].

The `evaluation` package shown in Table 3.3 contains two additional classes, which are used in the experimental evaluation of schedulability tests. Class `TestItem` represents a combination of a priority assignment algorithm and a feasibility test. In the framework implementation, all fixed-priority response time analyses and feasibility tests take into an instance of `TaskSet` class, which contains tasks that are ordered according to a certain priority assignment. Therefore, the usage of `TestItem` class ensures that a task set is ordered according to the desired priority assignment prior to applying the schedulability test. However, in a special case, when using the Audsley's optimal priority assignment (OPA) the schedulability decision can be obtained in a more efficient manner. This is due to the optimality of the OPA algorithm. More precisely, when the OPA algorithm is employed, response time is utilized in the process of assigning priorities. If the OPA algorithm cannot produce a priority assignment for a task set with given response-time analysis, task set is not feasible with regard to that analysis. Note that for such an usage of a response-time analysis in the OPA algorithm, a response-time analysis has to satisfy requirements of OPA compatibility [38]. In the context of mixed-criticality systems, examples of OPA compatible schedulability tests are **AMC-max**, **AMC-rtb**, **SMC-NO**, **SMC**, and hence they can be used with the OPA algorithm. However, the exact schedulability test devised in [37] is not OPA compatible as well as the sufficient test devised in this research.

When the schedulability test is not OPA compatible in a fixed-priority scheduled system, it is possible to use suboptimal heuristic assignments. An example of such an assignment that is used with the exact schedulability test and sufficient test devised in the thesis is implemented

in the framework in class `PriorityAssignmentNOPA`. This is in fact the assignment algorithm depicted in Alg. 3.

Similarly, note that the usage of priority assignment is not necessary for the dynamic priority assignment approaches since the information about priority assignment is incorporated in the schedulability test itself. From the implementation point of view, it is not necessary to sort the tasks when applying feasibility tests in dynamic environment since the priority of tasks in system is not fixed, yet it changes during the system operation. Examples of such tests are tests based on the EDF with virtual deadlines approach, i.e., the EDF with virtual deadlines test devised in [47] and Ekberg's schedulability test devised in [48].

In `taskgen` package, implementations of task generators are placed. At the moment, a single implementation of task generator algorithm is available, i.e., UUnifast algorithm devised in [39] is implemented. However, the implementation in this framework has several additional features that are useful in the task generation:

- Implementation enables adding a constraint on the hyperperiod of the generated task sets, i.e., if the hyperperiod of generated task set exceeds the given maximum value, it shall be discarded.
- Implementation enables adding a constraint that denotes the allowed deviation of the utilization. Note that since task sets that have integer, i.e., discrete parameters, the resulting utilization of a task set generated by the UUnifast algorithm can be different than the target utilization due to the rounding errors.
- Implementation enables adding a schedulability test as a filter for the resulting task sets, i.e., an instance of `IFeasibilityTest`. This is especially useful in cases when only feasible task sets with regard to a certain feasibility test have to be observed.
- Finally, the implementation enables blacklisting certain task sets from the generated set of task sets. This is useful when we want to generate two different sets of task sets.

Table 3.1: The organization of Java packages in the framework implementation

Package	Description
assignments	Package contains implementations of priority assignment algorithms for mixed-criticality systems.
evaluation	Package contains utilities for testing of priority assignments, feasibility tests and response-time analyses.
ftests	Package contains implementations of feasibility tests.
interfaces	Package contains definitions of interfaces for priority assignments, feasibility tests and response-time analyses.
models	Package contains classes which represent mixed-criticality job, task, task set, system state and task state.
rtimes	Package contains implementation of response-time analyses.
taskgen	Package contains implementation of task set generators.

Table 3.2: Classes in the `assignments` package

Class	Description
PriorityAssignmentDynamic	Class contains implementation of off-line assignment for dynamic priority system (does not change the order of tasks off-line).
PriorityAssignmentCrMPO	Class contains implementation of criticality-monotonic priority ordering (CrMPO) for dual-criticality systems.
PriorityAssignmentDM	Class contains implementation of deadline-monotonic priority assignment (DMPA).
PriorityAssignmentOPA	Class contains implementation of Audsley's optimal priority assignment (OPA) algorithm devised in [32].
PriorityAssignmentNOPA	Class contains implementation of non-optimal priority assignment (NOPA) for dual-criticality systems devised in [37].

Table 3.3: Classes in the `evaluation` package

Class	Description
TestItem	Class represents a single test item. In this context, test item is a combination of priority assignment and feasibility test.
TestUtils	Class contains implementation of methods which run testing with custom parameters.

Table 3.4: Classes in the `ftests` package

Class	Description
FeasibilityTestEDFWithVD	Class contains implementation of the earliest deadline first with virtual deadline feasibility test from [47].
FeasibilityTestEkbergGreedy	Class contains implementation of the Ekberg's schedulability test for EDF-scheduled dual-criticality systems as proposed in [48].
FeasibilityTestUBHL	Class contains implementation of the necessary schedulability test based on separate calculation of LO and HI response times (UBH&L) [47].
FeasibilityTestEfficientExact	Class contains implementation of the exact schedulability test for fixed-priority preemptive dual-criticality systems as proposed in [37] with corrections from [25] and this research.
FeasibilityTestEfficientExactWrong	Class contains implementation of the exact schedulability test for fixed-priority preemptive dual-criticality systems as proposed in [37] with possible errors.
FeasibilityTestResponseTime	Class contains implementation of the generic response-time schedulability test for fixed-priority preemptive dual-criticality systems, which compares the response time of each task to its respective deadline.

Table 3.5: Classes and interfaces in the `interfaces` package

Class/Interface	Description
PriorityAssignment	This is an abstract class that specifies priority assignment interface and additional methods.
IFeasibilityTest	Interface specifies methods that each feasibility test has to implement, i.e., a method that checks feasibility.
IResponseTime	Interface specifies methods that each feasibility test has to implement, i.e., a method that calculates response time of a task.

Table 3.6: Classes in the `rtimes` package

Class	Description
ResponseTimeSMCno	Class contains implementation of SMC-NO response time for fixed-priority preemptive dual-criticality systems as proposed in [29].
ResponseTimeSMC	Class contains implementation of SMC response time for fixed-priority preemptive dual-criticality systems as proposed in [30].
ResponseTimeAMCmax	Class contains implementation of AMC-rtb response time for fixed-priority preemptive dual-criticality systems as proposed in [30].
ResponseTimeAMCrtb	Class contains implementation of AMC-max response time for fixed-priority preemptive dual-criticality systems as proposed in [30].
ResponseTimeEfficientExact	Class contains implementation of the exact response time for fixed-priority preemptive dual-criticality systems as proposed in [37] with corrections from [25] and this research.
ResponseTimeClassic	Class contains implementation of response time for non mixed-criticality systems (see section 2.1.4).
ResponseTimePeriodic	Class contains implementation of the exact response time fixed-priority preemptive dual-criticality systems based on the exhaustive system state exploration, i.e., simulation.
ResponseTimeAMCtight	Class contains implementation of the response time analysis for fixed-priority preemptive dual-criticality systems devised in this research (see section 3.6).

Table 3.7: Classes in the `taskgen` package

Class	Description
UUniFastDiscard	Class contains implementation of UUnifast algorithm for task set generation from [39].

3.9.3 Example of implementation of response-time analysis

In this subsection, an example of implementation of response-time analysis is discussed. For this particular example, **AMC-rtb** analysis was chosen since it is significantly simpler than the **AMC-max** analysis and the exact test, but it still includes multiple system states, i.e., LO and HI, which are specific to the adaptive mixed-criticality scheme. As it was discussed earlier in the chapter, to determine schedulability using the **AMC-rtb** test, the following response times have to be calculated: response time in low-criticality mode $R_i(LO)$, response time in high-criticality mode $R_i(HI)$, and response time in case of a criticality switch from low to high-criticality mode $R_i(MC)$. The latter response times can be calculated using the recurrence relations given with (3.20), (3.21), (3.22).

The code shown in Listing 3.1 shows the implementation of the calculation of the $R_i(MC)$ for HI tasks, which is calculated in a loop iteratively. It can be seen that in `for` loop in line 7, it is assumed that the tasks with the index lower than task τ_i have higher priority, i.e., priority assignment implementations move tasks with higher priority into position with lower index. In line 13, it can be seen that two separate cases are handled, which correspond to two different sums in equation (3.22). Note that this code is somewhat rewritten here for conciseness and clarity in comparison with code in [45].

When adding a new response time analysis to the framework, an user has to provide a function similar to the code provided in Listing 3.1. Afterwards, when an instance of `FeasibilityTestResponseTime` is being instantiated, the user provides its implementation of response time to the constructor of the feasibility test class, which will ensure a proper comparison of calculated response times with the deadlines of tasks. Note that if boolean schedulability test is needed, user can directly implement a class that extends `IFeasibilityTest`. In addition, note that schedulability test can have both response-time and boolean variant. Moreover, in the current version of the framework, exact schedulability test devised in [37] has response-time based test and boolean test as shown in Table 3.6 and Table 3.4, i.e., classes `ResponseTimeEfficientExact` and `FeasibilityTimeEfficientExact` are implemented.

3.9.4 Customizing priority assignments

Similarly, as in case of feasibility tests new priority assignments can be added. Here, a brief illustration of implementation of the simplest priority assignment is provided, i.e., deadline-monotonic priority assignment (DMPA). Listing 3.2 shows the partial code of class that implements the DMPA assignment. Simply, by sorting a new list of tasks using the `comparatorDM`, which sorts tasks according to deadlines, in line 8 new priority assignment is obtained. By modifying `assign` method, new priority assignment algorithms can be easily added.

```
1 ...
2 // R_lo and R_hi calculated using
3 // with regard to C(LO) and C(HI)
4 while (R_mc != t && R_mc <= D) {
5   R_mc = t;
6   t = Ci_HI;
7   for (int j = 0; j < i; j++) {
8     MCTask tj = tasks.get(j);
9     int Cj_LO = tj.getWCET(0);
10    int Cj_HI = tj.getWCET(1);
11    int Tj = tj.getT();
12    int Lj = tj.getL();
13    if (Lj < L) {
14      t = t + Cj_LO *
15        Math.ceil(1.0 * R_lo / Tj);
16    } else {
17      t = t + Cj_HI *
18        Math.ceil(1.0 * R_mc / Tj);
19    }
20  }
21 }
22 ...
```

Listing 3.1: The implementation of **AMC-rtb** response time in the framework class. `ResponseTimeAMCrtb`.

```
1 public class PriorityAssignmentDM
2   implements IPriorityAssignment {
3
4   @Override
5   public MCTaskSet assign(MCTaskSet set) {
6     List<MCTask> tasks = new
7     ArrayList<MCTask>(set.getTasks());
8     tasks.sort(comparatorDM);
9     return new MCTaskSet(tasksSorted);
10  }
11  ...
12 }
```

Listing 3.2: The implementation of DMPA in the framework class `PriorityAssignmentDM`.

3.10 Usage of adaptive mixed-criticality schedulability tests in the industrial context

Nowadays, mixed-criticality in industrial context is often linked with the usage of hypervisors in various transportation domains such as automotive, railway or even aerospace. The result of plethora of different projects is a large number of different hypervisors and hypervisor modifications [49, 50, 51, 52, 53], which in theory enable application of adaptive mixed-criticality scheduling scheme to schedule the virtual machines of different criticality. The devised adaptive mixed-criticality schedulability test and the framework for schedulability testing are applicable in any such scenario under the following conditions:

- The system designer has to specify timing parameters from system model section 3.4 for the virtual machines.
- Scheduling can be described with the adaptive mixed-criticality scheduling model.

Applying the schedulability test is fairly trivial when the conditions above are satisfied. The system designer has to apply schedulability test in a similar manner as in Example 9.

3.11 Chapter summary

In this chapter, existing response-time analyses and schedulability tests are discussed and presented. It is shown how the sufficient response-time analysis of adaptive mixed-criticality systems can be improved. Improvements are based on more precise analysis of low and high-criticality interferences in case of a criticality switch for each high-criticality task. If we increase the complexity of the algorithm by allowing the choice of task that causes a criticality switch, we can reduce the worst-case response time. The systematic approach and comparison with different existing schedulability tests has enabled the detection and correction of inconsistencies in the existing exact analysis. Moreover, the devised framework enables consistent testing of the existing schedulability test as well as the devised test. In the end, conditions for usage of developed methods and techniques in the industrial context are presented.

Chapter 4

Method for harmonic period assignment in safety-critical part of real-time mixed-criticality systems

4.1 Context of the research

In the previous chapter, the focus was on the adaptive mixed-criticality scheduling scheme which effectively accomplishes the goal of increasing the schedulability of systems with tasks or functions with different criticality. Moreover, as discussed in the introduction to this thesis, such a technique is acceptable in systems in which the spatial or temporal independence between functions of different criticality can be reduced. However, in a large number safety-critical systems trade-off between maintaining the independence of safety-critical functions and achieving higher schedulability is rarely available. Furthermore, in most cases independence is strictly required by safety standards. In such cases, safety-critical functions have to be completely temporally isolated from non-critical functions. As it can be observed in the subsection 3.5.1, naive scheduling approaches, i.e., partitioned criticality approaches, which maintain temporal isolation between tasks, preform very poorly in terms of schedulability, especially regarding the low-criticality tasks. It is worth noting that in a mixed-criticality system in which functions with different criticality are partitioned, the amount of time available for execution of non-critical tasks corresponds to time in which safety-critical tasks are not executing. A method which would enable the design of safety-critical part of the system in a manner that it can guarantee a certain amount of time for execution of non-critical tasks would be applicable in this situation.

In this chapter, an optimization method is proposed, which can be used for period assignment of safety-critical tasks in such a manner that the final resulting utilization of the safety-critical tasks can be precisely regulated. By regulating the utilization of safety-critical tasks

in such a system, we can effectively control the amount of time that can be used for execution of non-critical functions in the system. Moreover, this method addresses several different challenges that exist in the design of safety-critical systems by means of period assignment.

In this chapter, along with providing a method for regulation of utilization of safety-critical tasks, the issue of assigning a fixed number of harmonic periods from period ranges to maximize utilization in real-time systems is studied. In the existing period assignment approaches, the number of different harmonic period values in the solution was not addressed. In this thesis it is demonstrated that in real-time systems in which the number of available task periods is restricted, such a constraint is crucial for efficient system design. In the chapter, this problem is formally defined in the context of existing harmonic period assignment research. It is shown that this problem is at least weakly NP-hard and an optimal algorithm and suboptimal heuristics are devised. Based on an extensive evaluation on synthetically generated task sets, it is concluded that the devised approach is efficient and applicable in a variety of real-world scenarios.

Note that the part of results presented in this chapter has been published in [54]. Here, as an extension, a more general definition of the harmonic period assignment with distinct number of different period values and arbitrary utilization is introduced. As it is mentioned before, the ability of the optimization procedure to tune the utilization to an arbitrary value is necessary prerequisite to control amount of time which can be used by high-criticality, i.e., safety-critical, and low-criticality tasks in the system.

4.2 Introduction to period optimization in safety-critical systems

As it was discussed earlier, in traditional industrial control systems, timeliness, stability and predictability are very important properties. Moreover, in safety-critical control systems, these properties are *condicio sine qua non* as they are required according to generic safety standards such as IEC 61508 and domain-specific safety standards, e.g., EN 50128 in the railway domain and ISO 26262 in the automotive domain. Efficiency and accuracy of the control algorithm depends to a large extent on the timeliness of underlying embedded computing platforms. Consequently, strict requirements are imposed on the design of operating systems as the controlling procedures have to be prompt and correct. In this context, selecting adequate sample times, i.e., task periods, is crucial for an appropriate behavior of a system.

Additionally, in systems with very strict safety requirements, system designers will often degrade performance of the system drastically in favor of safety characteristics. In this chapter, mixed-criticality systems with partitioned functionalities are observed. As it can be seen in the previous chapter, partitioning tasks with different criticality levels, degrades the schedulability. Therefore, in this chapter, the techniques are provided that can mitigate this effect by optimizing

task periods and utilization in the system. Using the devised period assignment, i.e., period optimization, techniques safety-critical utilization can be optimized. This enables system designers to ensure that enough processor time will remain for execution of non-critical tasks.

4.2.1 Related work

Period assignment is a well-studied topic in real-time system design since the choice of periods in sporadic or periodic task sets has a direct effect on system schedulability, efficiency and utilization. Additionally, harmonic period assignment is of special interest as it is well-known that any harmonic task set with processor utilization less or equal to one is schedulable by a rate-monotonic scheduler [55]. Research in this domain can be divided into three groups with respect to the particular focus of the research.

The first research area includes papers focused on the schedulability of real-time systems with arbitrary period selection, i.e., periods of task sets are not constrained to harmonic values. Early research in this context was done by Seto et al. in [56]. The authors devised algorithms for discovering feasible integer periods in fixed-priority systems with a fixed rate-monotonic and an arbitrary priority assignment. Moreover, in their approach periods are upper-bounded by the slowest task rate required by an application. According to authors in [57], the approach taken in [56] seems to be inefficient due to the combinatorial explosion for larger task sets. On the other hand, in [57], authors precisely formulate the feasibility region in the rate space and devise optimization algorithm for any convex objective. In their approach, periods are not constrained to be integers, while fixed-priority scheduling is assumed. However, there are no additional constraints regarding the period range for tasks in systems.

The second research area includes papers that exploit the harmonic relations between the periods of tasks in systems for determining schedulability. For instance, Han and Tyan in [58] devise a sufficient schedulability bound which is better than the one proposed by Liu and Layland [2]. The approach is based on two previously introduced algorithms Sr and DCT investigated in the context of distance-constrained real-time systems [59]. Similarly, an exact polynomial-time schedulability test for harmonic task sets with any fixed-priority assignment was devised in [60]. It is worth noting that the problem of determining schedulability of sporadic task sets with arbitrary periods is NP-hard [61].

The third research area includes papers which are focused on harmonic period assignment with period ranges. In recent research [62], authors determined that two classical harmonic period optimization problems labeled UHPA (utilization-maximizing harmonic period assignment) and CHPA (cost-minimizing harmonic period assignment) are in the NP-hard complexity class. Additionally, they devised approximation algorithms for the relaxed version of the CHPA problem, i.e., the constraints on period ranges are removed. These problems are formally defined later in the chapter since this research is focused on a variant of the UHPA problem with

additional constraints on period values. In [63], the authors introduce the notion of harmonic projection and devise an exponential time (in size of a task set) algorithm for determining harmonic periods for tasks with period ranges. Additionally, they devise period assignment algorithms such that the resulting utilization of a task set is equal to the lower or the upper bound utilization value. They expand on their work in [64]. The similarities and the differences with the approach devised as a part of this research are highlighted later in this chapter. Period selection and assignment were investigated in the context of minimizing the hyperperiod of task set in systems in which periods are closely harmonically related [65]. Similarly, in [66], the authors investigate the minimization of the hyperperiod by non-harmonic period assignment from period ranges.

The common motivation in the period assignment research is the real-time system and optimal controller co-design, in which the problem of selecting adequate sample times is directly linked to the problem of determining optimal periods [67]. In such approaches, the LQG (linear-quadratic-Gaussian) plant model is used [68, 69]. Moreover, the period assignment of harmonic period values is of great importance in many applications such as radar dwell tasks [70], mobile robotics [71], integrated modular avionics [72], and automotive applications [73].

4.2.2 Motivation and new challenges

As an additional motivation and rationale for imposing additional constraints on the classical harmonic period assignment problem, i.e., a variant of the UHPA, safety-critical software from real-world industrial scenarios is observed since it serves as the primary motivation for this research. Safety-critical embedded software for control applications typically has a modular composition in which each module, task or runnable executes with a predefined period which is determined off-line as a part of the application design. For instance, ANSYS SCADE Suite [74], HIMA SILworX [75] and KONČAR Grap Designer [76] provide automatic code generation based on a set of application modules. The application designer determines a range of periods for every module, i.e., task, in the application. In order to ensure function correctness, tasks have to be executed with periods belonging to their specified range. Moreover, it is in the interest of application and system designers that every task in the system executes with the highest possible frequency, i.e, the lowest possible period as this will ensure a higher quality of service, and consequently increase utilization. Thus, utilization is maximized. The number of tasks in an application can grow and be arbitrarily high. However, in many systems, e.g., the KONČAR Grap [76] operating system, or engine management systems [73] in automotive applications, the number of available periods, i.e., rates, is fixed or bounded and cannot be increased due to the specific architecture of the hardware and the operating system. For instance, the maximum number of different period values may be fixed to 4, or restricted to the interval from 4 to 8. It is worth noting that previous research regarding harmonic period assignment does not address the

number of distinct period values in the solution of the period assignment problem. The devised approach can be used by system and application designers to determine the optimal choice of task periods, even when the number of available periods in the system is limited.

4.2.3 Contributions and organization of the chapter

There are five main contributions in this chapter. Novel utilization-maximizing harmonic period assignment problem with a constrained number of distinct period values referred to as UDHPA is defined(i). It shown that the already studied UHPA problem is Turing reducible to the UDHPA problem. Additionally, using the complexity results for the UHPA problem from [62], it is determined that the UDHPA problem is at least weakly NP-hard by reduction from the well-known partition sum problem (ii). An optimal and heuristic algorithms for the UDHPA problem are devised accompanied with the appropriate time-complexity analysis, and the effectiveness of the approach is demonstrated with extensive evaluation on a large number of synthetically generated instances, which correspond to real-world motivational scenarios (iii). Devised optimal algorithm and heuristics are employed to solve instances of the UHPA problem and they are compare them to the existing approaches (iv). Moreover, a numerical example that illustrates a real-world period assignment problem is provided (v). These results, as presented in [54], do not address the problem when system has to be optimized to arbitrary utilization value that is smaller than the schedulability bound for rate-monotonic harmonic task system, i.e., $U = 1$. Therefore, additional contributions of this thesis: are formulation of the AUDHPA problem, i.e, arbitrary utilization-maximizing harmonic period assignment with a constrained number of distinct period values (vi), and its complexity analysis (vii).

The rest of the chapter is organized as follows. In section 4.3, the system model is introduced. In section 4.5, existing harmonic period assignment problems are reviewed, and then the UDHPA problem, and the more general AUDHPA problem are defined. In section 4.6, the complexity of the UDHPA problem is analyzed. Moreover, the complexity analysis of the AUDHPA problem is provided. In section 4.8, an optimal algorithm for the UDHPA problem, which is trivially modified to solve the AUDHPA problem, is devised. In section 4.9, the devised approaches are evaluated. Finally, in section 4.10, the concluding remarks are stated.

4.3 System model

In this chapter, system S is represented as a set $\mathscr{T}$ of n periodic tasks with no initial offset, i.e., a synchronous task set [77]. A task model that is used common for the period assignment with period ranges proposed in [62, 63], and [64]. Therefore, task τ_i is represented as a tuple $\tau_i = \{C_i, I_i\}$, where C_i is the worst-case execution time (WCET) of τ_i, and I_i is the period range

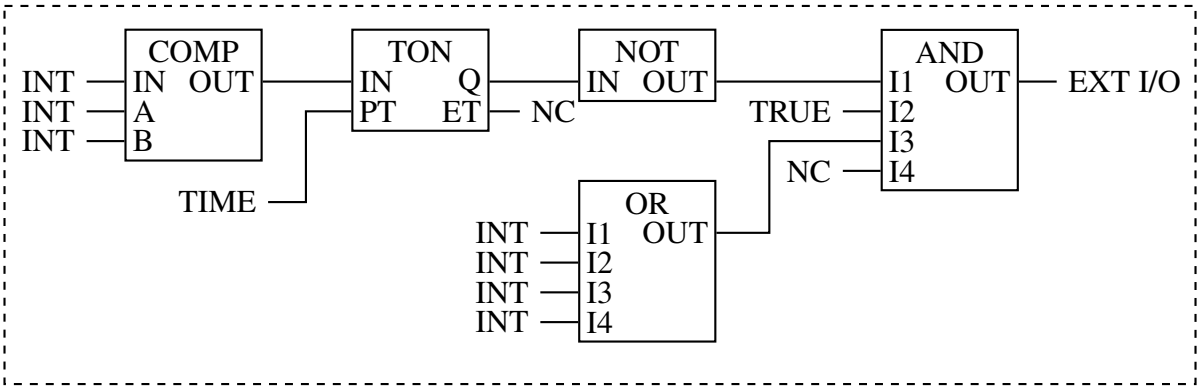


Figure 4.1: Example of part (module) of control software.

of allowed period values for τ_i. Period range $I_i = [p_i^{min}, p_i^{max}]$ is determined by the minimal p_i^{min} and the maximal p_i^{max} allowed period value. Actual period value of task τ_i is denoted as T_i. This is the value which is assigned to a task by solving a period assignment problem. Additionally, T_i is the deadline of τ_i, i.e., implicit-deadline task sets are considered. It is assumed that the WCET is a real number, i.e., $C_i \in \mathbb{R}$, and that task periods are integers, i.e., $T_i, p_i^{min}, p_i^{max} \in \mathbb{N}$. Moreover, periods are in harmonic relation, i.e., $\frac{T_i}{T_j} \in \mathbb{N} \vee \frac{T_j}{T_i} \in \mathbb{N}, i, j \in [1, n]$. Definition 18 determines the correctness of period assignment.

Definition 18. ***Correct period assignment.*** *Period assignment for task τ_i is correct iff period value T_i assigned to task τ_i is such that $p_i^{min} \leq T_i \leq p_i^{max}$.*

It is worth noting that the correctness of period assignment does not guarantee the feasibility of a task set. The feasibility of a harmonic task set is often expressed using the utilization of a task set, i.e., $U = \sum U_i = \sum \frac{C_i}{T_i} \leq 1$.

4.4 Mapping of the system model to the motivational applications

To further justify the imposing of additional constraints on the classical harmonic period assignment problem, i.e., variant of UHPA, a safety-critical software example from real-world industrial scenario, which primarily motivated this research, is introduced. Safety-critical embedded software for control applications typically has modular composition where each module is represented as a functional block diagram (FBD). Function block diagram (FBD) is a graphical programming language described in the third part of IEC 61131 standard [78]. FBD is used as a technique for design of safety-critical applications regardless the platform architecture. As mentioned before, industrial tools such as ANSYS SCADE Suite [74], HIMA SILworX [75] and KONČAR Grap Designer [76] provide automatic code generation, which is based on the FBD diagrams. Application consists of many modules which are executed in recurring or

cycling manner. Every module consists of smaller building blocks which correspond to the elements of FBD programming language. To better illustrate the notion of an application module, an example of module is depicted in Fig. 4.1. Output signal of a module `EXT I/O`, depends on several input variables. Elements of a module execute sequentially following predefined order of execution, e.g., COMP $\rightarrow$ TON $\rightarrow$ NOT $\rightarrow$ OR $\rightarrow$ AND. Application specification determines range of periods for every module in the application. For a function to be correct, module has to be executed with period belonging to the specified range. The number of modules in an application can grow and be arbitrarily high, however in many systems, e.g., KONČAR Grap [76], the maximum number of available periodic tasks is fixed and cannot be increased, due to the specific architecture of hardware and operating system. Therefore, application modules have to be scheduled using available periodic tasks. The algorithms devised in this chapter can be directly applied in such scenario as a tool for generating assignment of modules to tasks. Similar concept to module is *runnable* which is a basic workload unit in organizing industrial and automotive software. For instance, Amalathea framework [79] uses runnables in modeling in industrial and automotive environments. Execution of modules and runnables follows read-compute-write policy [80] similar as blocks in Simulink [81]. The proposed approach can be applied to those modeling approaches as well.

Note that in the model presented in the previous subsection, modules or runnables correspond to tasks and the notion of the periodic task is directly linked with period which has to be discovered. Here, an alternative model is provided, which might be more straightforward to the system designer of industrial application. This model is not used further in the thesis and it is introduced here primarily for clarity. In order to better fit the real-world scenario, i.e., modular application structure, the notion of module as the basic workload unit of a task is introduced. Therefore, tasks are composed of smaller runnable units which are often referred to as modules or runnables. A module (M_j) is described with a set $M_j = \{c_j, p_j^{min}, p_j^{max}\}$ where $c_j \in \mathbb{R}^+$ is the WCET of a module and $p_j^{min} \in \mathbb{N}$ and $p_j^{max} \in \mathbb{N}$ are its minimum and maximum period, respectively. With introduction of modules as the building blocks of a task, system model is changed to include module set $\mathscr{M}$. Therefore, system S can be represented as a tuple:

$$S = (\mathscr{M}, \mathscr{T}) \tag{4.1}$$

In order to be executed, a module is assigned to exactly one task. Furthermore, more than one module can be assigned to a task. The following definitions explain task and module hierarchy.

Definition 19. ***Correct module assignment.*** *Module M_j is assigned correctly iff it is assigned to task τ_i with period T_i such that $p_j^{min} \leq T_i \leq p_j^{max}$.*

As a task is composed of modules, its WCET depends on the sum of the WCETs of modules

assigned to it:

$$C_i = \sum_{j \in \mathscr{M}_i} c_j \tag{4.2}$$

where $\mathscr{M}_i$ is subset of $\mathscr{M}$ which contains modules assigned to task τ_i. Modules assigned to task execute sequentially when a job of a task is executing. This is depicted with pseudocode Alg. 5. Response time of a task in fixed-priority preemptive systems is given with well known equation [2]:

$$R_i = C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \tag{4.3}$$

where $hp(\tau_i)$ is set of tasks with priority higher than τ_i.

Definition 20. ***Task feasibility.*** *Task is feasible at some priority level iff it meets its deadline, i.e.,* $R_i \leq D_i$.

Based on the definitions of task feasibility and correct module assignment the notion of system feasibility can be derived.

Definition 21. ***System feasibility.*** *System S is feasible iff every module in* $\mathscr{M}$ *is assigned correctly and every task from* $\mathscr{T}$ *is feasible.*

Algorithm 5 Task concept

```
1: function TASK(M_i)
2:     for M_j in M_i do
3:         execute M_j
4:     end for
5: end function
```

Moreover, task utilization of a task is defined as ratio $U_i = \frac{C_i}{T_i}$, i.e., total system utilization is $\sum_{i=1}^{n} U_i$. From the modular system perspective total utilization can be expressed as:

$$U = \sum_{i=1}^{n} \sum_{M_i \in \mathscr{M}_i} \frac{c_j}{T_i} \tag{4.4}$$

Essentially, every module contributes to the utilization depending on the assigned period. Furthermore, since the observed systems can only have harmonic periods, feasibility can be expressed as $U \leq 1$ as it was mentioned before.

4.5 Problem formulation

4.5.1 Classical harmonic period assignment problems

Existing harmonic period assignment problems which are analyzed and discussed in the literature [62, 63, 64] are discussed in this subsection. Inputs of these problems are tasks described

with worst-case execution times C_i and period ranges $I_i = [p_i^{min}, p_i^{max}]$. The outputs are period values T_i assigned to each task. The utilization-maximizing harmonic period assignment (UHPA) problem is formulated as follows:

$$\begin{aligned}
&\text{maximize} \quad U = \sum_{i=1}^{n} U_i \\
&\text{subject to} \\
&\qquad T_i \in I_i, && i \in [1,n] \\
&\qquad \frac{T_i}{T_j} \in \mathbb{N} \quad \text{or} \quad \frac{T_j}{T_i} \in \mathbb{N}, && i,j \in [1,n] \\
&\qquad U \leq 1
\end{aligned}$$

As it can be seen, actual periods T_i are allowed to be in range I_i and have to be in harmonic relation. It is worth noting that in the related literature (e.g., [62]) period values T_i are not restricted to the integer values. However, the period ratios of the two consecutive integers $\frac{T_i}{T_j}$ are required to be integers. The second common problem formulation in the literature is the cost-minimizing harmonic period assignment (CHPA) problem which can be formulated as:

$$\begin{aligned}
&\text{minimize} \quad \sum_{i=1}^{n} w_i T_i \\
&\text{subject to} \\
&\qquad T_i \in I_i, && i \in [1,n] \\
&\qquad \frac{T_i}{T_j} \in \mathbb{N} \quad \text{or} \quad \frac{T_j}{T_i} \in \mathbb{N}, && i,j \in [1,n] \\
&\qquad U \leq 1
\end{aligned}$$

The CHPA problem is common in control co-design applications [68] where the goal function is a linear function of task periods. In the goal function, the weight w_i determines the contribution of each period to the total cost.

4.5.2 Formulation of harmonic assignment problem with a constrained number of distinct period values

In this chapter, the focus is on finding the solution to the utilization-maximizing harmonic period assignment problem with a constrained number of different period values. In the previous problems, the number of different period values in the solution is not constrained. For instance, an optimal solution can have any number of different period values, from only one up to n. As argued in the motivation for the research, the number of different period values can be smaller

than the number of tasks in the system. To address this, the problem in which the number of different period values is constrained, i.e., fixed has to be formulated. Such an approach enables a more flexible system design as it allows the system designer to regulate the number of distinct period values in the system. This is elaborated further using the numerical example in section 4.9.4. The problem can be formulated in several steps. Firstly, it is necessary to introduce the vector $\vec{p}$ which contains m different period values, where $m \leq n$. Secondly, the period assignment matrix $\mathbf{X}$, which contains mapping of period value p_j to period value T_i of task τ_i, has to be introduced. Value x_{ij} of the binary matrix $\mathbf{X}$ is determined as follows:

$$x_{ij} = \begin{cases} 1, \text{period value } p_j \text{ is assigned to task } \tau_i, \text{ i.e., } T_i \leftarrow p_j \\ 0, \text{otherwise} \end{cases} \tag{4.5}$$

In case $m = n$, one period value p_j maps to only one period value T_i, i.e., this is one to one mapping. In case $m \leq n$, period value p_j can be mapped to many tasks, i.e., generally, this is one to many mapping. However, period value p_j has to be mapped to at least one period value T_i. If p_j is not mapped to at least one task, then the number of different period values in the resulting period assignment would not be equal to m, and this is the requirement of the problem. To express this formally, the constraints given with equations (4.6) and (4.7) are introduced. First, it is necessary to restrict the assignment of only one period value p_j to period value T_i of task τ_i:

$$\sum_{j=1}^{m} x_{ij} = 1, \quad i \in [1,n] \tag{4.6}$$

The constraint (4.6) ensures that the i-th row of matrix $\mathbf{X}$ can contain only one non-zero element, i.e., only one p_j is mapped to T_i. Secondly, every period value p_j has to be assigned to at least one task:

$$\sum_{i=1}^{n} x_{ij} \geq 1, \quad j \in [1,m] \tag{4.7}$$

The constraint (4.7) ensures that the j-th column of matrix $\mathbf{X}$ has to contain at least one non-zero element, i.e., every period value p_j has to be assigned to a task. The latter constraints do not ensure the correctness of a period assignment. For instance, a period value which is too high or too low can be assigned to a task. To ensure that a period assignment is correct (Definition 18), it is necessary to restrict the assignment to period values from interval $[p_i^{min}, p_i^{max}]$. The correctness criteria can be expressed as:

$$x_{ij} \implies p_i^{min} \leq p_j \leq p_i^{max} \tag{4.8}$$

The latter logical expression can be translated to an arithmetic expression using the translation for logical implication to a linear constraint:

$$X \implies Y \rightarrow x \leq y \tag{4.9}$$

At this point, the binary constraint matrix $\mathbf{A}$ with $n \cdot m$ elements a_{ij} can be introduced. Value a_{ij} is determined as follows:

$$a_{ij} = \begin{cases} 1, p_i^{min} \leq p_j \leq p_i^{max} \\ 0, \text{otherwise} \end{cases} \tag{4.10}$$

To ensure the correctness of a period assignment, $x_{ij} \implies a_{ij}$ has to hold. Using the arithmetic counterpart for implication given with (4.9), the correctness criteria can be expressed as:

$$x_{ij} \leq a_{ij}, \quad \forall i \in [1,n], \forall j \in [1,m] \tag{4.11}$$

With the latter constraints in place, the problem can be formally expressed as:

$$\begin{aligned} &\text{maximize} \quad U = \sum_{j=1}^{m} \sum_{i=1}^{n} C_i \frac{x_{ij}}{p_j} \\ &\text{subject to} \\ &\quad x_{ij} \leq a_{ij}, && i \in [1,n], j \in [1,m] \\ &\quad \sum_{j=1}^{m} x_{ij} = 1, && i \in [1,n] \\ &\quad \sum_{i=1}^{n} x_{ij} \geq 1, && j \in [1,m] \\ &\quad p_j = k_j p_{j-1}, && k_j \in \mathbb{N}^+ \setminus \{1\}, j \in [2,m] \\ &\quad U \leq 1 \end{aligned}$$

where k_j is the integer ratio of two consecutive period values p_j and p_{j-1}. This problem is referred to as the utilization-maximizing harmonic period assignment with a constrained number of different period values (UDHPA). The outputs of the UDHPA problem are period vector $\vec{p}$ and period assignment matrix $\mathbf{X}$. These two variables determine period T_i for each task in system. Similarly as in the classical problems, the inputs of the problem are tasks represented with worst-case execution times C_i and period ranges I_i. The additional input in the UDHPA problem is the number of distinct period values in the solution m. In the more general version of the problem the utilization constraint, i.e., $U \leq 1$ is generalized. More precisely, $U \leq U_a$, where U_a represents the maximal allowed utilization of the system. This does not inflict on schedulability since U_a has to be in interval $[0,1]$.

4.6 Problem analysis: Turing reducibility and complexity

4.6.1 Turing reduction from UHPA to UDHPA

In order to show that the UHPA problem is Turing reducible to the UDHPA problem, it has to be shown that an oracle for the UDHPA problem can be used to solve the UHPA problem. Formally, this is stated with the following lemma.

Lemma 1. *The UHPA problem can be solved by solving the UDHPA problem n times.*

Proof. When the UHPA problem is considered, an optimal solution has an arbitrary number of distinct period values which can be lower than the number of tasks in the system. In the UDHPA problem, however, the number of distinct period values is fixed. Therefore, to solve the UHPA problem using an algorithm for the UDHPA problem, one has to solve the UDHPA problem for every number m of distinct period values from interval $[1,n]$. To better illustrate this, pseudocode is provided in Alg. 6. □

Algorithm 6 Turing reduction from UHPA to UDHPA

Input: $S = \mathscr{T}$
Output: U_{max} ▷ maximal utilization

```
1: function SOLVEUHPA(𝒯)
2:     U_max ← 0
3:     for m in range 1 to n do
4:         U ← SOLVEUDHPA(𝒯, m)
5:         if U > U_max then
6:             U_max ← U
7:         end if
8:     end for
9:     return U_max
10: end function
```

Theorem 3. *The UHPA problem is Turing reducible to the UDHPA problem.*

Proof. In Lemma 1, it can be seen that the UHPA problem is solved by invoking the oracle for the UDHPA problem in polynomial time, which proves the theorem. □

4.6.2 Complexity analysis

To derive the complexity of the UDHPA problem, the UHPA complexity results from [62] can be used. To show that the UHPA problem is at least weakly NP-hard, the authors provided many-one reduction from the partition sum problem (PART) to the UHPA problem. In other words, they provided a polynomial-time algorithm for reducing any given instance of the PART problem to an instance of the UHPA problem. Their proof can be directly applied to the UDHPA

problem. For completeness and clarity, important parts of the proof are produced in the continuation. For the complete proof, refer to section 3 in [62]. Firstly, the number partitioning problem (PART) has to be defined.

Definition 22. ***The PART problem***. *Let $A = \{a_1, ..., a_n\}$ be a set of n items with an associated size function $s : A \to \mathbb{N}$ which assigns a positive integer to each item. The problem is to determine whether A can be partitioned into two sets, A_1 and A_2, such that the total size of items in A_1 equals that of A_2. More formally, let S, S_1, and S_2 denote the sum of items for A, A_1, and A_2, respectively. That is,*

$$S = \sum_{a_i \in A} s(a_i) \tag{4.12}$$

$$S_1 = \sum_{a_i \in A_1} s(a_i) \tag{4.13}$$

$$S_2 = \sum_{a_i \in A_2} s(a_i) \tag{4.14}$$

Then, the problem is to decide whether A can be partitioned into A_1 and A_2 (i.e., $A_1 \cup A_2 = A$ and $A_1 \cap A_2 = \emptyset$), such that $S_1 = S_2$. An instance of this problem is said to be a positive one if such a partitioning exists [62].

The PART problem is known to be NP-complete, but solvable in pseudo-polynomial time [82].

At this point, the polynomial-time method for transforming any given instance of the PART problem to an instance of the UHPA problem is reproduced. To show that the proof is applicable to UDHPA problem as well, it has to be shown that the transformation of any PART instance to an instance of the UHPA is a transformation to an instance of the UDHPA problem as well.

Definition 23. ***PART transformation.*** *For any instance of the PART problem, the corresponding UHPA problem is specified by a set of $n+2$ tasks. The WCET of τ_i is determined as:*

$$C_i = \begin{cases} \frac{4s(a_i)}{3S+3}, 1 \leq i \leq n \\ \frac{2}{3S+3}, n+1 \leq i \leq n+2 \end{cases} \tag{4.15}$$

Period ranges for each τ_i are determined as:

$$I_i = \begin{cases} [1,2], 1 \leq i \leq n \\ [1,1], i = n+1 \\ [2,2], i = n+2 \end{cases} \tag{4.16}$$

Proposition 5. *An UHPA instance obtained using **PART transformation** is an UDHPA instance with $m = 2$.*

Proof. Any instance of the PART problem is transformed to an instance of UHPA problem with period ranges such that the allowed harmonic period values are either 1 or 2. There are always exactly two different period values in the resulting UHPA problem. Therefore, any such instance is an UDHPA instance with two different period values, i.e., $m = 2$. □

Lemma 2. *A given instance of the PART problem is positive (i.e., the given set can be partitioned) if and only if the UHPA problem instance obtained from **PART transformation** has a solution in which $U = 1$ [62].*

Proof. The latter lemma is proven in [62] and here the proof is reproduced since the proof will be further generalized later. Let T_i denote the period of task τ_i assigned in the solution to the UHPA problem. According to the specification of the problem, the periods of task τ_{n+1}, and τ_{n+2}, have to be 1 and 2, respectively. Periods of other tasks are required to be in interval $[1,2]$, thus there are only two options. In this way, two sets are formed:

$$\begin{aligned} J_1 &= \{i | T_i = 1, 1 \leq i \leq n\} \\ J_2 &= \{i | T_i = 2, 1 \leq i \leq n\} \end{aligned} \tag{4.17}$$

The total utilization that is achieved with periods defined above can be expressed as:

$$U = \sum_{i=1}^{n+2} \frac{C_i}{T_i} = \sum_{i \in J_1} C_i + \sum_{i \in J_2} \frac{C_i}{2} + C_{n+1} + \frac{C_{n+2}}{2} \tag{4.18}$$

Similarly, when C_i values are substituted as defined in (4.15), the following equations are obtained:

$$\begin{aligned} U &= \sum_{i \in J_1} \frac{4s(a_i)}{3S+3} + \frac{1}{2} \sum_{i \in J_2} \frac{4s(a_i)}{3S+3} + \frac{2}{3S+3} + \frac{1}{3S+3} \\ &= \frac{4}{3S+3} \left(\sum_{i \in J_1} s(a_i) + \frac{1}{2} \sum_{i \in J_2} s(a_1) \right) + \frac{3}{3S+3} \\ &= \frac{4}{3S+3} \left(\frac{1}{2} \sum_{i \in J_1} s(a_i) + \frac{1}{2} \sum_{i \in J_1} s(a_i) + \frac{1}{2} \sum_{i \in J_2} s(a_1) \right) + \frac{3}{3S+3} \\ &= \frac{4}{3S+3} \left(\frac{1}{2} \sum_{i \in J_1} s(a_i) + \frac{S}{2} \right) + \frac{3}{3S+3} \end{aligned} \tag{4.19}$$

Now, to show that a PART instance is positive if and only if $U = 1$:

$$1 = \frac{4}{3S+3} \left(\frac{1}{2} \sum_{i \in J_1} s(a_i) + \frac{S}{2} \right) + \frac{3}{3S+3} \tag{4.20}$$

which can be reduced to:

$$\frac{3S}{4} = \sum_{i \in J_1} \frac{s(a_i)}{2} + \frac{S}{2} \tag{4.21}$$

and finally:

$$\frac{S}{2} = \sum_{i \in J_1} s(a_i) \tag{4.22}$$

The final expression shows that when the utilization is equal to one, the corresponding PART instance is positive since there are indeed two subsets of input set A with sum that equals $\frac{S}{2}$. □

It is worth noting that Lemma 2 applies to the UDHPA problem as well since it is known from Theorem 3. that every instance of the UHPA problem obtained using **PART transformation** can be solved with an oracle for the UDHPA problem with $m = 2$, i.e., there is one to one mapping between instances of the UHPA and UDHPA problem with fixed $m = 2$.

Theorem 4. *The UDHPA problem is at least weakly NP-hard.*

Proof. Using the **PART transformation** and Lemma 2 any PART instance can be reduced to a corresponding UHPA instance. Additionally, using Proposition 5 and Theorem 3 it can be seen that this transformation is valid for the UDHPA problem as well. Therefore, any algorithm used for solving the UDHPA problem can be used for solving any instance of the PART problem after the **PART transformation**. Therefore, the UDHPA problem is at least hard as the PART problem. Moreover, the UDHPA problem is at least weakly NP-hard. □

In the remainder of the section, it is shown that a similar proof procedure can be applied for proving that the AUDHPA problem, i.e., the problem of assigning distinct harmonic periods with an arbitrary utilization, is at least NP-hard as well. Let us denote the maximal utilization with U_a. Then the transformation to the instance of the AUDHPA problem can be done with the following procedure.

Definition 24. ***PART2 transformation.*** *For any instance of the PART problem, the corresponding AUDHPA problem can be specified with task set with $n+2$ tasks with $m = 2$ distinct period values. The WCET is determined as:*

$$C_i = \begin{cases} \frac{4U_a s(a_i)}{3S+3}, 1 \leq i \leq n \\ \frac{2U_a}{3S+3}, n+1 \leq i \leq n+2 \end{cases} \tag{4.23}$$

Period ranges for each τ_i are determined with (4.16).

In comparison with **PART transformation**, **PART2 transformation** multiplies the WCET values with factor U_a.

Lemma 3. *A given instance of the PART problem is positive (i.e., the given set can be partitioned) if and only if the AUDHPA problem instance obtained from* ***PART transformation*** *has a solution in which* $U = U_a$.

Proof. Similarly as before, Let T_i denote the period of task τ_i assigned in the solution to the UHPA problem. According to the specification of the problem, the periods of task τ_{n+1}, and τ_{n+2}, have to be 1 and 2, respectively. Periods of other tasks are required to be in interval $[1,2]$, thus there are only two options. In this way, two sets are formed:

$$\begin{aligned} J_1 &= \{i | T_i = 1, 1 \leq i \leq n\} \\ J_2 &= \{i | T_i = 2, 1 \leq i \leq n\} \end{aligned} \tag{4.24}$$

The total utilization that is achieved with periods defined above can be expressed as:

$$U = \sum_{i=1}^{n+2} \frac{C_i}{T_i} = \sum_{i \in J_1} C_i + \sum_{i \in J_2} \frac{C_i}{2} + C_{n+1} + \frac{C_{n+2}}{2} \tag{4.25}$$

Similarly, when C_i values are substituted as defined in (4.23), the following equations are obtained:

$$\begin{aligned} U &= \sum_{i \in J_1} \frac{4U_a s(a_i)}{3S+3} + \frac{1}{2} \sum_{i \in J_2} \frac{4U_a s(a_i)}{3S+3} + \frac{2U_a}{3S+3} + \frac{U_a}{3S+3} \\ &= \frac{4U_a}{3S+3} \left(\sum_{i \in J_1} s(a_i) + \frac{1}{2} \sum_{i \in J_2} s(a_1) \right) + \frac{3U_a}{3S+3} \\ &= \frac{4U_a}{3S+3} \left(\frac{1}{2} \sum_{i \in J_1} s(a_i) + \frac{1}{2} \sum_{i \in J_1} s(a_i) + \frac{1}{2} \sum_{i \in J_2} s(a_1) \right) + \frac{3U_a}{3S+3} \\ &= \frac{4U_a}{3S+3} \left(\frac{1}{2} \sum_{i \in J_1} s(a_i) + \frac{S}{2} \right) + \frac{3U_a}{3S+3} \end{aligned} \tag{4.26}$$

Now, to show that a PART instance is positive if and only if $U = U_a$.

$$U_a = \frac{4U_a}{3S+3} \left(\frac{1}{2} \sum_{i \in J_1} s(a_i) + \frac{S}{2} \right) + \frac{3U_a}{3S+3} \tag{4.27}$$

Note that when U_a cancels out the latter expression corresponds to equation (4.20). Similarly, this can be reduced to:

$$\frac{3S}{4} = \sum_{i \in J_1} \frac{s(a_i)}{2} + \frac{S}{2} \tag{4.28}$$

and finally:

$$\frac{S}{2} = \sum_{i \in J_1} s(a_i) \tag{4.29}$$

The final expression shows that when the utilization is equal to U_a, the corresponding PART

instance is positive since there are indeed two subsets of input set A with sum that equals $\frac{S}{2}$. □

Theorem 5. *The AUDHPA problem is at least weakly NP-hard.*

Proof. Using the **PART2 transformation** and Lemma 3 any PART instance can be reduced to a corresponding AUDHPA instance. Therefore, any algorithm used for solving the AUDHPA problem can be used for solving any instance of the PART problem after the **PART2 transformation** that is computed in polynomial time. Therefore, the AUDHPA problem is at least hard as the PART problem. More precisely, the AUDHPA problem is at least weakly NP-hard. □

4.7 Existing suboptimal period assignment approaches from period ranges

In this section, existing suboptimal approaches for the UHPA problem are discussed. These approaches are based on the harmonic projection model [63, 64], which is discussed in the next subsection.

4.7.1 Harmonic projection model

In [63], authors introduce the notion of the projected harmonic zone, which is useful since it enables easier manipulation with the specified period intervals I_i for each task. In the following definitions, the period interval I_i is specified with two values, I_i^s that is the lowest value of the interval, and I_i^e that is the highest value of the interval. Projected harmonic zone can be defined as follows.

Definition 25. *The projected harmonic zone* $\chi^a_{I_1 \to I_2} : [\chi^s, \chi^e]$ *from interval* I_1 *to* I_2, $I_1^s \leq I_2^s$, *with multiplier* $a \in \mathbb{N}^+$, *is a range of numbers in* I_2 *that starts from* $\chi^s = \max\{I_2^s, aI_1^s\}$ *and ends to* $\chi^e = \min\{I_2^e, aI_1^e\}$, *and for any* $i_2 \in I_2$ *there exists at least one* $i_1 \in I_1$ *such that* $\frac{i_2}{i_1} \in \mathbb{N}$.

In addition, authors in [63] prove that all possible multipliers a are given with the following definition.

Definition 26. *For two intervals* I_1 *and* I_2, $I_1^s \leq I_2^s$, *the set of harmonic multipliers is defined as* $A_{I_1 \to I_2} = \{a_1, a_2, ..., a_z\}$ *where* $a_1 = \lfloor \frac{I_2^s}{I_1^e} \rfloor + 1, a_2 = a_1 + 1, a_3 = a_2 + 1, ..., a_z = \lfloor \frac{I_2^e}{I_1^s} \rfloor$. *If* $\lfloor \frac{I_2^s}{I_1^e} \rfloor \in \mathbb{N}$, *then* $a_1 = \lfloor \frac{I_2^s}{I_1^e} \rfloor$.

Intuitively, multipliers from set $A_{I_1 \to I_2}$ project the interval I_1 onto I_2 and in overlap of these two intervals harmonically related value $i_2 \in I_2$ is harmonically related to value $i_1 \in I_1$. By enumerating combinations of the multipliers for each specified pairs of intervals, i.e., I_1 and I_2, I_2 and I_3, projected harmonic zones can be found such that values $\frac{i_2}{i_1} \in \mathbb{N}, \frac{i_3}{i_2} \in \mathbb{N}, ..., \frac{i_n}{i_{n-1}} \in \mathbb{N}$ exist.

Forward search period assignment

In order to make the enumeration more efficient, authors in [63] propose that the projected harmonic zones, which overlap in the sense that the end of the first zone is higher than the beginning of the second zone, are combined into a single continuous projected harmonic zone. Moreover, they show that for multipliers such that:

$$a_x \geq \frac{I_1^s}{I_1^e - I_1^s} \tag{4.30}$$

all projected harmonic zones produced by a_x to a_z can be represented by a single projected harmonic zone, i.e., $\chi_{I_1 \to I_2}^{\{a_x, a_{x+1}, ..., a_z\}} = [a_x I_1^s, \min\{a_z I_1^e, I_2^e\}]$. Enumeration of the possible projected harmonic zones can be done using the Alg. 7. Alg. 7 constructs a graph in which vertices represent projected harmonic zones and edges represent harmonic multipliers that connect them. The resulting graph contains all possible connections between the projected harmonic zones. Moreover, each path through the graph structure which covers n vertices verifies the existence of a harmonic relation. For any such path, the result is a sequence of intervals $I : \{I_1, I_2, ..., I_n\}$. This approach is referred to as *forward search* since it starts with depth first search of harmonic projection zones from lower to higher period values.

Note that the solution generated by Alg. 7 is not the solution to the UHPA problem since intervals are produced rather than exact values of periods. To generate periods, authors propose using the low-utilization period assignment algorithm shown in Alg. 8 which will guarantee feasibility of the solution, but not the optimality.

As it can be seen the main setbacks of this approach is inability to tune the number of different period values and to ensure that the solution is optimal.

Backward search period assignment

In [64], a similar approach was proposed. The main difference is in the introduction of the notion of projection origin, which is similar to the projected harmonic zone. In this approach, the method starts from the last interval and attempts to find origins in the previous intervals. The motivation for such an approach is in the problem of transitivity of harmonic relation between intervals, i.e., if I_1 and I_2, are harmonically related and I_2 and I_3 are harmonically related that does not imply that I_1 and I_3 are harmonically related. This causes potential exponential complexity of the forward approach when all the harmonic projections are disjoint. Projection origins can be defined as follows.

Definition 27. *Projection origins of interval I_i from interval I_{i+1} ($I_i^s < I_{i+1}^s$) are denoted by $\psi_i = \{\psi_{i,1}, \psi_{i,2}, ..., \psi_{i,hi}\}$ where h_i is the number of sub-intervals inside ψ_i and have following*

Algorithm 7 Graph construction algorithm

Input: $\mathscr{T}, G, I_{v_{i,j}}$
Output: G

if $i = n$ or $T_{i+1}^{max} < I_{v_{i,j}}^{s}$ **then**
  return G
else
  $s \leftarrow \max\{T_{i+1}^{min}, I_{v_{i,j}}^{s}\}$
  $a_1 \leftarrow \left\lfloor \frac{T_{i+1}^{max}}{I_{v_{i,j}}^{s}} \right\rfloor$
  $a_z \leftarrow \left\lfloor \frac{s}{I_{v_{i,j}}^{e} - I_{v_{i,j}}^{s}} \right\rfloor$
  $m \leftarrow a_x - a_1$
  $a \leftarrow a_1$
  for $k = 1$ to m **do**
    $I_{v_{i,k}} \leftarrow [\max\{aI_{v_{i,j}}^{s}, s\}, \min\{aI_{v_{i,j}}^{e}, T_{i+1}^{max}\}]$
    $q = k + K_i$
    add vertex $v_{i+1,q}$
    add an edge between $v_{i,j}$ and $v_{i+1,q}$ to G
    $GCA(\tau, G, I_{v_{i,q}})$
    $a \leftarrow a + 1$
  end for
  $I_{v_{i,m+1}} \leftarrow [\max\{aI_{v_{i,j}}^{s}, s\}, \min\{a_z I_{v_{i,j}}^{e}, T_{i+1}^{max}\}]$
  $q = m + 1 + K_i$
  add vertex $v_{i+1,q}$
  add an edge between $v_{i,j}$ and $v_{i+1,q}$ to G
  $GCA(\tau, G, I_{v_{i,q}})$
end if

Algorithm 8 Low utilization period assignment algorithm

Input: I
Output: T

$T_n \leftarrow I_n^e$
for $i \leftarrow n-1$ down to 1 **do**
  $b_i \leftarrow \left\lfloor \frac{T_{i+1}}{I_i^e} \right\rfloor$
  $T_i \leftarrow \frac{T_{i+1}}{b_i}$
end for
return T

properties:

$$\forall x \in \psi_i, \exists y \in I_{i+1}; \quad \textit{such that} \quad \frac{y}{x} \in \mathbb{N} \tag{4.31}$$

$$\forall j,k \in 1,2,...,h_i, j \neq k; \quad \psi_{i,j} \cap \psi_{i,k} = \emptyset \tag{4.32}$$

$$\forall j, 1 \leq j \leq h_i : I_i^s \leq \psi_{i,j}^s \quad \textit{and} \quad \psi_{i,j}^e \leq I_i^e \tag{4.33}$$

The most important observation from [64], which is key to reducing the computational complexity, is that there at maximum two different non-overlapping projection origins in any ψ_i, i.e, $h_i \leq 2$. The algorithm is depicted in Alg. 9. As it can be seen the algorithm starts from the back, i.e., the last period interval, and in the first step finds the projection origins for I_{n-1}. In subsequent steps, projection origins for other intervals are determined. If at any step origins set ψ_i contains more than 2 sub-intervals, there are sub-intervals in ψ_i, which have to be merged (see line 15 in Alg. 9). On the other hand if at any step ψ_i is empty, there is no harmonic period set for given intervals. Moreover, this algorithm represents a necessary and sufficient condition for existence of harmonic period set as it is proven in [64].

Algorithm 9 Existence test for harmonic period assignment

Input: I - set of intervals
Output: {yes, no}, ψ - feasibility and set of origins

1: $\psi_n \leftarrow I_n$
2: $\psi \leftarrow \{\psi_n\}$
3: **for** $i \leftarrow n-1$ down to 1 **do**
4: $\quad \psi_i \leftarrow \emptyset$
5: $\quad$ **for** $j \leftarrow 1$ to h_{i+1} **do**
6: $\quad\quad$ calculate a_s and a_e according to Definition 26
7: $\quad\quad$ **if** $(a^s+1)I_i^e \leq \psi_{i+1,j}^e$ **then**
8: $\quad\quad\quad \psi_i \leftarrow I_i$
9: $\quad\quad\quad$ break
10: $\quad\quad$ **end if**
11: $\quad\quad$ **if** $a^s \leq a^e$ **then**
12: $\quad\quad\quad \psi \leftarrow \psi \cup [a^s I_i^s, I_i^e] \cup [a^e I_i^s, I_i^e]$
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: $\quad$ merge intervals in ψ_i so that (4.32) holds
16: $\quad$ **if** $\psi_i = \emptyset$ **then**
17: $\quad\quad$ **return** no
18: $\quad$ **end if**
19: $\quad \psi \leftarrow \psi \cup \{\psi_i\}$
20: **end for**
21: **return** yes, ψ

Similarly as in the forward search, the obtained intervals do not correspond to the solution of the UHPA problem. Again, a heuristic algorithm is used for period assignment. In [64],

authors propose an algorithm, which is efficient, but does not ensure that the solution is feasible in the sense of utilization, i.e., $U \leq 1$. The idea is similar to the algorithm depicted in the previous subsection, i.e., to assign the largest possible period to tasks to ensure that the obtained utilization is minimal. The algorithm is depicted in Alg. 10.

Algorithm 10 Heuristic period assignment algorithm

Input: ψ
Output: T

```
1: T_1 ← ψ^e_{1,h_1}
2: for i ← 2 to n do
3:     for j ← h_i down to 1 do
4:         a^e ← ⌊ψ_{i,j} / T_{i-1}⌋
5:         if a^e T_{i-1} ≥ ψ^s_{i,j} then
6:             T_i ← a^e T_{i-1}
7:             break
8:         end if
9:     end for
10: end for
11: return T
```

Similarly, as in the forward approach, this algorithm does not guarantee optimality with regard to the utilization. Moreover, it does not enable control over the number of different period values in the system.

4.8 An optimal algorithm for the UDHPA problem

The UDHPA problem cannot be easily solved by using existing mixed-integer or integer programming solvers. The utilization of the system, which is the goal function, is a non-linear, i.e., signomial, function. Methods for solving mixed-integer signomial problems exist, but do not guarantee to find a global solution [83].

In the approach proposed in this research, possible solutions are enumerated to find an optimal solution of the problem. The UDHPA problem can be split into two independent parts:

1. enumeration of potential harmonic period sets – this part is referred to as period enumeration (PE),
2. assignment of periods from a harmonic period set to tasks - this part is referred to as task assignment (TA).

In the first part, the possible harmonic period sets, which can be used for task assignment in the system, are enumerated. This is possible since the periods are constrained to be integer values. Still, in the worst case, enumerating all the possible harmonic period sets can lead to a combinatorial explosion due to the exponential growth in the period search space. Therefore, several propositions are introduced which drastically reduce search space in most use cases.

In the second part, it is assumed that harmonic period set $\vec{p}$ is known. It can be seen that the goal function of the UDHPA problem with known period values is linear, which makes such problem an integer linear program, i.e., zero-one linear program. The relaxed version of the UDHPA problem, i.e., an assignment problem with a known harmonic period set, is referred to as the task assignment (TA) problem. The TA problem can be expressed as follows:

$$\text{maximize} \quad U = \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{C_i}{p_j} x_{ij}$$

subject to

$$\begin{aligned} & x_{ij} \leq a_{ij}, && i \in [1,n], j \in [1,m] \\ & \sum_{j=1}^{m} x_{ij} = 1, && i \in [1,n] \\ & \sum_{i=1}^{n} x_{ij} \geq 1, && j \in [1,m] \\ & U \leq 1 \end{aligned}$$

In the TA problem, only the mapping of period values to tasks has to be determined since harmonic period set $\vec{p}$ is known in advance. In the devised approach, this observation is exploited. Firstly, possible harmonic period values sets are enumerated, and then the TA problem is solved for each enumerated harmonic period set.

4.8.1 Enumerating period values

The first step in determining m different period values is the choice of the value for the lowest period, i.e., p_1. Subsequent period values are determined by choosing the integer ratios $k_j > 1$ of two consecutive integer values p_j and p_{j-1}. Possible values for the first period depend on two specific values which can be determined from task period ranges. The first value is p_{min}^{min}, which is the minimal lower bound in all period ranges, i.e., $\min p_i^{min}$. The second value is p_{min}^{max}, which is the minimal upper bound in all period ranges, i.e., $\min p_i^{max}$.

Proposition 6. ***Choice of*** p_1**.** *In any feasible solution of the UDHPA problem, value of* p_1 *is in interval* $[p_{min}^{min}, p_{min}^{max}]$.

Proof. To prove this, it is necessary to consider cases in which p_1 is not in the proposed interval and to show that at least one of the constraints is violated. Firstly, let us assume that p_1 has a value which is lower than p_{min}^{min}. As the period value has to be assigned to at least one task, i.e., $\sum_{i=1}^{n} x_{ij} \geq 1, j \in [1,m]$, there is no feasible solution to the UDHPA problem since period value p_1 cannot be assigned to task τ_i with the minimal lower bound $p_i^{min} = p_{min}^{min}$, or to any other task. Secondly, if p_1 is greater than the minimal upper bound p_{min}^{max}, it is not possible to

assign any period value to task τ_i with $p_i^{max} = p_{min}^{max}$ due to the violation of the period range constraint. Therefore, for any feasible solution to the UDHPA problem, p_1 is within the interval $[p_{min}^{min}, p_{min}^{max}]$. □

Similarly, as the possible choices of p_1 are restricted, the choice of any subsequent period values p_j can be restricted as well. This can be done by introducing another specific value for the given period ranges, p_{max}^{max}, which denotes the maximal upper bound among the upper bounds of all tasks, i.e., $\max p_i^{max}$.

Proposition 7. ***A bound on choice of*** p_j**.** *In any feasible solution, there is no period value such that* $p_j > p_{max}^{max}$.

Proof. If $p_j > p_{max}^{max}$, it is not possible to assign p_j to any task since the range constraints will be violated. Therefore, in any feasible solution, every period p_j is less than or equal to p_{max}^{max}. □

Using the specific values obtained from period ranges, it is possible to determine the maximum number of distinct period values which can appear in the solution. The following proposition is useful as it restricts the period enumeration search space.

Proposition 8. ***The maximum number of distinct period values.*** *In any feasible solution the maximum number of distinct period values m is such that* $m \leq m_{max} = \lfloor \log_2 \frac{p_{max}^{max}}{p_{min}^{min}} + 1 \rfloor$.

Proof. Due to the harmonic relations of period values, it is evident that the smallest value of the largest period, i.e., p_m, is obtained when all consecutive integer ratios are such that $k_j = 2, j \in [1, m-1]$. Then, we have $p_m^{min} = p_{min}^{min} \cdot 2^{m-1}$. From Proposition 7, we know that $p_m^{min} \leq p_{max}^{max}$, and therefore $p_{min}^{min} \cdot 2^{m-1} \leq p_{max}^{max}$. When we solve the latter inequality for m, we get $m \leq \log_2 \frac{p_{max}^{max}}{p_{min}^{min}} + 1$, which proves the proposition. □

Proposition 9. ***Maximum integer factor*** k_j**.** *In any feasible solution, every integer factor* k_j *is such that* $k_j \leq \lfloor \frac{p_{max}^{max}}{p_{j-1} \cdot 2^{m-j}} \rfloor$.

Proof. Let $p_m \leq p_{max}^{max}$ be the last value of the period vector $\vec{p}$. Period p_m can be calculated as $p_m = p_1 \cdot k_2 \cdot k_3 \cdot \ldots \cdot k_j \cdot \ldots \cdot k_m$. Moreover, $p_m = p_{j-1} \cdot k_j \cdot k_{j+1} \cdot \ldots \cdot k_m$. Therefore, $k_j = \lfloor \frac{p_m}{p_{j-1} \cdot \prod_{i=j+1}^{m} k_i} \rfloor$. Factor k_j is maximal when the product $\prod_{i=j+1}^{m} k_i$ is minimal, i.e., $k_i = 2, i > j$. Therefore, $k_j \leq \lfloor \frac{p_m}{p_{j-1} \cdot 2^{m-j}} \rfloor$, which proves the proposition. □

Using the latter propositions, harmonic period sets can be enumerated in an efficient manner. Alg. 11 depicts a recursive algorithm for the enumeration of period sets. Firstly, the `Period Enumeration` function assigns values determined by Proposition 6 to the first period p_1 (line 6 in Alg. 11). The subsequent period values are determined according to Proposition 9 in a recursive manner (line 15 in Alg. 11). In the basic case, when all the period values are set to their respective values, i.e., when $j == m+1$, the TA problem is solved for the constructed

Algorithm 11 Algorithm for harmonic period enumeration

Input: $S = (\mathscr{T})$

1: **function** PERIOD ENUMERATION($\mathscr{T}$, m)
2: $p_{min}^{min} = \min p_i^{min}, \quad \forall \tau_i \in \mathscr{T}$
3: $p_{min}^{max} = \min p_i^{max}, \quad \forall \tau_i \in \mathscr{T}$
4: $p_{max}^{max} = \max p_i^{max}, \quad \forall \tau_i \in \mathscr{T}$
5: **for** i in range p_{min}^{min} to p_{min}^{max} **do**
6: $p_1 \leftarrow i$ ▷ according to Prop. 6
7: PERIOD ENUMERATION STEP($\vec{p}$, p_{max}^{max}, 2)
8: **end for**
9: **end function**
10: **function** PERIOD ENUMERATION STEP($\vec{p}$, p_{max}^{max}, j)
11: **if** j ==m+1 **then**
12: SOLVE TASK ASSIGNMENT($\vec{p}$, $\mathscr{T}$)
13: **return**
14: **end if**
15: **for** k_j in range 2 to $\lfloor \frac{p_{max}^{max}}{p_j \cdot 2^{m-j}} \rfloor$ **do** ▷ according to Prop. 9
16: $p_j \leftarrow k_j \cdot p_{j-1}$
17: PERIOD ENUMERATION STEP($\vec{p}$, p_{max}^{max}, $j+1$)
18: **end for**
19: **end function**

period set. Moreover, the number of different harmonic period sets with m distinct period values corresponds to the number in which the basic case is reached. The number of solutions with regard to m is further analyzed in section 4.9 in the context of feasibility evaluation. Now, time complexity of the period enumeration algorithm is analyzed. Firstly, it is worth noting that the asymptotic analysis with regard to m is not of any interest since it is known from Proposition 8 that m is bounded and that there are no feasible solutions for higher values of m. Therefore, time complexity is analyzed with regard to the highest period in the input p_{max}^{max}, as it is obvious that the number of steps in the algorithm increases when p_{max}^{max} increases (see loop bound in line 15 in Alg. 11). To provide an asymptotic upper bound on the time complexity of Alg. 11, a similar enumeration problem referred to as the DIVENUM problem can be observed.

Definition 28. ***The DIVENUM problem.*** *The enumeration problem, which is referred to as DIVENUM, is to output all m-tuples* $(k_1, ..., k_m)$ *such that:*

$$\Pi_{j=1}^{m} k_j \leq \chi, \quad k_j \in \mathbb{N}^+, \forall j \tag{4.34}$$

We can see that the DIVENUM problem is in fact very similar to the PE problem since in both problems we are looking for a set of m factors such that their product is lower than the specified bound, χ and p_{max}^{max}, respectively. In the PE problem, each factor k_j is greater than one. On the other hand, in the DIVENUM problem, k_j is a positive integer including one. Therefore, we know that the number of steps required to enumerate solutions to the DIVENUM problem is always higher than the number of steps in the PE problem. For the sake of completeness and

clarity, Alg. 12 is provided that depicts the enumeration for the DIVENUM problem. Moreover, the time complexity of the Alg. 12 is analyzed, and the obtained result will serve as an upper bound of the time complexity of the PE algorithm (Alg. 11).

Algorithm 12 Algorithm for the DIVENUM problem

Input: χ, j

```
1: function DIVISOR ENUMERATION(χ, j)
2:     if j == 0 then
3:         output tuple (k_1, ..., k_m)
4:         return
5:     end if
6:     for k_{m+1-j} in range 1 to χ do
7:         DIVISOR ENUMERATION(χ / k_{m+1-j}, j − 1)
8:     end for
9: end function
```

In Alg. 12, we see that the number of steps $T(\chi, m)$ required to enumerate all m-tuples of positive integers with product less than or equal to χ is given with:

$$T(\chi, m) = \sum_{k=1}^{\chi} T\left(\left\lfloor \frac{\chi}{k} \right\rfloor, m-1\right) \tag{4.35}$$

$$T(\chi, 0) = 1 \tag{4.36}$$

where $T(\chi, 0)$ is the number of elementary operations in the basic case. For any bound x, we know that:

$$T(x, 1) = \sum_{k=1}^{x} T(x, 0) = x \tag{4.37}$$

Moreover, for $T(x, 2)$ we have the following:

$$\begin{aligned} T(x,2) &= T\left(\frac{x}{1},1\right) + T\left(\frac{x}{2},1\right) + T\left(\frac{x}{3},1\right) + \ldots + T\left(\frac{x}{x},1\right) \\ &= \frac{x}{1} \qquad + \frac{x}{2} \qquad + \frac{x}{3} \qquad + \ldots + \frac{x}{x} \end{aligned}$$

We can see that this is in fact a finite partial sum of the harmonic series:

$$T(x, 2) = x \sum_{k=1}^{x} \frac{1}{k} = xH_x \tag{4.38}$$

where $H_x = \sum_{k=1}^{x} \frac{1}{k}$ is the x-th harmonic number. Now, with $T(x, 2) = xH_x$, $T(x, 3)$ can be

expressed as:

$$\begin{aligned} T(x,3) &= T\left(\frac{x}{1},2\right) + T\left(\frac{x}{2},2\right) + T\left(\frac{x}{3},2\right) + ... + T\left(\frac{x}{x},2\right) \\ &= xH_x \quad + \frac{x}{2}H_{x/2} \quad + \frac{x}{3}H_{x/3} \quad + ... + \frac{x}{x}H_1 \end{aligned}$$

Next, we can bound $T(x,3)$:

$$\begin{aligned} T(x,3) &= xH_x \quad + \frac{x}{2}H_{x/2} + \frac{x}{3}H_{x/3} + ... + \frac{x}{x}H_1 \\ &\leq xH_x\left(1 + \frac{1}{2} \quad + \frac{1}{3} \quad + ... + \frac{1}{x}\right) \\ &= xH_x^2 \end{aligned}$$

By induction, we get that $T(\chi,m) = \chi H_\chi^{m-1}$. Moreover, harmonic numbers can be approximated with an integral:

$$H_\chi = \int_1^\chi \frac{1}{t}dt = \ln\chi \tag{4.39}$$

Therefore, an upper bound on the time complexity for the DIVENUM problem is given with $O(\chi \log^{m-1}(\chi))$. Moreover, the time complexity of the PE algorithm is in $O(p_{max}^{max} \log^{m-1}(p_{max}^{max}))$.

4.8.2 Solving the TA problem

The TA problem is solved by enumeration of all possible period to task assignments with respect to the given harmonic period set. First off, polynomial-time algorithms which yield the lower and upper bounds for the goal function, i.e., the utilization of the system, are devised. Then, by using these bounds, an algorithm for the optimal task assignment is devised.

Bound algorithms

In order to determine the bounds, constraints $\sum_{i=1}^{n} x_{ij} \geq 1, j \in [1,m]$ can be relaxed to allow that some period values remain unused in the solution. This problem is referred to as TA*. In such a scenario, an algorithm that produces a lower bound of the utilization assigns the highest correct period to every task (note: correct with respect to Definition 18). This assignment is referred to as the HPF (highest period first) assignment, and it is depicted with Alg. 13. Similarly, the LPF (lowest period first) assignment yields an upper bound of the TA* problem. To obtain the LPF assignment using the Alg. 13, period values $\vec{p}$ have to be iterated from the lowest to the highest period value (line 6 in Alg. 13). Formally, the properties of the HPF assignment are stated in the continuation (Proposition 10 and 11). The properties of the LPF assignment are analogous with respect to an upper bound of utilization.

Algorithm 13 HPF algorithm

Input: $\vec{p}, \mathscr{T}$
Output: (correctness, U, $\mathbf{X}$) - correctness of assignment, utilization and period assignment matrix

```
1: function HPF ASSIGNMENT(p⃗, 𝒯)
2:     X ← [0], n ← |𝒯|, m ← |p⃗|
3:     U ← 0, correctness ← true
4:     for i in range 1 to n do
5:         assigned ← false
6:         for j in range m down to 1 do
7:             if p_i^min ≤ p_j ≤ p_i^max then
8:                 x_ij ← 1
9:                 U ← U + C_i/p_j
10:                assigned ← true
11:                break
12:            end if
13:        end for
14:        if not assigned then
15:            correctness ← false
16:            break
17:        end if
18:    end for
19:    return (correctness, U, X)
20: end function
```

Proposition 10. *The highest period first (HPF) assignment yields the tight lower bound for the TA* problem.*

Proof. Without loss of generality, we observe one task τ_i from task set $\mathscr{T}$ and two correct periods p_j and p_k such that $p_j < p_k$. Assume that U_0 is utilization of task set $\mathscr{T}$ without τ_i and that to each task a period value is assigned correctly. We observe two possible period assignments for τ_i. In the first case, $T_i \leftarrow p_j$, and system utilization is $U = U_0 + \frac{C_i}{p_j}$. In the second case, $T_i \leftarrow p_k$, and system utilization is $U' = U_0 + \frac{C_i}{p_k}$. As $p_j < p_k$, it follows that $\frac{C_i}{p_k} < \frac{C_i}{p_j}$, and consequently $U' < U$. Therefore, assignment to higher period p_k for each τ_i will yield a lower bound of utilization. Moreover, this lower bound is tight. □

Proposition 11. *The highest period first (HPF) period assignment yields a lower bound of utilization for the TA problem.*

Proof. Proposition 10 guarantees that the HPF will yield minimal utilization in the case when there are no restrictions on the number of the distinct period values which have to appear in the solution. The HPF algorithm tries to assign the highest period value to each task, but this is not possible if we have restriction on the number of distinct period values as all of the period values have to be used. Therefore, if the highest correct period value cannot be assigned to the task, the lower period value will be assigned to the task and consequently utilization will increase. Therefore, the HPF assignment yields a lower bound of the utilization for the TA problem. However, in this case, this bound may not be tight. □

Both, the HPF and the LPF, have polynomial-time complexity, which is evident from Alg. 13. For each task in a task set, i.e., in n steps, the highest or the lowest period is chosen in at most m steps. Therefore, the time complexity of these algorithms is $O(n \cdot m)$. Note that these algorithms are not suitable for solving UDHPA instances as they do not guarantee that the number of used values in the solution will be equal to m, i.e., some period values from $\vec{p}$ may remain unused. However, in cases when there is restriction only on the maximal number of period values, i.e., solution can have any number of period values from 1 to m_{max}, usage of these algorithms is appropriate. Moreover, usage of these algorithms is appropriate for UHPA instances as in the UHPA problem there are no constraints on the number of period values. In such cases, period enumeration is firstly used to find appropriate period sets, and HPF or LPF approach to find corresponding task assignments. Time complexity of the approach is pseudo-polynomial with regard to p_{max}^{max} as time complexity of PE algorithm is in $O(p_{max}^{max} \log^{m-1}(p_{max}^{max}))$, and polynomial regarding n, as time complexity of HPF and LPF is in $O(m \cdot n)$

The optimal task assignment algorithm

An optimal algorithm for enumeration of task assignments is depicted with Alg. 14. This algorithm is referred to as optimal task assignment (OTA). Prior to explaining the optimal task assignment algorithm, it is necessary to explain how the bounds calculated by the HPF and the LPF algorithm are used. Moreover, a mechanism for tracking the number of distinct period values during the enumeration process has to be explained as well.

Bounds of utilization. Using the HPF and the LPF assignment, the vector of lower bounds $\vec{b^l}$ and the vector of upper bounds $\vec{b^u}$ is constructed. These vectors are used to prune infeasible or suboptimal branches in the enumeration of task assignments. These vectors contain upper and lower bounds of subsets of task set $\mathscr{T}$. In this context, the i-th subset of task set $\mathscr{T}$ is set $\mathscr{T}_i = \{\tau_i, \tau_{i+1}, ..., \tau_n\}$. Therefore, the i-th value of vectors $\vec{b^l}$ and $\vec{b^u}$ can be expressed as:

$$b_i^l = U_{HPF}(\mathscr{T}_i), \quad b_i^u = U_{LPF}(\mathscr{T}_i), \quad i = 1, ..., n \tag{4.40}$$

$$b_{n+1}^l = 0, \qquad b_{n+1}^u = 0 \tag{4.41}$$

where $U_{HPF}(\mathscr{T}_i)$ corresponds to the utilization obtained by the HPF assignment for task set $\mathscr{T}_i$. Similarly, $U_{LPF}(\mathscr{T}_i)$ corresponds to the utilization obtained by the LPF assignment for task set $\mathscr{T}_i$. It is worth noting that the first values of both vectors, namely b_1^l and b_1^u, correspond to the lower bound and the upper bound of task set $\mathscr{T}$, i.e., $\mathscr{T} = \mathscr{T}_1$. Values $b_{n+1}^l = 0$ and $b_{n+1}^l = 0$ are introduced for valid comparison in the last step of recursion (see line 28 in Alg. 14).

Usage of every period in the assignment. To ensure that every period value p_j is used in a task assignment at least once, the number of used period values when constructing a task assignment has to be tracked. Therefore, the binary vector $\vec{l}$ with values l_j is introduced such

that:

$$l_j = \begin{cases} 1, \text{period value } p_j \text{ is assigned at least once} \\ 0, \text{otherwise} \end{cases} \tag{4.42}$$

Additionally, d is the number of currently assigned period values, i.e., $d = \sum_{j=1}^{m} l_j$. As all of the period values have to be used at least once, in a valid task assignment d must be equal to m.

Here follows a detailed explanation of the OTA algorithm depicted with Alg. 14. For brevity and ease of representation, variables U_{max}, $\mathbf{X}_{max}$, $\mathbf{X}$, $\vec{l_j}$, and d, flag `feasible`, and bounds $\vec{b^l}$, $\vec{b^u}$ are assumed to be global. U_{max} is the current maximal value obtained for assignment matrix $\mathbf{X}_{max}$, and flag `feasible` indicates the feasibility of the problem. Global variable $\mathbf{X}$ is current assignment matrix. The local variable u represents utilization at step i.

In the first part of the algorithm, i.e., `Task Assignment` function, the HPF assignment is used to determine the lower bound of utilization prior to enumerating all task assignments (lines 2 to 8 in Alg. 14). If the obtained lower bound is greater than one or the assignment is not correct, there is no need for enumeration of task assignments (line 6 in Alg. 14). In this way, a lot of period sets for which the task assignment is infeasible are efficiently pruned.

In the second part of the algorithm, i.e., `Task Assignment Step` function, task assignments are enumerated in a recursive manner. In the basic case (lines 12 to 18), if the utilization of the current task assignment u is larger than the current maximal value U_{max}, and the solution $\mathbf{X}_{max}$ is updated accordingly. In other cases, period values are conditionally assigned to the tasks (lines 21 to 36).

At the beginning of the loop, it is checked if the value p_j is assigned to any task at the previous recursion steps. If l_j is 0 at step i, value p_j is not assigned to any task τ_k such that $k < i$. On the other hand, if value p_j is used for the first time at step i, values of l_j and auxiliary variable ul are set to 1 (lines 22 to 25). The auxiliary variable ul keeps track of "locking" period value p_j at step i. Therefore, at the end of the loop (lines 32 to 35), if p_j was used for the first time at step i, l_j and ul have to be reset ("released").

Next, two groups of conditions in their respective `if` statements (line 26 and 28) can be analyzed separately. The first `if` statement checks validity of assignment. The first condition, i.e., $p_i^{min} \leq p_j \leq p_i^{max}$ corresponds to the correctness criteria (Definition 18). In the second condition, $m-d$ is the number of unused periods from the input period set $\vec{p}$, and $n-i$ is the number of tasks to which the period is not assigned. The condition requires that the number of unused periods is less than or equal to the number of tasks to which a period value is not assigned. In other words, if it is not possible to assign every period value in the next recursion steps, p_j cannot be assigned to τ_i. In line 27, period p_j is assigned to task τ_i, i.e., utilization for the next recursion step is incremented by $\frac{C_i}{p_j}$.

The second `if` statement (line 28) serves to test the feasibility and the bounds of the assignment. Condition $u' \leq 1$ ensures feasibility of the assignment. The second condition, i.e.,

$u' + b^l_{i+1} \leq 1$, checks if the sum of the current utilization and minimal utilization of task subset $\mathscr{T}_{i+1}$ is less than or equal to one. If this condition is false, we know that there is no assignment for which the final utilization will be less than one, because b^l_{i+1} is a lower bound. Similarly, the third condition, i.e., $u' + b^u_{i+1} \geq U_{max}$, checks if the sum of the current utilization and maximal utilization of task subset $\mathscr{T}_{i+1}$ is greater than or equal to current maximal value U_{max}. If this condition is not true, we know that there is no assignment for which the final utilization is greater than the current maximal value U_{max}, because b^u_{i+1} is an upper bound.

The time complexity of the OTA algorithm is evidently exponential in the number of tasks n. In the worst-case, when bounds are ineffective, one of the m period values will be assigned to each task. Therefore, the time complexity of the OTA algorithm is in $O(m^n)$. To find an optimal solution to an UDHPA instance, we have to use the OTA algorithm for each period set obtained using the PE algorithm. Complexity of such an approach is again pseudo-polynomial with regard to p^{max}_{max} since the time complexity of the PE algorithm is in $O(p^{max}_{max} \log^{m-1}(p^{max}_{max}))$, and exponential with regard to n since the time complexity of the OTA algorithm is in $O(m^n)$.

Application of the optimal algorithm to AUDHPA instances

As the period enumeration algorithm does not depend or impact the resulting utilization, i.e., the utilization of the task set with assigned periods, applicability of the combination of the period enumeration algorithm (PE) and optimal task algorithm (OTA) depends strictly on the task assignment itself. It can be seen that to solve an AUDHPA instance, OTA algorithm has to be minimally modified to include the target resulting utilization U_a. More precisely, to solve an AUDHPA instance, the constant maximal utilization of 1 in line 28 of Alg. 14 has to be replaced with the target utilization U_a.

4.9 Evaluation

To further investigate the UDHPA problem and the approach proposed in this research, an extensive evaluation of the developed algorithms on synthetically generated task sets is performed. Firstly, it is shown how the difficulty of the problem changes with regard to utilization, the number of distinct period values, the width of period ranges and the number of tasks in a task set. Furthermore, it is shown how the approach can be used for UHPA problem instances and the approach is compared with existing approaches. As the parameters used for synthetically generating task sets correspond to the parameters of task sets in motivational scenarios, it is demonstrated that the devised approach can be efficiently used in plenty of real-world scenarios. Moreover, a small numerical example that illustrates the benefits of the approach in motivational real-world scenarios is presented.

Algorithm 14 Optimal algorithm for the TA problem (OTA)

Input: $\vec{p}, \mathscr{T}$
Output: (feasible, U_{max}, $\mathbf{X}_{max}$)

```
1: function TASK ASSIGNMENT(p⃗, 𝒯)
2:     (correct, U, X) ← HPF ASSIGNMENT(p⃗, 𝒯)
3:     feasible ← false
4:     u ← 0
5:     U_max ← 0
6:     if not correct ∨ U > 1 then
7:         return
8:     end if
9:     TASK ASSIGNMENT STEP(p⃗, 1, 0)
10: end function
11: function TASK ASSIGNMENT STEP(p⃗, i, u)
12:     if i == n + 1 then
13:         if u > U_max then
14:             feasible ← true
15:             U_max ← u
16:             X_max ← X
17:         end if
18:         return
19:     end if
20:     ul ← 0
21:     for j in range 1 to m do
22:         if l_j == 0 then
23:             l_j ← 1
24:             ul ← 1
25:         end if
26:         if p_i^min ≤ p_j ≤ p_i^max ∧ m − d ≤ n − i then
27:             u' ← u + C_i/p_j                ▷ equivalent to T_i ← p_j or x_ij ← 1
28:             if u' ≤ 1 ∧ u' + b^l_{i+1} ≤ 1 ∧ u' + b^u_{i+1} ≥ U_max then
29:                 TASK ASSIGNMENT STEP(p⃗, i + 1, u')
30:             end if
31:         end if
32:         if ul == 1 then
33:             l_j ← 0
34:             ul ← 0
35:         end if
36:     end for
37: end function
```

4.9.1 Task set generation

In evaluation, task sets are generated using the UUnifast algorithm [39], which is commonly used in measuring the performance of algorithms in real-time systems. Using the UUnifast algorithm, utilizations of task sets are generated with regard to p_i^{max} of tasks in a set. Therefore, the target utilization for UUnifast method corresponds to the lowest utilization of a task set. This utilization is referred to as $U_{min} = \sum_i^n \frac{C_i}{p_i^{max}}$. After the utilizations are generated, p_i^{max} is chosen from the interval $[p_{down}, p_{up}]$ with uniform distribution. Then, based on the parameter σ, the lower bound of the period range for task τ_i is determined as $p_i^{min} = \lceil p_i^{max}\sigma \rceil$. Increasing σ decreases the width of the period range for tasks.

Task sets are generated with utilization U_{min} from interval $[0.2, 0.9]$ with an increment of 0.025. Moreover, $p_{down} = 1$, $p_{up} = 2048$, and $\sigma = 0.4$. In evaluation, 1000 task sets are generated per utilization factor, i.e., a total of $29 \cdot 1000 = 29000$ task sets. Every task set consists of $n = 20$ tasks. These are the default task set generation parameters unless noted otherwise. When it comes to algorithm runtime measurements, it is worth noting that implementations of algorithms are written in C++. Additionally, the specifications of the computing platform are given in Table 4.1. Furthermore, an algorithm is terminated the utilization value in the interval $[1 - 10^{-7}, 1]$ is obtained.

Table 4.1: Computing platform specifications

Processor	Intel(R) Core(TM) i7-7700HQ CPU 2.80GHz
RAM	8.00 GB
Operating system	Linux (64-bit)

4.9.2 Evaluation on UDHPA instances

In this section, the devised optimal approach which consists of the PE algorithm (Alg. 11) and the OTA algorithm (Alg. 14) is evaluated on UDHPA instances with regard to different problem parameters m, σ and n.

Firstly, the devised optimal approach is evaluated with regard to different number of distinct period values in the solution m. Fig. 4.2 shows the number of feasible systems for different utilization factors and for a different number of distinct period values in the solution. It can be seen that, for $m = 5$, the highest number of feasible solutions is obtained. For higher values of m, the number drops, and we know from Proposition 8 that the maximum value of distinct period values for the system to be feasible equals $\log_2 \frac{2048}{1} + 1 = 12$. Since the number of feasible systems for $m > 8, m < 3$ is lower than for $m = 8$, these graphs are not included in Fig. 4.2.

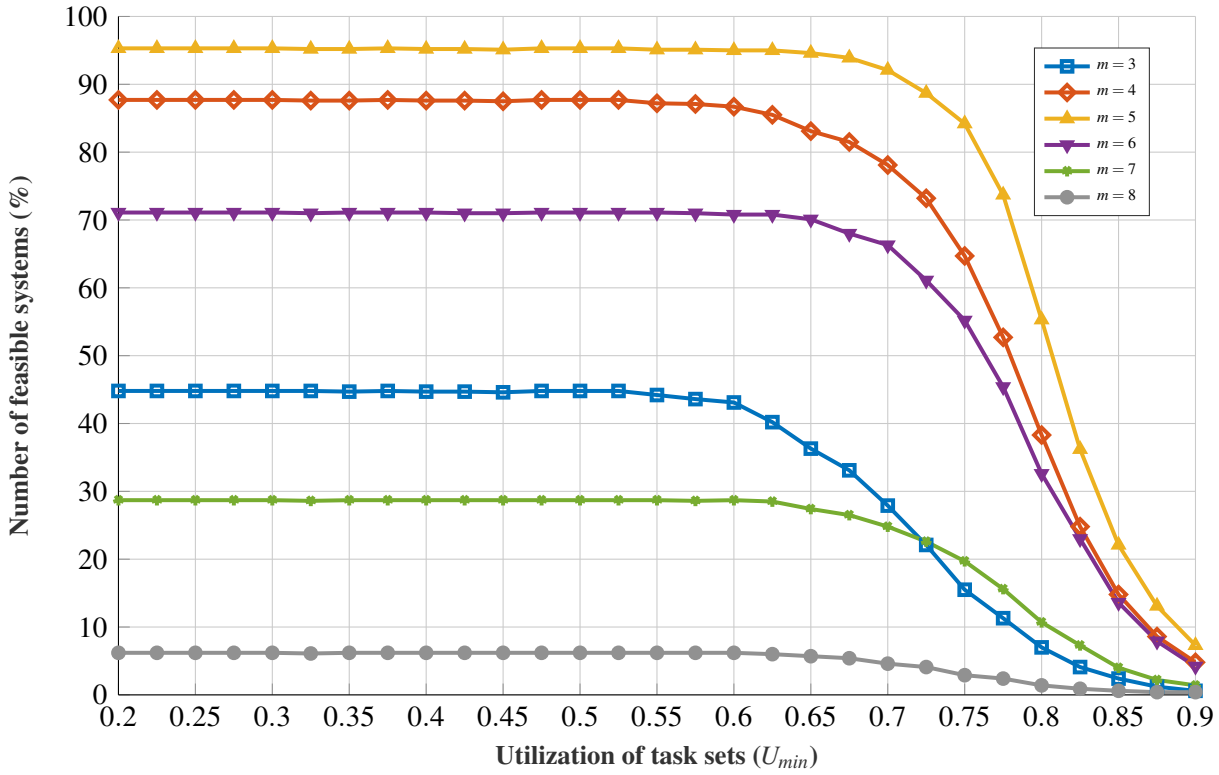


Figure 4.2: Number of feasible systems for different number of distinct period values m.

Moreover, Fig. 4.3 shows the average number of enumerated period sets with regard to m. It can be seen that, for higher values of different period values m, the number of harmonic period sets is reduced. Thus, increasing the number of different period values reduces the number of potentially feasible systems. On the other hand, although the number of enumerated period sets is higher for a lower m in Fig. 4.3, feasibility is reduced for a lower m as it is more difficult to find a lower number of harmonic period values that satisfy the correctness criteria (Definition 18) for each task in the system.

Fig. 4.4 shows the average resulting utilization for each utilization factor and a different number of distinct period values. In this particular evaluation, the resulting utilization of infeasible systems is set to zero. In this way, the information about the overall feasibility of systems is not lost. Again, it can be seen that, for $m = 5$, the best result are obtained. Dashed lines in the figure denote the utilization lower bound obtained by the HPF assignment for the corresponding number of distinct period values.

Fig. 4.5 shows the average utilization of feasible systems. Here, the resulting utilization of feasible systems for every m in interval $[3,8]$ is averaged. Additionally, Fig. 4.5 shows the lower bound and the upper bound obtained by the HPF and the LPF assignment, respectively. It can be seen that the resulting utilization of the devised optimal approach is very close to the upper bound for lower utilization values. When the upper bound is higher than 1, the devised optimal approach yields the highest possible utilization values lower or equal to 1.

Fig. 4.6 shows the average runtime per task set of the optimal algorithm for each utilization

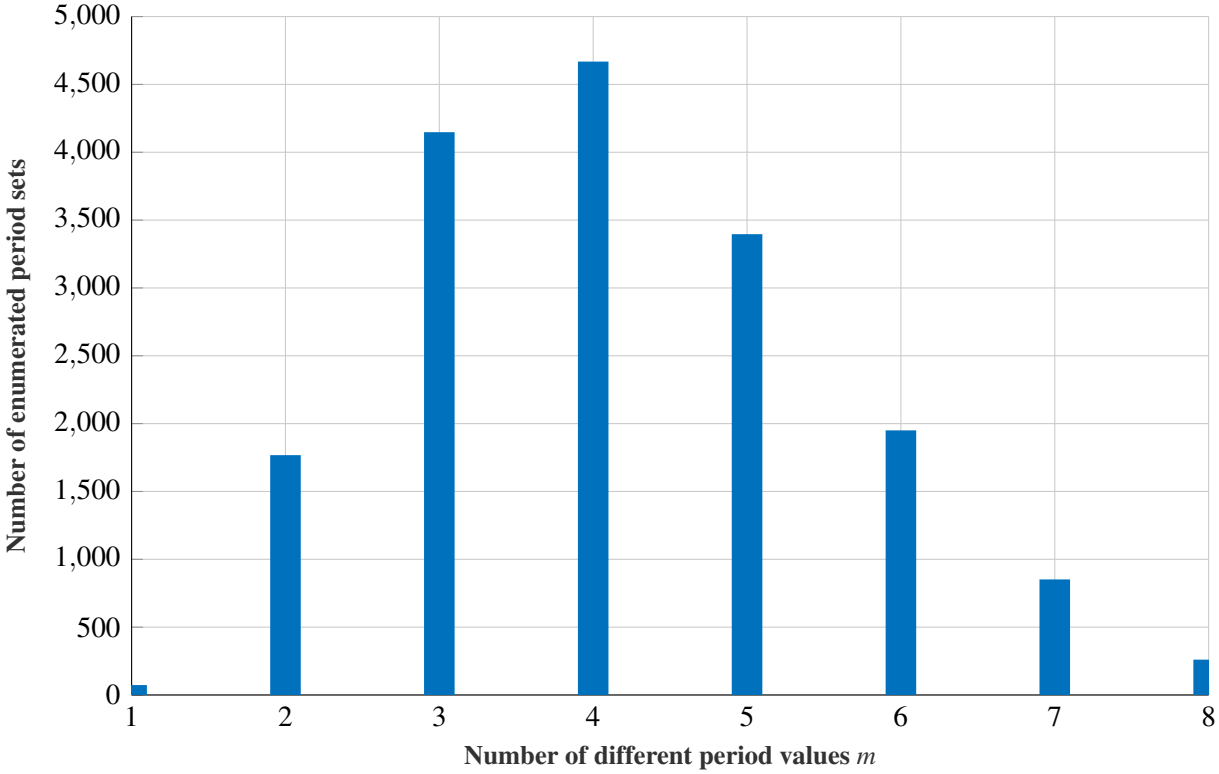


Figure 4.3: Number of period sets obtained in period enumeration w.r.t. m.

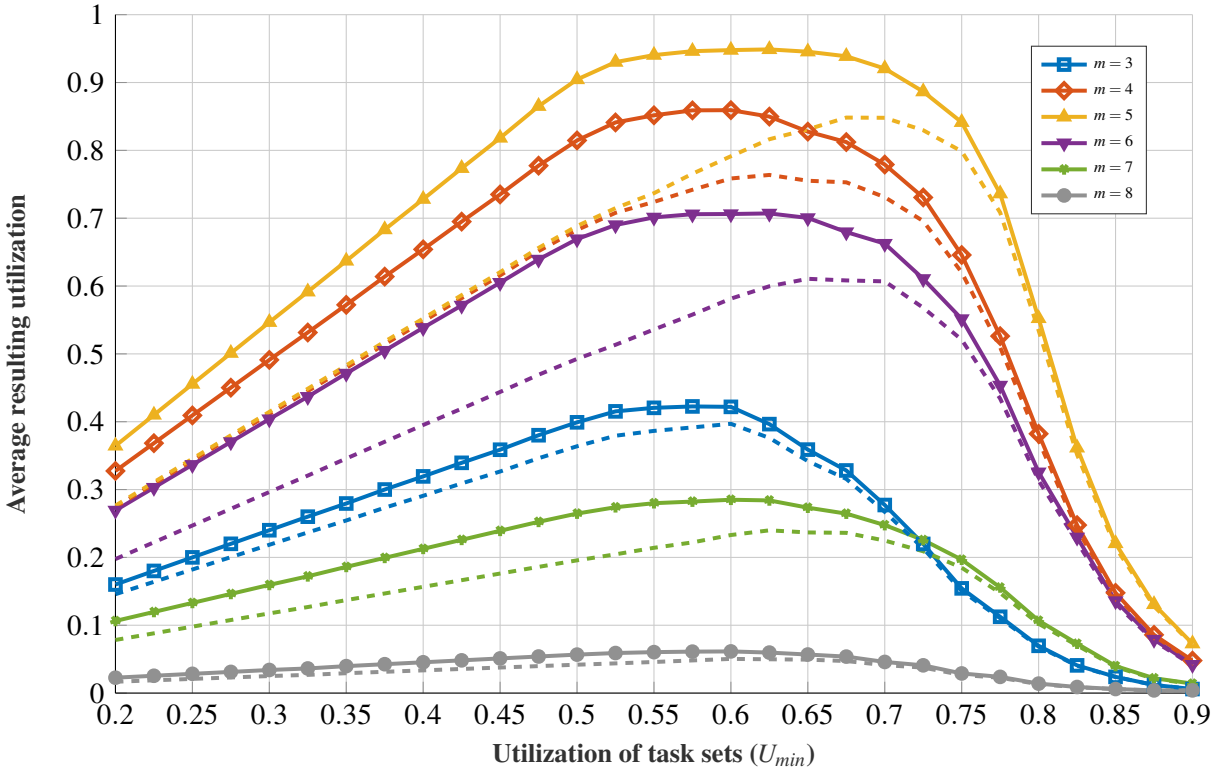


Figure 4.4: Average resulting utilization of task set for different number of distinct period values m.

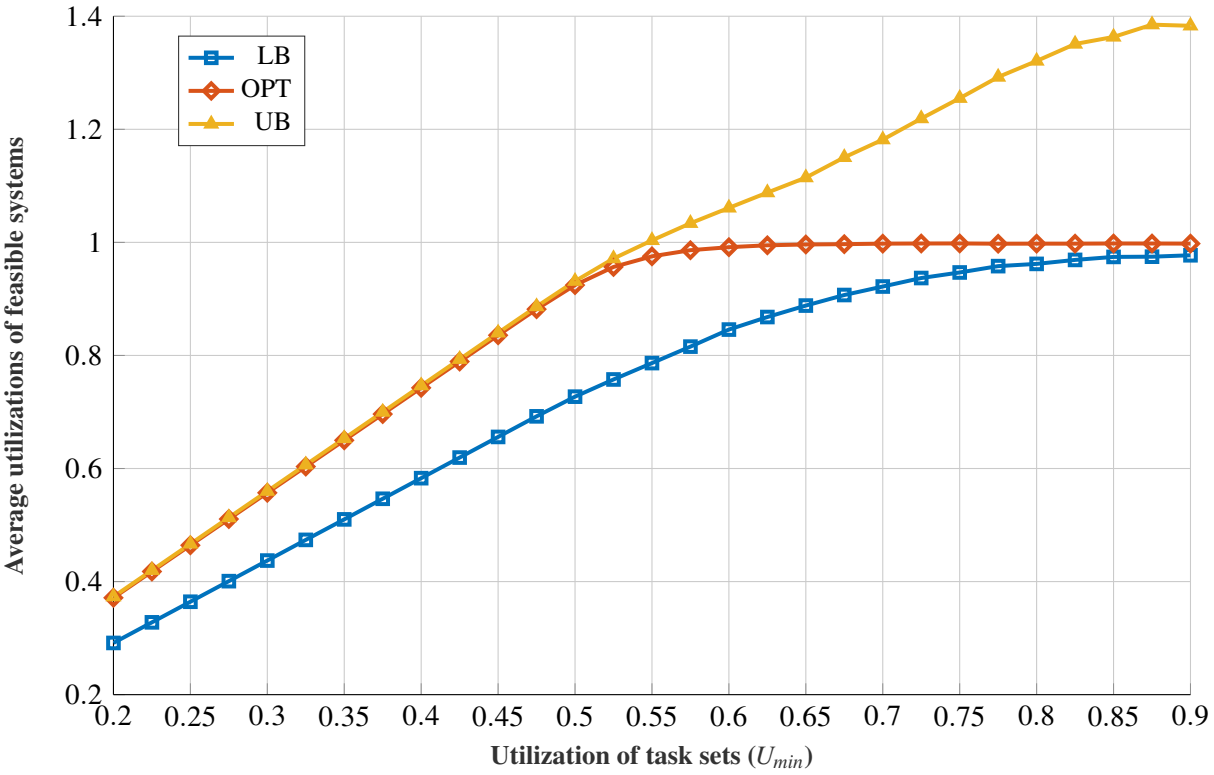


Figure 4.5: Average resulting utilization of the optimal assignment with corresponding bounds.

factor and a different number of distinct period values. It can be seen that, on average, the algorithm for larger numbers of different period values has a higher runtime. This is mostly due to a higher number of potential task assignments in solving the TA problem. Additionally, it is worth noting that the average runtime for $m = 8$ is lower than for $m = 7$ and $m = 6$. For $m = 8$, there is a smaller number of enumerated period sets, and therefore fewer TA problem instances have to be solved.

Figs. 4.7-4.9 show the effect of the period width σ on the overall feasibility, utilization and average runtime. The same parameters are used in task set generation as in the previous evaluation. However, in this case, the number of distinct period values is fixed, i.e., $m = 5$, and the period range width values σ from interval $[0.2, 0.7]$ are used. Additionally, 100 task sets for each utilization factor and for each σ are generated, which totals to $29 \cdot 6 \cdot 100 = 17400$ task sets.

Fig. 4.7 shows the feasibility for each utilization factor and for different period widths. It can be seen that, for higher σ, i.e., lower period range width, feasibility drops significantly. This is to be expected as the correctness constraints are more strict and there is less chance of finding a potentially feasible harmonic period set.

In Fig. 4.8, the average resulting utilization per utilization factor assuming that the utilization of infeasible system equals zero is shown. Larger period range width increases the number of feasible enumerated period sets, and consequently utilization is higher as the task assignments with high utilization can be discovered.

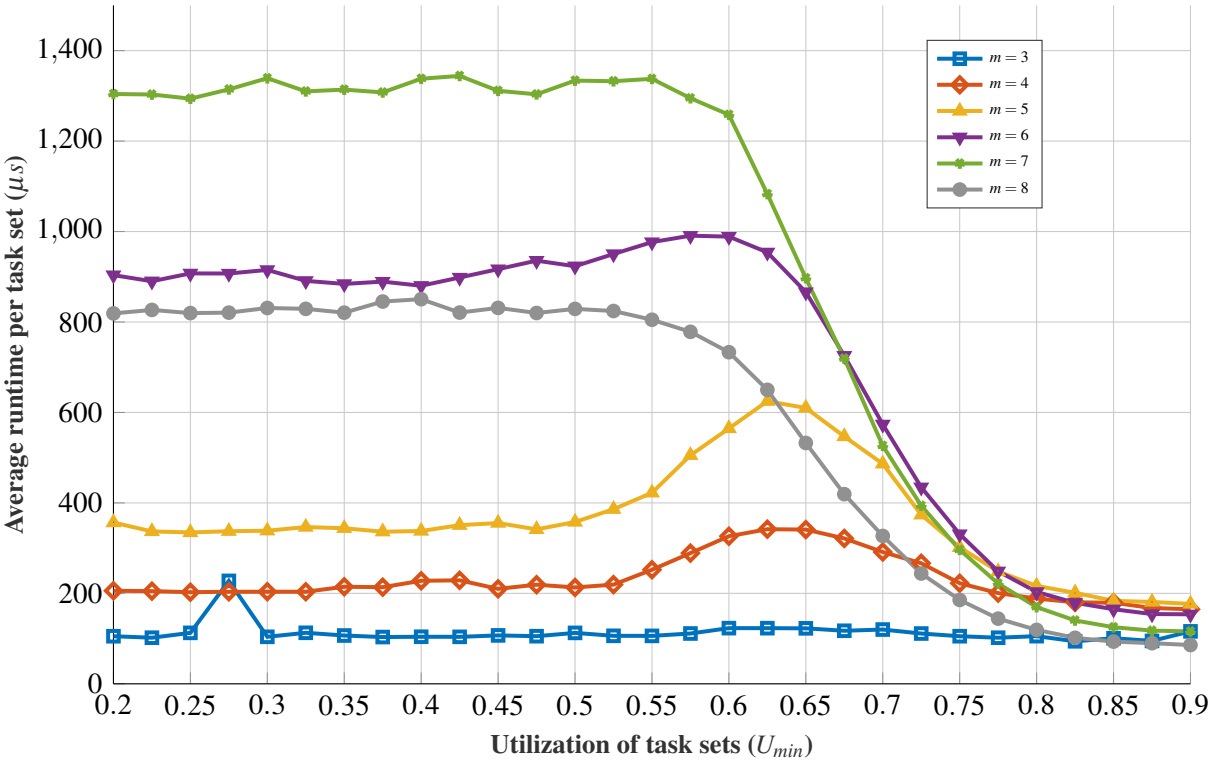


Figure 4.6: Average runtime of the optimal assignment for different number of distinct period values m.

In Fig. 4.9, we see, that for a larger period range width, i.e., $\sigma = 0.2$, runtime is of an order of magnitude higher than a smaller period range width. As it was already mentioned, the number of feasible solutions is higher for larger period range width and more enumerated period sets have to be explored. Thus, the runtime is increased. It is worth noting that runtime graphs for $\sigma \in [0.4, 0.7]$ cannot be distinguished, as they are much lower than the average runtime for $\sigma = 0.2$.

Finally, the evaluation of the devised optimal algorithm with regard to the size of a task set is investigated. In the previous evaluation, the number of tasks in a set was fixed to 20. For the purpose of this evaluation, alternative parameters were fixed, the period range width, i.e., $\sigma = 0.4$ and the number of distinct period values, i.e., $m = 5$. For Figs. 4.10-4.11, 200 task sets for each utilization factor and for each $n \in [20, 30, 40, 50]$ were generated, which totals to $29 \cdot 4 \cdot 200 = 23200$ task sets. Fig. 4.10 shows the number of feasible task sets with regard to n. It can be seen that, by increasing n, feasibility drops. This is the effect of adding more period range constraints in the systems for each task. Intuitively, it will be more difficult to find an appropriate period assignment with a higher number of constraints. For the same reason, the total average utilization is reduced when n is increased as depicted in Fig. 4.11. For the runtime evaluation with regard to n, the utilization factor was set to $U_{min} = 0.6$ and generated 200 task sets for every fifth number of tasks in range $[20, 100]$, i.e., a total of $200 \cdot 17 = 3400$ task sets. In Fig. 4.12, the PE + OTA graph corresponds to the devised optimal approach. The PE + EXH graph corresponds to an approach which consists of the PE algorithm (Alg. 11)

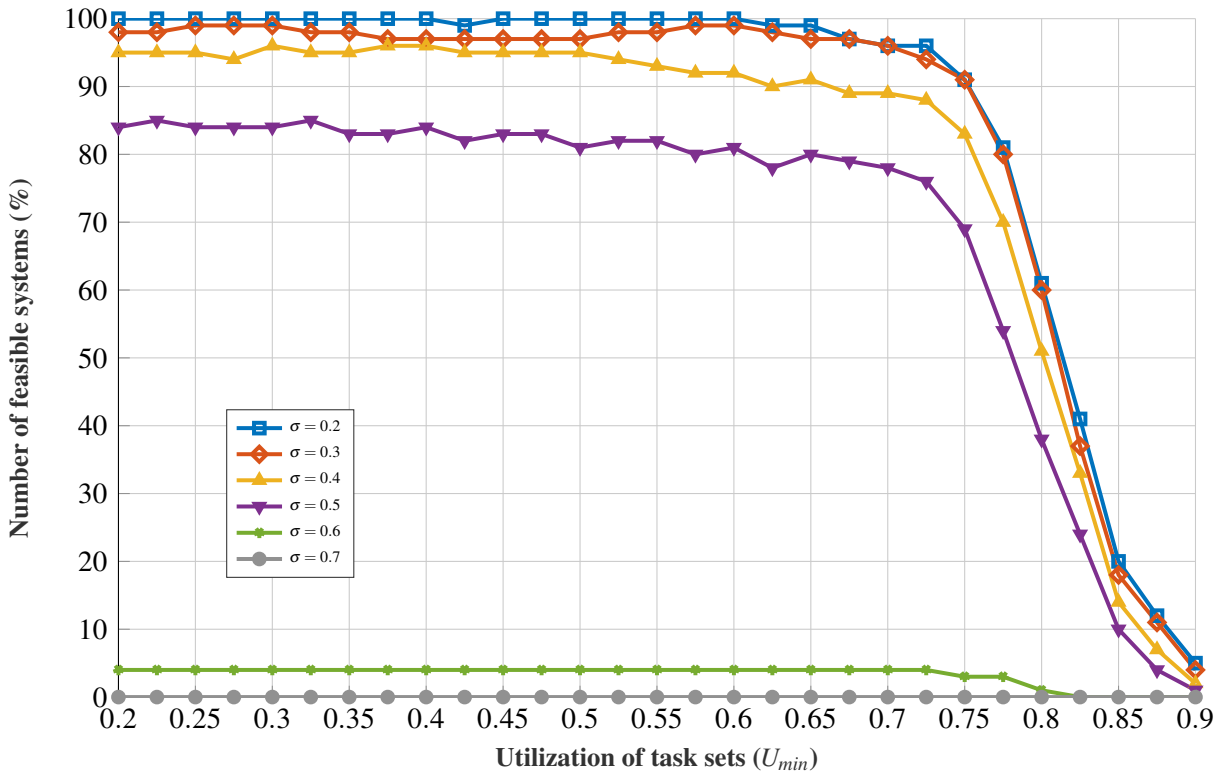


Figure 4.7: Number of feasible systems for different period ranges width σ.

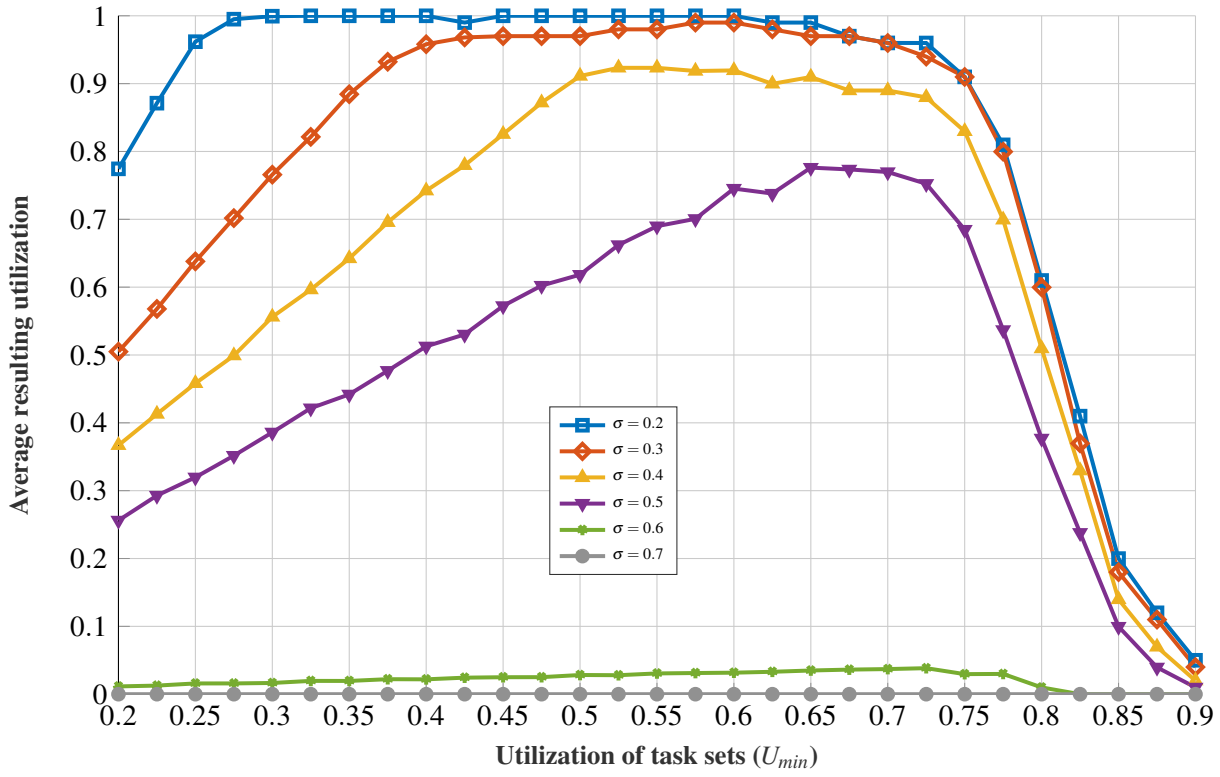


Figure 4.8: Average resulting utilization for different period range width σ.

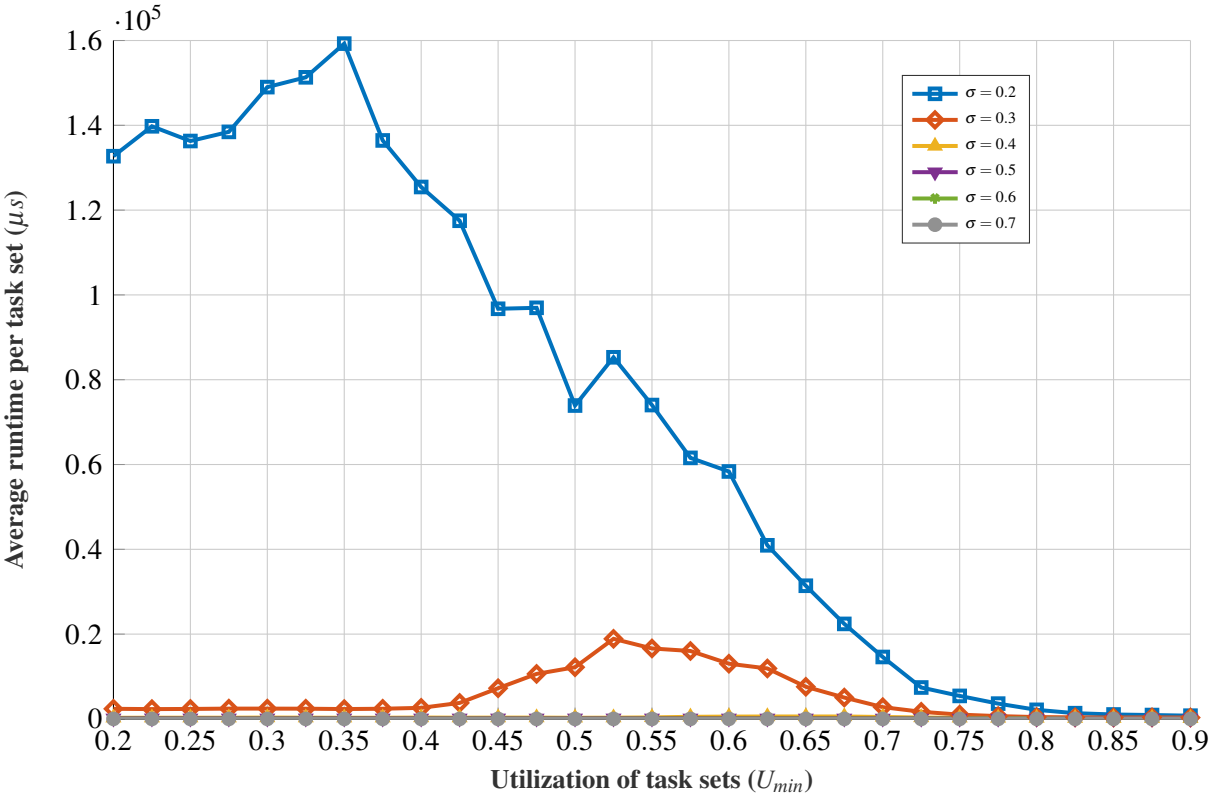


Figure 4.9: Average runtime of the optimal assignment for different period range width σ.

and an exhaustive search of task assignments, which does not employ utilization bounds and pruning rules devised in section 4.8.2. We can see that the runtime of exhaustive search rises exponentially with the number of tasks, which is an expected behavior, since, at worst, time complexity is in $O(m^n)$ (evaluation is not performed for $n \geq 70$). On the other hand, we see that the runtime of the PE + OTA approach has a reduced growth rate due to the usage of devised utilization bounds and pruning rules.

4.9.3 Evaluation in the context of existing UHPA approaches

On the basis of Theorem 3, we know that the UHPA problem is Turing reducible to the UDHPA problem, and therefore we can simply employ the devised optimal algorithm for UHPA instances. To solve an UHPA instance, we need to solve the corresponding UDHPA instances for every possible number of distinct period values m, which is given with Proposition 8. To optimally solve an UHPA instance, we use the PE algorithm in combination with the OTA algorithm for each m. In the UHPA problem, there is no restriction on the number of distinct period values. Thus, we can use the PE algorithm with the HPF algorithm to obtain the period assignment for each m. It is worth noting that it is possible that, while using the HPF algorithm, some values of the enumerated period set may remain unused. However, this is not a problem in the context of UHPA instances since there are no restrictions on the number of different period values. The devise approach is compared with existing UHPA approaches in the literature,

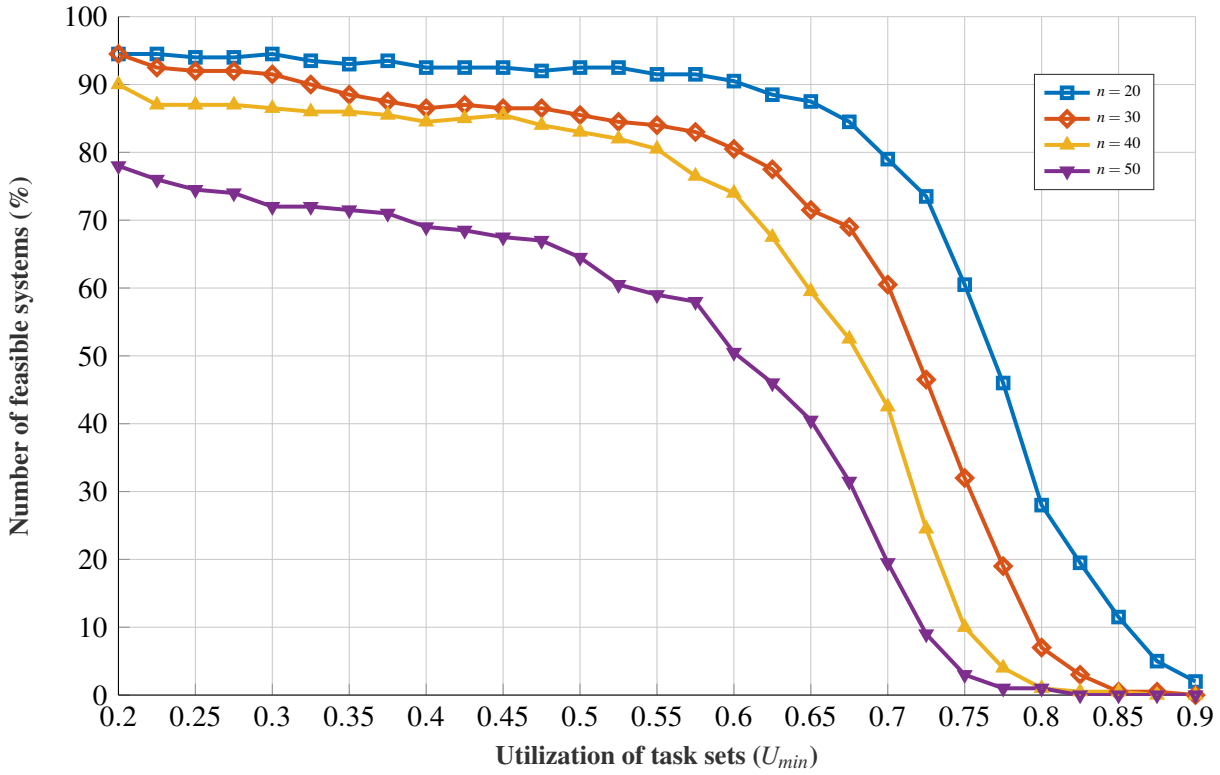


Figure 4.10: Number of feasible systems for different number of task in system n.

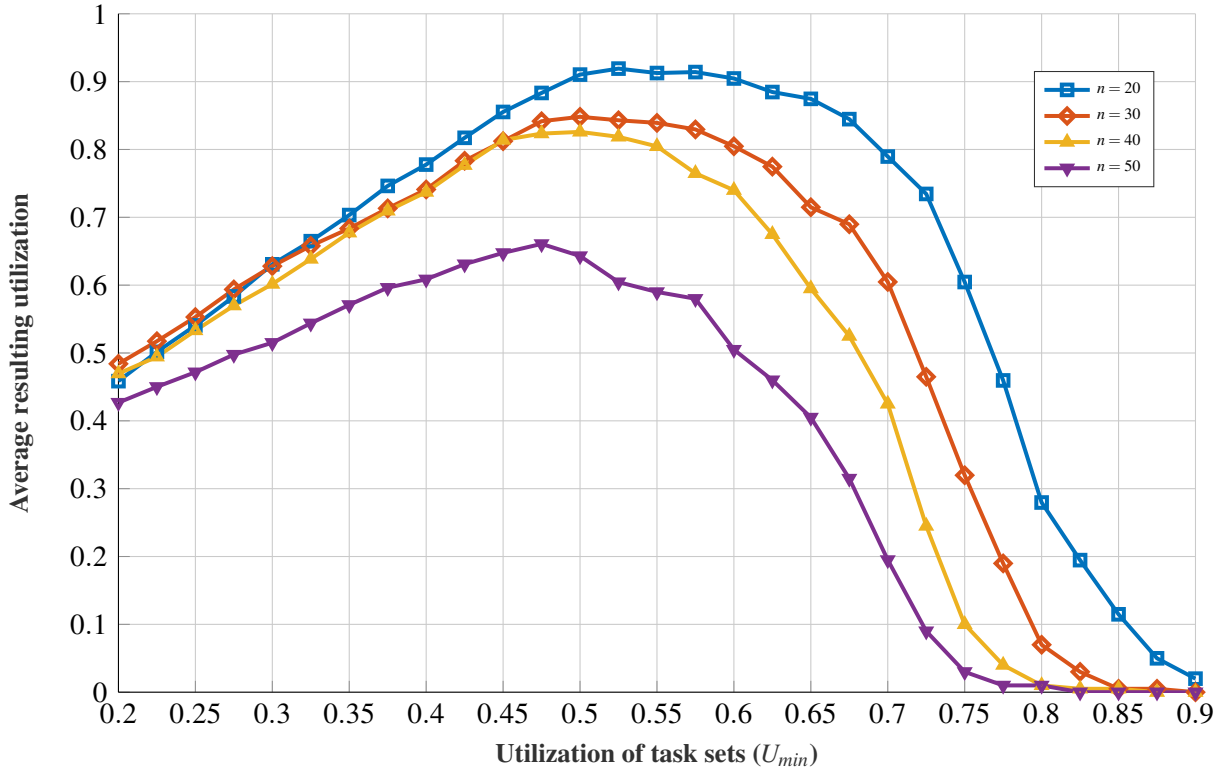


Figure 4.11: Average resulting utilization of task set for different number of tasks in system n.

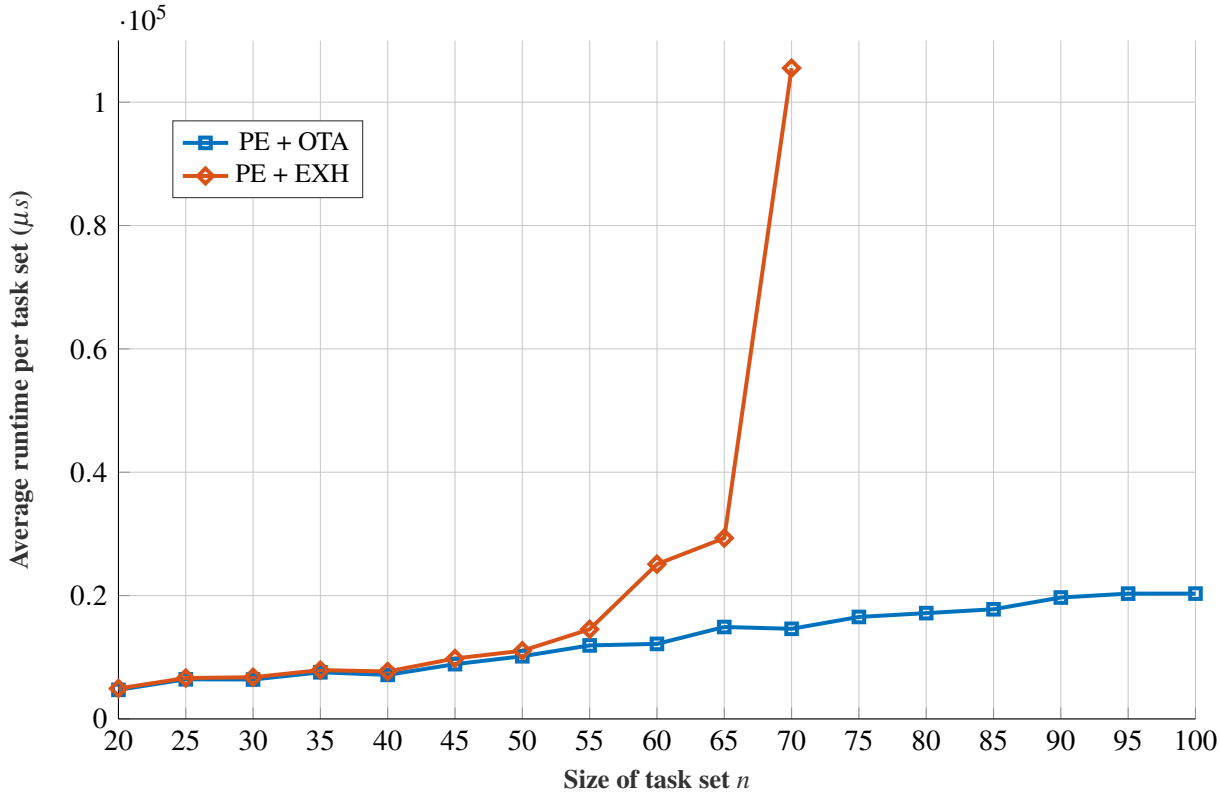


Figure 4.12: Average runtime of the optimal assignment for different size of task set n.

which are based on finding harmonic projections for given task period ranges [63, 64]. The algorithms employed in these approaches are generally of pseudo-polynomial time complexity, but in specific cases complexity can be reduced to linear or polynomial time. The approach in [63], referred to as *forward search*, consists of two parts, namely the graph construction algorithm (GCA), and a greedy heuristic for period assignment, which can yield low utilization (LU) or high utilization (HU). Here, the LU heuristic is employed since it increases the chance that the resulting harmonic period assignment will be feasible. The GCA part is analogous to the period enumeration part of the devised algorithm. Similarly, the HPF and OTA algorithms are counterparts to the LU heuristic. The approach from [64], referred to as *backward search*, is based on the harmonic period existence test and suboptimal heuristic period assignment. Figs. 4.13-4.17 show the performance of the devised approach in comparison with approaches from [63, 64]. Task sets were generated using the default task set generation parameters from the beginning of this section.

Fig. 4.13 shows the number of feasible systems for different period assignment approaches. It can be seen that the number of feasible task sets is higher when the devised optimal approach (PE + OTA) and heuristic approach (PE + HPF) are used, than when *forward search* or *backward search* are used. The number of feasible systems when the PE + HPF or PE + OTA approaches are used is the same because both algorithms are optimal regarding feasibility. However, the HPF algorithm yields the solution with the lowest utilization. More precisely, it yields the

lowest utilization for the TA* problem. As explained in section 4.8.2, the TA* problem does not restrict the number of different period values in the solution, and therefore the utilization obtained using the HPF assignment is minimal. Fig. 4.14 shows the resulting utilization for different period assignment approaches. As expected, the devised optimal approach (PE + OTA) yields the highest utilization. Moreover, the period enumeration with the HPF assignment (PE + HPF) dominates *forward search* and *backward search* as well.

Finally, the runtime results in Figs. 4.15-4.17 are shown. In Fig. 4.15 , it can be seen that the runtime for *forward search* and *backward search* is at least an order of magnitude lower than in the approach proposed in this research. Since the period ranges are relatively wide, i.e., $\sigma = 0.4$, both *forward* and *backward search* are time-efficient. Moreover, in Fig. 4.16, we can see that when the number of tasks in a set is increasing, runtime is higher for exhaustive search (PE + EXH) and the optimal approach (PE + OTA) than for the other approaches which cannot be distinguished in the figure. The PE + HPF approach is efficient when the number of tasks is increasing since the time complexity of the HPF algorithm is polynomial $O(n \cdot m)$ and the time complexity of the PE algorithm does not depend on the number of tasks in the system. However, in Fig. 4.17, we can see that the runtime of approaches which employ PE algorithm increases when the maximum period in the system p_{max}^{max} is increased. It is worth noting that although runtime can be significantly higher when the optimal approach (PE + OTA) and the heuristic approach (PE + HPF) are used, it is still relatively low, i.e., several milliseconds per task set. Since period assignment in practice is typically done off-line during the application design, this is more than acceptable.

4.9.4 Numerical real-world period assignment problem

To further emphasize and explain the benefits of the devised approach, a small numerical example of a real-world period assignment problem is provided. As it was already stated in the introduction, the structure of safety-critical control applications is modular and often each module, i.e., task, is developed by a different application designer. In the development process, based on the specific application requirements, application designers provide implementations of tasks with suggested execution rates, which are in this research and related literature modeled with period ranges. It is in the interest of every application designer that their module executes with the highest possible execution rate, i.e., the smallest period, in order to achieve a higher quality of service for a particular part of the application. Based on the input from the application designers, the system designer has to determine periods which shall be used in the system to achieve the highest utilization, i.e., quality of service. Thus, utilization is maximized. Table 4.2 shows the task set with task parameters. Such a table is an input to the system designer. However, due to the specific architecture of the system, i.e., the operating system and the underlying hardware, the system designer is restricted regarding the number of distinct period values which

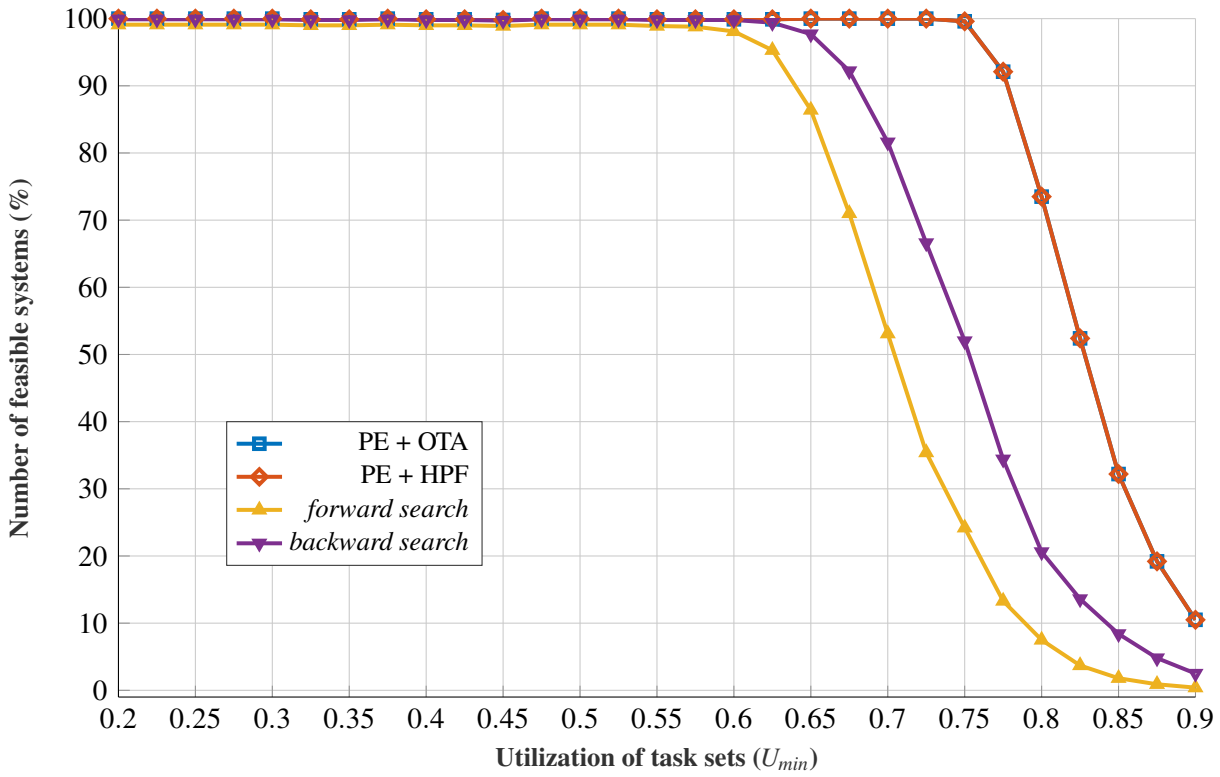


Figure 4.13: Number of feasible systems for UHPA instances for different period assignment approaches.

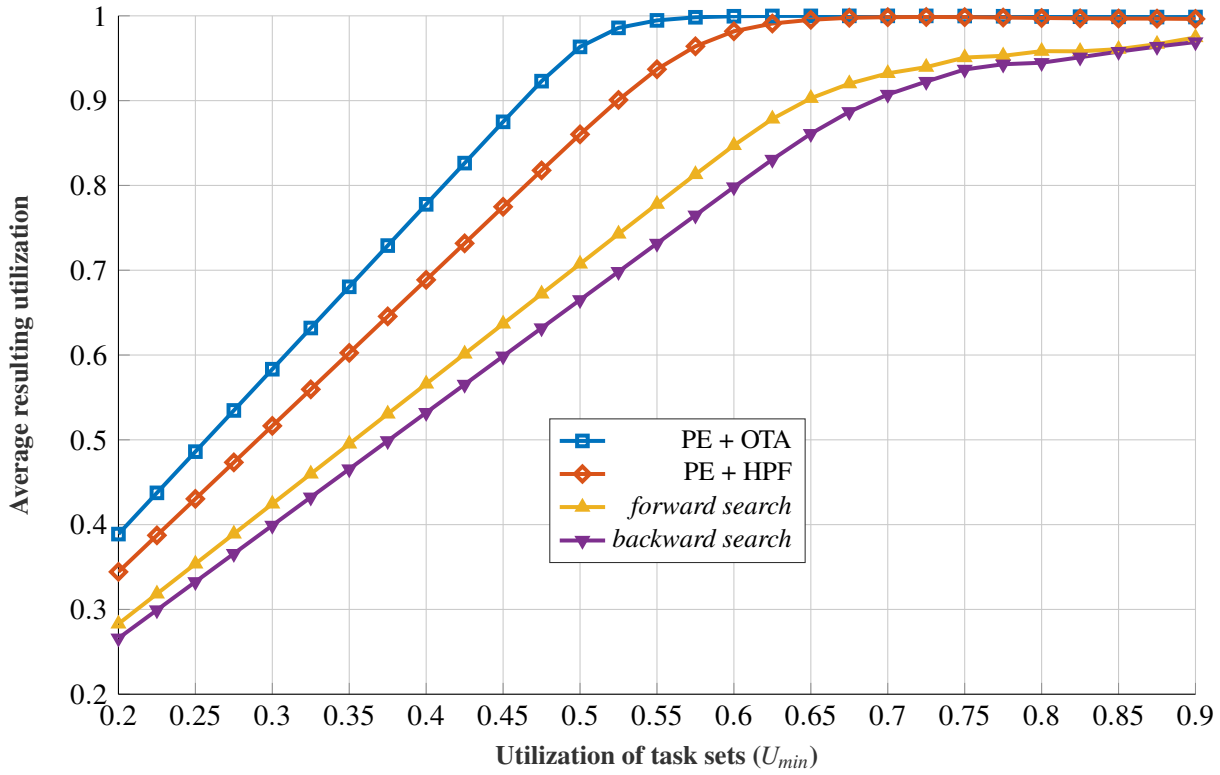


Figure 4.14: Average resulting utilization for different period assignment approaches.

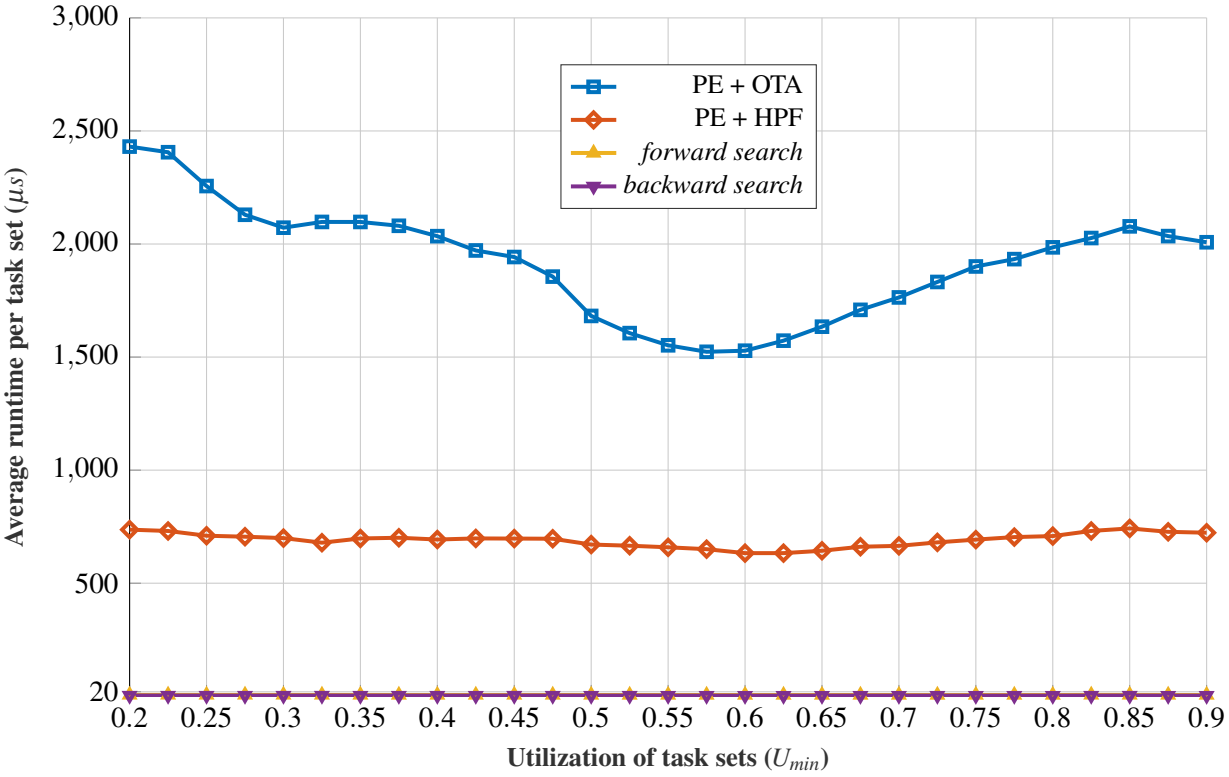


Figure 4.15: Average runtime for different period assignment approaches w.r.t. utilization.

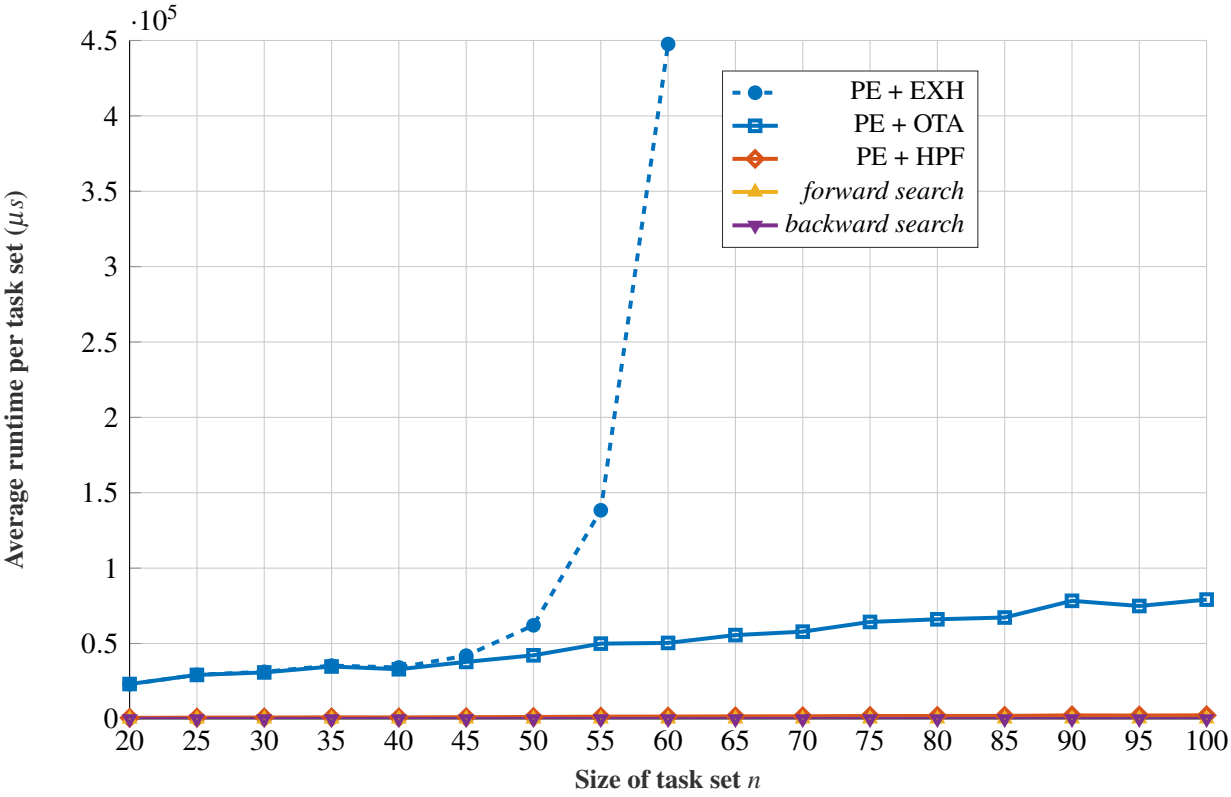


Figure 4.16: Average runtime for different period assignment approaches w.r.t. size of task set.

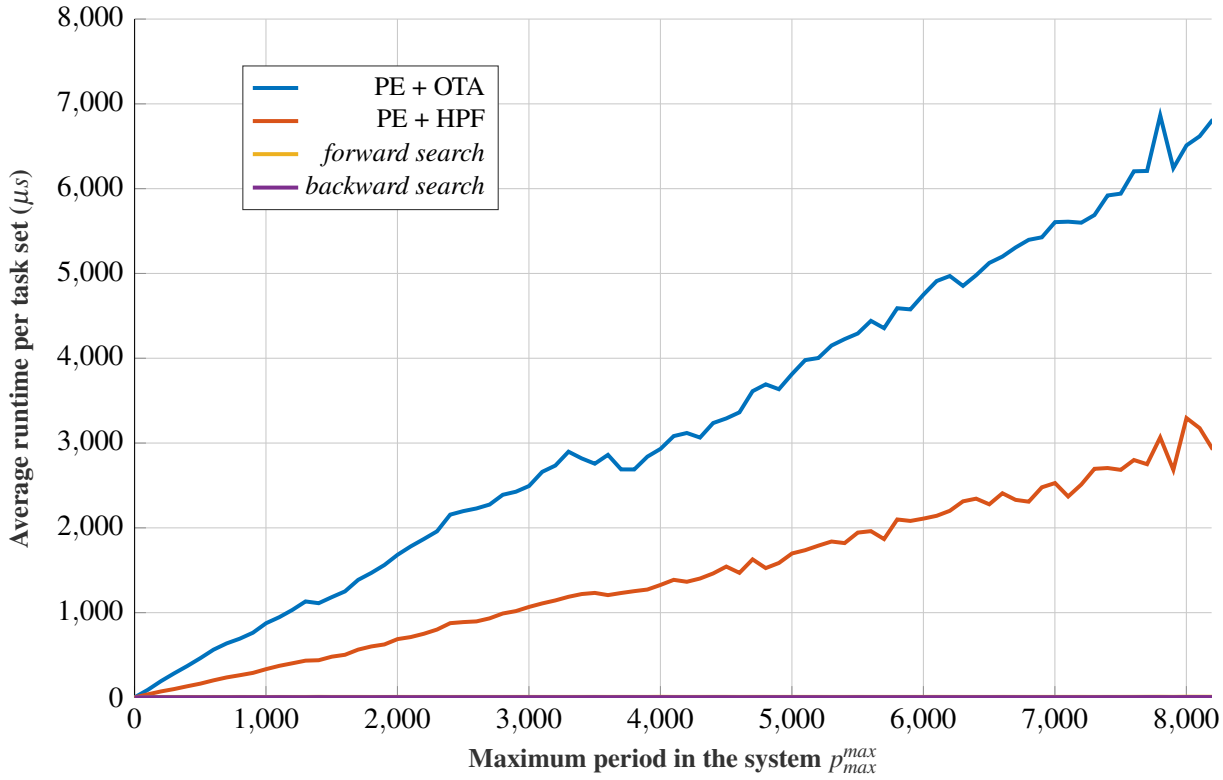


Figure 4.17: Average runtime for different period assignment approaches w.r.t. maximum period in the system p_{max}^{max}.

can be used in the solution. In this example, the maximum number of different period values is 4. Therefore, any number of different period values smaller or equal to $m_{max} = 4$ can be used.

Table 4.2: Task parameters for the application task set

Task	C_i	p_i^{min}	p_i^{max}
τ_1	1	2	5
τ_2	2	5	16
τ_3	2	13	42
τ_4	1	21	68
τ_5	13	36	118
τ_6	3	38	124

Table 4.3 shows the period values assigned to each task in the input. We can see that period enumeration with the optimal task assignment (PE + OTA) yields the most satisfying result since it produces the maximal utilization and uses no more than 4 period values. In order to achieve this, the system designer has to solve UDHPA instances using the PE + OTA approach for m

in the interval $[1, m_{max} = 4]$. We can see that period enumeration using the HPF algorithm (PE + HPF) also yields a satisfying result as the number of the distinct period values is lower than m_{max}. However, the utilization is lower than the value obtained using the OTA algorithm. The *forward* and the *backward search* do not yield satisfying results since they do not restrict the number of distinct period values in the solution. Moreover, utilization factors are significantly lower than when using both the PE + OTA and the PE + HPF approaches.

Table 4.3: Assigned periods, the number of distinct period values and resulting utilization per period assignment approach

	Assigned periods T_i per approach			
Task	PE + OTA	PE + HPF	*forward*	*backward*
τ_1	2	5	3	5
τ_2	14	5	15	15
τ_3	14	20	30	30
τ_4	42	60	60	60
τ_5	84	60	60	60
τ_6	84	60	120	120
m	**4**	**3**	5	5
U	**1.000**	**0.983**	0.791	0.658

4.9.5 Numerical real-world period assignment problem with arbitrary utilization

To show the importance of adding the target utilization rather than maximizing the utilization up to the utilization schedulability bound, a small numerical example similar to the one in the latter section is provided. Table 4.4 shows the task set with task parameters. Such a table is an input to the system designer. However, due to the specific architecture of the system, i.e., the operating system and the underlying hardware, the system designer is restricted regarding the number of distinct period values which can be used in the solution. In this example, the maximum number of different period values is 4. Therefore, any number of different period values smaller or equal to $m_{max} = 4$ can be used. In addition, the resulting utilization is restricted to be less or ideally equal to 0.8.

Table 4.4: Task parameters for the application task set

Task	C_i	p_i^{min}	p_i^{max}
τ_1	1	2	6
τ_2	1	6	17
τ_3	1	8	26
τ_4	5	11	34
τ_5	3	16	51
τ_6	2	33	108

Table 4.5 shows the period values assigned to each task in the input. We can see again, that the best results are obtained using the **PE + OTA** approach, i.e., optimum value of 0.8. **PE + HPF** approach satisfies the distinct number of period values and utilization constraints. However, the resulting utilization is somewhat lower than using the optimal approach. Again, *forward* and *backward* search perform worse than the proposed approaches. As they are agnostic about additional design constraints, they are inapplicable in these scenarios. Moreover, the proposed approaches are suitable for scenarios in which the utilization of critical tasks has to be tuned.

Table 4.5: Assigned periods, the number of distinct period values and resulting utilization per period assignment approach

	Assigned periods T_i per approach			
Task	PE + OTA	PE + HPF	*forward*	*backward*
τ_1	5	4	6	4
τ_2	15	16	12	8
τ_3	15	16	24	16
τ_4	15	16	24	16
τ_5	30	48	48	32
τ_6	60	48	96	96
m	**4**	**3**	5	5
U	**0.800**	**0.792**	0.583	0.865

4.10 Chapter summary

In this chapter, the utilization-maximizing harmonic period assignment problem with a constrained number of distinct period values (UDHPA) was defined. Moreover, this problem was extended to allow optimization with regard to an arbitrary utilization factor. The motivation for this problems arises from the observation of industrial control systems in which the number of different period values is either fixed or restricted. The problem was discussed in the context of the already studied UHPA problem and it was shown that the UHPA problem is Turing reducible to the UDHPA problem. Additionally, it is shown that UDHPA problem is at least weakly NP-hard. It was demonstrated that the more general problem of assigning periods with an arbitrary utilization (AUDHPA) is NP-hard as well. An optimal algorithm for the UDHPA problem is devised and its efficiency on a large number of synthetically generated task sets was demonstrated. Moreover, the developed algorithm was used for solving UHPA instances and the differences between the devised approach and the existing approaches were explained and discussed. The key benefit of the devised approach lies in the fact that, for a large variety of synthetically generated UDHPA and UHPA instances, the devised algorithm is time-efficient and optimal, and therefore more than suitable for use in real-world system design. Furthermore, as it was shown in numerical examples, the existing approaches are not applicable in systems with a restricted number of different period values and arbitrary target utilizations. The future

work will include the application of the devised algorithms to real-world scenarios and a further theoretical analysis of the relation between the optimal number of distinct period values and period ranges of tasks.

Chapter 5

Method for task scheduling for improvement of quality of service in real-time mixed-criticality systems based on genetic programming

5.1 Context of the research

In the previous chapters, formal methods for verification and design of mixed-criticality and safety-critical systems were discussed. In both chapters, the focus in the design and verification is on ensuring schedulability and maintaining the quality of service of high-criticality and safety critical tasks. In contrast, in this chapter, a method which would enable the system designer to devise a scheduling algorithm which would optimize the performance of the low-criticality tasks is devised. Moreover, while optimizing the quality of service of the low-criticality tasks, the schedulability of high-criticality tasks should not be impacted and must be ensured. The most important effect, which affects the mixed-criticality system in this context, is *overload*. Due to the increase in the utilization upon a criticality switch, overload occurs. Under overload, optimal techniques for scheduling are significantly different than in the normal conditions. Here, different mechanisms are investigated which can mitigate these effects such as priority scheduling, acceptance tests and job skipping. The main method that is exploited in this chapter is the genetic programming, which is often used in solving NP-hard scheduling problems in the literature. However, genetic programming methods have not been exploited yet in real-time scheduling problems. In this chapter, the framework is proposed that enables the evolution of priority functions for scheduling low-criticality tasks in the dynamic scheduling environment. The priority functions are evolved to optimize certain quality-of-service criteria, namely grade of service and penalty. Firstly, the single-objective genetic programming is

employed for the evolution. The method is then extended to allow multi-objective optimization, which enables the simultaneous evolution with regard to more than one quality-of-service metrics. In addition, the cooperative co-evolution approach is investigated for the evolution of acceptance tests for testing the task instances, i.e., jobs on the release. Furthermore, the method is applied in the partitioned environment in combination with the harmonic period assignment method devised in the previous chapter. Similarly as in the previous chapters, the method is extensively tested on synthetically generated task sets as well as on small numerical examples.

5.2 Introduction

As discussed before, recent years have brought a significant rise in demand for embedded systems which can execute functions with different levels of importance. The importance of a system function is determined by a system designer according to the application requirements. In the context of real-time task scheduling, the importance of a function is often related to the worst-case execution time (WCET) since larger worst-case execution times must be allocated for high-criticality tasks. This was expressed by Vestal in the seminal mixed-criticality (MC) scheduling paper [29].

Motivation. As it is discussed in the chapter 3, the initial Vestal's approach, referred to as static mixed-criticality (SMC) approach, and later adaptive mixed-criticality approach (AMC) devised by Baruah et al. [30], abandon low-criticality (LO) tasks in the high-criticality (HI) mode. This is justified in systems which do not use results of LO tasks in HI mode. However, such an assumption does not hold in many practical applications. Therefore, in order to increase the applicability of MC theory to real-world systems, numerous models emerged for systems which have a goal to improve quality of service (QoS) of LO tasks. These models are covered in the extensive survey by Burns et al. [28].

Initial observation about overload. In MC systems, HI tasks are typically assigned two different estimations of their WCET, which correspond to different methods of execution time profiling. Typically, thorough profiling will yield more conservative WCET values, and consequently more conservative bounds are used in the analysis of HI mode. Therefore, processor utilization of HI tasks upon a mode change (from LO to HI mode) will increase. If LO tasks are not discarded in HI mode, the total processor utilization could increase above the necessary schedulability bound, i.e., 100% for uniprocessor platforms. This makes MC systems potentially overloaded real-time systems from a task scheduling point of view. For clarity, a real-time system is overloaded if a feasible schedule does not exist, i.e., there is no schedule in which all the tasks will meet their deadlines.

5.2.1 Related work

This part of the research is related to several research areas in scheduling domain, namely **overloaded real-time systems**, **mixed-criticality systems** and **generating scheduling heuristics using genetic programming**. In the review of the related real-time systems literature, a common task model is assumed, i.e., jobs are released by sporadic or periodic tasks.

Overloaded real-time systems. Early research in overloaded real-time systems was done in papers such as [84, 85] where authors devise algorithms for on-line scheduling in overloaded systems and compare their *effective processor utilization* (EPU) to EPU of the optimal off-line clairvoyant algorithm. EPU is a metric which measures processor utilization for tasks which complete by their deadline. Generally, in real-time systems there are two types of overload: (1) transient overload due to task overruns or aperiodic events or (2) permanent overload in periodic task systems [86]. In this research, permanent overload is of a particular interest since periodic MC task systems are observed. Techniques for mitigation of the permanent overload are often divided into three groups: period adaptation, service adaptation and job skipping. There are a number of these approaches and they are reviewed in [86, 87]. In this research, job skipping is employed as a technique of mitigating overload. Initial research of job skipping was introduced in [88] where the skip-over task model was introduced. In the skip-over model each task has a skipping factor which determines the number of consecutive jobs of a task which have to be executed before their deadline. Moreover, a more general approach known as (m,k)-firm task model [89] specifies that each m jobs of k consecutive invocations have to meet their deadline. An interesting result from [88] is that the problem of determining feasibility of occasionally skippable task set is NP-hard. In addition, authors in [88] conjecture that constructing an off-line schedule, i.e., making an optimal use of skips in job skipping environment, is NP-hard as well. It is worth noting that these approaches are criticality-agnostic and cannot be easily incorporated in MC models.

Overloaded mixed-criticality systems. As pointed out earlier, MC systems can undergo permanent overload conditions due to the overrun of a job of HI task, i.e., upon a mode change. The first technique used to ensure feasibility and avoid the overload conditions was suspending LO tasks [30] in HI mode. As this can affect system performance in terms of quality of service, different period adaptation, service adaptation and job skipping techniques were investigated. **Period adaptation** in MC systems was investigated in several papers, which extend the elastic model of real-time systems for MC systems [90]. Similarly, in dual-mode control systems, periods for different operating modes are adapted to maximize the control performance [91]. **Service adaptation** includes reducing the computational time needed for some tasks to avoid overload. Based on this technique, imprecise mixed-criticality (IMC) models [92] emerged, which use reduced computational time for LO tasks in HI mode. **Job skipping** in MC systems is proposed by Burns et al. in [93] for increasing the robustness of a system. In this research,

systems allows job skipping of LO tasks in HI mode, and LO jobs are scheduled in the slack time of HI jobs. This approach is similar to the approach devised by Hikmet et al. in [94], where they schedule jobs of non-critical tasks in the slack time of tasks with a hard deadline. Job skipping was investigated in the context of mitigating overload of control tasks in [95] where authors devise a dynamic programming algorithm for determining optimal job skipping pattern for the hyperperiod of a certain task set. This approach may not be suitable for an on-line implementation in real-time embedded systems due to the time and memory overhead of the optimal algorithm. Naturally, this is the consequence of the computational complexity of making optimal use of skips, which is likely to be NP-hard [88, 95]. In approach devised in this research, genetic programming is used to discover heuristics which may be sub-optimal, but have two major advantages: they can be used in job skipping scenarios for different MC task sets, and they are computationally efficient and suitable for an on-line implementation in real-time embedded systems.

Generating scheduling heuristics using genetic programming. For systems that do not have hard deadlines as real-time systems, an effective approach to determine a schedule is to use heuristics, i.e., priority dispatching rules, evolved using genetic programming. These approaches are covered in the extensive survey by Nguyen et al. [96] on usage of genetic programming in production scheduling. An example of such an approach is presented in [97] where authors showed the dominance of heuristics evolved using genetic programming in minimization of weighted tardiness for different scheduling models (e.g., single machine, job shop) over common approaches such as the earliest due date (EDD). Similar approaches that employ genetic programming for generating heuristics for various single machine environments were investigated in [98, 99, 100]. The problem that is found in the production scheduling literature that is similar to job skipping problems discussed earlier in this section in the context of real-time systems, is the problem of minimizing the number of tardy jobs on a single machine with release times [101]. This problem is known to be NP-hard in the strong sense (see Table E.3 in Appendix E in [101]). However, the main point of difference between these problem formulations is that tardy jobs in the real-time setting are not allowed and they are discarded, whereas in the context of production scheduling tardy jobs are typically not discarded.

5.2.2 Contribution

In this chapter, genetic programming is used to evolve on-line scheduling heuristics for uniprocessor MC systems, which employ job skipping as a technique of mitigating overload. It is shown that the usage of evolved scheduling heuristics can improve user-defined quality of service metric for LO tasks in HI mode. It is demonstrated that these heuristics are computationally efficient, which makes them suitable for implementation in real-time embedded systems. Additionally, it is shown that these heuristics acquire generality in the training process and can

be used for different MC task sets and that the evolution process can be easily customized for different performance metrics.

5.3 Preliminaries

5.3.1 System model

In this chapter, synchronous dual-criticality task systems are investigated. It is assumed that time is discrete, i.e., any particular time instant t is integer. The system is comprised of task set $\mathscr{T}$, and each of n tasks is represented as a set $\tau_i = \{\vec{C}_i, T_i, \vec{D}_i, L_i\}$ where $\vec{C}_i \in \mathbb{N}^2$ is monotonic non-decreasing vector of WCETs, $T_i \in \mathbb{N}$ is the period, $\vec{D}_i \in \mathbb{N}^2$ is monotonic non-decreasing vector of deadlines and $L_i \in \{LO, HI\}$ is the criticality level of a task. Tasks can have either low criticality ($L_i = LO$) or high criticality ($L_i = HI$), $C_i(HI) \geq C_i(LO)$, $D_i(HI) \geq D_i(LO)$ and $D_i(HI) = T_i$. Tasks periodically release jobs and the currently active job of task τ_i is described with set $J_i = \{c_i, d_i\}$ where $c_i \in \mathbb{N}$ is the remaining execution time of the currently active job of task τ_i and $d_i \in \mathbb{N}$ is its remaining time to deadline at some time instant. Note that these variables, namely $c_i(t)$ and $d_i(t)$, are time dependent. However, for brevity, the time instant argument t is omitted. J_{ij} denotes the j-th released job of task τ_i.

During runtime, the system behaves like a classical adaptive mixed-criticality system as discussed in sections 3.4 and 3.5. Therefore, the system mode is described with criticality indicator Γ. The system starts in the LO mode ($\Gamma = LO$). The system switches to the HI mode ($\Gamma = HI$) when a job of HI task in the system executes for its LO execution time without signaling completion. In the proposed approach LO jobs are not discarded and their computation time and execution frequency are not reduced, which in turn can cause an overload condition. Although LO tasks in HI mode are not discarded, the schedulability definition for the whole system (Definition 29) remains the same as in the common approaches to MC scheduling.

Definition 29. ***MC-schedulability***. *A task set is MC-schedulable if jobs of all tasks meet their deadlines in LO mode and jobs of HI tasks meet their deadlines in HI mode.*

Here, we allow LO jobs to miss their deadline in HI mode, i.e., they are treated as "soft real-time" or non-critical tasks and they are scheduled in *best effort* fashion. When a job of LO task τ_i misses its deadline in HI mode, it is dropped, i.e., **skipped**, and new job of task τ_i is accepted. Note that as in the classical adaptive mixed-criticality approach in LO mode jobs of all tasks, LO and HI, have to be executed to completion as it is stated in Definition 29.

Scheduling of LO jobs in HI mode is done using the scheduling policy $\mathscr{S}$. The scheduling policy $\mathscr{S}$ can be described with a priority function $\pi_i(t)$ which at each time instant t assigns a priority to the active jobs in the system based on the current system state $\xi(t)$. The system state in this context at least includes a set of currently active jobs $\mathscr{J} = \{J_i, J_j, J_k, ...\}$, i.e.,

$\xi(t) = \{\mathscr{J}\}$. However, it can include any additional dynamical variables or parameters that are needed for evaluation of the priority function. This will be clarified further later in this chapter. Based on the current system state $\xi(t)$, a priority function returns a single real value for each active job in the system. Moreover, the lower the value of $\pi_i(t)$, the higher the priority of the currently active job of task τ_i. Such a definition enables defining scheduling policies in a relatively simple manner. For instance, the earliest deadline first (EDF) scheduling policy can be described with $\pi_i(t) = d_i$, i.e., the lower the deadline, the higher the priority. If two active jobs have equal priority, e.g., $\pi_i(t) = \pi_j(t)$, the priority of job, which is released earlier is higher. If jobs are released at the same time instant and have equal priority, i.e., $\pi_i(t) = \pi_j(t), i < j$, job of task with the lower index is higher, which in this case is τ_i. Note that throughout the chapter, scheduling policies are referred to as heuristics or priority assignment functions.

5.3.2 Motivational problems

To better illustrate the problem and provide adequate motivation for the approach that is investigated in this chapter, two small numerical examples are provided. In both examples, schedules are constructed and observed for the synchronous arrival sequence of all tasks in HI mode, i.e., with regard to HI WCET of tasks. For clarity, the synchronous arrival sequence of a task set is a sequence in which all tasks arrive with their minimum interarrival time, i.e., period [37]. Note that instances in both examples are overloaded in a sense that it is not possible to schedule all jobs of every task in the system. However, with an appropriate scheduling algorithm, schedulability of HI tasks in the system can be guaranteed. Such an on-line scheduling algorithm at some time instant can be described with two steps:

1. Schedule jobs of HI criticality tasks according to the EDF scheduling policy if they are active.
2. Schedule jobs of LO criticality tasks according to scheduling policy $\mathscr{S}$ if no jobs of HI criticality tasks are active.

Although the schedulability of LO criticality tasks cannot be guaranteed, scheduling policy $\mathscr{S}$ has to reduce the total number of skips in the system.

Example 11. *Consider the MC task set given in Table 5.1. The goal is to find a scheduling policy which minimizes the total number of skips of LO criticality tasks in the hyperperiod of the synchronous arrival sequence in HI mode of the input task set. Firstly, we can show that it is not possible to schedule all jobs since the system is overloaded, i.e.,* $U(HI) > 1$*:*

$$U(HI) = \sum_{i}^{n} \frac{C_i(HI)}{T_i} = \frac{2}{10} + \frac{4}{6} + \frac{1}{5} \approx 1.07 \tag{5.1}$$

Therefore, at the minimum at least one job has to be skipped. The first, i.e., a naive or an

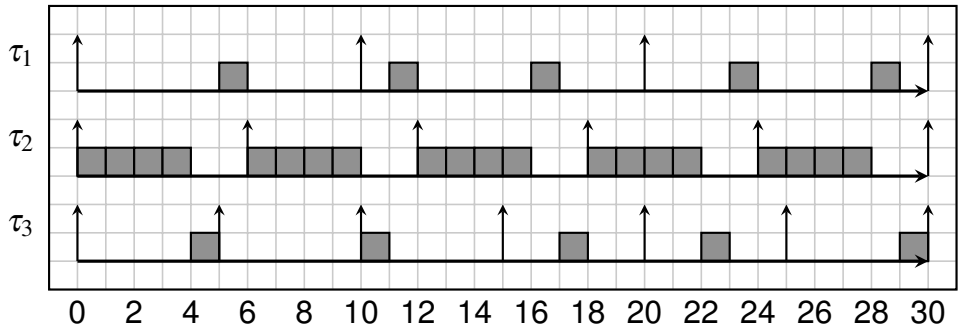


Figure 5.1: Low-criticality tasks scheduled according to the EDF scheduling policy for task set in Table 5.1.

intuitive, approach would be to choose the EDF scheduling policy for scheduling the instances, i.e., jobs, of LO criticality tasks. Such a schedule is shown in Fig. 5.1. Note that LO jobs are executed only when there are no active jobs of HI task τ_2. Vertical arrows in scheduling diagrams in Figs. 5.1-5.5 denote the periodic arrival of jobs.

Table 5.1: Task set parameters for Example 11.

Task	$C_i(LO)$	$C_i(HI)$	$T_i = D_i$	L_i
τ_1	2	2	10	*LO*
τ_2	2	4	6	*HI*
τ_3	1	1	5	*LO*

If we count the skipped instances in Fig. 5.1, we see that 2 jobs are skipped, i.e., J_{11} and J_{32}, i.e., they did not execute to completion, which requires 2 and 1 time units, respectively. The second approach is to try to find alternative scheduling policy, which would minimize the number of skips. If we observe the available time carefully, we can see that if we schedule the LO jobs according to their remaining execution time c_i, we might be able to reduce the number of skipped jobs. Indeed, such a schedule produced by the shortest remaining time first (SRTF) scheduling policy is shown in Fig. 5.2. We can see that the number of skipped instances is reduced to the minimum and that only one instance is skipped, i.e., J_{11}.

In the latter example, we can see that by choosing the SRTF policy, we managed to reduce the number of skips to the minimum. Naturally, it is unlikely that this is an optimal approach. It is demonstrated that the SRTF policy is not optimal in the following example.

Example 12. *Consider the MC task set given in Table 5.2. The goal is to find a scheduling policy which minimizes the total number of skips of LO criticality tasks in the hyperperiod of the synchronous arrival sequence in HI mode of the input task set. Firstly, we can show that it*

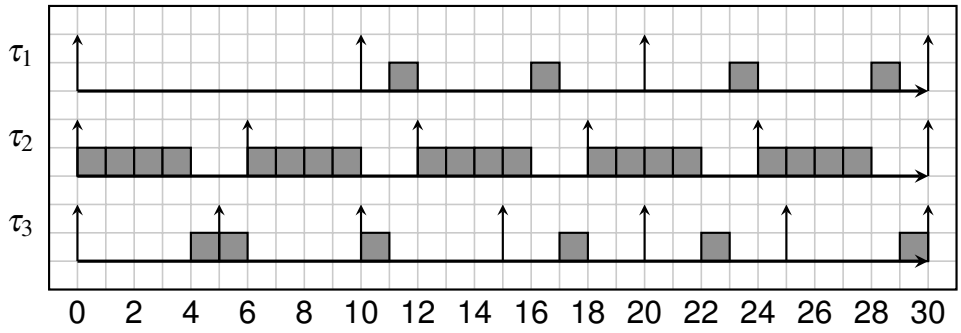


Figure 5.2: Low-criticality tasks scheduled according to the SRTF heuristic for task set in Table 5.1.

is not possible to schedule all jobs since the system is overloaded, i.e., $U(HI) > 1$*:*

$$U(HI) = \sum_{i}^{n} \frac{C_i(HI)}{T_i} = \frac{2}{10} + \frac{1}{15} + \frac{4}{5} \approx 1.07 \tag{5.2}$$

Table 5.2: Task set parameters for Example 12.

Task	$C_i(LO)$	$C_i(HI)$	$T_i = D_i$	L_i
τ_1	2	2	10	*LO*
τ_2	1	1	15	*LO*
τ_3	2	4	5	*HI*

Similarly as in the Example 11., we can apply the scheduling algorithms to the task set. Obtained schedules for the EDF and the SRTF policy are depicted in Fig. 5.3 and Fig. 5.4, respectively. We can see that both policies yield the same number of skips, i.e., the total of 2 *skipped instances. In the case of the EDF policy, jobs* J_{12} *and* J_{13} *are skipped, and in the case of the SRTF policy,* J_{11} *and* J_{13} *are skipped. At this point, it may not be trivial to point to a priority function, i.e., scheduling policy, which would minimize the total number of skips. However, consider this priority function for scheduling of LO jobs:*

$$\pi_i(t) = \max\left(\frac{c_i + d_i}{d_i - \sum_{k=1|k\neq i}^{n} d_k}, c_i\right) \tag{5.3}$$

The proposed method of acquiring this priority function will be explained later in the chapter. Note that $\sum_{k=1|k\neq i}^{n} d_k$ *denotes the sum of remaining times to deadline of all LO jobs which are currently active and which belong to tasks other than task* τ_i*. The schedule obtained using the policy given with equation (5.3) is shown in Fig. 5.5. It can be seen that the total number of skips using this policy is minimal, and that only one job is skipped, i.e.,* J_{12}*. Fig. 5.5 contains values of the priority function given with equation (5.3) for LO tasks* τ_1 *and* τ_2 *at instants when*

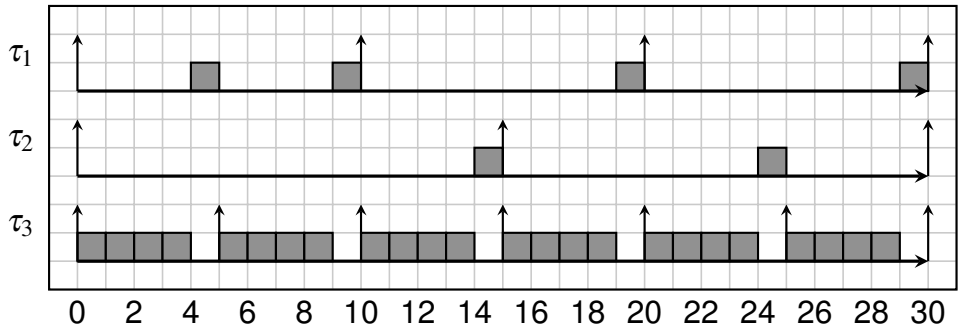


Figure 5.3: Low-criticality tasks scheduled according to the EDF heuristic for task set in Table 5.2.

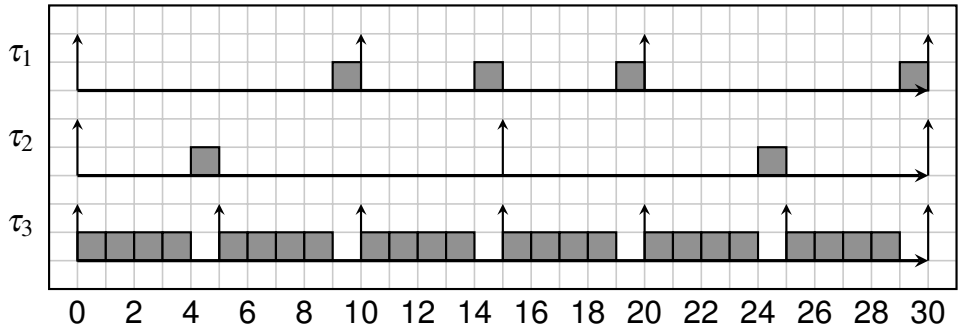


Figure 5.4: Low-criticality tasks scheduled according to the SRTF heuristic for task set in Table 5.2.

HI task is not executing. For instance, the priority of τ_1 *at time instant* 4 *is calculated as:*

$$\pi_1(4) = \max\left(\frac{2+6}{6-11}, 2\right) = 2 \tag{5.4}$$

Note that the division is protected, i.e., if denominator is 0*, we divide by* 1.

The latter examples show that an optimal solution in terms of the total number of skipped tasks may be obtained using an adequate priority function. Alternatively, an exhaustive algorithm could be devised, which would find assignments of LO jobs to the available execution time, i.e., the time when HI tasks are not executing. Although such an approach guarantees optimality for each problem instance, it has several significant drawbacks:

- The usage of such an approach in runtime, i.e., on-line scheduling, is limited since the computation time may be high. This makes the approach especially inapplicable in embedded systems with limited computing power, which are mostly used in the implementation of safety-critical and mixed-criticality systems.
- The static scheduling table generated by an exhaustive approach is optimal only for the given problem instance, i.e., task set.
- In case of MC task set, we would have to provide a scheduling table for any possible overload sequence, and not only for the synchronous arrival sequence. Since the potential number of overload sequences is generally very high as they can occur upon overrun of any HI task, this makes such an approach highly inadequate.

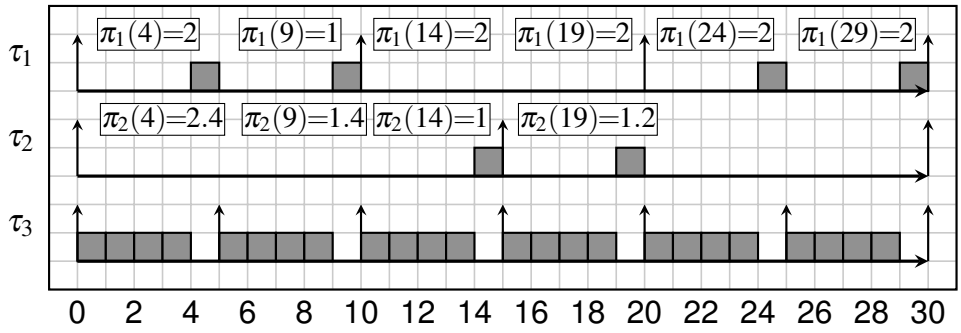


Figure 5.5: Low-criticality tasks scheduled according to the custom priority function given with equation (5.3) for task set in Table 5.2.

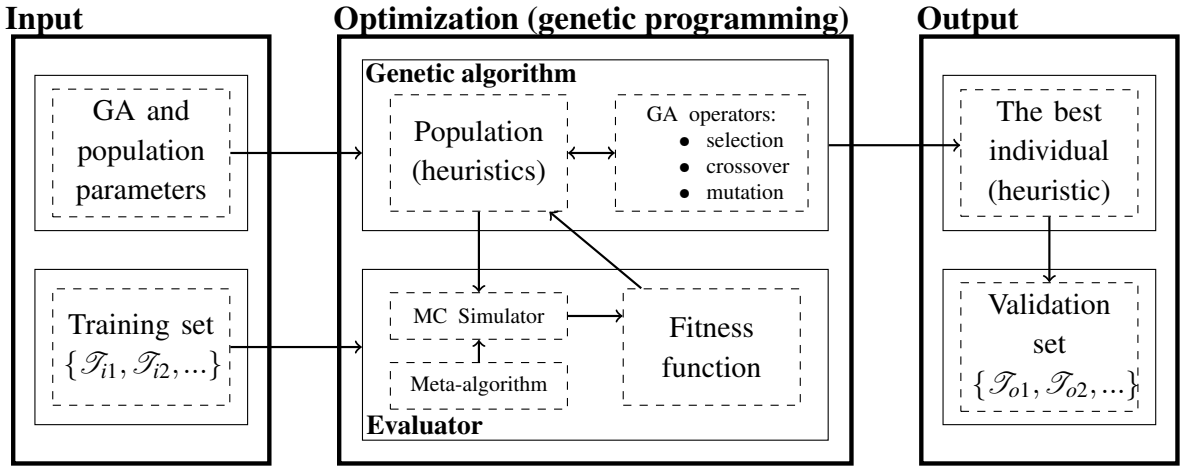


Figure 5.6: Genetic programming approach for design of heuristics for overloaded MC task sets.

- The size of the scheduling table grows exponentially with the hyperperiod which makes this approach unusable for large hyperperiods.

On the other hand, if an adequate scheduling policy, i.e., heuristic, is chosen for a task set or group of task sets during the design of the system, it may be used for different problem instances with similar properties, and similarly for different overload scenarios. Moreover, priority functions similar to the priority function of the EDF, i.e., $\pi_i(t) = d_i$, or the SRTF, i.e., $\pi_i(t) = c_i$, or even a priority function given with equation (5.3) can be efficiently computed during the runtime thus introducing the minimal overhead. Therefore, in the approach devised in this chapter, heuristics are designed off-line during the design of the system, and these heuristic can be efficiently implemented and used in an on-line approach, e.g., implemented as priority functions in real-time operating system.

5.4 Genetic programming for evolving priority functions

This section contains description of the genetic programming approach that is used for design of heuristics, i.e., priority functions, which are suitable for use in overloaded mixed-criticality systems with skipping. The usage of genetic programming for evolution of scheduling heuristics

is widely used in production scheduling [96]. In related literature, this approach was not used for scheduling of overloaded real-time or mixed-criticality systems. Genetic programming is a hyper-heuristic optimization approach [102]. Therefore, instead of searching the space of solutions, i.e., particular schedule for a task set, hyper-heuristics search the space of heuristics which, when adequately applied, generate schedules. A formal description of hyper-heuristic search problem can be found in [103] (see chapter 6.2).

5.4.1 Optimization framework overview

The proposed approach is depicted in Fig. 5.6. Inputs of the optimization process shown on the left-hand side in Fig. 5.6 are

- parameters of the genetic algorithm (GA) and parameters of the initial population,
- and training set of task sets.

The output of the optimization shown on the right-hand side in Fig. 5.6 is the best heuristic found in the optimization process. When the optimization process is finished, the performance of the best individual is evaluated on the validation set of task sets.

The optimization process consists of two distinct parts, i.e., a genetic algorithm and an evaluator. By applying the selection, crossover and mutation operators, genetic algorithm advances the generation of the population based on the fitness of the individuals, which is assigned by the evaluator. The evaluator itself defines the behavior of the system. More precisely, the evaluator simulates the behavior of the input task sets for each scheduling heuristic, i.e., priority function, in the population. Moreover, the evaluator assigns fitness to each heuristic from population based on its performance in the simulation. An important part of evaluator is the meta-algorithm which defines the scheduling algorithm in the system. More precisely, it defines how the heuristic, which is represented as an individual in the population, is applied in a problem instance. An example of a meta-algorithm has been already presented in the motivational examples, which specifies that heuristic is used for determining the priority of LO jobs when there are no HI jobs available. Similar, but more formal definition of the meta-algorithm, will be provided in the next subsection.

5.4.2 Population and individual representation in genetic programming

A priority function $\pi_i(t)$ can be represented as a tree structure. Tree structures are commonly used in solving problems using genetic programming including priority based scheduling [97]. Trees which represent heuristics, i.e., priority functions, consist of function and terminal nodes. Terminal nodes correspond to leafs in a tree and represent static and dynamic information about a job. Function nodes define operations over child nodes. Table 5.3 contains the description of function and terminal nodes used in the proposed approach. Traversing a tree yields a number

Table 5.3: The function and terminal node set

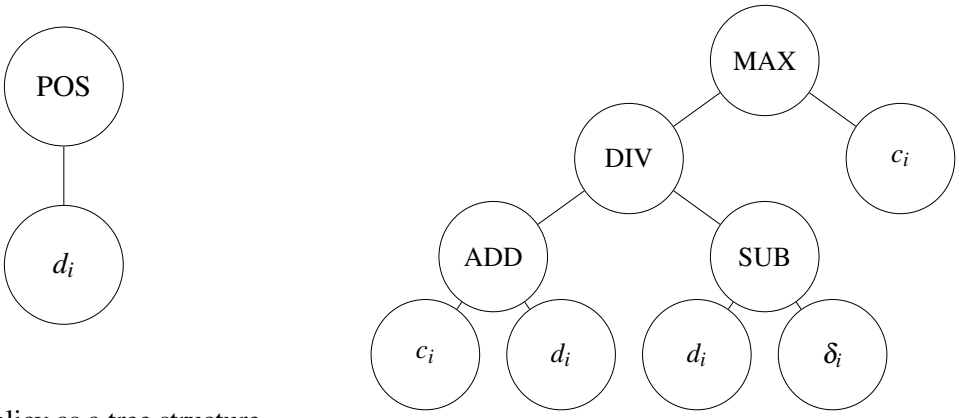
Function name	**Description**
ADD, SUB, MUL	addition, subtraction and multiplication of child nodes
DIV	protected division (returns first argument if division by zero is detected)
MAX, MIN	returns child node with larger or smaller value, respectively
POS	returns positive value of the single child node
Terminal name	**Description**
c_i	remaining execution time of active job of task τ_i
γ_i	sum of remaining execution times of jobs of LO tasks other than τ_i
d_i	remaining time to deadline of active job of task τ_i
δ_i	sum of remaining times to deadline of jobs of LO tasks other than τ_i
s_i	amount of time jobs of LO task τ_i were executing in HI mode in slack of HI jobs (used slack)
g_i^d	demanded grade of service of task τ_i
$g_i(t)$	grade of service of task τ_i at time t
σ_i	sum of grade of services of LO tasks other than task τ_i, i.e., $\sum_{j=1\|j\neq i}^{n} g_j(t)$

which indicates a priority level of a job in the system. As mentioned before in this chapter, it is assumed that a larger number, which is obtained by priority function, implies a lower priority.

For instance, Fig. 5.7a depicts the EDF policy. By traversing the tree in Fig. 5.7a we obtain the equation $pos(d_i)$ which evaluates to the remaining time to deadline of the active job of i-th task in a system, i.e., larger the deadline lower is the priority level. An example of a more complex tree structure is depicted in Fig. 5.7b. Note that tree shown in Fig. 5.7b corresponds to priority function given with equation (5.3) that is used in Example 12.

5.4.3 Genetic algorithm

To evolve heuristics, the generic genetic algorithm, which employs the k-tournament selection, sub-tree swapping crossover and sub-tree mutation, is used. Sub-tree swapping crossover operator randomly selects and swaps sub-trees of parent individuals. Sub-tree mutation operator replaces a randomly selected sub-tree with a new randomly generated sub-tree. As in the common genetic programming approaches, the maximum number of nodes in a tree and the maximum tree depth are limited before running the algorithm.



(a) EDF policy as a tree structure.

(b) Heuristic from Example 12 as a tree structure.

Figure 5.7: Representation of heuristics with trees

5.4.4 Synchronous MC task set simulator

To evaluate the performance of heuristics in the population, behavior of the synchronous MC task set is simulated, i.e., tasks are released periodically with no initial offset. In simulation, all tasks execute for the respective WCET depending on the system mode, i.e., tasks execute for their LO WCET in LO mode, and their HI WCET in HI mode. By simulating execution of jobs in HI mode for their respective HI WCET, overload scenarios are simulated. Moreover, all possible HI scenarios are simulated, i.e., scenarios which occur upon a criticality switch due to an overrun of HI task, up to the hyperperiod of the system. Note that if it is necessary to find heuristics for task sets with probabilistic behavior, i.e., probabilistic WCET, only the modification of this behavior in the simulator is needed without changes in genetic algorithm, fitness functions, etc.

In simulation, it is necessary to keep track of system state $\xi(t)$ for each HI mode scenario which is simulated. As it is stated before, system state $\xi(t)$ at least includes dynamical information about jobs. In addition to this basic information, the system state includes terminal node values which are specified in Table 5.3. These system state variables are used in formulation of a priority function and fitness function, i.e., performance metric.

5.4.5 Scheduling meta-algorithm

In this chapter, the design of heuristics for preemptive dynamical scheduling on synchronous uniprocessor MC platforms is investigated. The proposed scheduling approach consists of a manually defined meta-algorithm depicted in Alg. 15 and scheduling heuristics, i.e., priority functions, for LO jobs in HI mode (line 6 in Alg. 15). Scheduling algorithm is composed of two parts, namely LO part (lines 1-2) and HI part (lines 3-6). The LO part of the algorithm schedules all tasks in LO mode and guarantees fully operational behavior of the system, i.e., ensures that there are no deadline misses. The appropriate choice for a scheduling policy in

Algorithm 15 Scheduling meta-algorithm

```
1: if system in LO mode then
2:     schedule jobs using EDF w.r.t. LO deadlines
3: else if HI jobs available then
4:     schedule HI jobs using EDF w.r.t. HI deadlines
5: else if LO jobs available then
6:     schedule LO jobs using heuristic
7: end if
```

LO mode is the EDF policy with LO mode deadlines of HI tasks obtained by the greedy tuning approach devised by Ekberg and Wang in [48]. As in the other EDF with virtual deadlines approaches [104], the usage of virtually lower deadlines for HI tasks in LO mode significantly increases the schedulability. The HI part of the algorithm, schedules jobs in HI mode in the following manner: if there are active HI jobs in the job queue, schedule them using EDF policy w.r.t. conservative (true) deadlines, otherwise schedule LO jobs according to the desired policy, i.e., LO jobs are scheduled in slack time of HI jobs. This approach will not inflict on schedulability and the schedulability test obtained by applying corrections to deadlines devised in [48] is still sufficient. Moreover, Ekberg's greedy approach is relatively efficient in terms of feasibility [48]. As it is stated before, when the system is overloaded, LO jobs will not be able to complete by their deadlines. Albeit deadlines are missed, the schedule is still viable and certain performance metrics can be determined for LO tasks. Therefore, the goal is to find a heuristic that will schedule LO jobs as best as possible with regard to the defined performance metric. For instance, in examples 11 and 12, the performance metric is the total number of skipped LO jobs.

5.4.6 Fitness functions

Generally, the choice of the fitness function depends on particular requirements of an application. Moreover, quality of service itself depends highly on the nature of an application. Therefore, potential fitness functions are lateness, tardiness, the number of skipped jobs, or similar quality of service metrics. In this chapter, the *grade of service* (GoS) metric is used as a fitness function. Grade of service is often used as a metric in telecommunication systems [94], where it is defined as the ratio of the number of completed calls and the total number of calls. In this research as in [94], GoS of a LO task is the ratio of the number of jobs completed by their deadline and the total number of job releases in some time interval. GoS of task τ_i at time instant t can be defined as:

$$g_i(t) = \frac{\text{number of releases} - \text{number of skips}}{\text{number of releases}} \tag{5.5}$$

Average GoS of a task set $\mathscr{T}$ can be defined as:

$$G(\mathscr{T}) = \frac{1}{M}\sum_{j=1}^{M} G(\mathscr{T}_j^{HI}) \tag{5.6}$$

where M is the number of different HI, i.e., overloaded scenarios. $G(\mathscr{T}_j^{HI})$ is average grade of service of the j-th overloaded scenario and can be defined as:

$$G(\mathscr{T}_j^{HI}) = \frac{1}{n}\sum_{i=1|L_i=LO}^{n} g_{ij}(H) \tag{5.7}$$

where $g_{ij}(H)$ is the grade of service of task τ_i for any given HI scenario $\mathscr{T}_j^{HI}$ at the end of hyperperiod H.

In many applications, maximizing grade of service, may not be of benefit especially if it is known in advance that some functions have to execute with different level of GoS to satisfy user requirements. Therefore, similarly as in [94], *penalty* $p_i(t)$ at time instant t for a LO task in a set is introduced:

$$p_i(t) = g_i^d - \min(g_i(t), g_i^d) \tag{5.8}$$

where g_i^d denotes the minimal GoS level demanded by the i-th task in a task set. More precisely, if the actual grade of service $g_i(t)$ is lower than the demanded grade of service g_i^d, the penalty equivalent to the difference of these values is introduced. The term $\min(g_i(t), g_i^d)$ in the latter equation ensures that penalty is non-negative. Average penalty, i.e., $P(\mathscr{T})$, of task set is defined in a similar manner as average GoS, i.e., $G(\mathscr{T})$. More precisely:

$$P(\mathscr{T}) = \frac{1}{M}\sum_{j=1}^{M} P(\mathscr{T}_j^{HI}) = \frac{1}{M}\frac{1}{n}\sum_{j=1}^{M}\sum_{i=1|L_i=LO}^{n} p_{ij}(H) \tag{5.9}$$

where $p_{ij}(H)$ is the penalty of task τ_i for any given HI scenario $\mathscr{T}_j^{HI}$ at the end of hyperperiod H. Average grade of service, i.e., $G(\mathscr{T})$, and average penalty, i.e., $P(\mathscr{T})$, are the fitness functions. These metrics are used since it is easy to understand their relation to the performance of the system. Note that for brevity, $g_i(t)$ and $p_i(t)$ are denoted as g_i and p_i.

5.4.7 Flexibility of the approach

A great benefit of the proposed approach on an architecture level is its flexibility, modularity and automatism. This is especially important at the time of the system design. Consider a scenario in which a system designer has to design a scheduling approach which would minimize the total number of skips for LO tasks in a task set during the overload, i.e., HI scenario. In the classical real-time system design approach, one would either try to find an optimal or suboptimal

heuristic if possible, or an exhaustive search algorithm for a particular problem of minimizing the number of skips. Although this may be an effective approach, its main disadvantage is that it is potentially time consuming. Moreover, if the metric is changed from the total number of skips to the average penalty as defined in the previous section, it is highly plausible that devised heuristics and algorithms may not be applicable to those scenarios. However, with the proposed genetic programming approach, a heuristic can be generated for an arbitrarily chosen performance metric by changing only the fitness function in the evaluation and rerunning the training process. Similarly, let us assume that there are multiple different system states or that it is necessary to introduce acceptance tests for jobs, only the change in the scheduling meta-algorithm is required followed by rerun of the optimization process. Moreover, by manipulating the input, i.e., the training set of task sets, system designer can effectively custom-tailor the priority functions for scenarios of interest. For instance, system designer may have the need to provide a specific priority function which will only be used with a specific task set, thus having only one task set in a training set. In a different use-case, system designer would have to provide a more generic solution, using task sets with similar parameters as the training set. Using this approach, an optimal heuristic in Example 12 was found. The task set specified in Table 5.2 was used as the input, and the total number of skips as a fitness function. Moreover, the synchronous arrival sequence of tasks in HI mode was simulated. This demonstrates that the approach can be used for generation of optimal heuristics for specific problem instances. Similar approaches for determining optimal heuristics have been studied in the literature [105].

5.5 Evaluation

In this section, the effectiveness of the approach in generating heuristics is demonstrated in the context of maximizing the average grade of service defined with equation (5.6) and minimizing the average penalty defined with (5.9). As mentioned before, for obtaining these results, only the fitness function in the evaluation has to be modified. The performance of generated heuristics is evaluated with regard to the utilization of task sets in HI mode ($U(HI) = \sum_i^n \frac{C_i(HI)}{T_i}$), the number of tasks in the system (n), and the maximal allowed period (T_{max}) of a task in a task set. Additionally, two simple, but efficient, generated heuristics are dissected and explanation for their effectiveness is provided with regard to their respective metrics.

5.5.1 Genetic algorithm parameters

In the process of evolving heuristics, the number of generations was limited to 15. The tree depth was limited to 4, and consequently the number of nodes in a tree was limited to $2^{4+1} - 1 = 31$ node. The maximum initial depth of the tree was set to 4. There were 100 individuals in

population. Probability of mutation was set to 0.2. The size of the selection tournament was 3. The justification for usage of these parameters is provided in section 5.5.6.

5.5.2 Task set generation

For evaluation of the proposed approach, task sets were generated using several established methods for generation of utilizations and periods of task sets. First off, UUnifast algorithm [39] was used for generation of utilizations similarly as in the previous chapters since it is often used in measuring the performance of scheduling algorithms or schedulability tests in real-time systems. Generated utilizations correspond to the high-criticality utilizations, i.e., $u_i(HI) = \frac{C_i(HI)}{T_i}$, for each task in a task set. Secondly, for generation of harmonic periods the backward search approach from [64], which was discussed in the previous chapter was used. This method is adequate since the systems of interest, i.e., mixed-criticality and safety-critical systems, often employ harmonic periods due to the their beneficial properties such as predictability, schedulability and stability [54, 63]. The harmonic period assignment method developed in [64] enables generation of harmonic periods from period ranges, which is frequently needed in practice [62]. The method devised in this research was not used since additional parameters such as the distinct number of different period values do not have to be specified. Here, period ranges are generated in the following manner:

1. Firstly, a random period value T_i^r is chosen from the interval $[T_{min}, T_{max}]$ with the uniform distribution for each task in a task set, where T_{min} and T_{max} are the minimal and maximal allowed period of any task in any task set.
2. Secondly, an interval $\left[\frac{T_i^r}{2}, T_i^r\right]$ is formed from which a harmonic period is obtained using the method from [64] for each task in a task set.

The resulting task sets have harmonic periods and a hyperperiod which is upper bounded by the maximal allowed period in a task set, i.e., T_{max}. The low-criticality execution time $C_i(LO)$ for each task set is obtained as $\left\lceil \frac{C_i(HI)}{CF} \right\rceil$, where CF is the constant equal for each task, which is referred to as criticality factor and it is set to 2. The training and validation sets contain only task sets that are schedulable according to Definition 29. Schedulability is tested using Ekberg's schedulability test [48]. When the Ekberg's schedulability test is employed, LO deadlines of HI tasks are corrected to smaller values as in similar EDF with virtual deadlines approaches. This is important since in practice only feasible task sets are considered, i.e., task sets for which the execution of HI tasks is guaranteed under overrun of a HI task in any scenario.

In each experiment, a certain parameter is varied, i.e., $U(HI)$, n, or T_{max}, to see the effectiveness of the proposed method. For training, a smaller number of training task sets is generated in contrast to a larger number of generated validation task sets in order to show that the learned behavior is generalized. Separate experiments are run for each varied parameter. For each parameter, the genetic programming is run 10 times and the average performance of the best

individuals, i.e., heuristics, found in these consecutive runs is recorded. The default values of these parameters which are used for generating the task sets are:

- high-criticality utilization $U(HI) = 1.2$,
- the number of tasks in a task set $n = 10$,
- the maximal allowed period $T_{max} = 500$,

unless otherwise noted, i.e., unless the parameter is varied. Other task set parameters which remain fixed throughout the experiments are:

- criticality ratio CP , i.e., the fraction of HI criticality tasks in a task set, $CP = 0.5$,
- demanded grade of service g_i^d for each generated task, chosen from interval $[0,1]$ with uniform distribution.

5.5.3 Experimental results

In this section, the experimental results are shown and the devised method is compared to some naive approaches and approaches that can be found in the literature [94]. It has already been demonstrated in Example 12 that scheduling jobs of LO tasks in HI mode using the **EDF** or **SRTF** heuristics is suboptimal and that these are in fact naive approaches. More complex approaches for scheduling of non-critical jobs in an overloaded scenario that can be found in the related work [94] are:

- **LSUF** - *Least Slack Used First* heuristic schedules the job of a task which used the least amount of slack of HI tasks, i.e., s_i, at some time instant t.
- **LGoSF** - *Lowest Grade of Service First* schedules the job of a task with the lowest grade of service, i.e., $g_i(t)$, at some time instant t.
- **HPF** - *Highest Penalty First* schedules the job of a task with the highest penalty, i.e., $p_i(t)$, at some time instant t.

The common property of all these approaches is that they are single variable based, i.e., they take into consideration only one parameter or observed variable in the system to make a scheduling decision, which can be a possible pitfall as it can be seen in examples 11 and 12. Generated heuristics, on the other hand, use more information which may result in a better scheduling decision or at least in a better sequence of scheduling decisions. In this section, the experimental evidence for these claims is provided with regard to all varied parameters $U(HI)$, n, and T_{max} for the average grade of service and average penalty as quality of service metrics.

First off, scheduling heuristics are evaluated with regard to the utilization $U(HI)$ of the task sets. In experiment, 10 different task sets are generated for each utilization in interval $[1.05, 1.4]$ with 0.05 increment, which totals to $(\frac{1.4-1.05}{0.05}+1)\cdot 10 = 80$ task sets in a training set. Similarly, completely new task sets are generated for validation, i.e., it is not allowed for task sets from training set to be in the validation set. For validation, 1000 different task sets are generated for each utilization in interval $[1.05, 1.4]$, which totals to 8000 different task sets, i.e., the validation

set is 100 times larger than the training set. Fig. 5.8 shows the performance of all heuristics on the validation sets with regard to average grade of service at the end of hyperperiod. Note that **GP-avg** graph denotes the proposed genetic programming approach. As mentioned before, it corresponds to the average performance of the best heuristics generated over 10 consecutive runs of the genetic programming. It can be seen that the heuristics generated using the approach dominate other heuristic in terms of average grade of service of LO tasks in all overloaded scenarios. In the next subsection, the architecture of a single heuristic found in the experiments that dominates the greedy **SRTF** approach is discussed, and a brief explanation for the dominance is provided. The performance of that single heuristic is denoted with **GP-GoS**. Note that multiple heuristics were found that dominated the **SRTF** heuristic, but this specific heuristic was chosen due to its over-the-average performance, simplicity, and interpretability. It is interesting to notice that the **LSUF** heuristic yields slightly higher average grade of service than the **EDF** heuristic for higher utilization values, but still worse than the **SRTF** and evolved heuristics.

To obtain results shown in Fig. 5.9, the experiment is repeated using the average penalty as a fitness function. Similarly as before, the average performance of the heuristics, denoted with **GP-avg**, which is now generated for a different fitness function, i.e., penalty, dominates other approaches for scheduling LO jobs in HI mode. It can be seen that in this case, the **SRTF** policy performs worse than in case of maximizing grade of service. This is mainly due to the fact that minimizing the penalty and maximizing the grade of service are essentially different requirements. By scheduling the job with the shortest remaining time, the policy is biased to increase the grade of service of short jobs, thus starving the tasks with longer execution time or high grade of service demand, but maintaining high average grade of service. In contrast, it is obvious that such an approach is not efficient when different grade of service is required for each task, and in such a scenario evolved heuristics perform better. Moreover, in both cases, the best heuristics are found using the proposed genetic programming approach. Similarly, as in the case of the average grade of service, the **GP-Pen** graph represents the performance of the penalty minimizing heuristic, which is presented in the next subsection. Among all heuristics that are found that dominate the **SRTF** policy, this heuristic was chosen due to its performance and interpretability. In addition, it can be seen that the other evaluated heuristics, i.e., **HPF**, **LGOSF**, **LSUF**, perform slightly better than the **EDF** but significantly worse than the **SRTF** and evolved heuristics with regard to average penalty.

Note that in both cases, the increase in utilization decreases the grade of service and increases the average penalty, which is an expected behavior since the higher utilization corresponds to the higher overload in the system.

The performance of heuristics with regard to the different number of tasks in a task set is analyzed hereafter. Similarly as before, task sets are generated for each integer n in interval $[5,15]$ which totals to $(\frac{15-5}{1}+1)\cdot 10 = 110$ task sets for the training process and $(\frac{15-5}{1}+1)\cdot 1000 =$

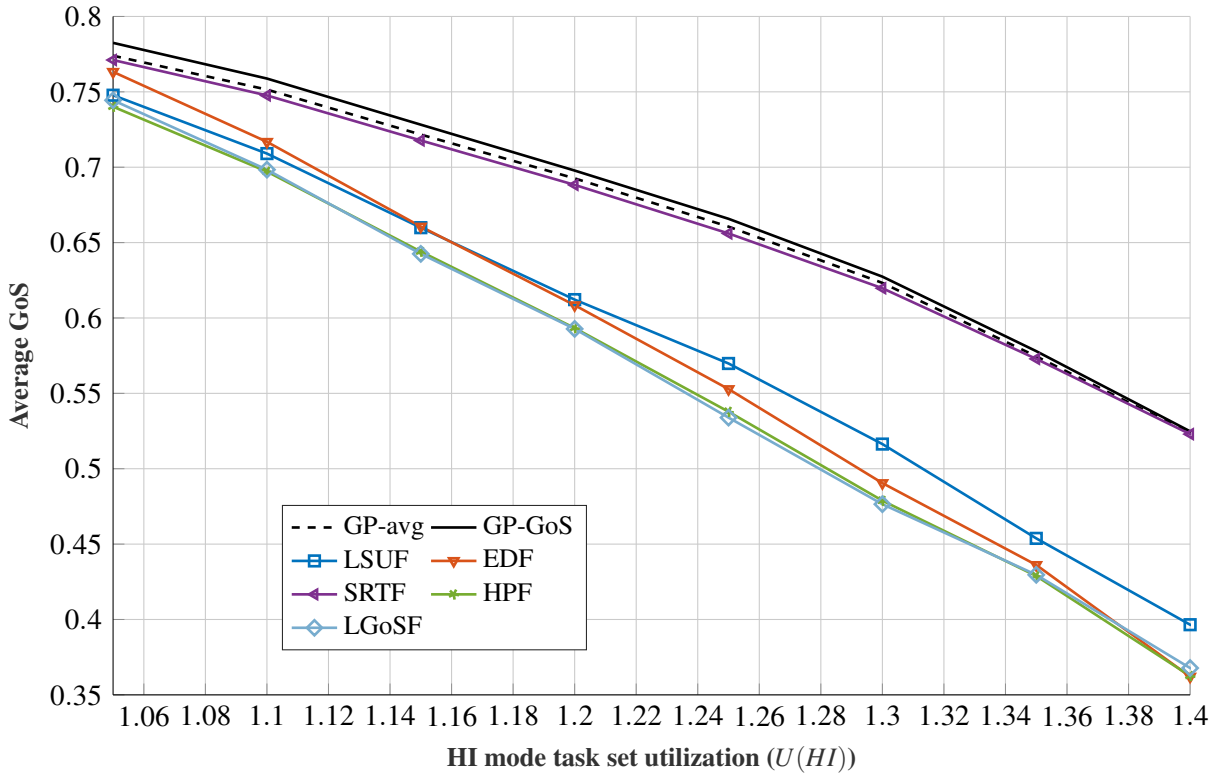


Figure 5.8: Average GoS for different HI mode utilization factors

11000 for the validation. The utilization of task sets in HI was set to 1.2, which makes the observed task sets overloaded. Fig. 5.10 shows the average grade of service of heuristics with regard to the different number of tasks in a task set. Similarly as before, the generated heuristics on average (denoted with **GP-avg**) and the heuristic **GP-GoS** achieve the best performance in comparison with other scheduling policies. After rerunning the training process with the average penalty as the fitness function, new results were obtained that are depicted in Fig. 5.11. The results show that, once again, the best performance is achieved by using the scheduling policies obtained by the proposed genetic programming approach. When other scheduling policies are observed, it can be seen that **LSUF** performs slightly better than other heuristics in terms of average grade of service, and **HPF** performs slightly better than other heuristics in terms of average penalty.

Note that increase in the number of tasks in a task set causes a slight increase in grade of service and decrease in the average penalty. This due to the fact that in larger task sets that are generated, there are more jobs of low-criticality tasks, which can be successfully executed.

Finally, the performance of the approach is evaluated with regard to the different values of maximum allowed period in a task set. For training, the total of 100 task sets with maximum allowed period $T_{max} = 500$ was generated. For validation, 100 task sets were generated for each maximum value of period T_{max} in interval $[500, 5000]$ with 500 increment which totals to $(\frac{5000-500}{500} + 1) \cdot 1000 = 10000$ task sets. Figs. 5.12-5.13 show that even when the maximum

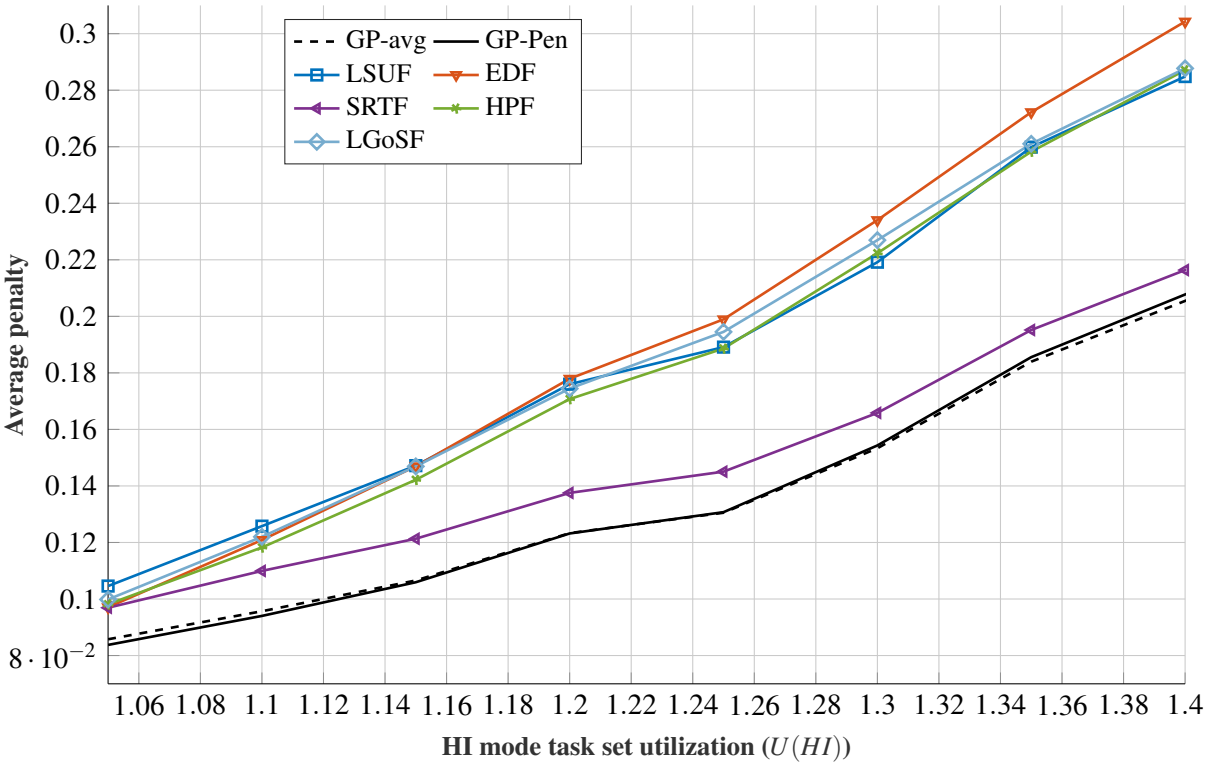


Figure 5.9: Average penalty for different HI mode utilization factors

allowed period is varied, the generated heuristics generalize well and perform better than the other approaches regardless of which quality of service metric is being used.

Generally, if the performance of the proposed approach is compared to the performance of the best single variable based heuristic, i.e., the **SRTF**, it can be seen that the relative increase in the grade of service achieved by the proposed approach is somewhat smaller than the decrease in the penalty. This is acceptable since the penalty is a metric that is more likely to be used in practice. However, for both performance metrics we find the best heuristics using the proposed approach. On the other hand, it can be seen the **SRTF** heuristic performs worse when the performance metric is changed. In addition, it is interesting to see that the wrong choice of single variable based heuristic can significantly degrade the performance of the system. For instance, if the **EDF** heuristic is chosen in a system in which the goal is to minimize the average penalty, the performance is degraded up to 10% in comparison with the proposed approach as it can be seen in Fig. 5.9. Moreover, in Fig. 5.8 the proposed approach yields heuristics which can improve performance up to 15% with regard to other single variable based heuristics. Note that in a general case, i.e., for an arbitrary performance metric, without thorough and time-consuming analysis, we do not know which heuristic has to be used. As it is demonstrated, the proposed approach can discover efficient heuristics for different objectives, i.e., fitness functions.

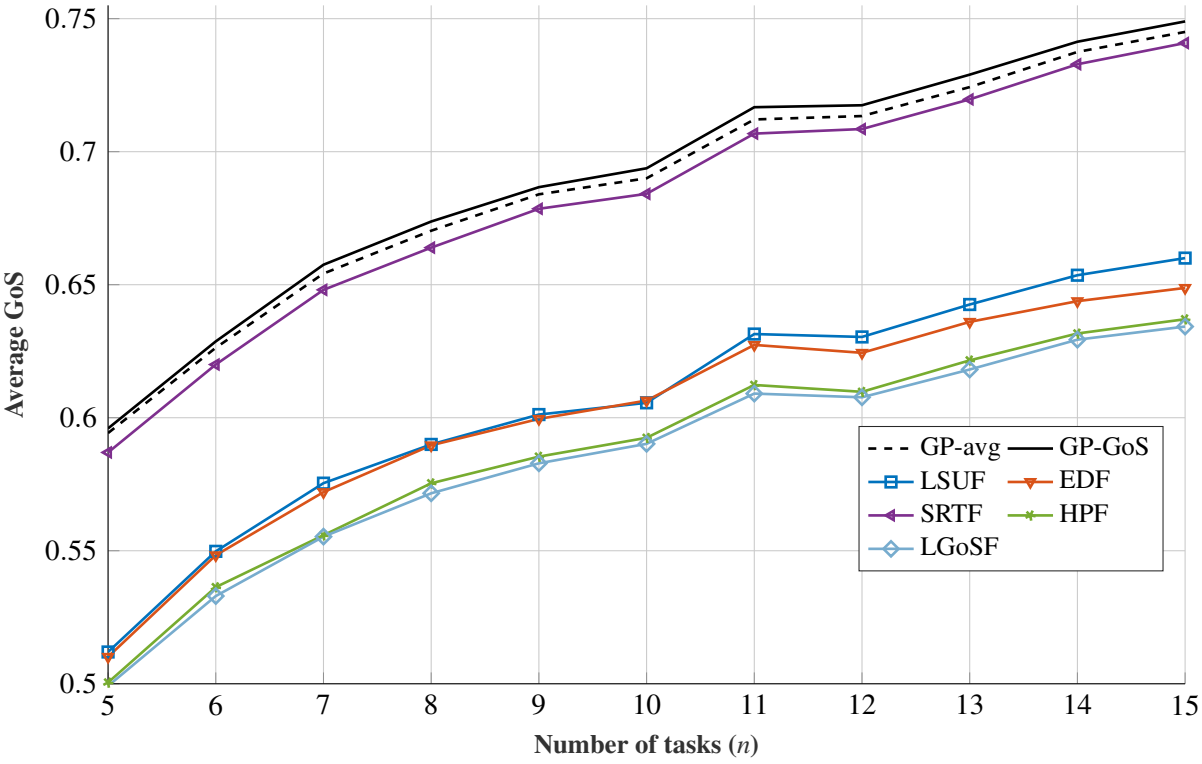


Figure 5.10: Average GoS for different number of tasks in the system

5.5.4 Dissecting the best heuristics

To better illustrate the inner workings of generated heuristics, the architecture of two heuristics is discussed. Namely, GoS maximizing heuristic, i.e., **GP-GoS**, and penalty minimizing heuristic, i.e., **GP-Pen**, shown as trees in Figs. 5.14-5.15, that we generated in the experiments. These heuristics dominate other naive scheduling policies in terms of the respective average performance metrics. Although they are not always the best choice tha was found, i.e., there are more complex priority functions with slightly better performance, they are dissected here due to their simplicity, efficiency, and interpretability.

The corresponding priority function $\pi_i(t)$ of the heuristic shown in Fig. 5.14 can be expressed as:

$$\pi_i(t) = \max(c_i, d_i) + \sigma_i + \min\left(\frac{d_i}{\gamma_i}, \sigma_i\right) \tag{5.10}$$

This priority function can be viewed as an approach that is greedier than the **SRTF** approach since it increases the priority of jobs which are more likely to increase the overall grade of service. The first term in the equation, i.e., $\max(c_i, d_i)$, will decrease priority of jobs with long remaining execution time c_i or a large time to deadline d_i. Moreover, in the case when the time to deadline is smaller than its execution time, the priority is decreased since the max function yields the bigger value, i.e., the execution time. This behavior can be viewed as a rudimentary acceptance test, i.e., it decreases the priority of jobs such that $c_i > d_i$. Adding σ_i, i.e., the sum of grade of service of other jobs will decrease the priority of jobs when there are jobs with larger

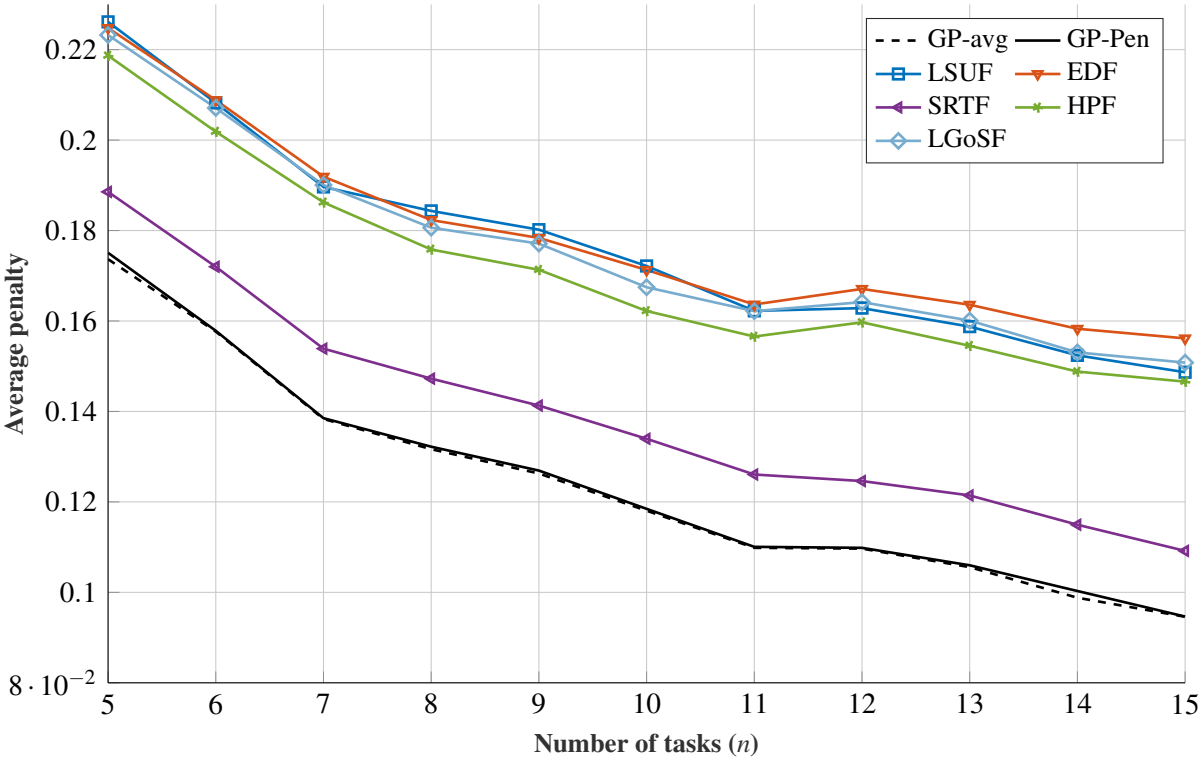


Figure 5.11: Average penalty for different number of tasks in the system

cumulative grade of service in the queue. For instance, if there are two jobs ready for execution which variables differ only in σ_i, the job with higher grade of service shall be executed, i.e., the corresponding σ_i is lower for the task with higher grade of service, and therefore its priority is higher. The last term in the equation is min function which can yield either σ_i or $\frac{d_i}{\gamma_i}$. There are two cases in which the term $\frac{d_i}{\gamma_i}$ is yielded by the expression. Either the deadline, i.e., d_i is low, or the remaining execution time of other jobs, i.e., γ_i is high. This, again, is a greedy rule which will favor jobs with the short time to deadline or short execution time. Alternatively, min function can return σ_i, which has a similar effect as described above. We can see that the generated heuristic exploits additional information about the system, and does not focus on a single observed parameter or variable. In this case, unlike the **SRTF** policy that uses only the remaining execution time c_i for priority assignment, the generated heuristic uses information about the remaining time to deadline d_i, the remaining execution time of other jobs γ_i, and the grade of service of other jobs σ_i.

Similarly, in the case of minimizing penalty, a simple generated heuristic shown in Fig. 5.15, i.e., **GP-Pen**, dominates the best single parameter policy, i.e, the **SRTF**, as it can be seen in Figs. 5.9, 5.11, 5.13. The corresponding priority function can be expressed as:

$$\pi_i(t) = \frac{\max(c_i, d_i)}{g_i^d} \tag{5.11}$$

An obvious pitfall of the **SRTF** heuristic is that it always chooses the job of a task with the

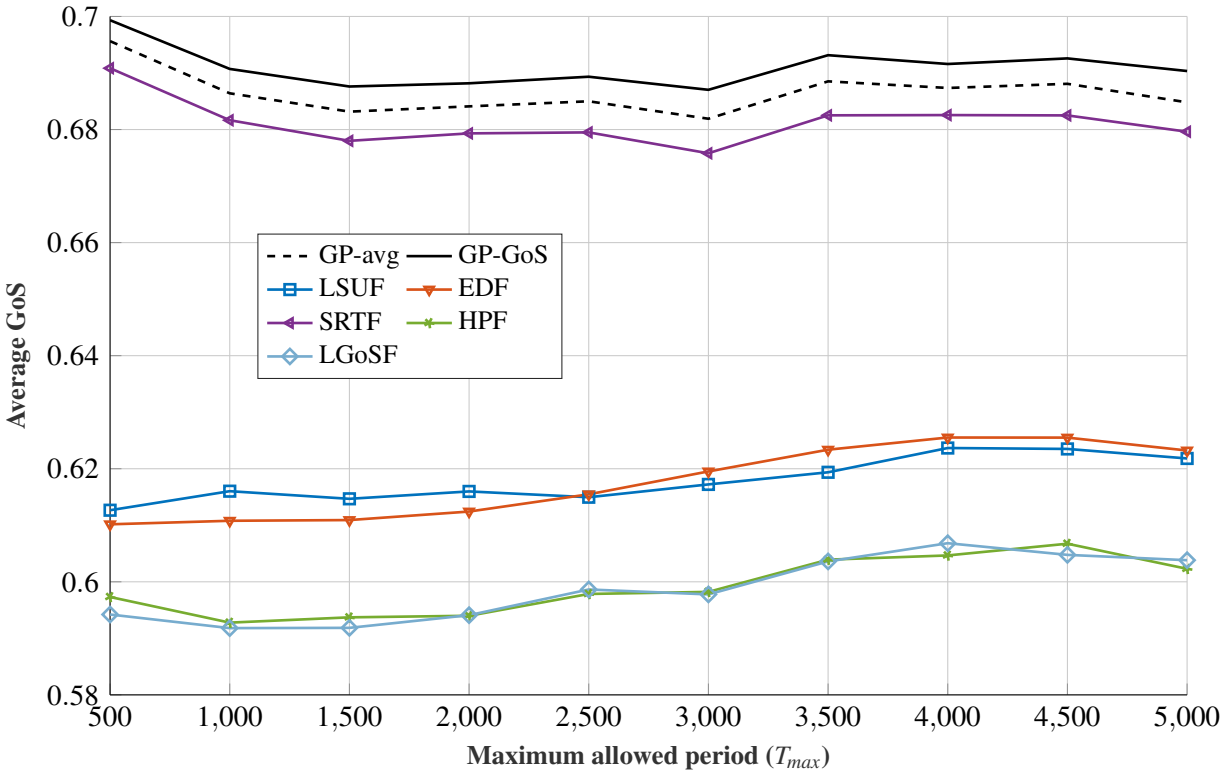


Figure 5.12: Average GoS for different values of the maximum allowed period

lowest remaining execution time. In a scenario in which a task with the shortest remaining execution time has met its grade of service demand, its execution will not decrease penalty. Moreover, its execution will prevent execution of other tasks which did not meet its demanded grade of service, which causes increase in the average penalty. The generated heuristic represented with equation (5.11) takes into consideration the demanded grade of service g_i^d. As it can be seen in (5.11), the lower the demanded grade of service the lower the priority of τ_i since $g_i^d \in [0,1]$. In (5.11), the term $\max(c_i, d_i)$ is used similarly as in the case of the grade of service maximizing heuristic given with (5.10). This further improves the performance since the information about remaining time to deadline d_i is utilized as well.

5.5.5 Runtime performance of the genetic programming

As it is demonstrated in the latter subsections, priority level of a job is obtained in constant time by traversing a tree or evaluating the corresponding equation, which makes the generated heuristics suitable for implementation and on-line usage in embedded systems. Since priority function of scheduling heuristics can be effectively computed in constant time, we do not investigate their timing performance. Note that in practice, i.e., in real-time operating systems, the priority of jobs is computed upon every invocation of scheduler which typically occurs on a millisecond timescale, and the computation time of the priority functions that are evolved using the proposed approach is much smaller. For instance, five operations are needed for implemen-

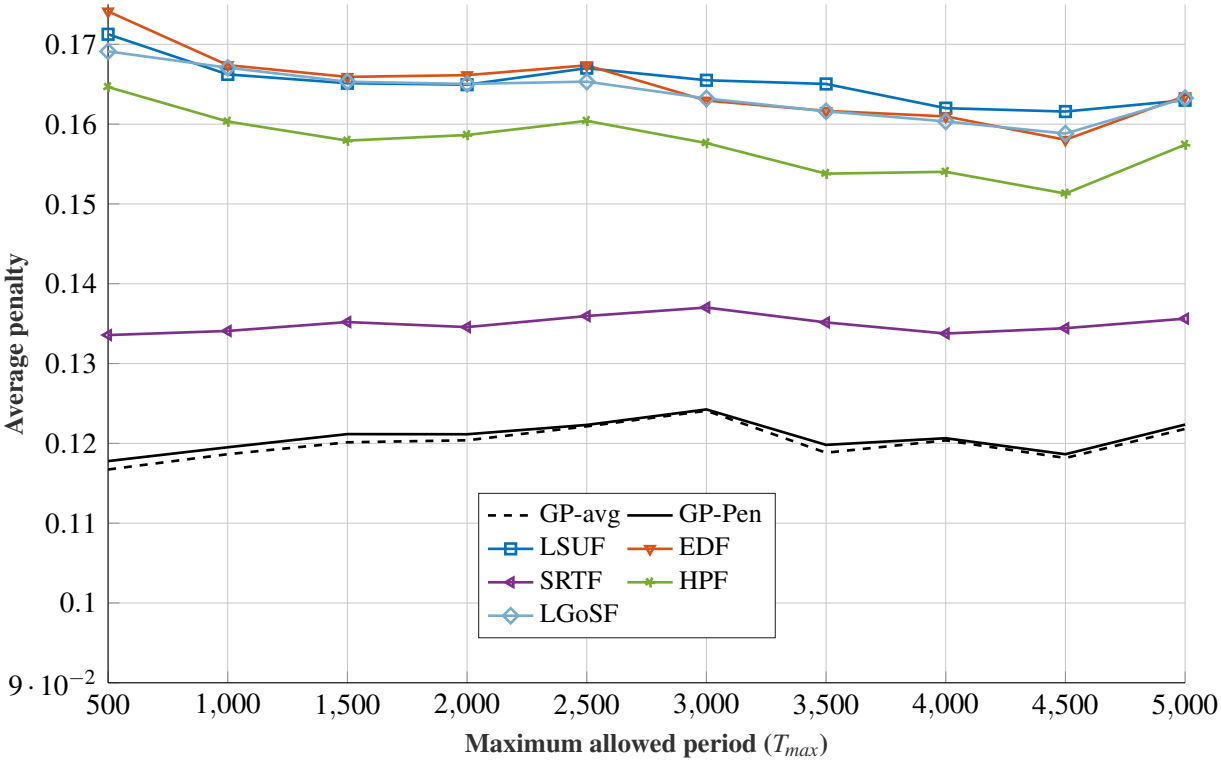


Figure 5.13: Average penalty for different values of the maximum allowed period

tation of the priority function **GP-GoS**. Moreover, for a full tree with the depth 4, $2^4 + 1 = 17$ operations are needed for the implementation.

On the other hand, the optimization procedure that is used off-line to generate heuristics is computationally intensive. Even though this is an off-line approach, it is important to show that the optimization procedure, i.e., genetic programming, is relatively efficient in terms of runtime for the systems of interest. This is important in the real-world applications since the system designer has to be able to rerun the training process repeatedly after changing the training set, the fitness function or the meta-algorithm in the procedure. Here, the runtime results are presented and discussed with regard to the number of tasks in a task set n, and the maximum period T_{max}, since these task set parameters influence the genetic programming runtime. Other task set generation parameters are set to their default values as specified in the end of section 5.5.2 and there were 100 task sets in the training set. Regarding the runtime measurements, it is worth noting that the in the experiments the *Evolutionary Computation Framework* (ECF) (current version available in [106]) which is written in C++ is used for implementation of the genetic programming optimization procedure. Moreover, implementations of the evaluator and meta-algorithm were written in C++, and each component was compiled with the highest speed optimization, i.e., `O3`. Additionally, the specifications of the computing platform are given in Table 4.1 in the previous chapter.

Fig. 5.16 shows the dependency of the average runtime per advancement of a single generation to the number of tasks in a task set. The most of the time for advancing the generation

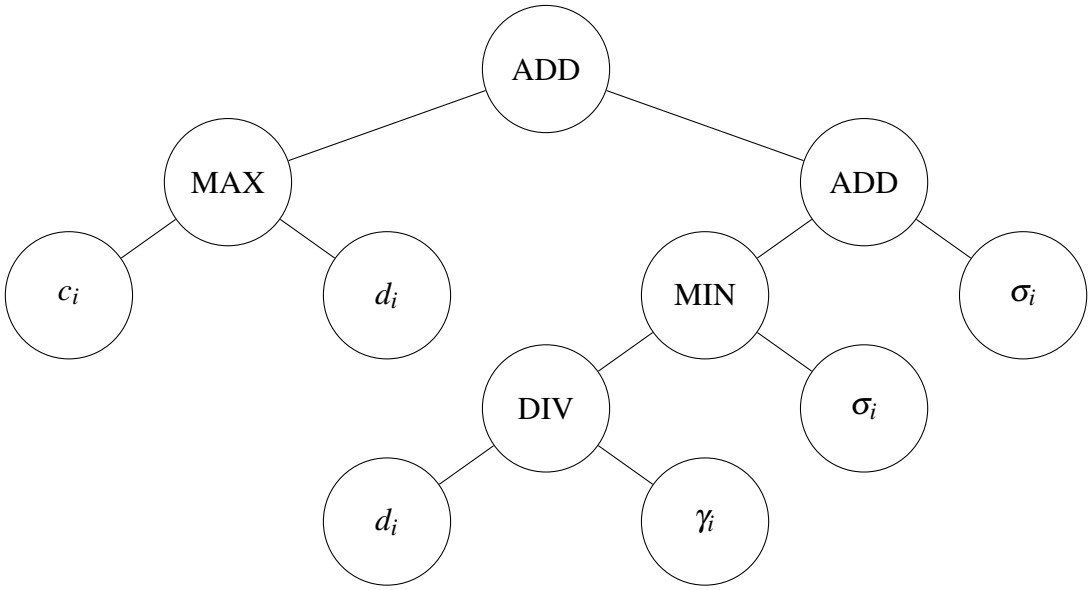


Figure 5.14: An example of an efficient heuristic (**GP-GoS**) found for maximizing average grade of service.

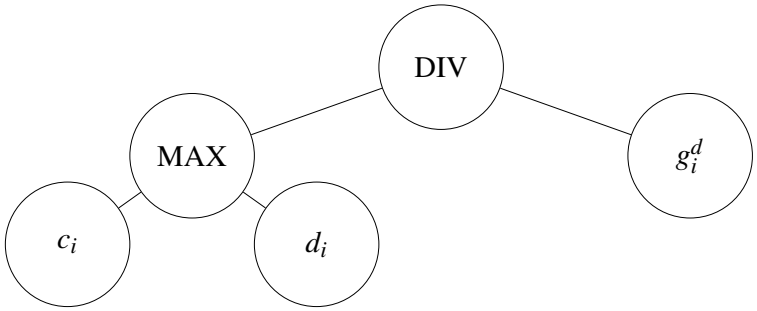


Figure 5.15: An example of an efficient heuristic (**GP-Pen**) found for minimizing average penalty.

is spent on the evaluation of the population, i.e., in simulation of the mixed-criticality system. It can be seen that the advancement of a single generation for the training set of 100 task sets comprised of 10 tasks takes approximately 6 seconds, which totals to 1 minute and 30 seconds when the number of generations is limited to 15. This is a reasonable off-line runtime. A larger growth in the runtime can be noticed when increasing the maximum allowed period as shown in Fig. 5.17. This is expected behavior since increasing the maximum period, and consequently the hyperperiod, increases the simulation time exponentially. Note that since the learned behavior generalizes well as it is shown in the experimental evaluation, task sets with lower number of tasks n and maximum period T_{max} can be used for discovering of efficient heuristics, which can reduce the training time if necessary.

5.5.6 The choice of genetic programming parameters

The choice of genetic programming parameters is important for obtaining adequate solutions, i.e., scheduling heuristics, with generalized behavior that yields good results in terms of a quality of service metric for different task sets or overload scenarios. To justify the choice of genetic

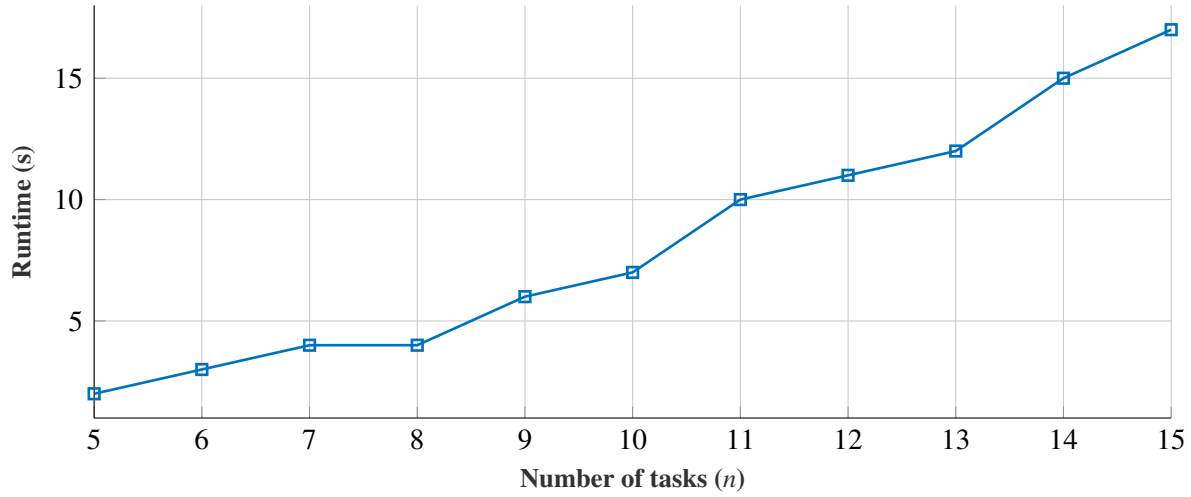


Figure 5.16: Influence of the number of tasks in a task set (n) on average runtime of a single generation.

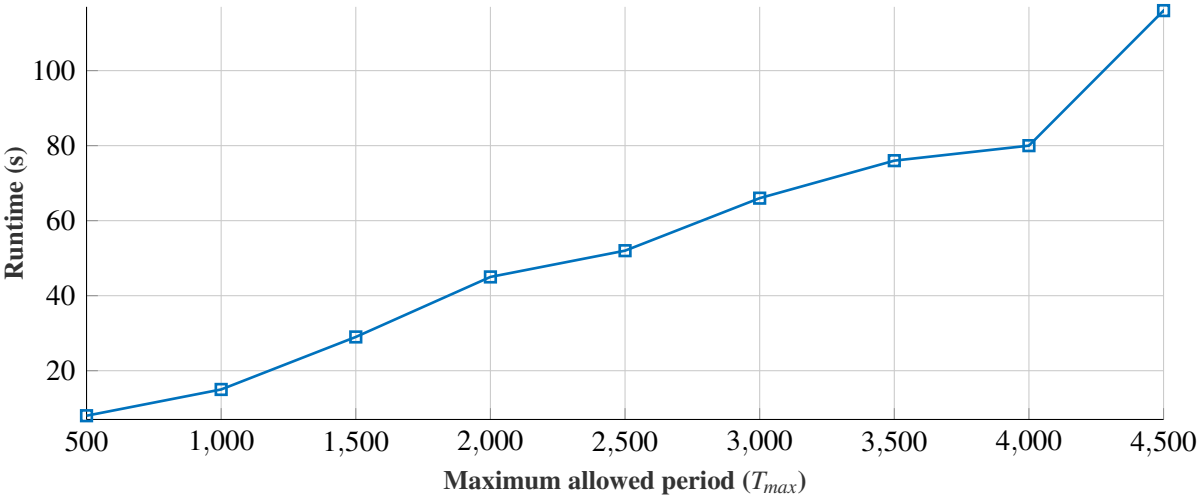


Figure 5.17: Influence of the maximum allowed period of task in a task set (T_{max}) on average runtime of a single generation.

programming parameters used in the previous section, the influence of the crucial parameters important for the efficiency of the approach is explored. As in the similar genetic programming approaches [97, 107, 108], the following parameters are identified as critical to runtime and the performance of the approach:

- the number of generations that is a stopping criterion for the optimization is initially set to 30,
- the genotype size, i.e., tree depth, initially set to 14,
- the population size initially set to 150,
- the training set size initially set to 200.

The task set generation parameters in experiments are set to their default values as specified in the end of section 5.5.2.

Figs. 5.18-5.21 show the average fitness functions values, i.e., grade of service and penalty, of the best individuals found in 10 consecutive experiments with regard to the observed genetic

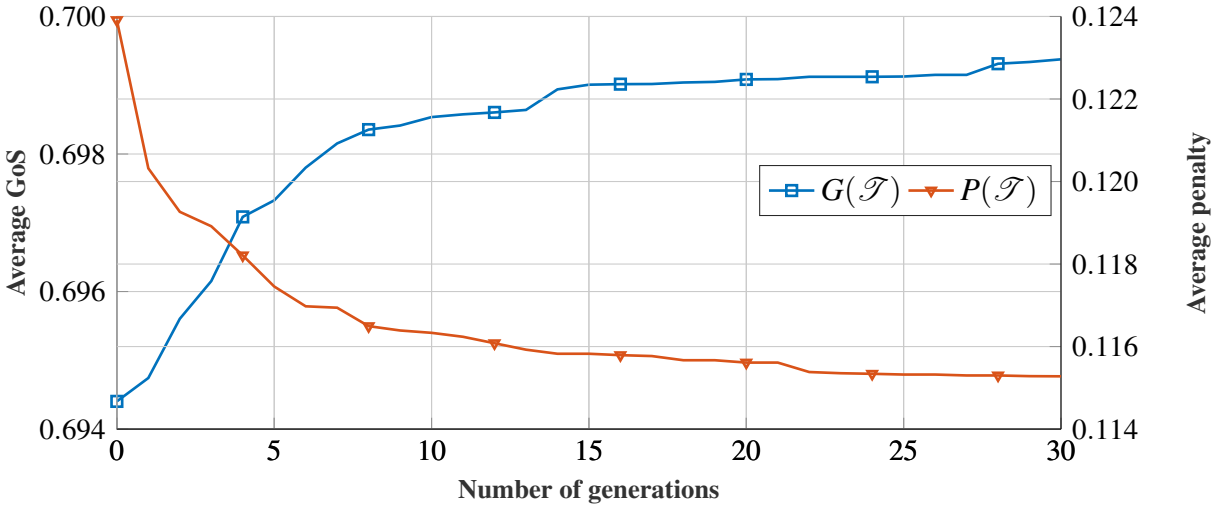


Figure 5.18: Influence of the number of generations on fitness.

programming parameter. In experiments, all parameters remained fixed except the parameter that was tuned. The tuning process was started with tuning the number of generations, and ended with tuning of the training set size.

Firstly, the number of generations was studied. In Fig. 5.18 it can be seen that the fitness functions do not drastically change after approximately 15 generations, and therefore the number of generations in experiments was set to 15 as a trade-off between the runtime and the fitness of the solution. Note that this number of generations is somewhat smaller but still on the same order of magnitude as in the literature [97, 107, 108]. Smaller number of generations indicates that terminal and function nodes have been chosen correctly and that initial random populations were a good starting point for the evolution.

Secondly, the influence of the maximum tree depth on performance of the approach was studied. In Fig. 5.19, it can be seen that larger tree depths, i.e., greater than 4, do not increase the fitness significantly. This is beneficial for the use cases of interest since smaller trees are preferred to larger trees due to the requirements for implementation in embedded systems, i.e., efficiency of priority function computation. Note that heuristics with zero tree depth correspond to the best single variable based heuristics, i.e., the **SRTF**.

Thirdly, the population size was studied. In Fig. 5.20, it can be seen that the increase in population size causes the increase in average grade of service, and similarly the decrease of average penalty. Although larger population sizes produce better fitness, a population size of 100 was chosen for experiments as a trade-off between the runtime of genetic programming and fitness of the solution.

Finally, the training set size was tuned. In Fig. 5.21, we see the performance of the best individuals, i.e., heuristics, on the validation set of size 10000. Note that this is a completely new validation set generated for study of the influence of the training set size on the fitness, i.e., it is different than validation sets used in section 5.5.3. It can be seen that for training set sizes

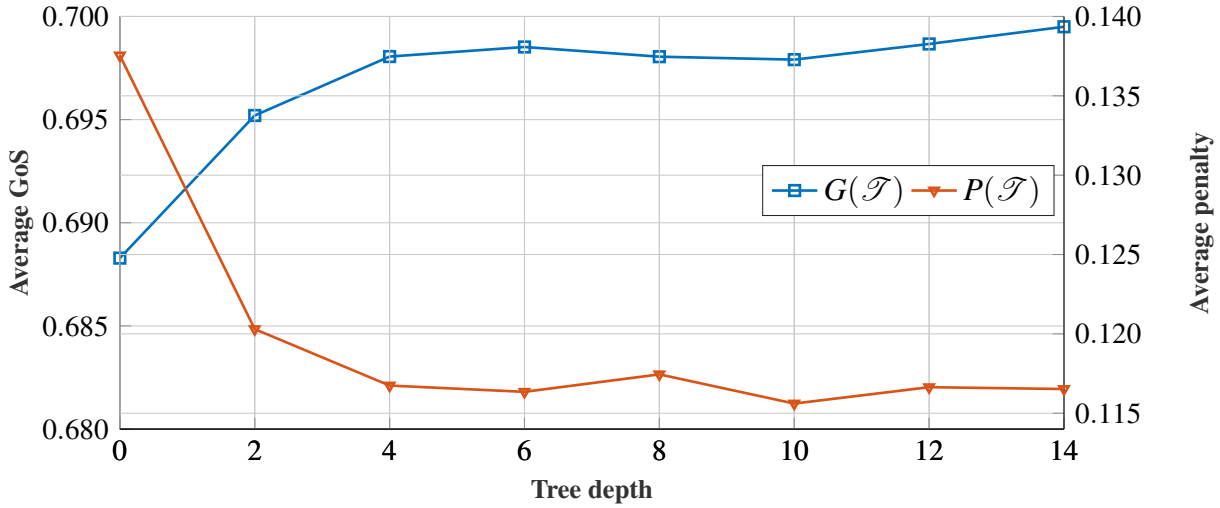


Figure 5.19: Influence of the tree depth on fitness.

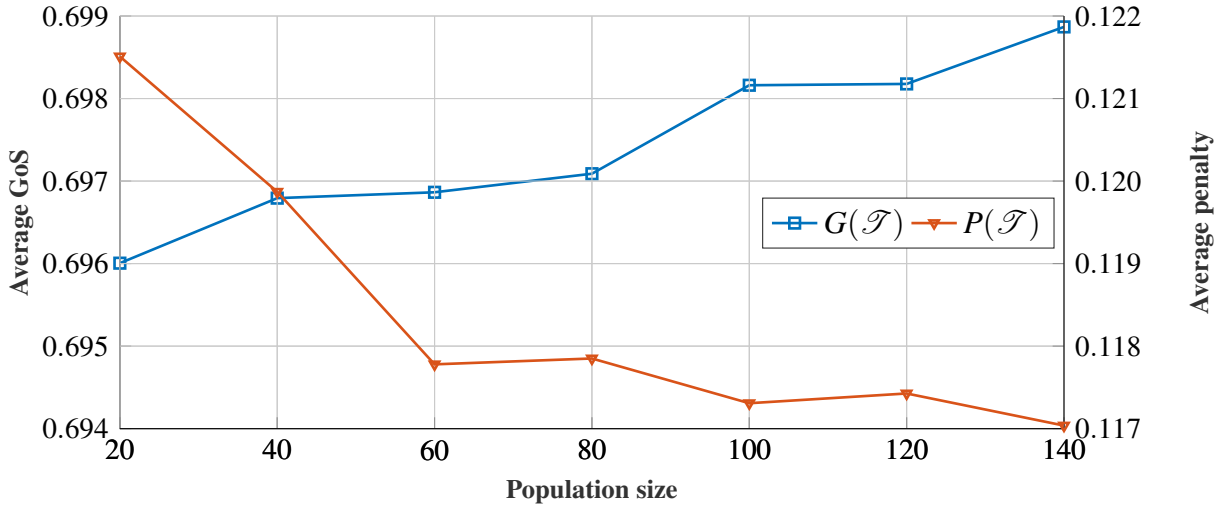


Figure 5.20: Influence of the population size on fitness.

larger than 50, the fitness of the validation set, which is much larger, does not drastically change. Furthermore, this is confirmed in the previous experiments in section 5.5.3 where training sets with approximately 100 task sets were used, and validation sets that were 100 times larger. The relatively small number of task sets that are required for the evolution of the generalized behavior may be connected to the large number of overload scenarios which are generated by a single task set. As it was mentioned before, in the simulation of a task set execution, overloaded scenarios are generated upon every possible criticality switch, i.e, any time a HI task depletes its allocated execution time without signaling completion.

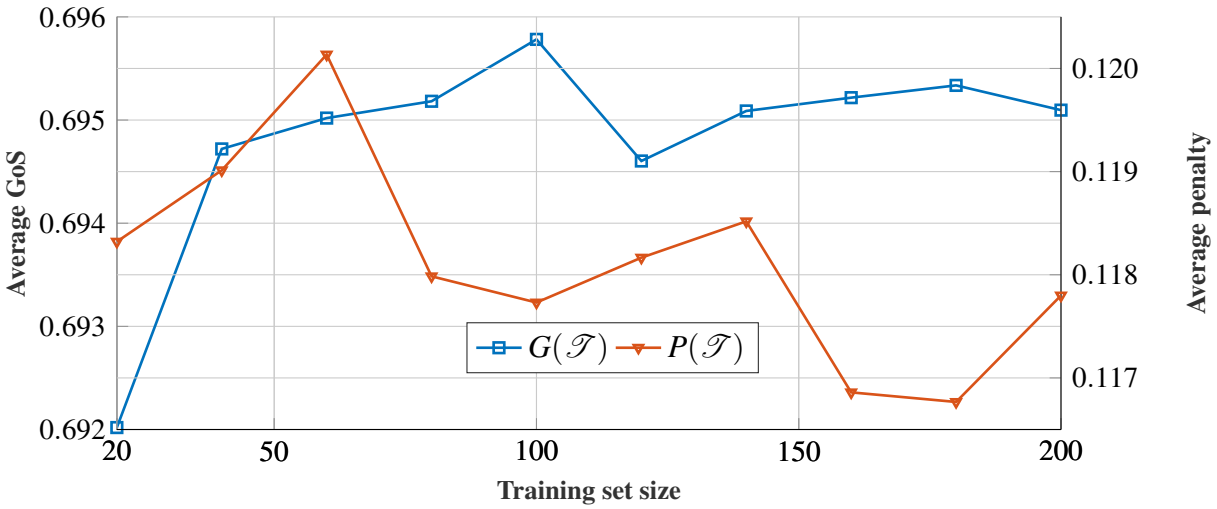


Figure 5.21: Influence of the training set size on fitness.

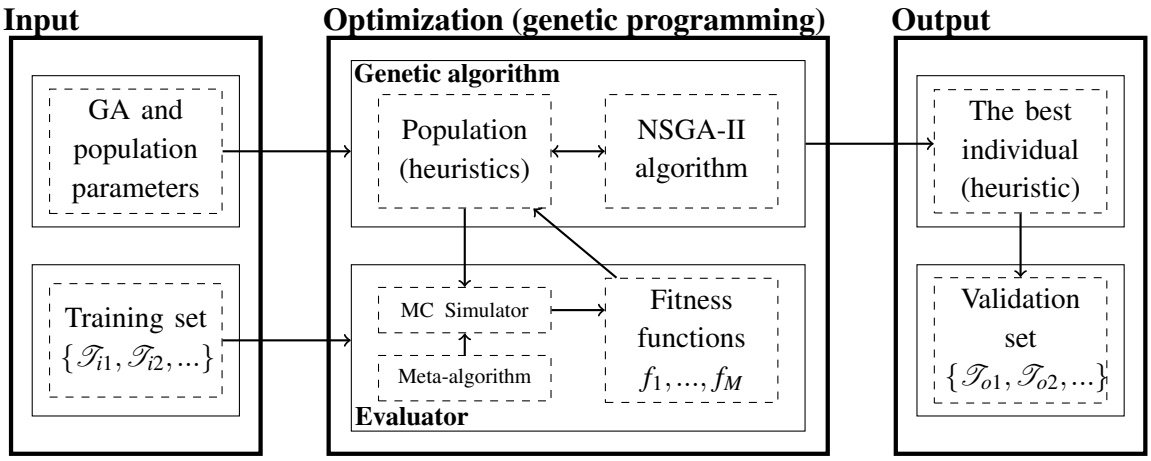


Figure 5.22: Genetic programming approach for design of heuristics for overloaded MC task sets with multiple objectives.

5.6 Optimizing heuristics for multiple objectives

An issue with the aforementioned genetic programming technique is the usage of a single fitness function in the design of the heuristic. In the previous section, it was explained how the usage of average grade of service as a fitness function makes the **SRTF** heuristic efficient. However, the grade of service of tasks with long execution time will be decreased since the **SRTF** heuristic prefers jobs with the shortest remaining execution time, and consequently the lower WCET. To resolve this issue, a multi-objective optimization approach is needed, which enables adding the fairness as the second objective in the optimization. Since the entire proposed optimization framework is modular, minimal changes are necessary to modify the approach. The revised framework is depicted in Fig. 5.22. Note that in comparison with Fig. 5.6, only the genetic algorithm is changed and an additional fitness for the population is added.

As it can be seen in Fig. 5.22, the genetic algorithm used for the multi-objective optimization is non-dominated sorting genetic algorithm (NSGA-II) introduced in [109]. The NSGA-II algorithm is frequently used as a means for stochastic multi-objective optimization in a lot of domains [110] including the genetic programming for evolving dispatching rules for scheduling problems [111, 112]. For completeness and clarity, a brief clarification of the inner workings of the algorithm, which is depicted in Alg. 18, is provided in the following subsection.

5.6.1 Non-dominated sorting algorithm (NSGA-II)

The NSGA-II algorithm introduced in [109] is based on the fast non-dominated sorting approach which makes it efficient in comparison with similar approaches. Additional reason for the long-term popularity of the algorithm is its elitism and the efficient parameter-less procedure for maintaining the diversity based on the crowding-distance computation.

Prior to illustrating the mechanism of the algorithm, some basic notation has to be intro-

duced:

- P_t denotes the current population at t-th generation,
- Q_t denotes the offspring population at t-th generation,
- the population P_t consists of individuals denoted with I_i,
- the individuals are sorted into sets that are typically called fronts $\mathscr{F}$,
- the vector $\vec{f}$ contains M different fitness functions that are optimized, N denotes the number of individuals in the population.

Non-dominated sorting. Non-dominated sorting is a procedure for sorting a population into a different non-domination levels. The dominance itself can be defined as follows.

Definition 30. ***Dominance.*** *An individual I_i dominates the other individual I_j if the following conditions are true:*

$$\begin{aligned} f_k(I_i) &\geq f_k(I_j), \forall k \in [1,...,M] \\ f_k(I_i) &> f_k(I_j), \exists k \in [1,...,M] \end{aligned} \tag{5.12}$$

In other words, I_i dominates I_j if it is not worse than I_j for any objective, and I_i performs better with regard to at least one objective. For clarity, note that in the context of multi-objective optimization individuals correspond to the solution of the problem. Formally, the domination of I_i over I_j is denoted as $I_i \preceq I_j$.

The time complexity of the naive algorithm for sorting the population into the non-dominated sets is in $O(MN^3)$. Such an approach requires at least MN^2 comparisons for creating a front, i.e., MN^3 steps to create N fronts. The authors in [109] proposed an algorithm for fast non-dominated sorting with better time complexity, i.e., $O(MN^2)$. This algorithm is depicted in Alg. 16. The fast non-dominated sorting algorithm is used in the implementation of the NSGA-II algorithm depicted in Alg. 18 in line 7. In the first part of the algorithm, the sets S_p of individuals which are dominated by individual p are determined. In addition, each individual is assigned a domination count, which tracks the number of individuals that dominate it. Based on these sets and the domination count in the second part of the algorithm, the fronts are generated iteratively.

Algorithm 16 Fast non-dominated sorting

Input: P - population

Output: $\mathscr{F}$ - non-dominated sets

```
1:  for each p ∈ P do
2:      S_p ← ∅
3:      n_p ← 0
4:      for each q ∈ P do
5:          if p ≺ q then
6:              S_p = S_p ∪ {q}
7:          else if q ≺ p then
8:              n_p ← n_p + 1
9:          end if
10:     end for
11:     if n_p = 0 then
12:         p_rank ← 1
13:         F_1 = F_1 ∪ {p}
14:     end if
15: end for
16: i ← 1
17: while F_i ≠ ∅ do
18:     Q ← ∅
19:     for each p ∈ F_i do
20:         for each q ∈ S_p do
21:             n_q = n_q − 1
22:             if n_q = 0 then
23:                 q_rank ← i + 1
24:                 Q ← Q ∪ {q}
25:             end if
26:         end for
27:     end for
28:     i ← i + 1
29:     F_i ← Q
30: end while
31: return F
```

Crowding distance assignment. Apart from the sorting algorithm, the authors in [109] introduced the algorithm for crowding distance calculation which is depicted in Alg. 17. The

crowding distance formula is implemented in line 7 of the algorithm. As it can be seen, the crowding distance of the individual I_i corresponds to the normalized sum of distances between the two individuals, the previous solution I_{i-1} and the next solution I_{i+1} for each objective m. To get a better intuition about the crowding distance, observe Fig. 5.23. The crowding distance of individual I_i corresponds to the average side length of the rectangle denoted with dashed line [109]. Using this algorithm we can differentiate between the solutions in the same front and preserve solutions with larger crowding distances.

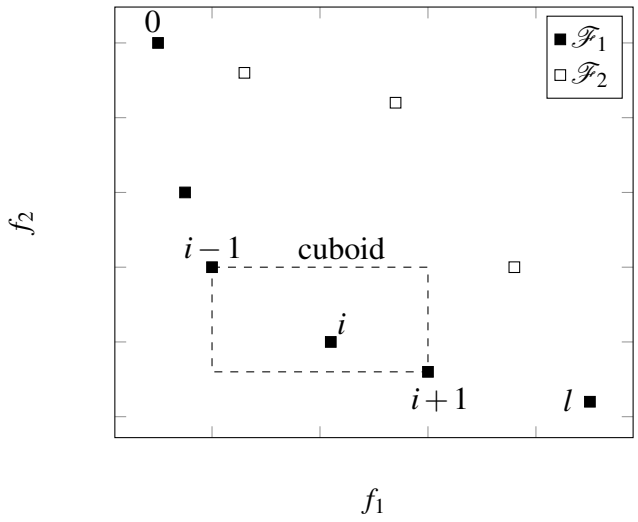


Figure 5.23: Representation of crowding-distance calculation in bi-objective case. Points denote solutions belonging to the same front. The figure is inspired by Fig. 1 in [109].

Algorithm 17 Crowding distance assignment

Input: I - non-dominated

Output: $\mathscr{F}$ - non-dominated set with assigned crowding distance values

1: $l = |I|$
2: $I_i \leftarrow 0, \forall i$
3: **for** each objective m **do**
4: $\quad I \leftarrow= sort(I,m)$ ▷ sort I w.r.t. objective m
5: $\quad I_1^{distance} \leftarrow I_l^{distance} \leftarrow \infty$
6: $\quad$ **for** $i = 2$ to $(l-1)$ **do**
7: $\quad\quad I_i^{distance} \leftarrow I_i^{distance} + \frac{I_{i+1}.m - I_{i-1}.m}{f_m^{max} - f_m^{min}}$
8: $\quad$ **end for**
9: **end for**

NSGA-II algorithm. The complete NSGA-II algorithm is shown in Alg. 18. Briefly, the algorithm operates in the following manner with regard to $t-th$ generation. First off, it sorts the current population and the offspring population in non-dominated sets $\mathscr{F}$. The next

generation P_{t+1} is created from the obtained sets in line 11. When a front, i.e., non-dominated set, cannot entirely fit in the new population, it takes $N - |P_{t+1}|$ individuals sorted according to the crowded-comparison $\prec_n$ operator, which is defined as follows.

Definition 31. ***Crowded-comparison operator.*** *Crowded-comparison creates a partial order $\prec_n$ between individuals p and q such that:*

$$p_{rank} < q_{rank} \vee (p_{rank} = q_{rank} \wedge p_{distance} > q_{distance}) \implies p \prec_n q \tag{5.13}$$

In other words, the better solution has the lower rank, and between two solutions with the same rank, better solution has the lower crowding distance.

When population P_{t+1} is generated, a new population Q_{t+1} is created using selection, crossover and mutation operators. Note that the employed selection operators should use the crowded-comparison operator in the selection process. In the subsequent iteration, the entire previous population is included in the sorting, which ensures elitism.

Algorithm 18 NSGA-II algorithm

1: $t \leftarrow 0$
2: generate initial population P_0
3: sort P_0 based on non-domination
4: create a new population Q_0 using selection, crossover and mutation operators
5: **while** termination condition not reached **do**
6: $R_t \leftarrow P_t \cup Q_t$
7: $\mathscr{F} = fast\_non\_dominated\_sort(R_t)$
8: $P_{t+1} \leftarrow= \emptyset, i = 1$
9: **while** $|P_{t+1}| + |\mathscr{F}_i| \leq N$ **do**
10: crowding_distance_assignment $(\mathscr{F}_i)$
11: $P_{t+1} = P_{t+1} \cup \mathscr{F}_i$
12: $i \leftarrow i + 1$
13: **end while**
14: sort$(\mathscr{F}_i, \prec_n)$ - sort according to the $\prec_n$ operator
15: $P_{t+1} = P_{t+1} \cup \mathscr{F}_i[1 : (N - |P_{t+1}|)]$
16: create a new population Q_{t+1} using selection, crossover and mutation operators
17: $t \leftarrow t + 1$
18: **end while**

5.6.2 Introduction of fairness as an objective

In the previous evaluation results, it could be seen that in some cases trivial scheduling rules such as the **SRTF** yielded good results in a sense of average grade of service. However, it is obvious that when the **SRTF** heuristic is used, the scheduler will be biased towards scheduling the short jobs, leaving no execution time for the long jobs. Although this will improve overall average grade of service or reduce the average penalty, this is probably not a desired behavior, and fairness has to be enforced to ensure that all jobs are executed at least a portion of the available time.

Formally, the *Gini coefficient* is introduced as an additional objective in the system. The Gini coefficient is often used in economics as a measure of statistical dispersion intended to represent the income inequality or wealth inequality within a nation or any other group of people [113]. From a mathematical standpoint, the Gini coefficient is half of the relative mean absolute difference, which is the mean absolute difference divided by the arithmetic mean. Formally, for an observed set of values $\mathscr{X} = \{x_1, ..., x_n\}$ this can be expressed as:

$$G = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} |x_i - x_j|}{2n^2 \bar{\mathscr{X}}} \tag{5.14}$$

where $\bar{\mathscr{X}} = \frac{1}{n} \sum_{i}^{n} x_i$, i.e., the arithmetic mean. When the resources in the system are equally distributed, i.e., the mean absolute difference is equal to zero, then the Gini coefficient G is also equal to zero $G = 0$. In the opposite extreme case, i.e., all the resources are assigned to a single part of the system, which is in this research a task, the Gini coefficient is equal to $G = 1 - \frac{1}{n}$. This means that in practice, the goal is to minimize the Gini coefficient since this will ensure fair share of the available execution time for all tasks. However, ensuring the fair share of the execution time is not enough since generally it is not acceptable to create a scenario in which all tasks perform equally bad. Therefore, in a lot of situations we ought to optimize the average of the target performance metric and its Gini coefficient.

Gini coefficient was chosen as a fairness metric, i.e., a measure of dispersion in a task set, due the following reasons:

- Gini coefficient was used as a fairness metric in similar mixed-criticality research [94].
- In comparison with measures such as standard deviation, Gini coefficient is scale independent, while standard deviation, unless scaled, preserves the original units [114].
- In comparison with standard deviation, Gini coefficient is bounded, i.e., $G \in [0, 1]$, thus providing an instant insight into the fairness of the observed system [114].

5.6.3 Evaluation results for multi-objective optimization

In this section, the results of the multi-objective optimization are presented. So far, in the experiments, two performance metrics were observed, grade of service and penalty. From now on, the focus will be on penalty since it is more likely to be used in real-world problems as it is argued before. More precisely, using the multi-objective optimization approach presented earlier in this section, the average penalty and its Gini coefficient shall be optimized, which should ensure that the overall average penalty is minimal and that generated heuristics are not biased to discard any task. The approach is evaluated in a similar manner as in section 5.5.3, i.e., with regard to $U(LO)$, n, and T_{max}. The function and terminal nodes as well as the optimization parameters, i.e., the number of generations, the population size, the tree depth, and the training set size are the same as in the single-objective approach.

Firstly, the evaluation with regard to the utilization is observed. Figs. 5.24-5.25 show the average performance of the multi-objective genetic programming denoted with **MOGP-avg** acquired in 10 consecutive runs as well as the approaches discussed in the previous experimental results. Note that this is solution with the minimal penalty, and other Pareto optimal solutions are discussed afterwards. In Fig. 5.24, it can be seen that the penalty performance is not significantly improved when the multi-objective approach is employed. The multi-objective approach did not reduce the penalty more than the single-objective approach. However, when the Gini coefficient is observed in Fig. 5.25, it can be seen that on average multi-objective approach yields solutions with increased fairness, i.e., reduced Gini coefficient. Note that the **SRTF** approach performs worse with regard to both metrics. As mentioned before, the **SRTF** heuristic is an extremely unfair policy due to the bias for short jobs. In addition, it is interesting to notice that the **GP-Pen** heuristic performs better in terms of fairness, but slightly worse in terms of penalty. This is a general effect which can be observed for all Pareto optimal solutions, which are yielded by the multi-objective approach.

In Figs. 5.26-5.28, this effect is demonstrated in detail. In Fig. 5.28, the performance of Pareto optimal solutions from the last run of the experiment is shown. These solutions are yielded by the multi-objective approach, i.e., NSGA-II algorithm, and all of these solutions have the same rank, i.e., they belong to the same non-dominated set. Therefore, these solutions can be considered "equally good". It is interesting to notice how minimizing the average penalty on x-axis decreases the fairness, i.e., increases the Gini coefficient, on y-axis. This is further illustrated in Figs. 5.26-5.27, where a few Pareto optimal solutions from the last run of the experiment are shown, namely **MOGP-X** where **X** designates the position, i.e., index, of the solution in the population. The first solution, i.e., **MOGP-1** in the population yields the minimal penalty and the last solution, i.e., **MOGP-100** yields the maximal fairness since at the end of the optimization procedure a single front remains. An interesting observation is that **MOGP-30** solution performs slightly worse in terms of penalty, but significantly better in terms of fairness.

This is important since it enables flexibility in the system design by leveraging the trade-off between the fairness and the penalty. Moreover, note that generally **MOGP-30** performs better than the **SRTF** in most cases in terms of penalty, but it is significantly better in terms of fairness. In general case, we cannot know which metric is more important for the specific application, but the multi-objective approach allows that the choice of heuristic is done in a straightforward manner. It is important to note that with the solution with higher fairness tasks will perform "equally good", but since the penalty for these solutions is significantly reduced, a more appropriate description for their performance is "equally bad". Again, this can be easily resolved in the design of specific application.

Generally, it can be observed that the Gini coefficient is relatively high. This is probably the consequence of lack of sufficiently large interval of slack time, which is needed for execution of longer low-criticality jobs. Therefore, since longer low-criticality jobs are rarely executed, fairness is decreased.

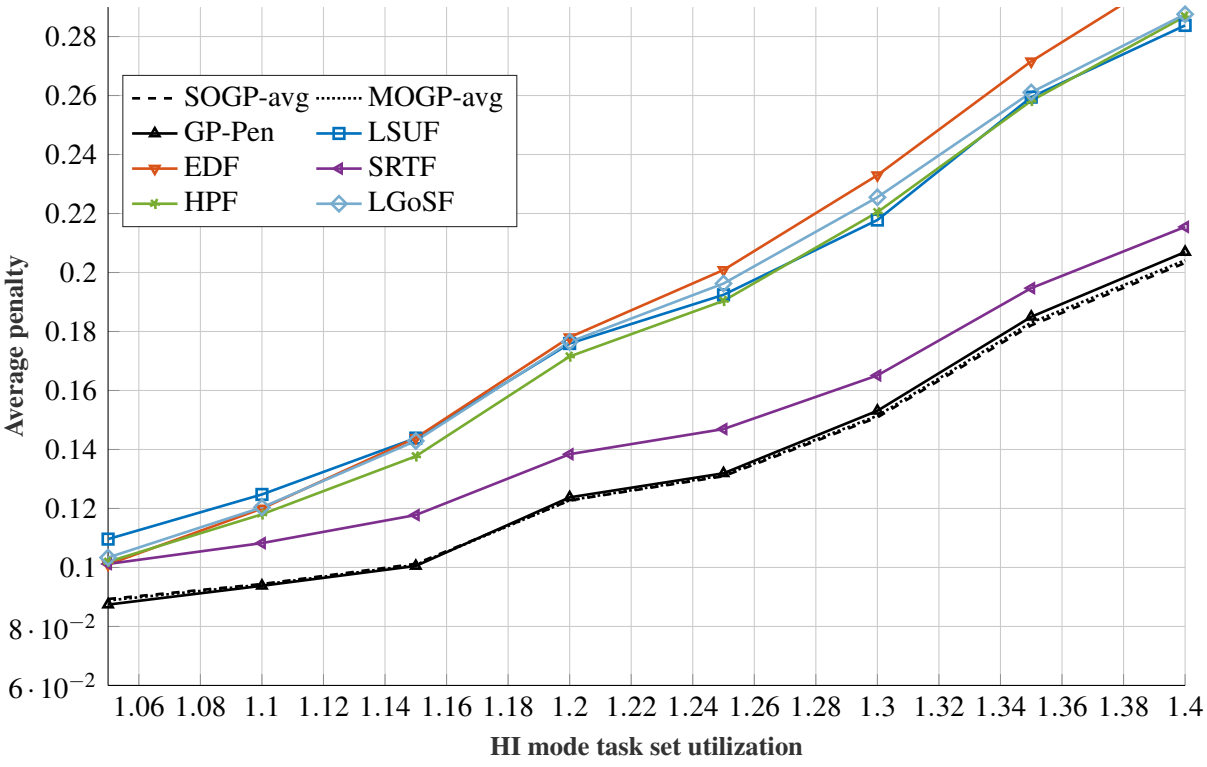


Figure 5.24: Average penalty for different HI mode utilization factors.

In Figs. 5.29-5.32, when varying the number of tasks and maximum allowed period in the system, we can see similar observations about the multi-objective approach as in the previous experiments. Thus, average penalty is not reduced, and the fairness is somewhat increased as it can be seen in Fig. 5.32. Similarly as before, the fairness can be tuned by choosing an appropriate solution from the set of non-dominated solutions with the same rank.

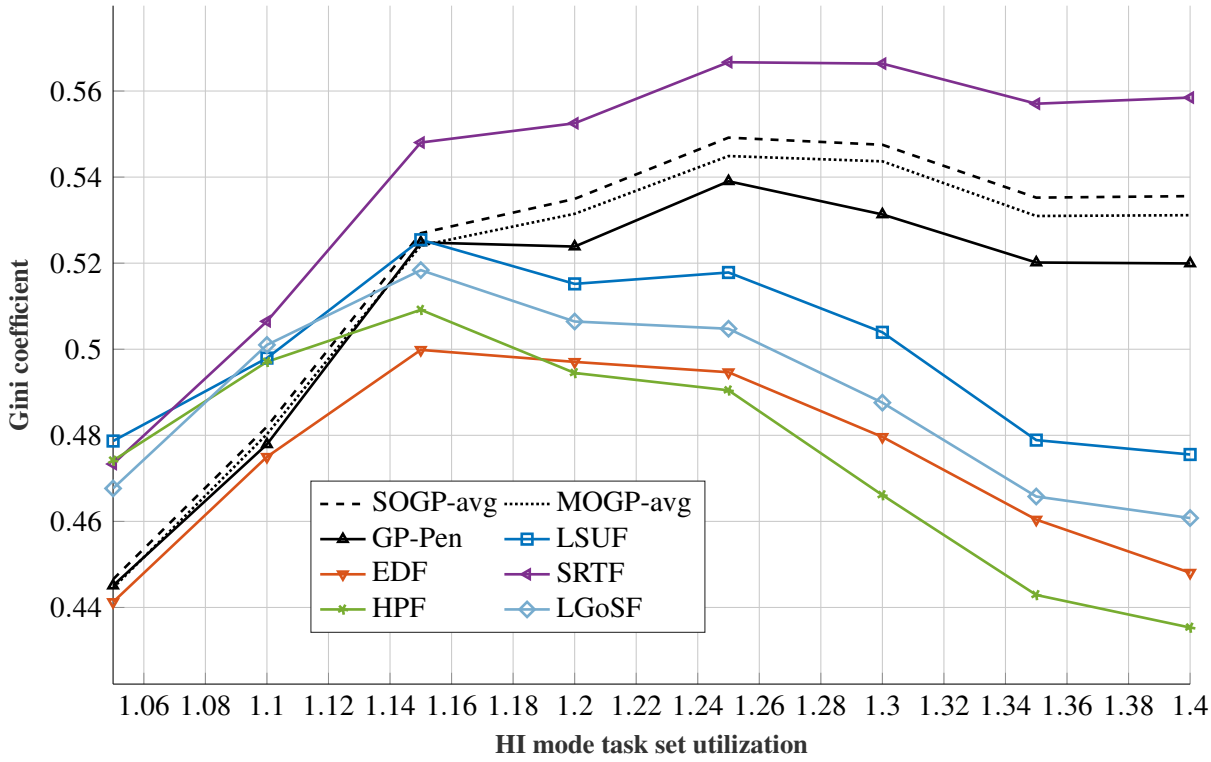


Figure 5.25: Gini coefficient for different HI mode utilization factors.

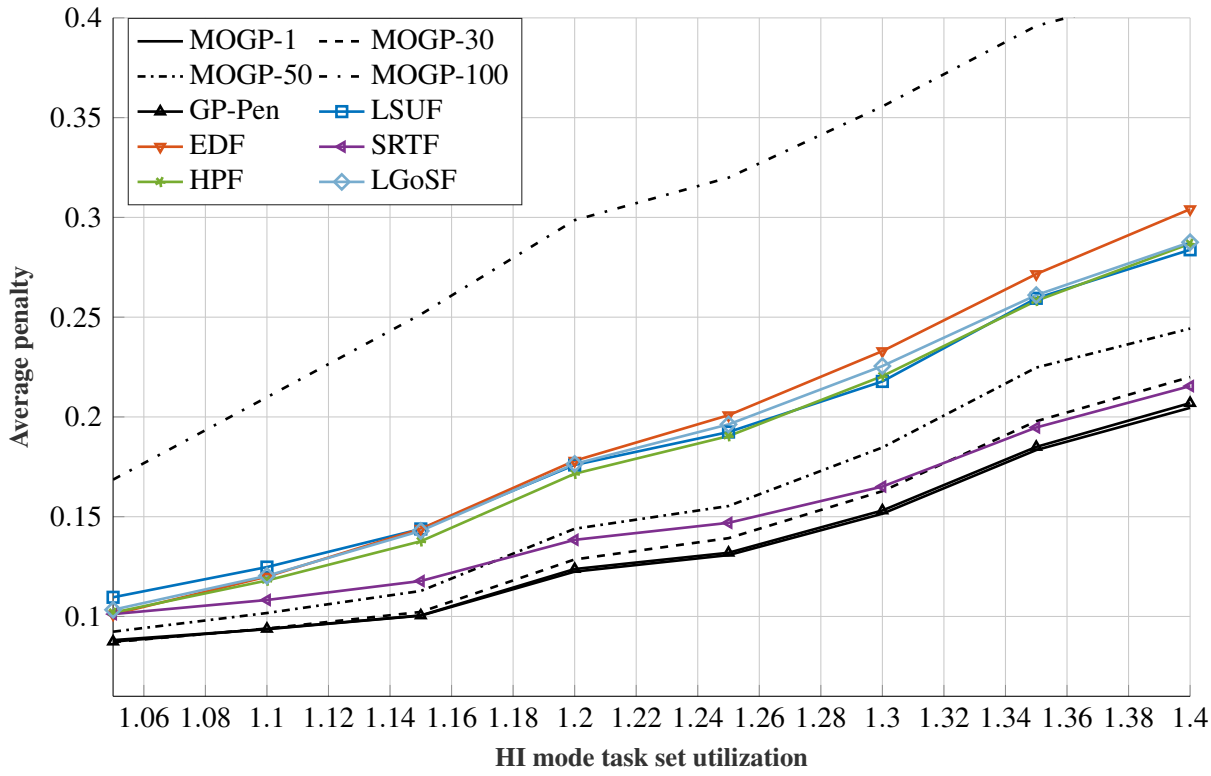


Figure 5.26: Average penalty for different HI mode utilization factors including several Pareto optimal solutions.

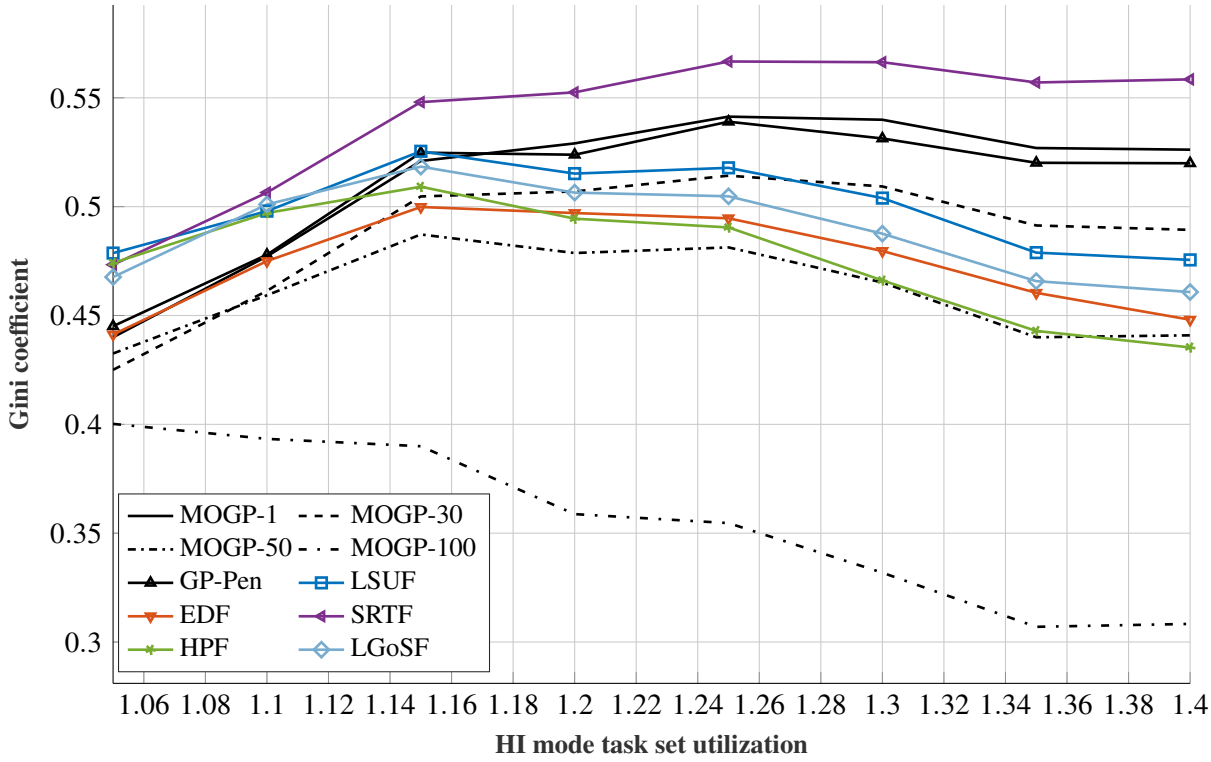


Figure 5.27: Gini coefficient for different HI mode utilization factors including several Pareto optimal solutions.

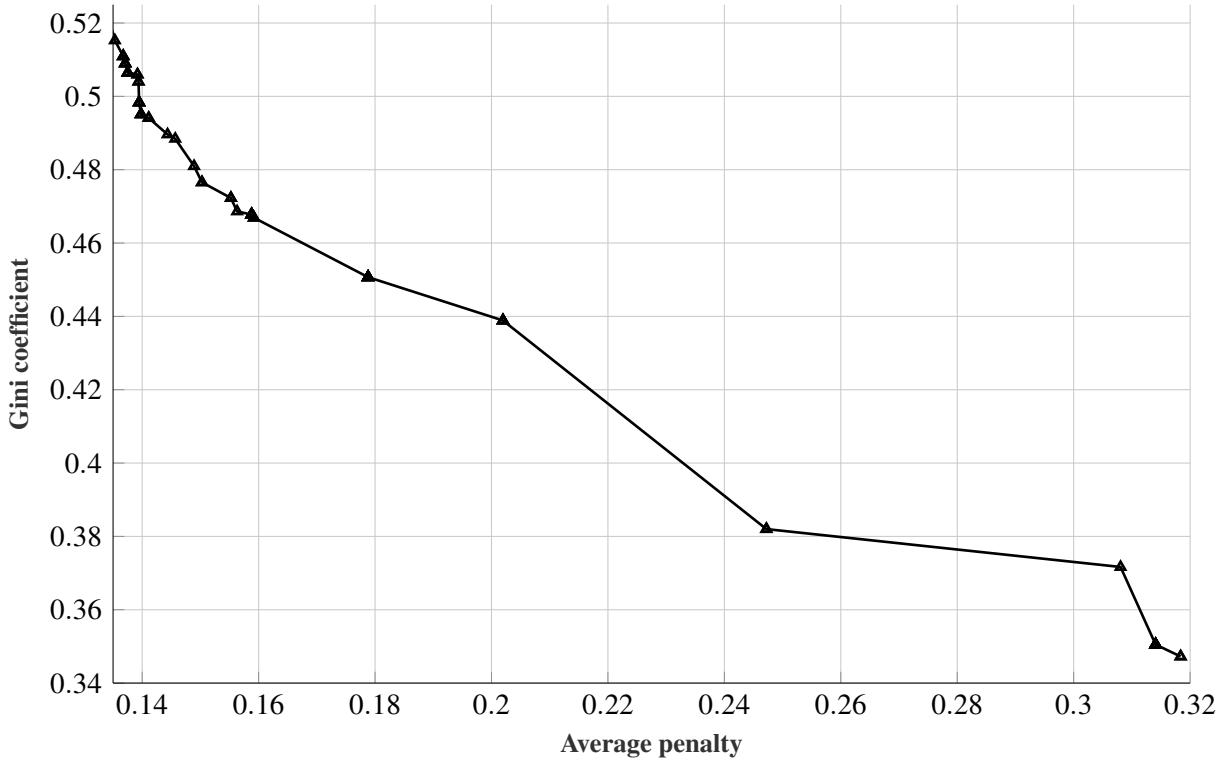


Figure 5.28: Pareto optimal solutions with regard to average penalty and Gini coefficients.

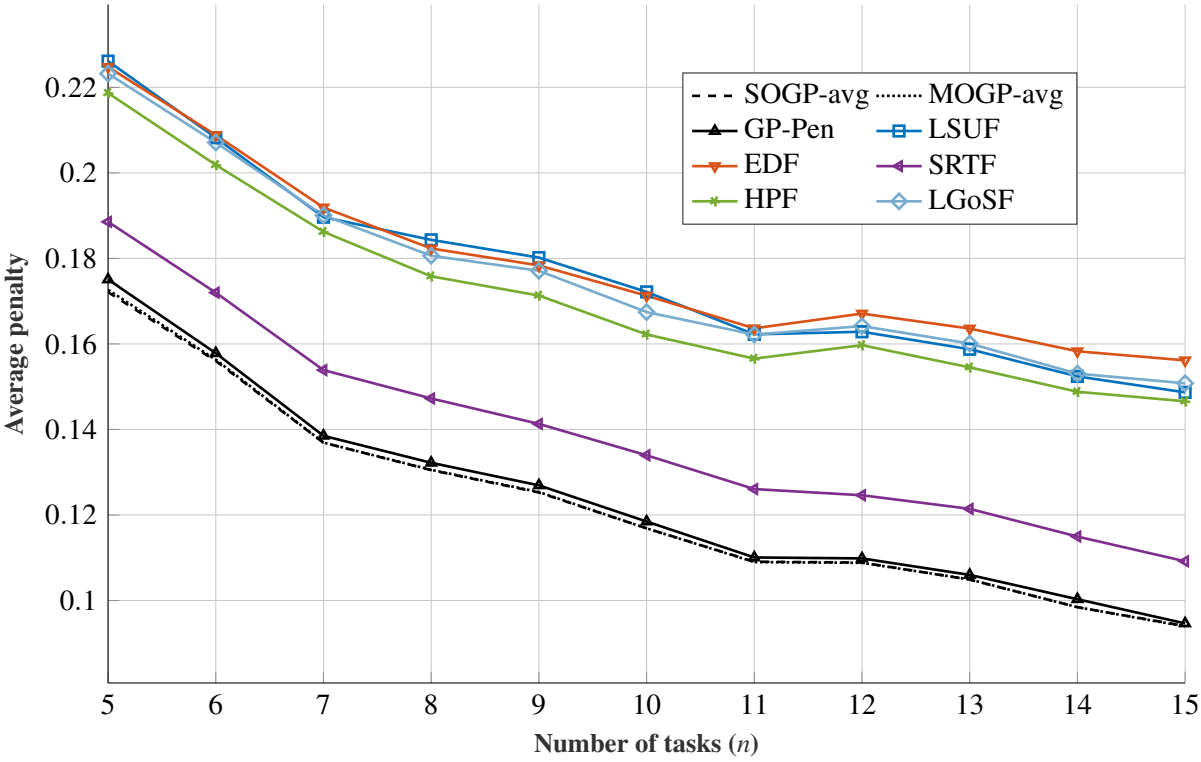


Figure 5.29: Average penalty for different number of tasks in a task set.

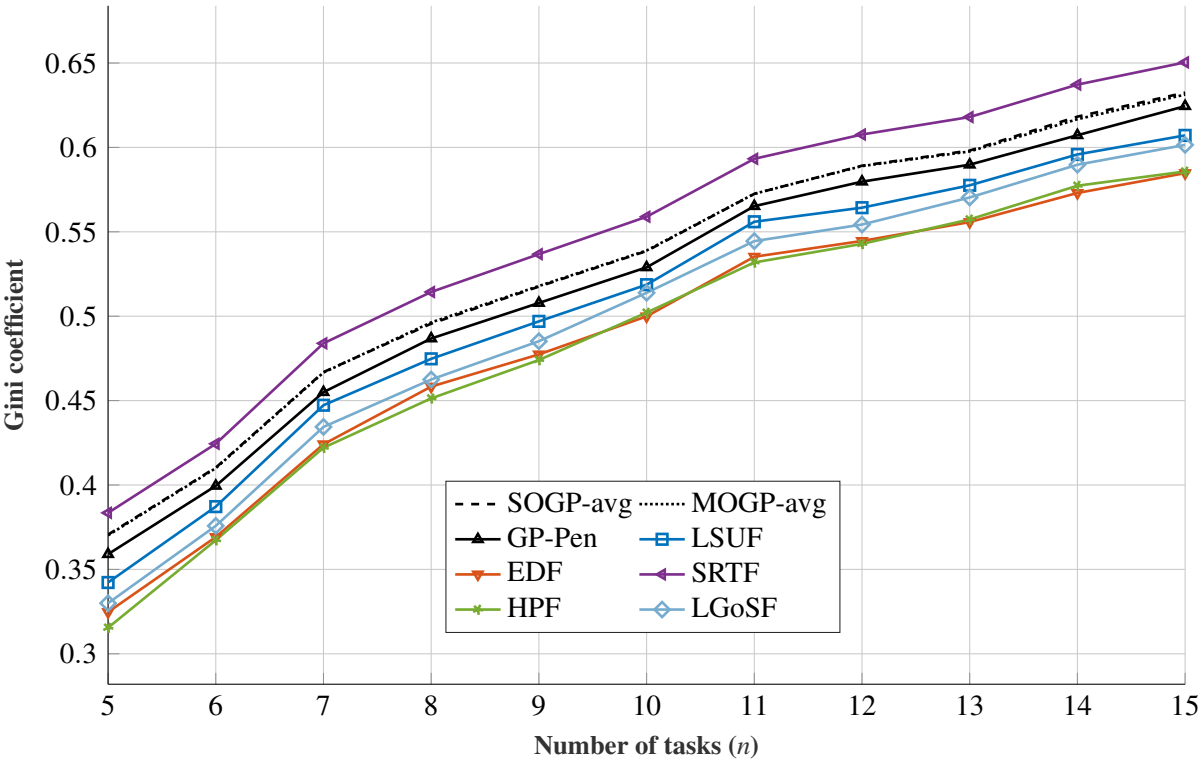


Figure 5.30: Gini coefficient for different number of tasks in a task set.

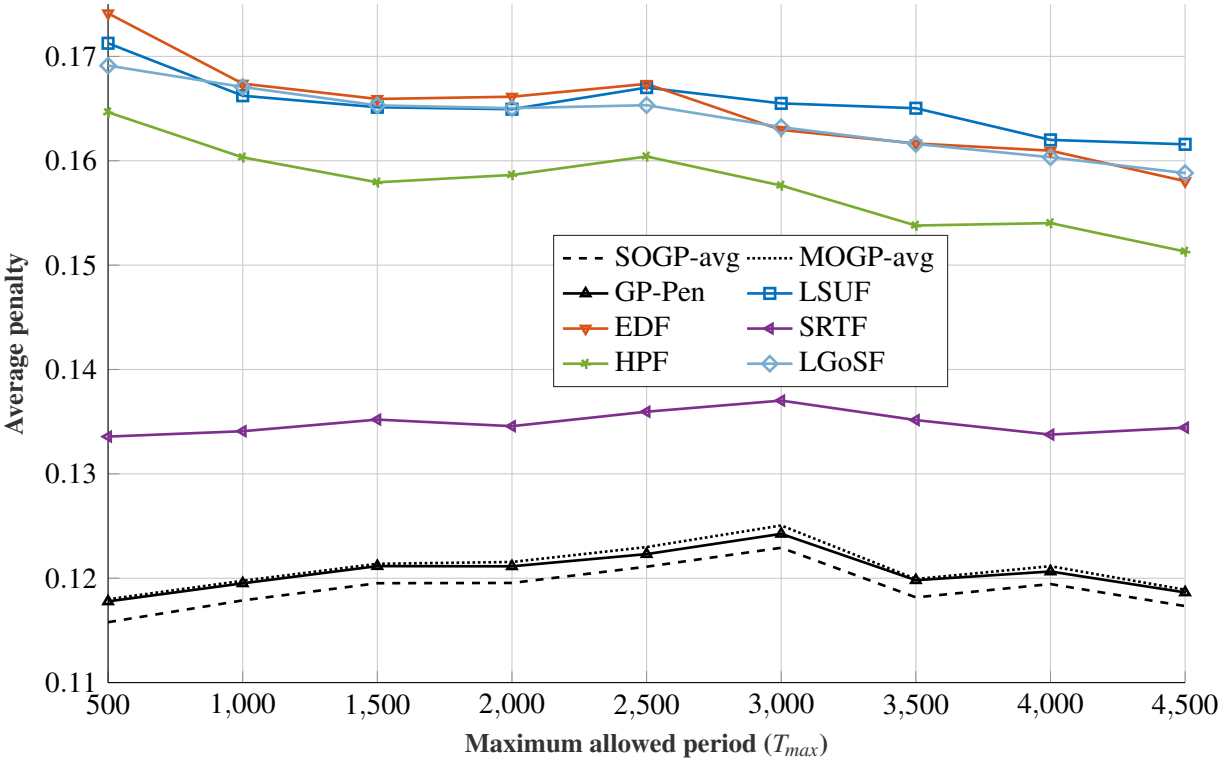


Figure 5.31: Average penalty for different maximum allowed period.

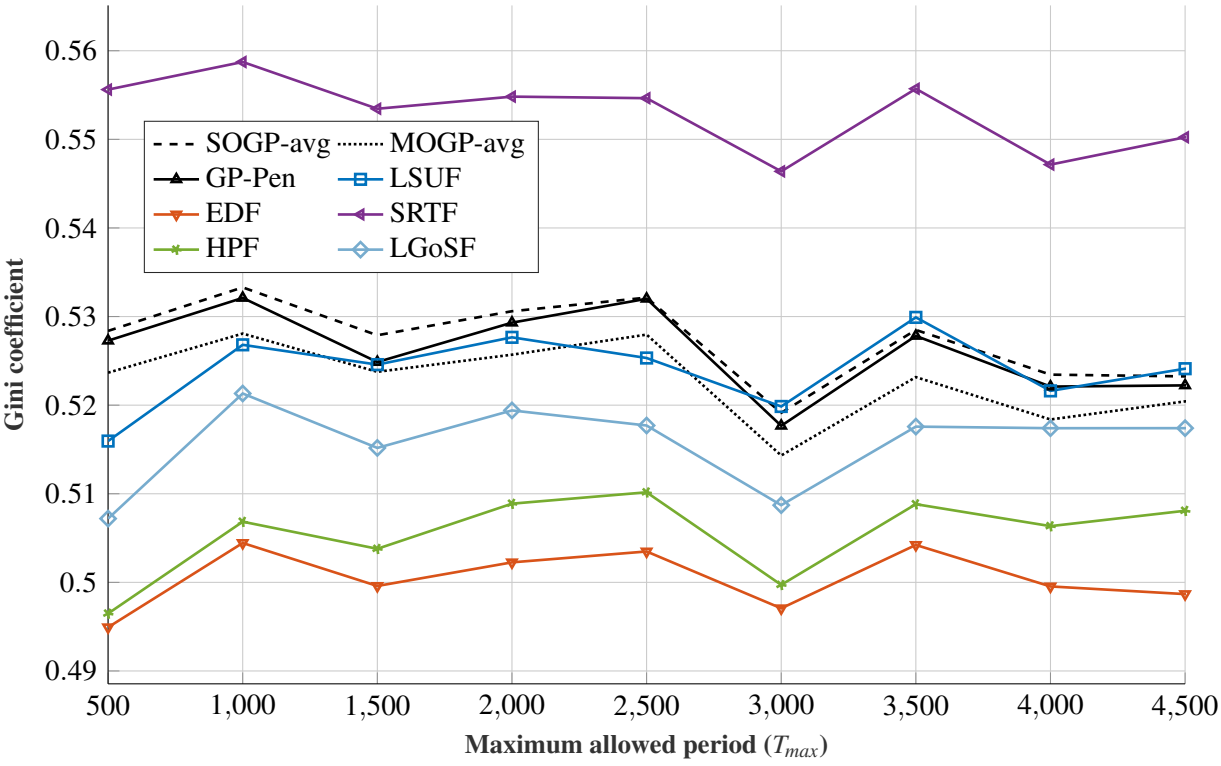


Figure 5.32: Gini coefficient for different maximum allowed period.

5.7 Acceptance tests on job release

Another significant drawback of the approaches in the latter sections is that in the scheduling meta-algorithm, priorities are assigned to each active job. However, it is possible that a job with the highest assigned priority cannot be feasibly scheduled. Therefore, a reasonable approach is to utilize an acceptance test prior to calculating the priorities of low-criticality tasks. To achieve this, it is necessary to change the scheduling meta-algorithm. The modified scheduling meta-algorithm is depicted in Alg. 19. A simple acceptance test, which can be used can be based

Algorithm 19 Scheduling meta-algorithm with acceptance test

```
1:  if system in LO mode then
2:      schedule jobs using EDF w.r.t. LO deadlines
3:  else if system in HI mode then
4:      if LO job is released then
5:          add job to queue if it passes acceptance test
6:      end if
7:      if HI jobs available then
8:          schedule HI jobs using EDF w.r.t. HI deadlines
9:      else if LO jobs available then
10:         schedule LO jobs using heuristic
11:     end if
12: end if
```

on the remaining execution time c_i and the time to deadline d_i. We know that job such that $c_i > d_i$ cannot be successfully executed. Moreover, executing such a job would only waste the processor time for jobs that can be successfully executed. However, there is a possibility that all jobs pass such a test. Therefore, the goal is to find a more suitable acceptance test, which would minimize the wasted processor time. In general case, an effective acceptance test could be determined using the same approach that is used for determining the scheduling heuristic. However, there are several possible approaches for doing this. Using the approach from the latter sections, the priority function can be fixed in scheduling meta-algorithm, and the heuristic for accepting and discarding jobs can be optimized. The evolved tree, i.e., heuristic, is not used for calculating the priority of a job. The value that the heuristic yields is compared to a fixed value, and the obtained boolean value represents the decision, i.e., true if the job is accepted, and false otherwise. Formally, this can be stated as follows:

$$a_i(t) = \begin{cases} 1, \alpha_i(t) > 0 \\ 0, \text{otherwise} \end{cases} \tag{5.15}$$

where $a_i(t)$ represents the decision value of the acceptance test represented with the acceptance function $\alpha_i(t)$. In the following numerical example, it is illustrated how such an approach can

improve the behavior of the system.

Example 13. *Consider the MC task set given in Table 5.4. The goal is to find an acceptance function $\alpha_i(t)$ which minimizes the total number of skips of LO criticality tasks in the hyperperiod of the synchronous arrival sequence in HI mode of the input task set with the EDF used for priority assignment, i.e., the priority function is $\pi_i(t) = d_i$. Firstly, it can be shown that it is not possible to schedule all jobs since the system is overloaded, i.e., $U(HI) > 1$:*

$$U(HI) = \sum_{i}^{n} \frac{C_i(HI)}{T_i} = \frac{4}{5} + \frac{4}{10} + \frac{3}{20} \approx 1.35 \tag{5.16}$$

Table 5.4: Task set parameters for Example 13.

Task	$C_i(LO)$	$C_i(HI)$	$T_i = D_i$	L_i
τ_1	2	4	5	1
τ_2	4	4	10	0
τ_3	3	3	20	0

In Fig. 5.33, a schedule without the acceptance test is shown. It can be seen that 3 jobs are skipped, i.e., J_{21}, J_{22}, and J_{31} since they did not finish their execution before the deadline. By introducing the acceptance function given with:

$$\alpha_i(t) = d_i - \delta_i \tag{5.17}$$

the total number of skips can be reduced as it can be seen in Fig. 5.34. The acceptance function causes the acceptance of tasks for which the time to deadline is larger or equal to remaining time to deadline of other tasks. In this particular case, this causes the skip on release of job J_{21} since $d_2 < d_3$. This ensures that enough slack time can be used for execution of J_{31}, which is successfully executed. Thus, only two jobs are skipped, i.e., J_{21}, J_{22}, which is the minimum since there is not enough slack time for execution of jobs of task τ_2. In addition, note that when there is no acceptance test, all available slack time is wasted. On the other hand, when the acceptance test is used, only one time unit is wasted for execution, which is spent for execution of J_{23}.

The possible issue with evolving the acceptance test only is that the priority heuristic that is used in the scheduling meta-algorithm is not compatible with the evolved acceptance test and may cause poor overall performance of the scheduling policy. An alternative approach includes simultaneous evolution of both the priority function, i.e., $\pi_i(t)$ and the acceptance function, i.e.,

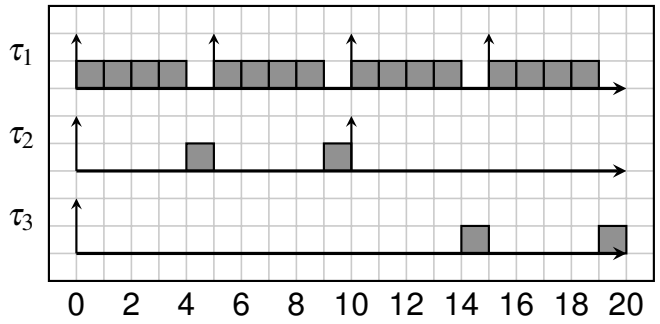


Figure 5.33: Low-criticality tasks scheduled according to the EDF heuristic for task set in Table 5.4.

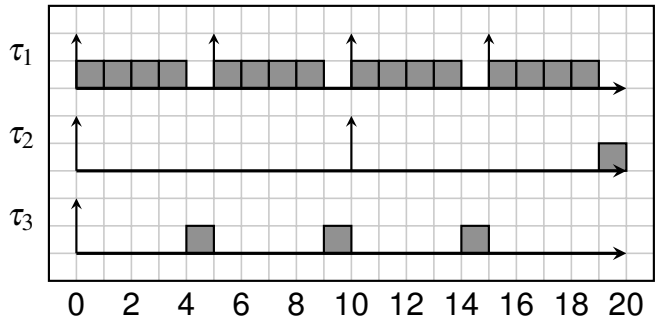


Figure 5.34: Low-criticality tasks scheduled according to the EDF scheduling policy for task set in Table 5.4 with acceptance function given with (5.17).

$\alpha_i(t)$. An approach that is often used in the literature for the simultaneous evolution of two or more different populations is known as cooperative co-evolution. The cooperative co-evolution mechanism was firstly introduced in work of Potter and De Jong [115] for function minimization, and a brief survey can be found in [116]. The cooperative co-evolution is used in the similar research for evolving dispatching rules in various scheduling environments [112, 117, 118]. For completeness and clarity, the mechanism of cooperative co-evolution optimization algorithm is discussed in the next subsection.

5.7.1 Cooperative co-evolution

Artificial cooperative co-evolution is an evolutionary approach that is inspired from the interaction of different species in nature. Species in nature often interact in several different ways:

- competing for the same resources, e.g., food or water,
- cooperating for solving the specific problem, e.g., the reproduction of plants is often performed by means of various species of insects.

Cooperation mechanism can be usefully simulated in an artificial environment, which allows us to combine individuals from different populations to a single complete solution of a problem. To solve problems using the cooperative approach, problems are divided in smaller complementary components. To each component of the problem, i.e., subproblem, typically a single population is assigned. By appropriate choice of the collaborators from each population, the complete

solution is obtained. The complete solution then can be evaluated and assigned appropriate fitness.

The generic algorithm for cooperative co-evolution is depicted in Alg. 20 (based on the algorithm in [116]). The algorithm starts in a classical genetic algorithm manner, i.e., initializing population. In this case, multiple populations are initialized. This is followed by the evaluation of the individuals by choosing collaborators from the other species for each individual. In the t-th generation, genetic operators are applied and the created offspring population is evaluated. Using the previous population and the newly generated offspring population, the next generation is created. Although in this pseudocode, a simple usage of genetic operators is presented, in a general case, any genetic algorithm procedure can be employed as long as its operators are applied to individuals in each species, and collaborators are chosen for evaluation of every individual.

Algorithm 20 Cooperative co-evolution

```
1: t ← 0
2: for each species s do
3:     randomly initialize population P_s(t)
4: end for
5: for each species s do
6:     evaluate P_s(t) by choosing collaborators from the other species for each individual
7: end for
8: while termination condition not reached do
9:     for each species s do
10:        select parents from P_s(t)
11:        apply genetic operators
12:        evaluate the offspring population by choosing collaborators for each individual
13:        select survivors for P_s(t+1)
14:    end for
15:    t ← t+1
16: end while
```

In the implementation in this research, collaborators are chosen in a straightforward manner. In every step of the algorithm, the choice of collaborators is fixed, i.e., the individuals with the same index from the first and the second species are chosen. Therefore, this can be viewed as the evolution of a single individual with two different trees in genotype. The first tree is used for obtaining the priority level, and the second tree for testing a job for acceptance. The function and terminal nodes as well as the optimization parameters, i.e., the number of generations, the population size, the tree depth, and the training set size are the same as in the single-objective

and multi-objective approaches discussed in the previous sections.

5.7.2 Measuring the impact of the acceptance test

As it is mentioned before, the acceptance test should reduce the amount time that the scheduler assigns to tasks that will end up not executed by their deadline, thus wasting the processor time. A simple metric that can track the misuse of the available execution time can be defined as:

$$w = \frac{\text{total available execution time} - \text{time used for successful job execution}}{\text{total available execution time}} \tag{5.18}$$

A metric $e = 1 - w$, which is complementary to this metric is similar to the effective processor utilization, which measures that amount time that was spent for the successful execution of jobs [84, 85].

5.7.3 Evaluation results for scheduling with acceptance tests

In this section, experimental results of the acceptance-test based approaches are presented. This includes the approach with standalone acceptance tests and the approach with cooperative co-evolution. So far, in the experiments, two aspects of the performance were observed, the average penalty and its Gini coefficient. Additionally, in the following evaluation, the wasted processing time is observed as well. Therefore, apart from introducing the acceptance test in the meta-algorithm, when using the multi-objective approach the penalty and the wasted processing time are optimized rather than the Gini coefficient. Thus, we study the following configurations:

- single-objective optimization of priority function (SO),
- single-objective optimization of acceptance function with **GP-Pen** as priority function (SOATO),
- single-objective optimization with cooperative co-evolution of priority function and acceptance test (SOAT),
- multi-objective optimization of priority function (MO),
- multi-objective optimization with cooperative co-evolution of priority function and acceptance test (MOAT).

As it is argued before, minimal modifications in optimization framework depicted in Figs. 5.6-5.22 are necessary to evaluate all of these configurations. Similarly as in previous experiments, approaches are evaluated with regard to $U(HI)$, n, and T_{max}.

Firstly, the performance of the approaches with regard to varying utilization $U(HI)$ can be observed in Figs. 5.35-5.37. It can be seen that the **MOGP** approach with no acceptance test performs slightly better than other approaches in terms of all metrics, i.e., penalty, fairness and wasted processing time. Note that the graph denoted as **GP-Pen/SOATO** corresponds to the

heuristic **GP-Pen** with and without acceptance test, i.e., optimization of the acceptance test did not introduce any improvement and the performance corresponds to the performance of the **GP-Pen** heuristic when all jobs are accepted. Since this configuration did not yield any new results, it is not investigated further in the evaluation. Furthermore, it can be seen that introduction of the acceptance tests did not introduce any significant improvements for any of the proposed metrics on average. Moreover, the performance of the approaches with acceptance test is slightly worse with regard to all metrics. Although this is contrary to the initial expectations, it is in fact a reasonable behavior. There are two crucial cases that are identified that drastically influenced configurations with acceptance tests, namely:

1. The evolved acceptance test function $\alpha_i(t)$ in the most of the studied solutions is always positive.
2. When the evolved acceptance test function $\alpha_i(t)$ is mostly negative, a large amount of jobs is discarded.

In the first case, when the evolved acceptance test function is positive, the configuration is equivalent to the configuration without acceptance tests since all jobs are accepted. Thus, only additional overhead is introduced in the evolution process since two populations have to be evolved. In the second case, when the acceptance test function is mostly negative, jobs are discarded although it is possible that they could be executed to completion, which consequently increases the penalty. Therefore, it seems that it is better to allow all jobs to be executed rather than discard job to reduce wasted processing time.

In Figs. 5.38-5.40, it can be noticed that solutions with different penalty and wasted processing time that belong to the same non-dominated set can be chosen. In practice, depending on the application specification, system designer can choose appropriate solution from the resulting non-dominated set. In addition, we can see that the introduction of the acceptance tests in **MOATGP** configuration does not bring any significant improvements in comparison with **MOGP** configuration.

In Figs. 5.41-5.43 it can be seen that there is no significant difference between the performance of the proposed configurations with regard to the number of tasks n. Similarly, there is no difference in the performance between the approaches when the maximal allowed period T_{max} is varied as it can be seen in Figs. 5.44-5.46.

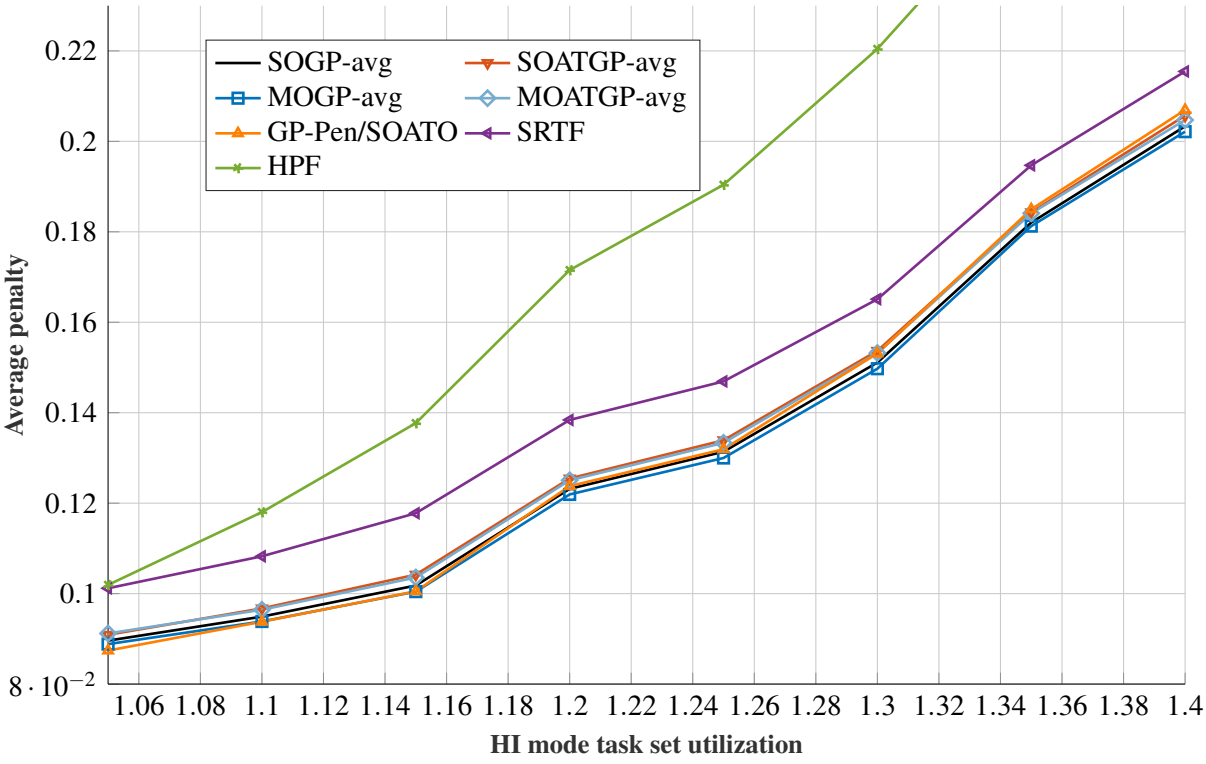


Figure 5.35: Average penalty for different HI mode utilization factors.

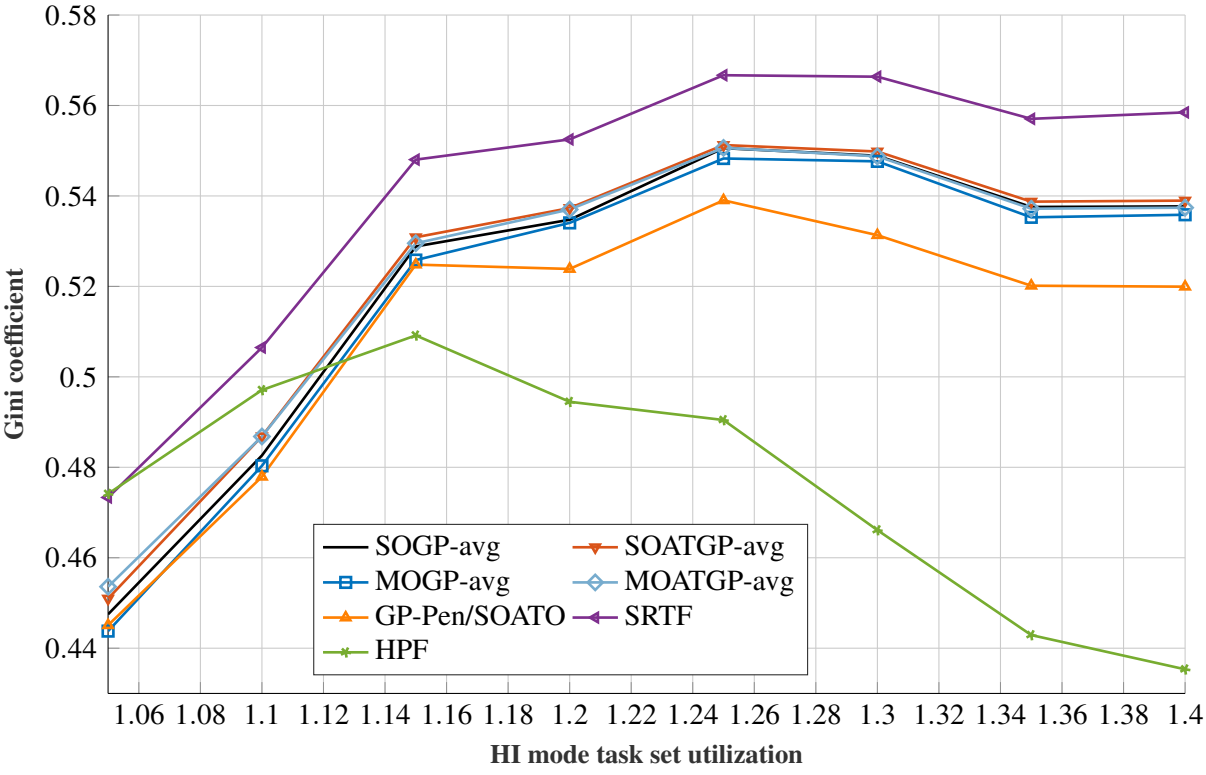


Figure 5.36: Gini penalty for different HI mode utilization factors.

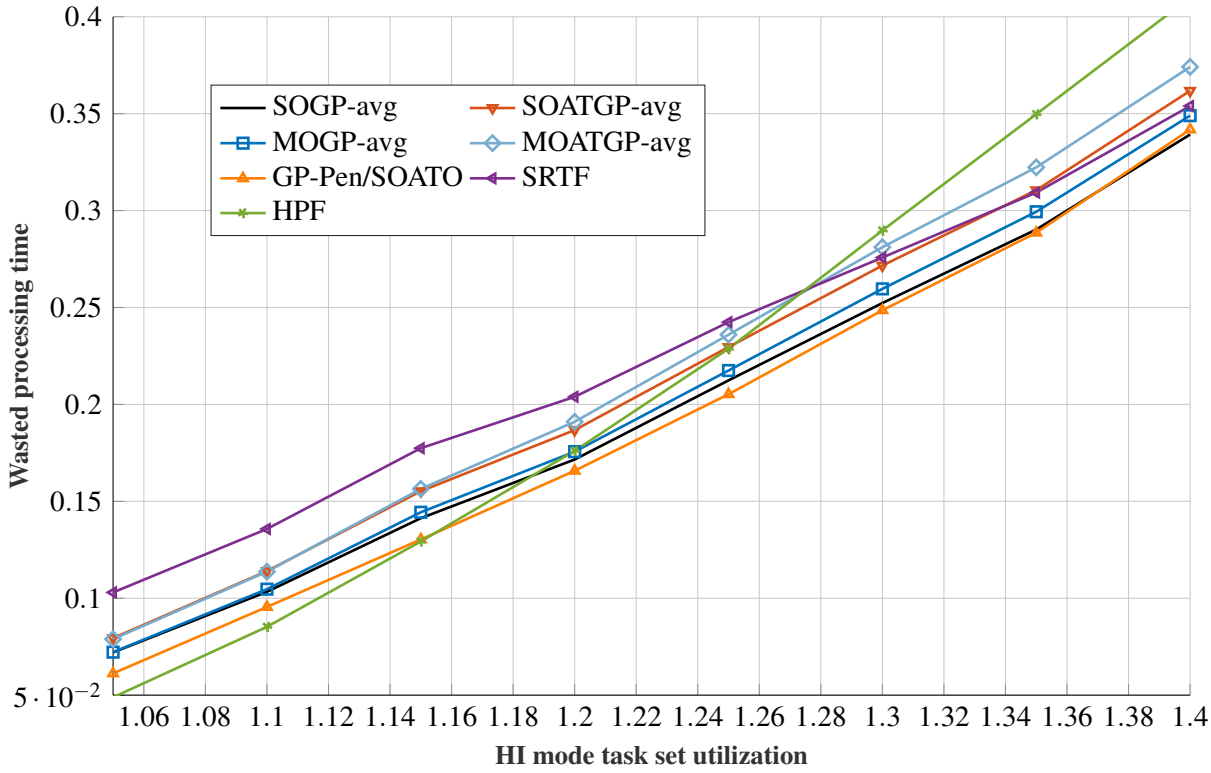


Figure 5.37: Wasted processing time for different HI mode utilization factors.

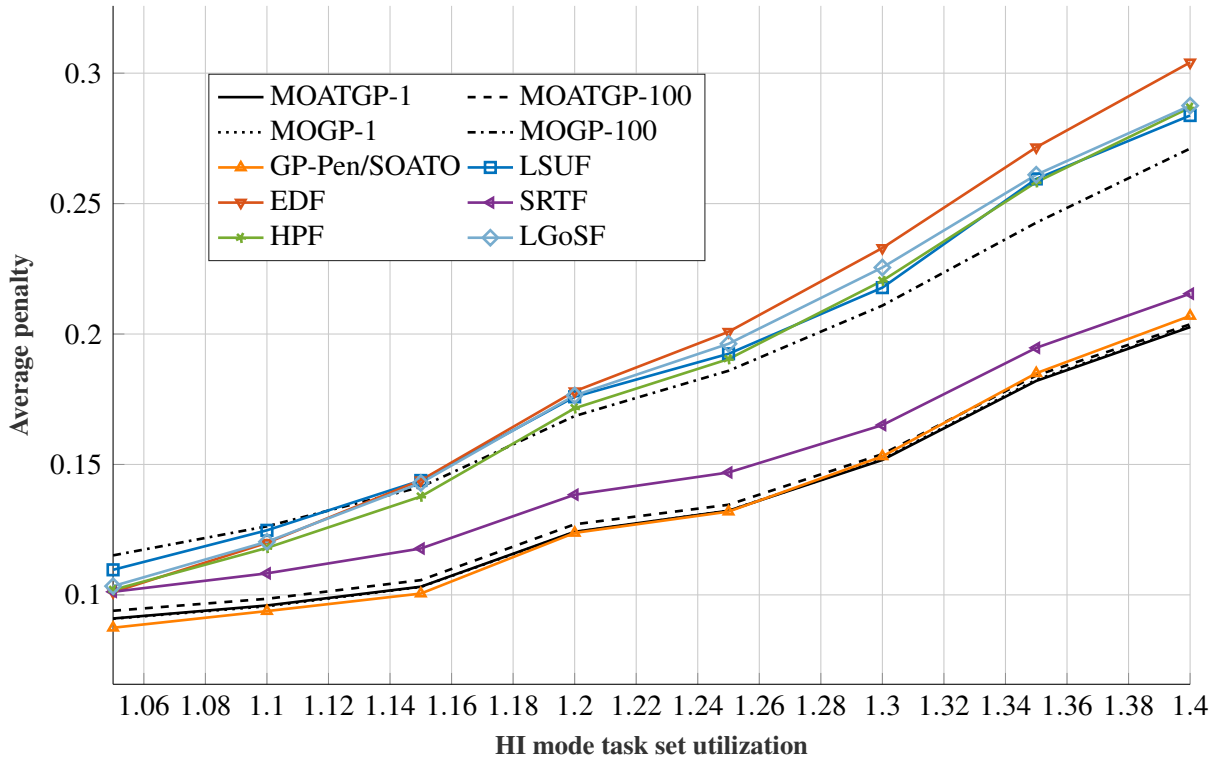


Figure 5.38: Average penalty for different HI mode utilization factors including several Pareto optimal solutions.

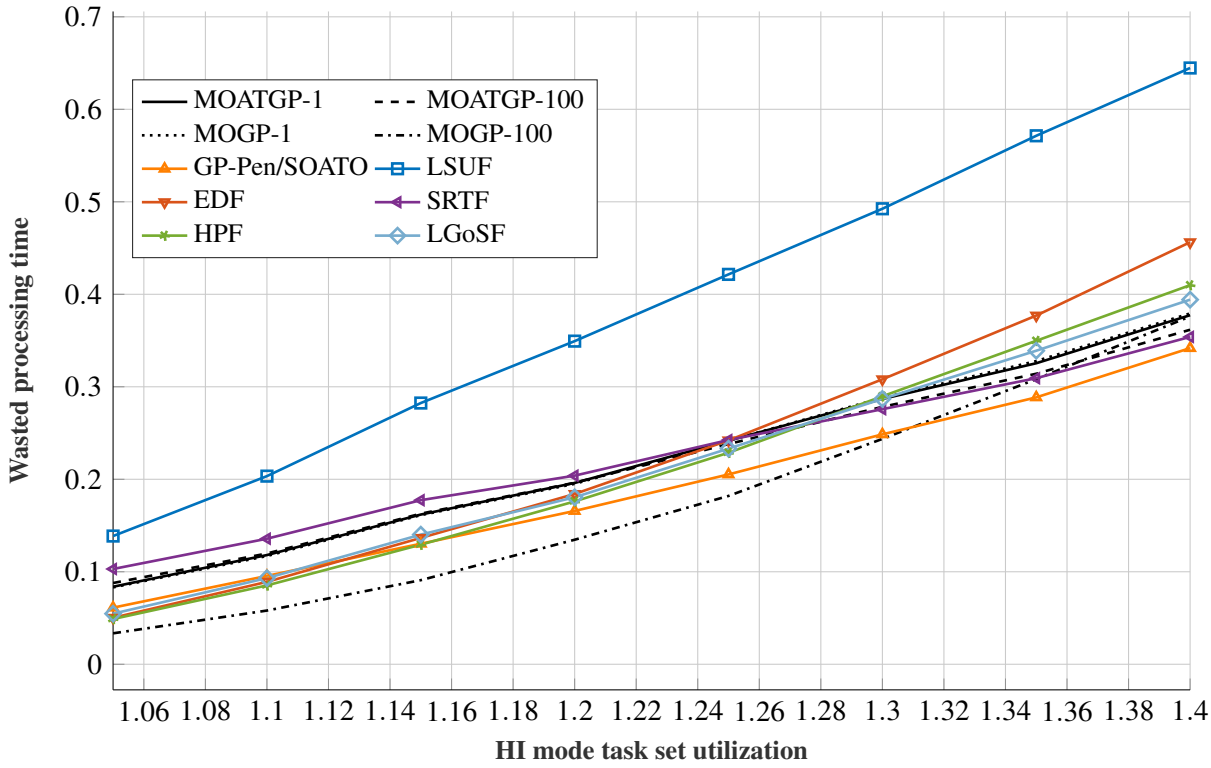


Figure 5.39: Wasted processing time for different HI mode utilization factors including several Pareto optimal solutions.

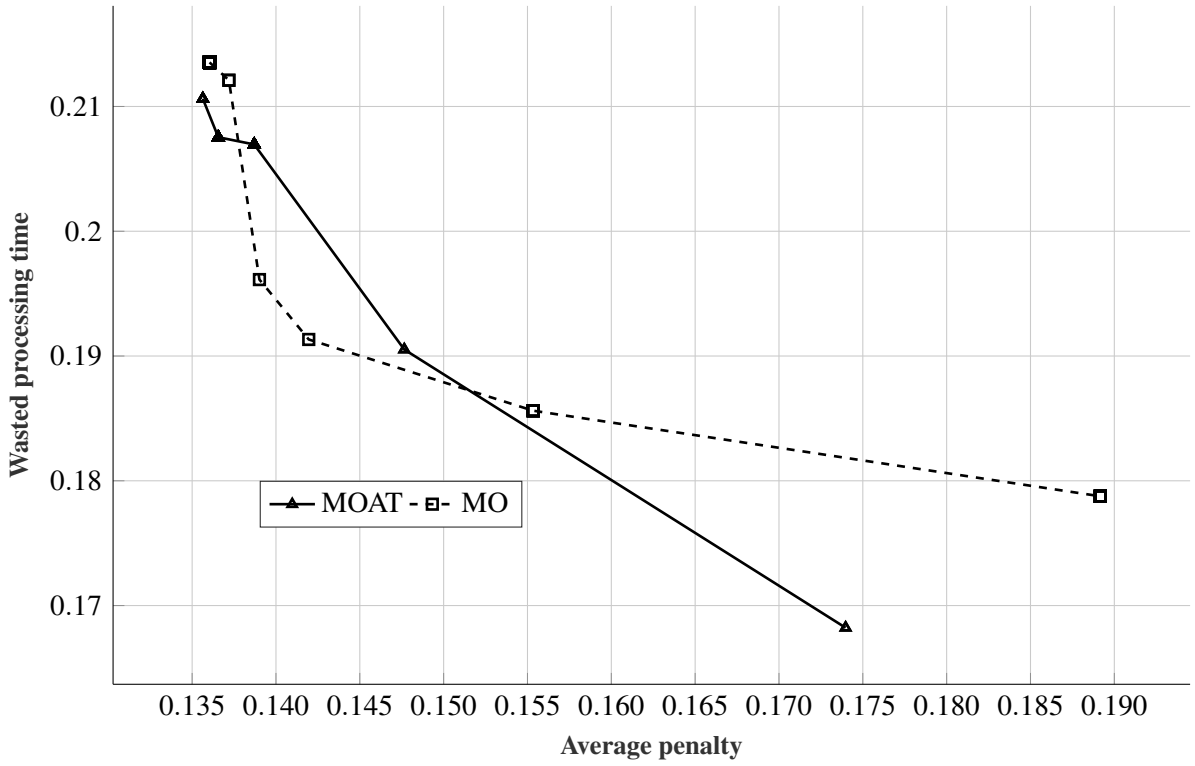


Figure 5.40: Pareto optimal solutions with regard to average penalty and wasted processing time.

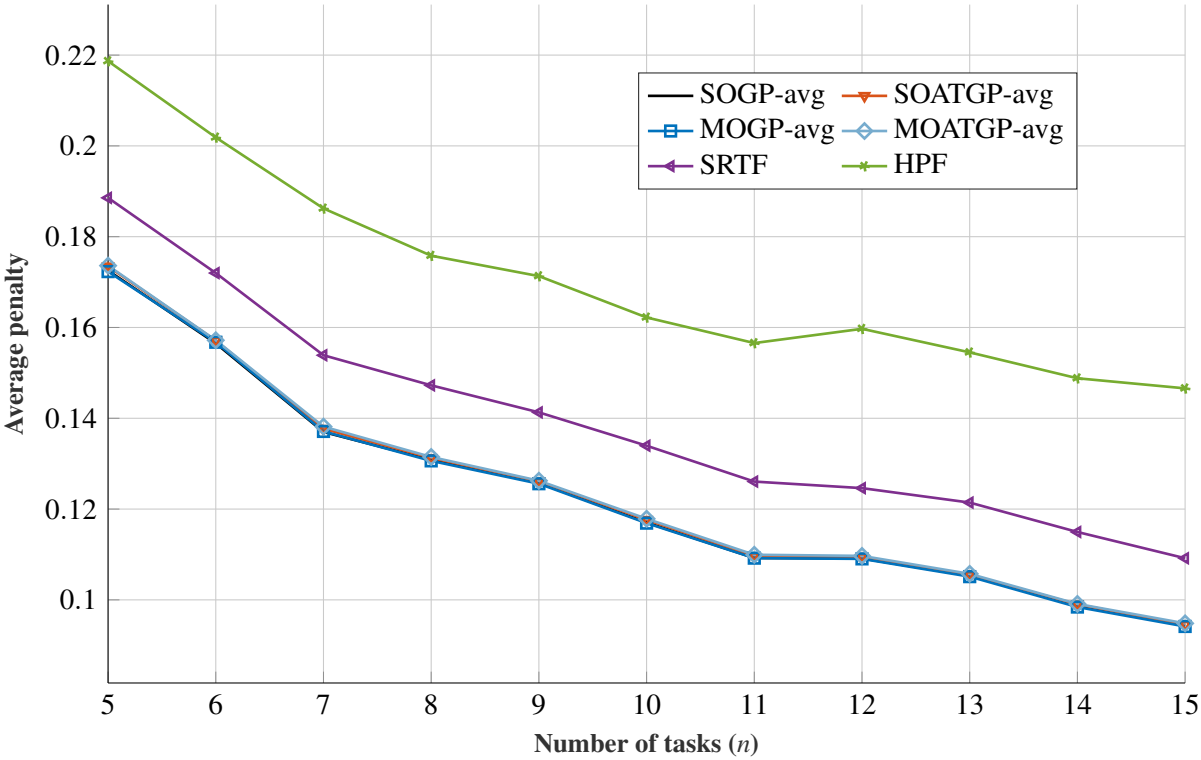


Figure 5.41: Average penalty for different number of tasks in a task set.

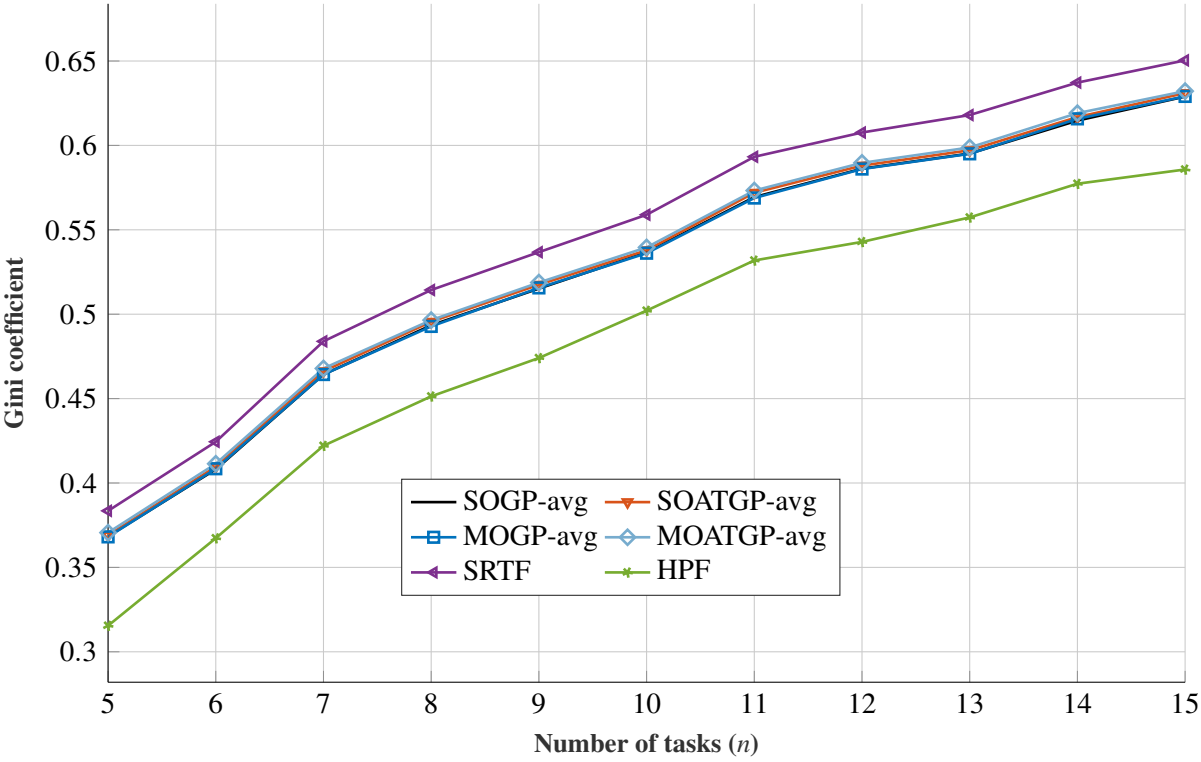


Figure 5.42: Gini coefficient for different number of tasks in a task set.

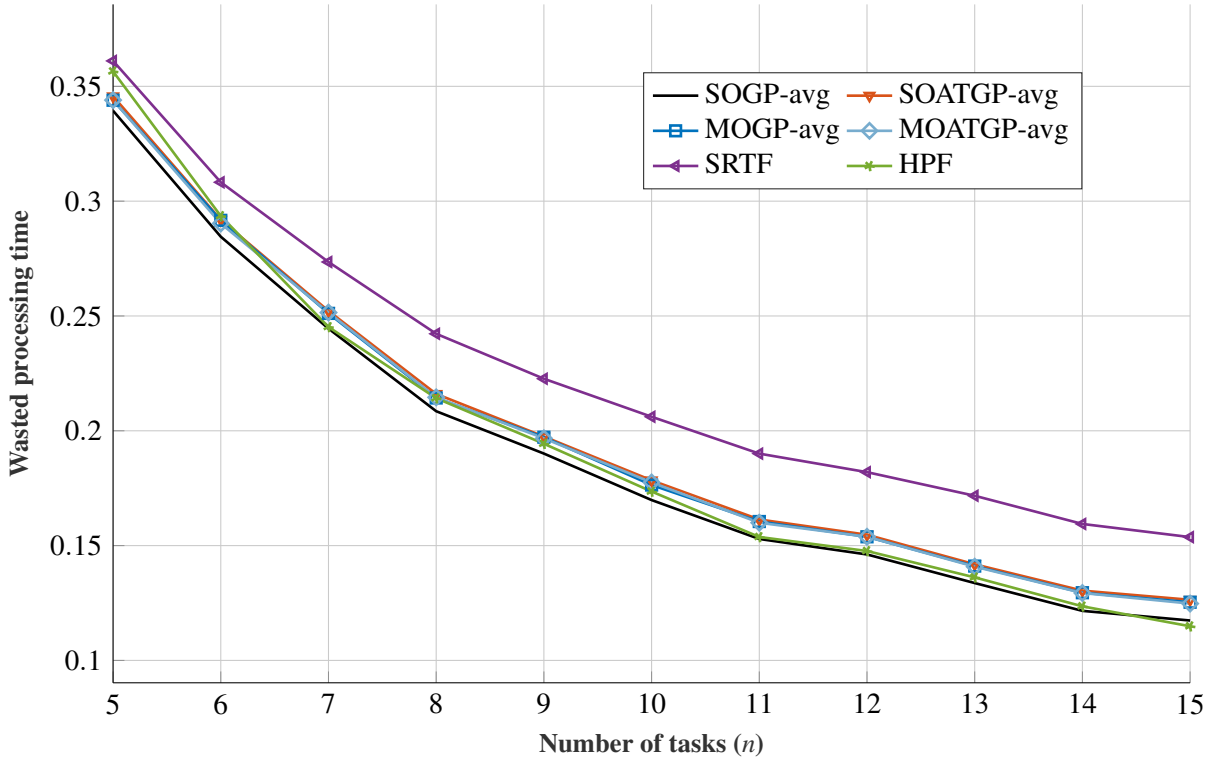


Figure 5.43: Wasted processing time for different number of tasks in a task set.

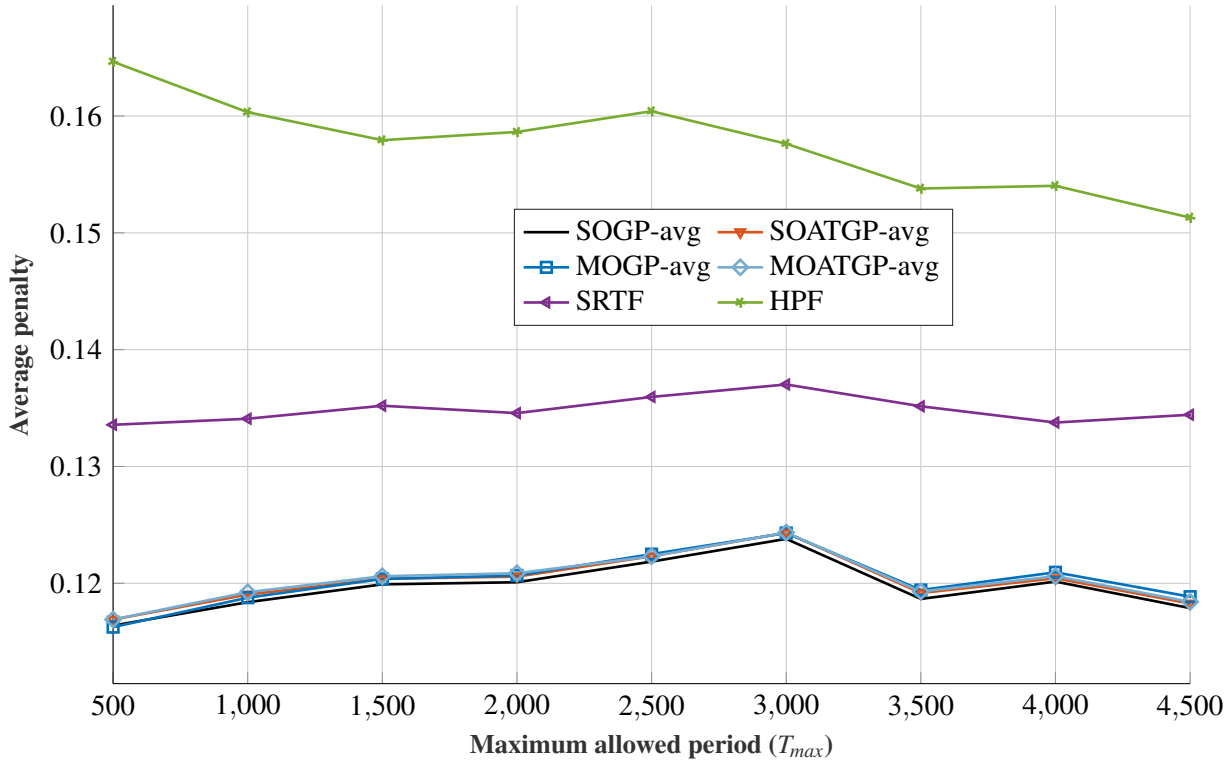


Figure 5.44: Average penalty for different maximum allowed period.

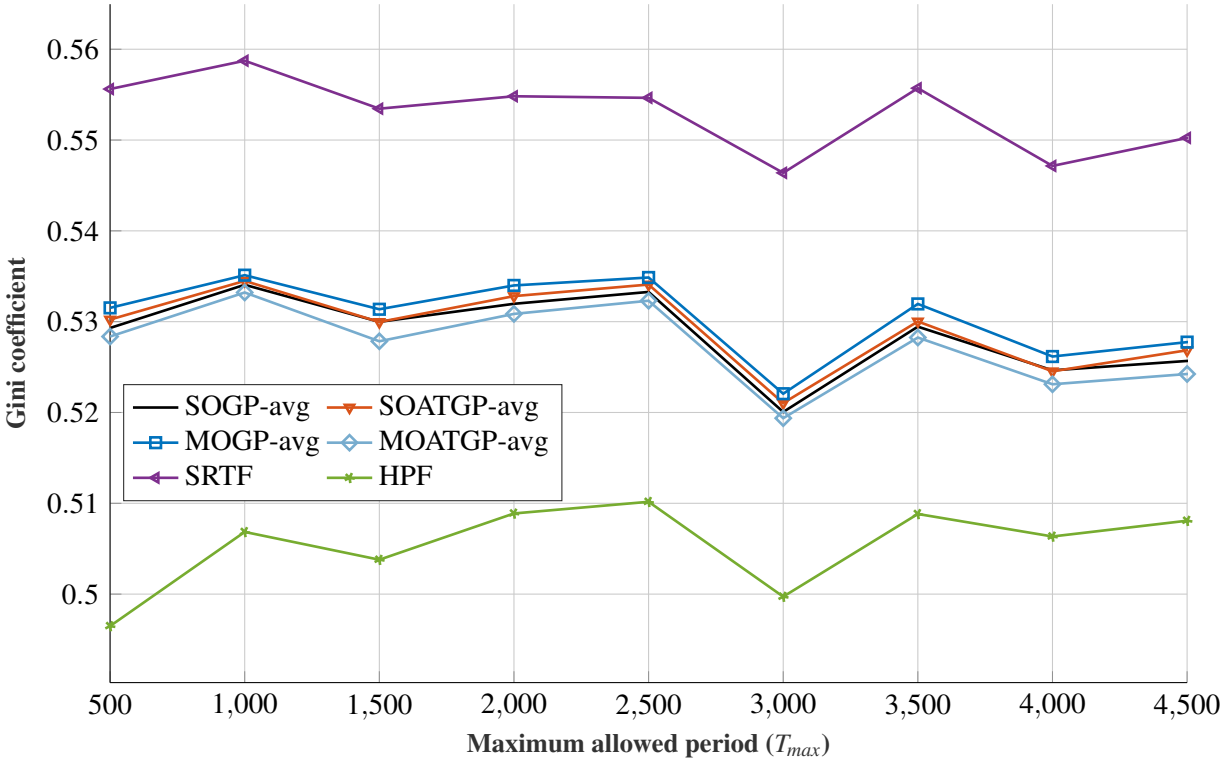


Figure 5.45: Gini coefficient for different maximum allowed period.

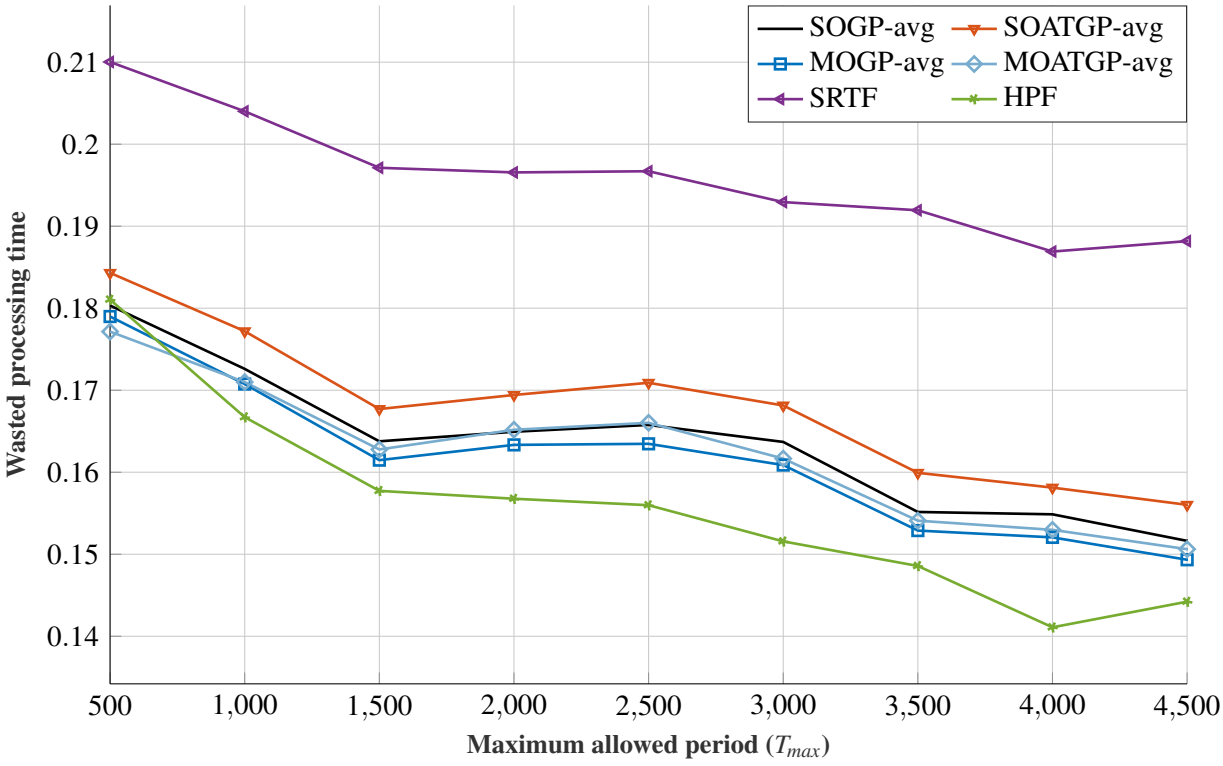


Figure 5.46: Wasted processing time for different maximum allowed period.

5.8 Designing operating parameters for real-time operating system with partitioned scheduler

In chapter 4, task periods were identified as crucial operating parameters, which have to be determined in safety-critical control applications. In addition, in this chapter, the generation of heuristic for low-criticality tasks for maximizing the quality of service in adaptive mixed-criticality systems is addressed. These two configurations are not compatible, however the methods can be used for simultaneous design of the mixed-criticality system with the partitioned scheduler that is depicted in Fig. 2.5b. This can be especially useful when simple operating system and computing platforms are used. For instance, consider a system which has a partitioned scheduler $\mathscr{S}$, which schedules the tasks in the following manner:

1. Schedule safety-critical tasks according to rate-monotonic scheduler $\mathscr{S}_{RM}$ if there are safety-critical tasks to be executed.
2. Schedule non-critical tasks in slack time of safety-critical task using the heuristic scheduler $\mathscr{S}_H$.

The design of operating parameters, i.e., task periods, in the first part of the system can be performed using the harmonic period assignment method proposed in chapter 4. Assigning harmonic periods will ensure the schedulability for any desired utilization of the processor lower than 1. Moreover, using the method, the amount of time which is left for the execution non-critical tasks that are executed in the slack time using the heuristic scheduler $\mathscr{S}_H$ can be regulated. When the periods are optimized, the approach proposed in this chapter can be used for obtaining the adequate heuristic for scheduling of tasks. Note that the scheduling procedure described above, once again corresponds to the scheduling meta-algorithm, i.e., it describes the system operation. An example of operating system, which has a scheduling meta-algorithm as described above, is KONČAR Grap operating system [76]. In Grap, safety-critical tasks must have harmonic periods and they are scheduled by rate-monotonic scheduler. In addition, Grap implements the *background* scheduler, which executes non-critical or low-criticality tasks, when there are no active safety-critical tasks. In the following example, it is briefly illustrated how these two methods can be used in practice simultaneously.

Example 14. *Consider the MC task set given in Table 5.5. The first goal is to determine periods of safety-critical tasks from period ranges given with p_i^{min} and p_i^{max} such that at maximum* 4 *different periods are used and utilization has to be less or equal to* 0.8*, leaving* 20% *of processor time for execution of LO tasks. This is easily accomplished using the* ***PE+OTA*** *as presented in example in section 4.9.5. Obtained period values are shown in bold in Table 5.5 in column $T_i = D_i$. In addition, goal is to find a scheduling policy that minimizes the number of skips of LO tasks in the hyperperiod of execution. This can be done in similar manner as it was demonstrated before by exploiting the devised genetic programming approach. Again, note that*

the system is overloaded:

$$U = \sum_{i}^{n} \frac{C_i}{T_i} = \frac{1}{5} + \frac{1}{15} + \frac{1}{15} + \frac{5}{15} + \frac{3}{30} + \frac{2}{60} + \frac{2}{15} + \frac{1}{30} + \frac{5}{60} + \frac{2}{60} \approx 1.08 \tag{5.19}$$

Table 5.5: Task set parameters for Example 14.

Task	C_i	p_i^{min}	p_i^{max}	$T_i = D_i$	L_i
τ_1	1	2	6	**5**	HI/SIL4
τ_2	1	6	17	**15**	HI/SIL4
τ_3	1	8	26	**15**	HI/SIL4
τ_4	5	11	34	**15**	HI/SIL4
τ_5	3	16	51	**30**	HI/SIL4
τ_6	2	33	108	**60**	HI/SIL4
τ_7	2	–	–	15	LO/SIL0
τ_8	1	–	–	30	LO/SIL0
τ_9	5	–	–	60	LO/SIL0
τ_{10}	2	–	–	60	LO/SIL0

In Figs. 5.47-5.48, schedule according to the EDF and the SRTF heuristic is shown. It can be seen that 3 *jobs are skipped in both cases, i.e., J_{71}, J_{74}, J_{82} for the EDF, and J_{71}, J_{73}, J_{91} for the SRTF, since they did not finish their execution before the deadline. By introducing the priority function given with:*

$$\pi_i(t) = \frac{d_i}{\gamma_i} \tag{5.20}$$

the total number of skips can be reduced to 2 *as it can be seen in Fig. 5.49 where jobs J_{71} and J_{91} are skipped.*

These two methods enable design of a partitioned system in a way that both parts of the system can be optimized. Firstly, the optimal periods are chosen for the safety-critical part of the system which ensure schedulability and high quality of service of the critical tasks in a sense of the frequency of safety-critical tasks execution, i.e., the utilization is maximized. Secondly, a scheduling heuristic is chosen for scheduling of non-critical tasks, which optimizes the quality of service that corresponds to the total number of skips of non-critical tasks.

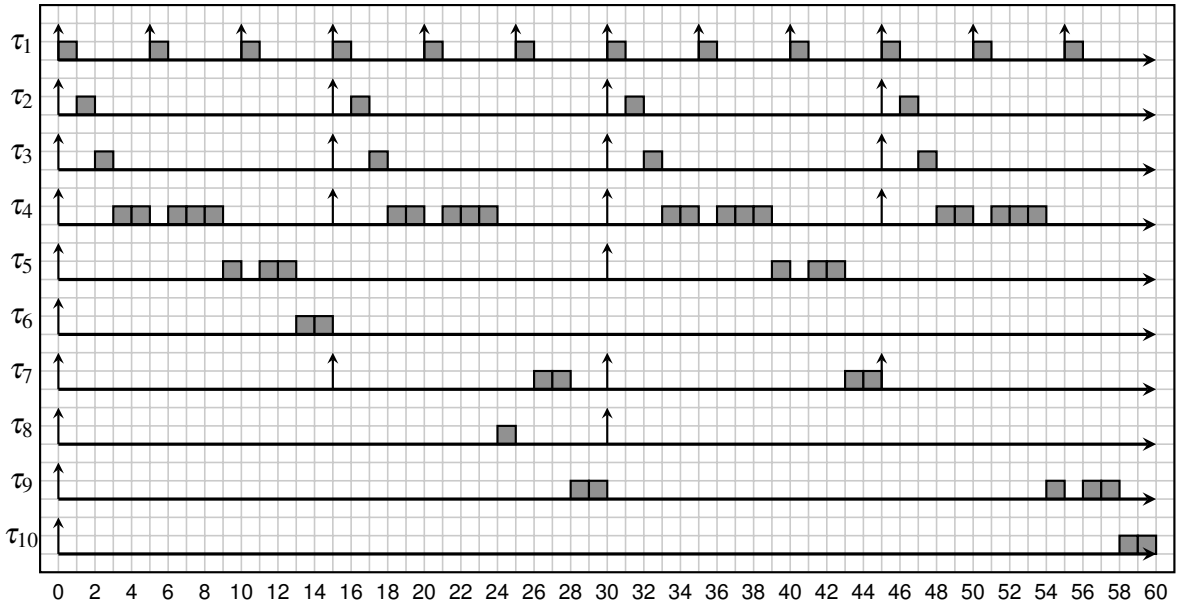


Figure 5.47: Low-criticality tasks scheduled according to the EDF heuristic for task set in Table 5.5.

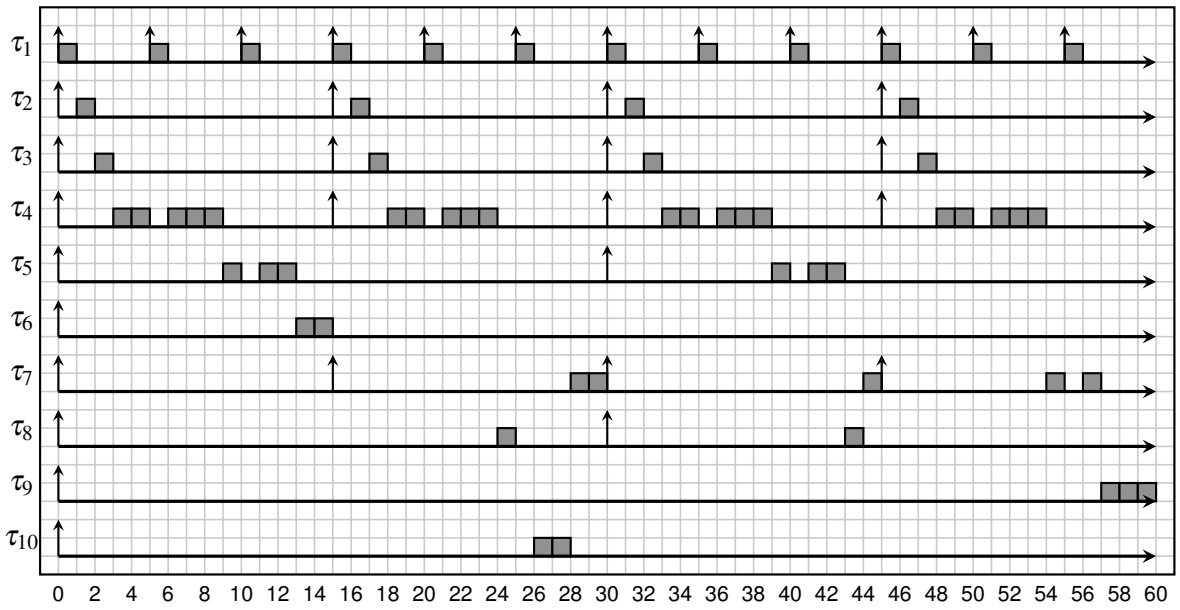


Figure 5.48: Low-criticality tasks scheduled according to the SRTF heuristic for task set in Table 5.5.

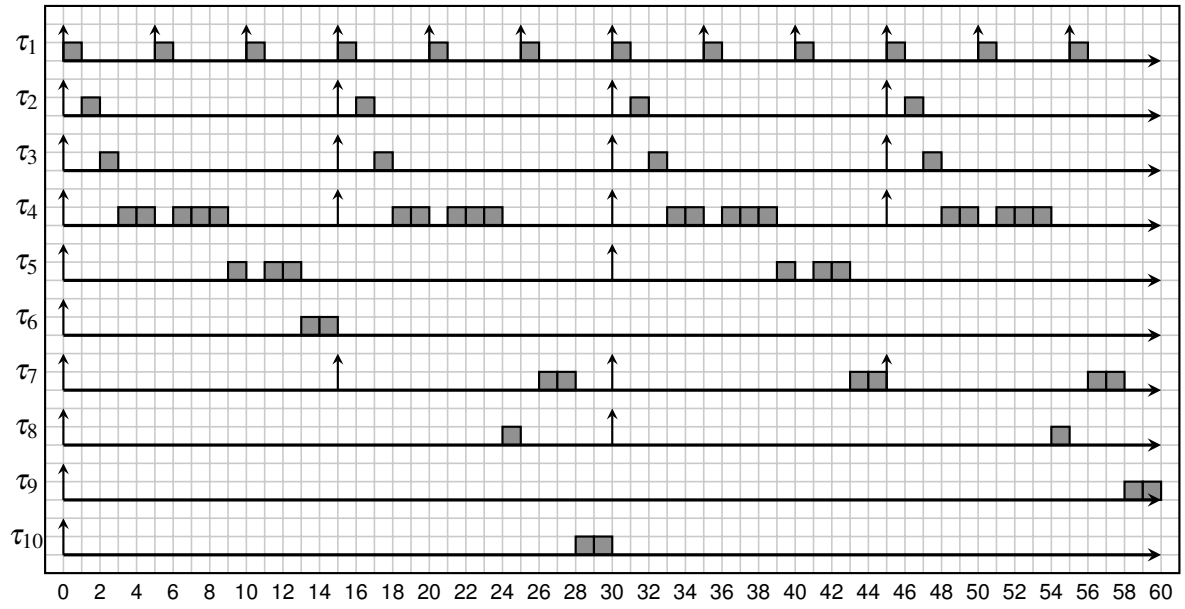


Figure 5.49: Low-criticality tasks scheduled according to the scheduling policy given with (5.20) for task set in Table 5.5.

5.9 Chapter summary

In this chapter, a genetic programming method was devised, which can be used to evolve scheduling heuristics that can mitigate the effects of overload in mixed-criticality systems by increasing the custom performance metric of low-criticality tasks. It was shown that the devised genetic programming approach in the context of mixed-criticality embedded system design has several significant advantages in comparison to the exhaustive optimal approaches that are often computationally demanding since the observed scheduling problem is likely to be NP-hard. The great benefits of the approach are automatism and flexibility, which allow independent and straightforward customization of scheduling environment, scheduling meta-algorithm and performance metrics for low-criticality tasks. Moreover, scheduling heuristics that are designed off-line can be efficiently implemented and computed, which makes them suitable for on-line implementation in mixed-criticality embedded systems which often have limited computing power. In this chapter, the effectiveness of the approach was demonstrated in generating heuristics for dynamically scheduled synchronous mixed-criticality systems which employ job skipping as technique of mitigating overload. Based on the experimental evaluation, it is concluded that generated heuristics can optimize custom performance metrics such as average grade of service or average penalty of low-criticality tasks in high-criticality mode. Moreover, generated heuristics show better performance than the other heuristics and naive approaches in the literature. Furthermore, it was demonstrated that these heuristics acquire generality in the training process, which makes them reusable for scheduling in different overload scenarios and for different task sets. In addition, the possibility of usage of the multi-objective optimization approach based on NSGA-II algorithm was studied as well as evolving acceptance test using

cooperative evolution. These approaches show slight improvement over the single objective approach in some scenarios. Moreover, it was demonstrated how the method can be used in combination with the period assignment method devised in chapter 4 for design of mixed-criticality system with partitioned scheduler.

Chapter 6

Conclusion

In this research various aspects of real-time mixed-criticality systems were investigated in order to provide new methods for analysis and design of real-time mixed-criticality systems. In this chapter the concluding remarks of the research will be systematically stated. In contrast to summary at the end of each chapter, the high-level conclusion with regard to the hypotheses and the contributions stated in the introduction will be provided.

6.1 The goal of the research and hypotheses revisited

The goal of the proposed research was to provide methods for improving schedulability and quality of service in real-time mixed-criticality systems.

The main hypotheses of the research were the following:

1. Schedulability tests for fixed-priority adaptive mixed-criticality system model can be improved.
2. Existing harmonic period assignment approaches can be improved by imposing additional constraints on a system design.
3. Heuristics generated using genetic programming can be used in mixed-criticality systems to increase the quality of service of non-critical tasks.

Each hypothesis implies that there is an optimization method for schedulability or quality of service in design of real-time mixed-criticality system. Therefore, the focus of this research was on providing these methods as scientific contribution.

6.2 Research results

In chapter 3, the sufficient schedulability test for adaptive mixed-criticality system was devised. It was devised by making the existing analysis more precise and reducing both the low-

criticality and high-criticality interference. It was shown that the test dominates other sufficient schedulability tests, such as AMC-max, AMC-rtb, SMC and SMC-NO in terms of admittance of schedulability, and that it dominates exact schedulability tests in terms of runtime performance and time complexity. The dominance was demonstrated theoretically, on simple numerical examples, and with the evaluation on the large number of synthetically generated task sets. Moreover, a framework for schedulability testing which enables the reproducibility of results was developed. Using the framework, errors and inconsistencies in the existing exact schedulability test for fixed-priority adaptive mixed-criticality systems were discovered and corrected. These corrections were validated and verified with examples and extensive evaluation.

In chapter 4, the method for harmonic periods assignment with distinct number of different period values was devised to address the problems encountered in safety-critical system design. The new harmonic period assignment problem referred to UDHPA and AUDHPA were formulated and analyzed. These problems were shown to be more flexible than the existing UDHPA problem since they have the constraints on the different number of distinct period values and allow arbitrary utilization. The existing approaches that are based on forward and backward search of harmonic projections were shown to be inadequate in instances of these problems, and therefore in this research a novel optimal harmonic period assignment was devised. Using the extensive evaluation on synthetically generated task sets, it was shown that the devised method performs better than other approaches from the literature in terms of the total resulting utilization, schedulability and usability in systems with constraints on different number of period values. The method enables optimization of periods in systems with arbitrary utilization as well, unlike the other methods in the literature.

In chapter 5, the method for generating heuristics using the genetic programming for scheduling of non-critical or low-criticality tasks in mixed-criticality system for improving the quality of service was devised. It was shown that the usage of genetic programming for generation of priority functions increases the performance of non-critical or low-criticality tasks with regard to certain performance metrics. Moreover, a great benefit of the proposed approach is that these heuristics, i.e., priority functions, can be easily employed in the real-time operating systems due to effectively constant time complexity. Furthermore, the devised framework is modular due to the independence of the definition of the scheduling problem through different scheduling meta-algorithms and the optimization, i.e., genetic algorithm. Therefore, various configurations of the method were tested. It was shown that usage of this method enabled generating heuristics which perform better than single-variable based heuristics that can be found in the literature. This was verified on a small numerical example, as well as on the extensive evaluation on the large number of synthetically generated task sets.

6.3 Future work

There are few possible directions for the future work based on this research, such as:

- weakly hard real-time systems,
- genetic programming in overloaded weakly hard-real time systems,
- investigation of harmonic period assignment methods,
- investigation of harmonic adaptive mixed-criticality systems.

The area of mixed-criticality systems is tightly coupled with the area of weakly hard real-time systems where the hard real-time and soft real-time constraints are not modeled in terms of criticality. Moreover, there is a lot of ongoing research that attempts to extend the weakly hard real-time models to mixed-criticality and vice-versa with the aim of further increasing the expressiveness and usability of the models.

As shown in the research, genetic programming is a useful method for generating heuristics in different system configurations. Although the research was focused only on the scheduling problems with timing constraints, it is possible to employ this method in systems with energy constraints as well.

The area of the harmonic period assignment is especially important in safety-critical systems. In this research the period enumeration algorithm was introduced which can find any possible m-tuple of period values such that their product is less than some upper bound value. In analytic number theory, a similar problem exists and it is known as Piltz divisor problem. In future work, the connection between these problems shall be investigated.

Finally, a possible research direction is in the domain of the adaptive mixed-criticality systems with harmonic periods. It is possible that adding the harmonic period constraints in the system can significantly reduce the complexity of the analysis and increase the schedulability in fixed-priority systems.

6.4 Final thoughts

Three methods for optimization of schedulability and quality of service in real-time mixed-criticality systems were devised as the part of this research, and each method addresses a particular problem within the specific system configuration. Since the increase of different embedded computing platforms and possible system configurations is nowadays somewhat overwhelming, the need for research of methods encompassed by this disseration is obvious. This was well highlighted and emphasized in the introductory motivation for this research. Although the methods devised in this thesis are tailored to suit system models of interest, i.e., models for real-time safety-critical and mixed-criticality systems, they have a great potential to be extended and reapplied for possible alternative models. Starting with the extensible framework for static and

adaptive mixed-criticality schedulability testing in chapter 3, to deep insights about enumerating m-tuples of harmonic values in chapter 4 and to modular and modifiable framework for generation of heuristics using the genetic programming in chapter 5. The advances in all of the aforementioned areas are certainly needed and lot of future research can certainly be conducted to extend the findings of this thesis.

On the other hand, one can pose a question whether such research is really needed. From a practical point of view the most of well-established scheduling algorithms commonly used in industrial real-time systems and especially in safety-critical systems, have been already devised in early real-time research days in 1970s and 1980s. For instance, the rate-monotonic and the earliest deadline first scheduling algorithms are overwhelmingly used in safety-critical systems. More advanced models, that were introduced in last 40 years, are used in systems in a proportion that is significantly smaller than the formerly mentioned algorithms. Additionally, it is well known that for a specific product, e.g., safety-critical embedded computer in railway, engineers dictate which implementation of software shall be used. The answer to "which approach to use" is predominantly, and for a good reason, the simplest approach available. Therefore, not the ideal or a more efficient, but the simplest and one that is proven to work. Although more efficient and better approaches exist, they will be pragmatically discarded due to cost-efficiency, backward compatibility etc. In the context of safety-critical systems, the time consuming process of system certification further influences engineers to take the simplest possible implementations. Although in majority of cases the novel methods and algorithms shall not be implemented, in a substantial minority some methods will certainly be extensively used and they will replace the less efficient variants. For instance in context of this thesis, one can refer to the method for sufficient schedulability testing of periodic adaptive mixed-criticality system that, in comparison with exhaustive test, significantly reduces the computation time. Similarly, the method for period assignment devised in this research is efficient in a sense of time complexity and increases degrees of freedom in the system design. Hence, although the research impact on industrial practices is not extensive it can be used in particular systems of interest.

There are two additional observations in regard to the relevance of this and similar researches. First off, a lot of devised techniques and gained insights cannot be immediately reused in practice but they gain relevance in the same or similar research areas over the time. Secondly, the relevance can sometimes be mitigated due to the significant increase in publishing of research papers which can lead to a loss of the central premises in the field in some papers. In the context of this research and this particular field, the latter phenomenon can be noticed. As argued in the introductory part, this phenomenon is detachment from the initial notion of criticality that is found in the safety standards, which is primarily linked to a system safety assessment process and following a guidelines provided by a specific safety standards in system

design. On the other hand, a large portion of the research devised in academia is primarily linked to the notion of epistemic uncertainty with regard to the knowledge of the worst-case execution time of tasks as discussed in the introduction. This misconception has sometimes led to research results that are in the end inapplicable in practice. Nevertheless, as mentioned before, a lot of concepts introduced in this and similar researches are vital to the better understanding of the processes in safety-critical systems, mixed-criticality systems, embedded systems, etc.

In that regard, this research attempts to provide a theoretical basis, in form of methods and algorithms, which can be used in development of tools for design and analysis of real-time mixed-criticality systems. In order to enable easier utilization of this research, a lot of small numerical examples have been provided. On the other hand, from purely theoretical perspective, research provides a lot of useful observations about the nature of the observed systems such as the schedulability bounds, the maximum number of different period values in a certain range, and existence of optimal heuristics for certain problem instances. Although some of these conclusions may not be applicable directly in practical engineering problems or industrial context, they tell us about the boundaries of the behavior of the system. The knowledge of the boundaries for any particular system is crucial for the efficient system design. As Harry Callahan once said: "A man’s got to know his limitations.".

Bibliography

[1] "Safetram: System for increased driving safety in public urban rail traffic," 2019, accessed: 2020-09-24. [Online]. Available: https://www.koncar-institut.hr/en/content-center/projects/safetram/

[2] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973. [Online]. Available: http://doi.acm.org/10.1145/321738.321743

[3] A. K. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," USA, Tech. Rep., 1983.

[4] E. Fersman, P. Krcal, P. Pettersson, and W. Yi, "Task automata: Schedulability, decidability and undecidability," *Inf. Comput.*, vol. 205, no. 8, p. 1149–1172, Aug. 2007. [Online]. Available: https://doi.org/10.1016/j.ic.2007.01.009

[5] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The digraph real-time task model," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2011, pp. 71–80.

[6] S. Baruah, "The non-cyclic recurring real-time task model," in *2010 31st IEEE Real-Time Systems Symposium*, 2010, pp. 173–182.

[7] S. K. Baruah, "Dynamic- and static-priority scheduling of recurring real-time tasks," *Real-Time Syst.*, vol. 24, no. 1, p. 93–128, Jan. 2003. [Online]. Available: https://doi.org/10.1023/A:1021711220939

[8] N. Tchidjo Moyo, E. Nicollet, F. Lafaye, and C. Moy, "On schedulability analysis of non-cyclic generalized multiframe tasks," in *Proceedings of the 2010 22nd Euromicro Conference on Real-Time Systems*, ser. ECRTS '10. USA: IEEE Computer Society, 2010, p. 271–278. [Online]. Available: https://doi.org/10.1109/ECRTS.2010.24

[9] S. Baruah, D. Chen, S. Gorinsky, and A. Mok, "Generalized multiframe tasks," *Real-Time Syst.*, vol. 17, no. 1, p. 5–22, Jul. 1999. [Online]. Available: https://doi.org/10.1023/A:1008030427220

[10] A. K. Mok and D. Chen, "A multiframe model for real-time tasks," *IEEE Transactions on Software Engineering*, vol. 23, no. 10, pp. 635–645, 1997.

[11] A. Esper, G. Nelissen, V. Nélis, and E. Tovar, "How realistic is the mixed-criticality real-time system model?" in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, ser. RTNS '15. New York, NY, USA: ACM, 2015, pp. 139–148. [Online]. Available: http://doi.acm.org/10.1145/2834848.2834869

[12] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan 2004.

[13] D. P. Siewiorek and R. S. Swarz, *Reliable Computer Systems (3rd Ed.): Design and Evaluation*. Natick, MA, USA: A. K. Peters, Ltd., 1998.

[14] J.-C. Geffroy and G. Motet, *Design of Dependable Computing Systems*. Hingham, MA, USA: Kluwer Academic Publishers, 2002.

[15] L. D. Gowen, J. S. Collofello, and F. W. Calliss, "Preliminary hazard analysis for safety-critical software systems," in *Eleventh Annual International Phoenix Conference on Computers and Communication [1992 Conference Proceedings]*, April 1992, pp. 501–508.

[16] "Fault tree analysis (FTA)," International Electrotechnical Commission, Geneva, CH, Standard, 2006.

[17] "Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)," International Electrotechnical Commission, Geneva, CH, Standard, Mar. 2006.

[18] "DO-178C, Software Considerations in Airborne Systems and Equipment Certification," Radio Technical Commission for Aeronautics, U.S.A, Standard, Jan. 2012.

[19] "Functional safety of electrical/electronic/programmable electronic safety-related-system," International Electrotechnical Commission, Geneva, CH, Standard, 2010.

[20] "Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems (EN 50128:2011)," International Electrotechnical Commission, Brussels, Belgium, Standard, 2013.

[21] "Road vehicles – Functional safety," International Organization for Standardization, Geneva, CH, Standard, Nov. 2011.

[22] "Automotive safety integrity level," 2020, accessed: 2020-09-06. [Online]. Available: https://en.wikipedia.org/wiki/Automotive_Safety_Integrity_Level

[23] "Unleash the unparalleled power and flexibility of zynq ultrascale+ mpsocs," 2016, accessed: 2021-04-07. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp470-ultrascale-plus-power-flexibility.pdf

[24] I. Pavić and H. Džapo, "Virtualization in multicore real-time embedded systems for improvement of interrupt latency," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, pp. 1405–1410.

[25] ——, "Commentary to: An exact schedulability test for fixed-priority preemptive mixed-criticality real-time systems," *Real-Time Systems*, vol. 56, no. 1, pp. 112–119, 2020.

[26] ——, "Framework for evaluation of schedulability tests for mixed-criticality systems," in *2021 44th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2021.

[27] 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," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, May 2008. [Online]. Available: https://doi.org/10.1145/1347375.1347389

[28] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 82:1–82:37, Nov. 2017.

[29] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, Dec 2007, pp. 239–243.

[30] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *2011 IEEE 32nd Real-Time Systems Symposium*, Nov 2011, pp. 34–43.

[31] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1140–1152, Aug 2012.

[32] N. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," 1991.

[33] F. Dorin, P. Richard, M. Richard, and J. Goossens, "Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities," *Real-Time Systems*, vol. 46, no. 3, pp. 305–331, Dec 2010. [Online]. Available: https://doi.org/10.1007/s11241-010-9107-4

[34] S. Baruah and A. Burns, "Implementing mixed criticality systems in ada," in *Proceedings of the 16th Ada-Europe International Conference on Reliable Software Technologies*, ser. Ada-Europe'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 174–188.

[35] K. Agrawal and S. Baruah, "Intractability issues in mixed-criticality scheduling," in *ECRTS*, 2018.

[36] V. Bonifaci and A. Marchetti-Spaccamela, "Feasibility analysis of sporadic real-time multiprocessor task systems," *Algorithmica*, vol. 63, no. 4, pp. 763–780, Aug 2012. [Online]. Available: https://doi.org/10.1007/s00453-011-9505-6

[37] S. Asyaban and M. Kargahi, "An exact schedulability test for fixed-priority preemptive mixed-criticality real-time systems," *Real-Time Syst.*, vol. 54, no. 1, pp. 32–90, Jan. 2018.

[38] R. I. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Systems*, vol. 47, no. 1, pp. 1–40, Jan 2011. [Online]. Available: https://doi.org/10.1007/s11241-010-9106-5

[39] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1, pp. 129–154, May 2005. [Online]. Available: https://doi.org/10.1007/s11241-005-0507-9

[40] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: A flexible real time scheduling framework," *Ada Lett.*, vol. XXIV, no. 4, p. 1–8, Nov. 2004. [Online]. Available: https://doi.org/10.1145/1046191.1032298

[41] Y. Matsubara, Y. Sano, S. Honda, and H. Takada, "An open-source flexible scheduling simulator for real-time applications," in *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, 2012, pp. 16–22.

[42] A. Manacero, M. B. Miola, and V. A. Nabuco, "Teaching real-time with a scheduler simulator," in *31st Annual Frontiers in Education Conference. Impact on Engineering and Science Education. Conference Proceedings (Cat. No.01CH37193)*, vol. 2, 2001, pp. T4D–15.

[43] A. Diaz, R. Batista, and O. Castro, "Realtss: a real-time scheduling simulator," in *2007 4th International Conference on Electrical and Electronics Engineering*, 2007, pp. 165–168.

[44] C. Yaashuwanth and R. Ramesh, "Web-enabled framework for real-time scheduler simulator (a teaching tool)," in *2010 Second International Conference on Computer Research and Development*, 2010, pp. 826–830.

[45] "Mc-evaluation - a framework for mixed-criticality schedulability testing," 2019, accessed: 2021-02-24. [Online]. Available: https://github.com/dumpram/mceval

[46] R. I. Davis, A. Zabos, and A. Burns, "Efficient exact schedulability tests for fixed priority real-time systems," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1261–1276, 2008.

[47] S. K. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, *Mixed-Criticality Scheduling of Sporadic Task Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 555–566.

[48] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-Time Systems*, vol. 50, no. 1, pp. 48–86, Jan 2014.

[49] R. West, Y. Li, E. Missimer, and M. Danish, "A virtualized separation kernel for mixed-criticality systems," *ACM Trans. Comput. Syst.*, vol. 34, no. 3, Jun. 2016. [Online]. Available: https://doi.org/10.1145/2935748

[50] G. Li and S. Top, "I/o sharing in a multi-core kernel for mixed-criticality applications," in *Embedded Systems: Design, Analysis and Verification*, G. Schirner, M. Götz, A. Rettberg, M. C. Zanella, and F. J. Rammig, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 331–342.

[51] C. Evripidou and A. Burns, "Scheduling for Mixed-criticality Hypervisor Systems in the Automotive Domain," in *WMC 2016 4th International Workshop on Mixed Criticality Systems*, Porto, Portugal, Nov. 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01419143

[52] A. Tellabi, M. Parekh, C. Ruland, and M. Ezziyyani, "A case study of virtualization used in mixed criticality systems," in *Advanced Intelligent Systems for Sustainable Development (AI2SD'2019)*, M. Ezziyyani, Ed. Cham: Springer International Publishing, 2020, pp. 1–11.

[53] S. Trujillo, A. Crespo, A. Alonso, and J. Pérez, "Multipartes: Multi-core partitioning and virtualization for easing the certification of mixed-criticality systems," *Microprocessors and Microsystems*, vol. 38, no. 8, Part B, pp. 921–932, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933114001380

[54] I. Pavić and H. Džapo, "Optimal harmonic period assignment with constrained number of distinct period values," *IEEE Access*, vol. 8, pp. 175 697–175 712, 2020.

[55] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. USA: Kluwer Academic Publishers, 1997.

[56] D. Seto, J. P. Lehoczky, and Liu Sha, "Task period selection and schedulability in real-time systems," in *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, Dec 1998, pp. 188–198.

[57] E. Bini and M. Di Natale, "Optimal task rate selection in fixed priority systems," in *26th IEEE International Real-Time Systems Symposium (RTSS'05)*, Dec 2005, pp. 11 pp.–409.

[58] C. . Han and H. . Tyan, "A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms," in *Proceedings Real-Time Systems Symposium*, Dec 1997, pp. 36–45.

[59] Ching-Chih Han, Kwei-Jay Lin, and Chao-Ju Hou, "Distance-constrained scheduling and its applications to real-time systems," *IEEE Transactions on Computers*, vol. 45, no. 7, pp. 814–826, July 1996.

[60] V. Bonifaci, A. Marchetti-Spaccamela, N. Megow, and A. Wiese, "Polynomial-time exact schedulability tests for harmonic real-time tasks," in *2013 IEEE 34th Real-Time Systems Symposium*, Dec 2013, pp. 236–245.

[61] P. Ekberg and W. Yi, "Fixed-priority schedulability of sporadic tasks on uniprocessors is np-hard," in *2017 IEEE Real-Time Systems Symposium (RTSS)*, Dec 2017, pp. 139–146.

[62] M. Mohaqeqi, M. Nasri, Y. Xu, A. Cervin, and K.-E. Årzén, "Optimal harmonic period assignment: complexity results and approximation algorithms," *Real-Time Systems*, vol. 54, pp. 830–860, 2018.

[63] M. Nasri, G. Fohler, and M. Kargahi, "A framework to construct customized harmonic periods for real-time systems," in *2014 26th Euromicro Conference on Real-Time Systems*, July 2014, pp. 211–220.

[64] M. Nasri and G. Fohler, "An efficient method for assigning harmonic periods to hard real-time tasks with period ranges," in *2015 27th Euromicro Conference on Real-Time Systems*, July 2015, pp. 149–159.

[65] J. Xu, "A method for adjusting the periods of periodic processes to reduce the least common multiple of the period lengths in real-time embedded systems," in *Proceedings of 2010 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*, 2010, pp. 288–294.

[66] I. Ripoll and R. Ballester-Ripoll, "Period selection for minimal hyperperiod in periodic task systems," *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1813–1822, Sep. 2013.

[67] Y. Xu, A. Cervin, and K. Årzén, "Harmonic scheduling and control co-design," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug 2016, pp. 182–187.

[68] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén, "Feedback–feedforward scheduling of control tasks," *Real-Time Systems*, vol. 23, pp. 25–53, 07 2002.

[69] E. Bini and A. Cervin, "Delay-aware period assignment in control systems," in *2008 Real-Time Systems Symposium*, Nov 2008, pp. 291–300.

[70] Chi-Sheng Shih, S. Gopalakrishnan, P. Ganti, M. Caccamo, and Lui Sha, "Scheduling real-time dwells using tasks with synthetic periods," in *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, Dec 2003, pp. 210–219.

[71] Huan Li, J. Sweeney, K. Ramamritham, R. Grupen, and P. Shenoy, "Real-time support for mobile robotics," in *The 9th IEEE Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings.*, 2003, pp. 10–18.

[72] A. Easwaran, I. Lee, O. Sokolsky, and S. Vestal, "A compositional scheduling framework for digital avionics systems," in *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2009, pp. 371–380.

[73] S. Anssi, S. Kuntz, S. Gérard, and F. Terrier, "On the gap between schedulability tests and an automotive task model," *Journal of Systems Architecture*, vol. 59, no. 6, pp. 341 – 350, 2013.

[74] "Ansys scade suite: Model-based development," https://www.ansys.com/products/embedded-software/ansys-scade-suite, 2019, accessed: 2019-05-25.

[75] "Silworx: Simplify safety engineering," 2019, accessed: 2019-06-17. [Online]. Available: https://www.hima.com/en/products-services/silworx

[76] "Grap-ide - powerful graphical programming, service and diagnostic tool," 2019, accessed: 2019-05-25. [Online]. Available: https://www.koncar-institut.hr/en/?solution_group=programming-tools

[77] I. Lee, J. Y.-T. Leung, and S. H. Son, *Handbook of Real-Time and Embedded Systems*, 1st ed. Chapman and Hall/CRC, 2007.

[78] "Programmable controlers - Part 3: Programming Languages," International Electrotechnical Commission, Geneva, CH, Standard, 2014.

[79] C. Wolff, L. Krawczyk, R. Höttger, C. Brink, U. Lauschner, D. Fruhner, E. Kamsties, and B. Igel, "Amalthea — tailoring tools to projects in automotive software development," in *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, vol. 2, Sep. 2015, pp. 515–520.

[80] I. Sañudo, P. Burgio, and M. Bertogna, "Schedulability and Timing Analysis of Mixed Preemptive-Cooperative Tasks on a Partitioned Multi-Core ystem," in *WATERS - International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2016.

[81] M. Di Natale, "Optimizing the multitask implementation of multirate simulink models," in *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, April 2006, pp. 335–346.

[82] I. P. Gent and T. Walsh, "Analysis of heuristics for number partitioning," *Computational Intelligence*, vol. 14, no. 3, pp. 430–451, 1998. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/0824-7935.00069

[83] A. Lundell and T. Westerlund, "Global optimization of mixed-integer signomial programming problems," in *Mixed Integer Nonlinear Programming*, J. Lee and S. Leyffer, Eds. New York, NY: Springer New York, 2012, pp. 349–369.

[84] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha, "On-line scheduling in the presence of overload," in *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, Oct 1991, pp. 100–110.

[85] G. Koren and D. Shasha, "D/sup over/; an optimal on-line scheduling algorithm for overloaded real-time systems," in *[1992] Proceedings Real-Time Systems Symposium*, Dec 1992, pp. 290–299.

[86] G. C. Buttazzo, "Handling overload conditions in real-time systems," in *Real-Time Systems, Architecture, Scheduling, and Application*, S. M. Babamir, Ed. Rijeka: IntechOpen, 2012, ch. 7.

[87] A. Queudet-Marchand and M. Chetto, "Quality of service scheduling in the firm real-time systems," in *Real-Time Systems, Architecture, Scheduling, and Application*, S. M. Babamir, Ed. Rijeka: IntechOpen, 2012, ch. 9.

[88] G. Koren and D. Shasha, "Skip-over: algorithms and complexity for overloaded systems that allow skips," in *Proceedings 16th IEEE Real-Time Systems Symposium*, Dec 1995, pp. 110–117.

[89] P. Ramanathan, "Overload management in real-time control applications using (m, k)-firm guarantee," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, pp. 549–559, 1999.

[90] H. Su, D. Zhu, and D. Mossé, "Scheduling algorithms for elastic mixed-criticality tasks in multicore systems," in *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, Aug 2013, pp. 352–357.

[91] X. Dai, W. Chang, S. Zhao, and A. Burns, "A dual-mode strategy for performance-maximisation and resource-efficient cps design," *ACM Trans. Embed. Comput. Syst.*, vol. 18, no. 5s, Oct. 2019.

[92] D. Liu, N. Guan, J. Spasic, G. Chen, S. Liu, T. Stefanov, and W. Yi, "Scheduling analysis of imprecise mixed-criticality real-time tasks," *IEEE Transactions on Computers*, vol. 67, no. 7, pp. 975–991, July 2018.

[93] A. Burns, R. I. Davis, S. Baruah, and I. Bate, "Robust mixed-criticality systems," *IEEE Transactions on Computers*, vol. 67, no. 10, pp. 1478–1491, Oct 2018.

[94] M. Hikmet, M. M. Kuo, P. S. Roop, and P. Ranjitkar, "Mixed-criticality systems as a service for non-critical tasks," in *2016 IEEE 19th Int. Symp. on Real-Time Distributed Computing (ISORC)*, May 2016, pp. 221–228.

[95] T. Yoshimoto and T. Ushio, "Optimal arbitration of control tasks by job skipping in cyber-physical systems," in *IEEE/ACM 2nd Int. Conf. on Cyber-Physical Systems*, April 2011, pp. 55–64.

[96] S. Nguyen, Y. Mei, and M. Zhang, "Genetic programming for production scheduling: a survey with a unified framework," *Complex & Intelligent Systems*, vol. 3, no. 1, pp. 41–66, Mar 2017.

[97] D. Jakobović and K. Marasović, "Evolving priority scheduling heuristics with genetic programming," *Applied Soft Computing*, vol. 12, no. 9, pp. 2781 – 2789, 2012.

[98] C. Dimopoulos and A. Zalzala, "Investigating the use of genetic programming for a classic one-machine scheduling problem," *Advances in Engineering Software*, vol. 32, no. 6, pp. 489 – 498, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0965997800001095

[99] C. D. Geiger, R. Uzsoy, and H. Aytug, "Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach," *J. of Scheduling*, vol. 9, no. 1, p. 7–34, Feb. 2006. [Online]. Available: https://doi.org/10.1007/s10951-006-5591-8

[100] D. Jakobović and L. Budin, "Dynamic scheduling with genetic programming," in *Genetic Programming*, P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 73–84.

[101] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. Springer Publishing Company, Incorporated, 2008.

[102] E. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, pp. 1695–1724, 07 2013.

[103] N. Pillay and R. Qu, *Hyper-Heuristics: Theory and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2018.

[104] S. K. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, "Mixed-criticality scheduling of sporadic task systems," in *Algorithms – ESA 2011*, C. Demetrescu and M. M. Halldórsson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 555–566.

[105] R. Hunt, M. Johnston, and M. Zhang, "Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, 2014, pp. 618–625.

[106] "Ecf - evolutionary computation framework," 2017, accessed: 2021-01-04. [Online]. Available: http://ecf.zemris.fer.hr/

[107] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[108] I. Kushchu, "Genetic programming and evolutionary generalization," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 431–442, 2002.

[109] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[110] C. Coello, S. B. Gonzalez Brambila, J. Figueroa Gamboa, M. Castillo Tapia, and R. Hernández Gómez, "Evolutionary multiobjective optimization: open research areas and some challenges lying ahead," *Complex and Intelligent Systems*, vol. 6, 06 2019.

[111] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Learning reusable initial solutions for multi-objective order acceptance and scheduling problems with genetic programming," in *Proceedings of the 16th European Conference on Genetic Programming*, ser. EuroGP'13. Berlin, Heidelberg: Springer-Verlag, 2013, p. 157–168. [Online]. Available: https://doi.org/10.1007/978-3-642-37207-0_14

[112] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 193–208, 2014.

[113] C. Gini, "Measurement of inequality of incomes," *The Economic Journal*, vol. 31, no. 121, pp. 124–126, 1921. [Online]. Available: http://www.jstor.org/stable/2223319

[114] F. G. De Maio, "Income inequality measures," *Journal of Epidemiology & Community Health*, vol. 61, no. 10, pp. 849–852, 2007. [Online]. Available: https://jech.bmj.com/content/61/10/849

[115] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Parallel Problem Solving from Nature — PPSN III*, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 249–257.

[116] C. Stoean and R. Stoean, *Cooperative Coevolution*. Cham: Springer International Publishing, 2014, pp. 57–73. [Online]. Available: https://doi.org/10.1007/978-3-319-06941-8_5

[117] K. Miyashita, "Job-shop scheduling with genetic programming," in *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO'00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, p. 505–512.

[118] J. Park, Y. Mei, S. Nguyen, G. Chen, M. Johnston, and M. Zhang, "Genetic programming based hyper-heuristics for dynamic job shop scheduling: Cooperative coevolutionary approaches," in *Genetic Programming*, M. I. Heywood, J. McDermott, M. Castelli,

E. Costa, and K. Sim, Eds. Cham: Springer International Publishing, 2016, pp. 115–132.

List of Figures

List of Tables

Biography

Ivan Pavić was born on March 2nd, 1994 in Zagreb, Croatia. In 2012 he enrolled in the bachelor degree undergraduate program at the University of Zagreb, Faculty of Electrical Engineering and Computing. He completed the program and graduated in module Electronic and Computer Engineering in 2015. In 2017, he obtained the Master's degree in Electronic and Computer Engineering with the highest honors with the thesis entitled "Virtualization in Real-Time Embedded Systems". During the graduation studies, he worked as embedded software intern at Byte Lab, Inc., Zagreb.

In June 2017, he started working as a young researcher at the Faculty of Electrical Engineering and Computing on the project SafeTRAM. In November 2017, he enrolled in PhD program at the same faculty. His activities on the project including the research and design of various embedded systems platforms with special emphasis on real-time systems and safety-critical systems. During the project, he cooperated extensively with KONČAR Electrical Engineering Institute. He was actively engaged in teaching activities in the following courses: Programming industrial embedded systems, Computer Aided Design of Electronic Systems and Electronic Equipment Design. Currently, he is working as a senior researcher on A-UNIT project. His interest include scheduling, real-time systems, digital signal processing, and digital design. During the doctoral research, he published two papers in CC-indexed journals and three international conference papers.

List of published works

Papers in journals

1. I. Pavić and H. Džapo, "Commentary to: An exact schedulability test for fixed-priority preemptive mixed-criticality real-time systems," *Real-Time Systems*, vol. 56, no. 1, pp. 112–119, 2020.
2. I. Pavić and H. Džapo, "Optimal harmonic period assignment with constrained number of distinct period values," *IEEE Access*, vol. 8, pp. 175 697–175 712, 2020.

Papers at international scientific conferences

1. I. Pavić and H. Džapo, “Virtualization in multicore real-time embedded systems for improvement of interrupt latency,” in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, pp. 1405–1410.
2. I. Pavić and H. Džapo, “Energy-aware real-time scheduling for energy-harvesting sensor nodes,” in *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2020, pp. 1–6.
3. I. Pavić and H. Džapo, “Framework for evaluation of schedulability tests for mixed-criticality systems,” in *2021 44th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2021.

Životopis

Ivan Pavić rođen je 2. ožujka 1994. u Zagrebu. Upisao je preddiplomski studij na Sveučilištu u Zagrebu, Fakultetu elektrotehnike i računarstva 2012. godine. Preddiplomski studij elektroničkog i računalnog inženjerstva završio je 2015. godine. Diplomski studij elektroničkog i računalnog inženjerstva završio je 2017. godine s najvišim počastima uz diplomski rad "Virtualizacijski postupci u sustavima za rad u stvarnom vremenu". Tijekom studija radio je kao pripravnik za ugradbene računalne sustave u tvrtki Byte Lab.

U 6. mjesecu 2017. godine počeo je raditi kao mlađi istraživač na Fakultetu elektrotehnike i računarstva na projektu SafeTRAM. U studenom iste godine, upisao je doktorski studij na istom fakultetu. Njegove aktivnosti na projektu uključivale su istraživanje i razvoj različitih ugradbenih računalnih sustava s posebnim naglaskom na sustave za rad u stvarnom vremenu i sigurnosno kritične sustave. Tijekom projekta intenzivno je surađivao s tvrtkom KONČAR Institut za elektrotehniku. Bio je aktivno uključen u nastavne aktivnosti na kolegijima "Programska potpora industrijskih ugradbenih sustava" i "Konstrukcija elektroničkih uređaja". Trenutno radi kao iskusni istraživač na A-UNIT projektu. Njegovi interesi uključuju raspoređivanje, sustave za rad u stvarnome vremenu, digitalni dizajn i digitalnu obradu signala. Tijekom doktorskog istraživanja objavio je dva rada u časopisima indeksiranim u bazi CC i tri rada na međunarodnim znanstvenim konferencijama.