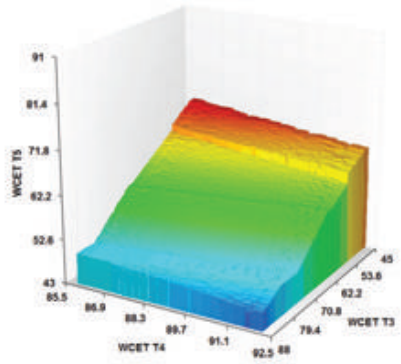
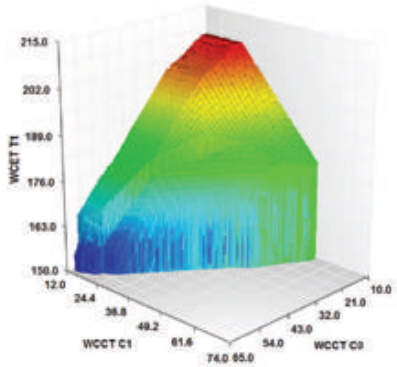
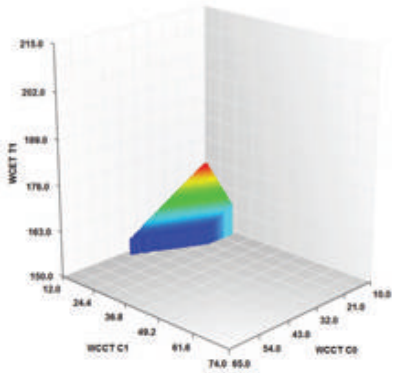
Arne Hamann

# Iterative Design Space Exploration and Robustness Optimization for Embedded Systems




Cuvillier Verlag Göttingen
Internationaler wissenschaftlicher Fachverlag

# Iterative Design Space Exploration and Robustness Optimization for Embedded Systems

Von der Carl-Friedrich-Gauß-Fakultät
Technische Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades
Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation


| | |
|---|---|
| von: | Dipl.-Inform. Arne Hamann |
| geboren am: | 3. August 1978 |
| in: | Eckernförde |



| | |
|---|---|
| eingereicht am: | 25. Juli 2008 |
| mündliche Prüfung am: | 6. Oktober 2008 |

| | |
|---|---|
| Vorsitzender: | Prof. Dr.-Ing. Lars Wolf |
| Referent: | Prof. Dr.-Ing. Rolf Ernst |
| Korreferent: | Prof. Dr.-Ing. Lothar Thiele |

**Bibliografische Information der Deutschen Nationalbibliothek**
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.
1. Aufl. - Göttingen : Cuvillier, 2008
Zugl.: (TU) Braunschweig, Univ., Diss., 2008

978-3-86727-819-5


Nonnenstieg 8, 37075 Göttingen
Telefon: 0551-54724-0
Telefax: 0551-54724-21
www.cuvillier.de


1. Auflage, 2008
Gedruckt auf säurefreiem Papier

978-3-86727-819-5

# Iterative Design Space Exploration and Robustness Optimization for Embedded Systems

ARNE HAMANN
Institute of Computer and Communication Network Engineering
Department of Electrical Engineering and Information Technology
Technical University of Braunschweig
Braunschweig, Germany

# Abstract

An embedded system is a micro computer system that is embedded into another technical device that itself does not appear as a computer. Embedded systems can be found everywhere and are going to be even more pervasive in the near future. Examples are telecommunication devices, consumer electronics, automotive systems, building technology, etc. Embedded systems have a very high influence on the system industry. Nowadays, modern products cannot be realized in a competitive manner without embedded micro computer systems.

In order to meet growing productivity demands and cost pressure, embedded systems need to be designed very efficiently. Obviously, due to the increasing complexity and distributed nature of modern systems this is by far no easy task. One key factor to increase the development process efficiency of complex distributed embedded systems is reuse. Concepts like the platform-based design style and standardization efforts on the software level (e.g. the AUTOSAR initiative in the automotive domain) allow to conceive whole product families and variants based on the same set of reusable components and sub-systems. However, while reuse helps to increase design efficiency at the functional level, it does not solve another key embedded systems challenge that arises with system complexity and sub-system integration: the control of non-functional properties, such as timing, power consumption, or dependability.

The benefits of controlling non-functional properties early in the design flow are clear. First, it helps the designer to take the right design decisions before proceeding to implementation, and thus to decrease the design risk and costs. Second, it assists the designer in conceiving systems with performance head-room for future extensions, updates, and bug-fixes. Achieving this is not trivial, since system performance is fragile with respect to modifications to the specification. Even slight changes of properties, such as execution demands, communication volumes, data rates, etc., can have severe impact on system performance. As a consequence, it is desirable that the system exhibits some sort of "robustness" towards changes.

A promising starting point to overcome these challenges are formal performance verification methods that are capable of accurately determining required performance data based on abstract application and execution platform models of a given embedded system. However, until now these methods have been mainly applied to characterize and solve isolated performance issues like, for instance, timing analysis of single ECUs, bus bottleneck detection, or bus configuration. A systematic and continuous approach allowing to control and optimize non-functional system properties is still missing.

This thesis introduces several techniques allowing to control and optimize non-functional system properties during the design flow and the lifetime of an embedded system.

- The first contribution is a flexible exploration framework that takes into account the iterative nature of performance verification during modern concurrent design flows. The proposed framework allows performing partial explorations and provides the possibility to dynamically extend and restrict the search space without loosing previously obtained results. It can, therefore, be iteratively applied to successively explore given systems taking into account data availability and responsibilities along the supply chain.

- The second contribution is the definition of expressive system robustness metrics for different assumptions and design scenarios. The proposed metrics are based on sensitivity analysis. In particular, robustness metrics for system properties with and without performance dependencies are distinguished.

- Especially the robustness metrics for the dependent case are computationally expensive, since multi-dimensional sensitivity analysis is necessary to capture interdependencies between involved properties. In order to circumvent this complexity problem, a scalable stochastic sensitivity analysis approach is developed that is capable of efficiently approximating robustness characteristics by formulating sensitivity analysis as multi-criterion optimization problem. This represents the third contribution of this thesis.

- As the fourth contribution of this thesis, it is shown how the stochastic sensitivity analysis can be used to efficiently approximate the proposed robustness metrics with upper and lower robustness bounds. Furthermore, methods are developed that utilize the derived bounds to significantly speed up robustness optimization.

- It is known that already for small systems different state-of-the-art performance analysis approaches exhibit remarkable differences in terms of accuracy. As a consequence, it can be stated that the expressiveness of the techniques proposed in this thesis highly depends on the choice of the right abstraction, i.e. analysis engine, for the considered system. For this reason, the last contribution of this thesis concerns solving the problem of providing accurate and expressive performance predictions for general embedded systems. Rather than searching for a unified more complex analysis model, it is proposed to exploit the so-called *compositional performance analysis methodology* to couple modular performance analysis techniques. In so doing, the advantages offered by the individual models of computation can easily be integrated into a cross-domain analysis that allows modeling and analyzing each system component with the best fitting method.

# Kurzfassung

Eingebettete Systeme sind Mikrocomputersysteme, die in andere technische Geräte, welche selbst nicht als Computer erscheinen, eingebettet sind. Eingebettete Systeme sind allgegenwärtig und werden in der Zukunft eine immer wichtigere Rolle spielen. Beispiele sind Telekommunikationsgeräte, Verbraucherelektronik, Automobilsysteme, Gebäudetechnik, usw. Eingebettete Systeme haben einen hohen Einfluss auf die Systemindustrie. Ohne eigebettete Systeme können moderne Produkte heutzutage nicht mehr wettbewerbsfähig hergestellt werden.

Um dem immer steigenden Produktivitätsanforderungen und Kostendruck erfolgreich zu begegnen, müssen eingebettete System sehr effizient entwickelt werden. Aufgrund der steigenden Komplexität und dem verteilten Aufbau moderner Systeme ist dies jedoch bei weitem keine einfache Aufgabe. Eine Schlüsseltechnologie, um die Effizienz von Entwicklungsprozessen für eingebettete Systeme zu steigern, ist Wiederverwendung. Konzepte wie plattformbasierter Entwurf sowie Standardisierungsinitiativen auf der Softwareebene (z.B. die AUTOSAR Initiative im Automobilsektor) erlauben es, ganze Produktfamilien und Varianten basierend auf den gleichen Komponenten und Subsystemen zu realisieren. Nichtsdestotrotz, während durch Wiederverwendung die Entwicklungseffizienz auf funktionaler Ebene gesteigert wird, bleibt eine weitere zentrale Herausforderung, welches durch Systemkomplexität und Subsystemintegration bedingt ist, bestehen: die Kontrolle nichtfunktionaler Eigenschaften wie z.B. Timing, Energieverbrauch oder Verlässlichkeit.

Die Vorteile nichtfunktionale Eigenschaften bereits früh im Entwurfsprozess zu kontrollieren sind klar. Erstens hilft es dem Entwickler die richtigen Entwurfsentscheidungen zu treffen, bevor er in die Implementierungsphase übergeht, wodurch das Entwurfsrisiko sowie Kosten gesenkt werden. Zweitens unterstützt es den Entwickler, Systeme mit Performancereserven für Erweiterungen, Updates und Bug-fixes auszulegen. Dies zu erreichen ist allerdings nicht trivial, da Systemperformance sehr sensitiv bzgl. Spezifikationsänderungen ist. Bereits kleine Änderungen an Systemeigenschaften wie z.B. Ausführungszeiten, Kommunikationsvolumina, Datenraten, etc., können schwerwiegenden Einfluss auf die Systemperformance haben. Aus diesem Grund ist es erstrebenswert, dass das System eine gewisse Art von “Robustheit” gegenüber solchen Änderungen aufweist.

Ein vielversprechender Ausgangspunkt, um diesen Herausforderungen zu begegnen, sind formale Ansätze zur Performanceverifikation, welche basierend auf abstrakten Modellen der Applikation und der Ausführungsplattform in der Lage sind, die benötigten Performancedaten eines eingebetteten Systems akkurat zu bestimmen. Bis zum heutigen Tag wurden diese Methoden jedoch hauptsächlich zur Charakterisierung und Lösung isolierter Performanceproblematiken, wie z.B. Timinganalyse einzelner

ECUs, Flaschenhalserkennung, Buskonfiguration, etc. , eingesetzt. Ein durchgängiger und systematischer Ansatz zur Kontrolle und Optimierung nichtfunktionale Systemeigenschaften fehlt immer noch.

Diese Dissertation führt mehrere Techniken ein, die es erlauben, nichtfunktionale Systemeigenschaften im Entwurfsprozess und während der gesamten Lebenszeit eines Systems zu kontrollieren und zu optimieren.

- Der erste Beitrag ist ein flexibles Explorationsframework, welches die iterative Natur von Performanceverifikation in modernen nebenläufigen Entwicklungsprozessen berücksichtigt. Das vorgeschlagene Framework erlaubt es, partielle Explorationsläufe durchzuführen und bietet die Möglichkeit den Suchraum dynamisch zu erweitern oder einzuschränken, ohne bereits erhaltene Ergebnisse zu verlieren. Es kann daher iterativ eingesetzt werden, um sukzessiv ein gegebenes System unter Berücksichtigung von Datenverfügbarkeit und Verantwortlichkeiten entlang der Zulieferkette zu explorieren.

- Der zweite Beitrag ist die Definition von ausdrucksvollen Robustheitsmetriken für unterschiedliche Annahmen und Entwurfsszenarien. Die vorgeschlagenen Metriken basieren auf Sensitivitätsanalyse, wobei insbesondere Metriken für Systemeigenschaften mit und ohne Abhängigkeiten bzgl. Systemperformance unterschieden werden.

- Die Robustheitsmetriken für den letzteren Fall sind dabei besonders rechenintensiv, da multi-dimensionale Sensitivitätsanalyse notwendig ist, um die gegenseitigen Abhängigkeiten zwischen den betrachteten Systemeigenschaften zu erfassen. Um dieses Komplexitätsproblem zu umgehen, wird eine skalierbare stochastische Sensitivitätsanalyse entwickelt, welche in der Lage ist, Robustheitscharakteristiken effizient zu approximieren, indem sie Sensitivitätsanalyse als multi-kriterielles Optimierungsproblem umformuliert. Dies stellt den dritten Beitrag dieser Dissertation dar.

- Der vierte Beitrag dieser Dissertation zeigt wie die stochastische Sensitivitätsanalyse eingesetzt werden kann, um die vorgeschlagenen Robustheitsmetriken effizient mit oberen und unteren Robustheitsschranken zu approximieren. Des Weiteren, werden Methoden entwickelt, welche diese Schranken einsetzen, um die Exploration der Robustheitsmetriken signifikant zu beschleunigen.

- Es ist bekannt, dass moderne Ansätze zur Performanceanalyse bereits für kleine Systeme beachtliche Unterschiede bzgl. Analysegenauigkeit aufweisen können. Aus diesem Grund hängt die Aussagekraft der in dieser Dissertation präsentierten Methoden stark von der Wahl der geeigneten Abstraktion, d.h. Analysemethode, für das jeweilige betrachtete System ab. Der letzte Beitrag dieser Dissertation beschäftigt

sich daher mit der Problematik, akkurate und aussagekräftige Performancevorhersagen für beliebige eingebettete Systeme zu liefern. Anstatt nach einem einheitlichen, komplexeren Analysemodell zu suchen, wird vorgeschlagen die sogenannte *kompositionelle Performanceanalysemethodik* auszunutzen, um modulare Performanceanalysetechniken zu koppeln. Auf diese Weise können die Vorteile der einzelnen Berechnungsmodelle leicht zu einer domänenübergreifenden Analyse kombiniert werden, welche es erlaubt, jede Systemkomponente mit der jeweils besten Methode zu modellieren und zu analysieren.

# Danksagung

An dieser Stelle möchte ich mich herzlich bei meinem Mentor Prof. Dr. Rolf Ernst bedanken. Er leitete mich mit zahlreichen aufschlussreichen Diskussionen, in die er seine umfangreichen Kenntnisse, seine große Erfahrung sowie sein sicheres Gespür für geeignete und praxisrelevante Forschungsrichtungen einbrachte, sicher durch meine Promotion.

Weiterer Dank gebührt Herrn Prof. Dr. Lothar Thiele, mit dessen Team an der ETH Zürich sich während meiner Promotion interessante Forschungskooperationen ergeben haben, die zu dieser Arbeit beigetragen haben, und der sich bereit erklärt hat den Part des Korreferenten zu übernehmen. Vielen Dank auch an Herrn Prof. Dr. Lars Wolf für die Leitung meiner Promotionskommission.

Insgesamt habe ich die Arbeit im SymTA/S Projekt am IDA sehr genossen, was daran lag, dass ich mit vielen sehr netten und talentierten Leuten zusammenarbeiten durfte. Das IDA stellte damit, nicht nur für die fachliche Arbeit, sondern auch für zahlreiche gemeinsame Freizeitaktivitäten, ein perfektes Umfeld dar. Insbesondere ist da natürlich mein Bürokollege und DFG Projektmitstreiter Razvan Racu zu nennen. Nicht minder zu erwähnen sind jedoch auch Rafik Henia, Simon Schliecker, Sean Whitty, Steffen Stein, Kai Richter, Marek Jersak, Matthias Ivers, Henning Sahlbach, Jörn Braam, Judita Kruse, Sven Heithecker, Amilcar Lucas, Jonas Diemer, Jonas Rox und Maurice Sebastian. Vielen Dank für die schönen Jahre am IDA.

Großer Dank geht auch an meine Frau Cécile. Sie hat sich mit viel Liebe und Hingabe um unsere kleine Familie gekümmert und mir damit den Freiraum gegeben, der für die Verfassung dieser Arbeit nötig war. Diese Arbeit ist daher zu einem großen Teil auch ihr Verdienst.

Besonderer Dank gilt meinen Eltern, Angelika und Wolfgang, die mich während des gesamten Studiums in allen erdenklichen Weisen unterstützt haben, und die immer für mich da waren als ich sie brauchte. Bessere Eltern kann man sich nicht wünschen.

Zu guter Letzt möchte ich noch meinen Schwiegereltern Jacques und Jacqueline Barrère sowie Jeanne Peyresaubes danken, die mir in Südfrankreich immer einen ruhiges und angenehmes Arbeitsumfeld mit entsprechend guter Verpflegung zur Verfügung gestellt haben, in dem auch ein großer Teil dieser Arbeit verfasst wurde.

# Contents

Chapter 1

# INTRODUCTION

An embedded system is a micro computer system that is embedded into another technical device that itself does not appear as a computer. Embedded systems can be found everywhere. Examples are telecommunication devices, consumer electronics, automotive systems, building technology, etc. Embedded systems have a very high influence on the system industry. Nowadays, modern products cannot be realized in a competitive manner without embedded micro computer systems.

Embedded systems have to fulfill a large variety of requirements to be fully functional and accepted. One important requirement that is imposed on embedded systems is dependability. Dependability is important since embedded systems are often safety-critical. Examples are aircrafts, cars, and trains. The notion of dependability covers several important aspects reaching from reliability and availability to safety and security.

Another important requirement is efficiency. There are many different metrics for efficiency that are applied to embedded systems. One important metric, that is often also related to dependability, is run-time efficiency. For instance, the environment may impose certain timing constraints on the system, meaning that the system must not only produce correct results, but in addition must deliver these timely. In this case we speak about an *embedded real-time system*. In this thesis we focus on complex embedded systems that are subject to such real-time constraints. However, there are a large variety of additional efficiency metrics that are important, including energy consumption, code-size, weight, cost, etc.

Today, many embedded system applications are implemented using distributed architectures, consisting of several hardware nodes intercon-

nected in a network. Thereby, each hardware node consists of a processor, memory, interfaces to I/O and to the network. The networks are arbitrated by specialized communication protocols that depend on the application area. For example, in the automotive electronics area communication protocols such as CAN [15, 1], Flex Ray [33], and TTP [120, 60] are common.

## 1.1 Design flow today

In this section we give a short overview of modern design flows for embedded systems. This overview is not meant to be exhaustive. It rather introduces the basic concepts and imposed requirements, which are then discussed using two popular example design flows: the Y-model known from hardware-software co-design, and the V-model utilized in the automotive industry. Based on these two design flows we will later motivate the techniques and contributions presented in this thesis.

### 1.1.1 Basic concepts

A coarse-grain overview of the embedded design flow is shown in Figure 1.1 (compare e.g. [26]).

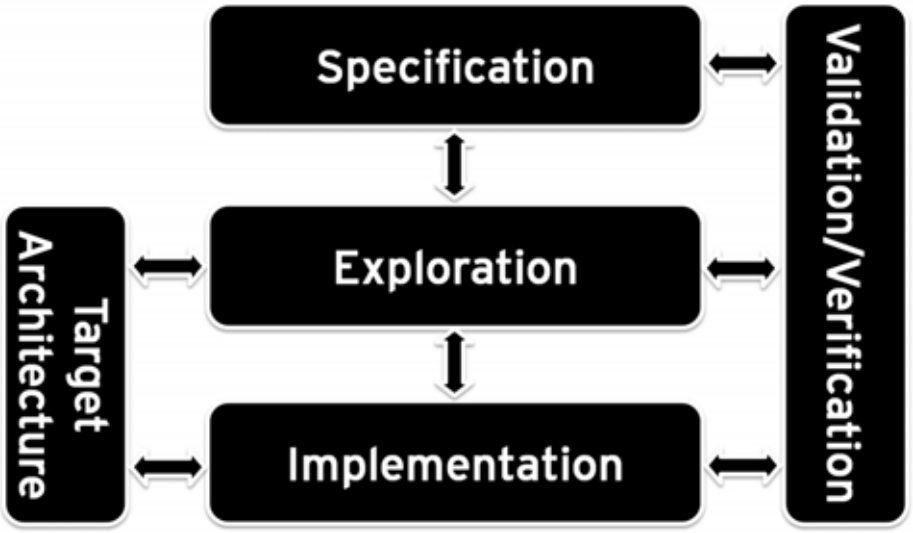



Fig. 1.1: Embedded Systems Design Flow - Coarse-grain overview

First of all the functionality that shall be realized by an embedded system must be specified. This is generally done using different specification languages with specialized models of computation. Examples are state charts [44], the ESTEREL programming language [9], data flow process networks [66], etc. Thereby, the choice of the utilized specification language very much depends on the type of application that shall be realized. State charts, for instance, are well suited for design-

ing reactive applications (e.g. safety in the car: ABS, airbag, etc.), whereas data flow process networks best fit transformative applications (e.g. video processing, digital signal processing, etc.). However, usually not all needed components are designed from scratch. Some components may be re-used from previous designs (legacy code), or are purchased as intellectual property (IP) from external suppliers.

Often the developed specifications represent executable models of the desired system functionality. These so-called *executable specifications* enable the designer to perform early system optimization and design space exploration. For instance, data flow process networks are well suited to optimize buffer requirements and throughput of filtering applications. Generally, design space explorations based on such executable specifications help the designer to choose between functional alternatives, perform hardware-software partitioning, take scheduling decisions, etc.

The specification phase is usually performed without explicit consideration of the target architecture. This is attractive, since it allows the designer to focus on functional correctness ignoring the verification of the concrete implementation. However, once the executable specifications satisfy functional requirements, the focus shifts to target architecture design and implementation. Often parts of the architecture are fixed. Reasons include the need to utilize standard components (processor, memory, bus, RTOS, etc.), and maximum cost and size constraints per unit. During the implementation phase the main challenge is to ensure functional correctness, while successfully integrating all components onto the target architecture under the constraints imposed by the limited service capacity of the available resources.

At the current state-of-the-art, none of the above mentioned design steps can be guaranteed to be correct. Therefore, usually a test-bench is developed in parallel to validate the correctness of the intermediate or final design representations and implementations. Typically, various properties need to be validated to ensure the correctness of the developed system, including performance, dependability, energy consumption, etc.

In case the designers encounter difficulties during implementation, i.e. for instance non-compliance of the resulting system with required performance properties, they need to get back to the exploration or specification phase to find better alternative implementations. In the worst-case parts of the functionalities need to be re-designed, or the target architecture must be modified.

### 1.1.2 The Y-model

As a popular representative of an iterative design flow with successive refinements we now shortly discuss the Y-model known from hardware-software co-design [30]. A simplified version of the Y-model is shown in Figure 1.2.

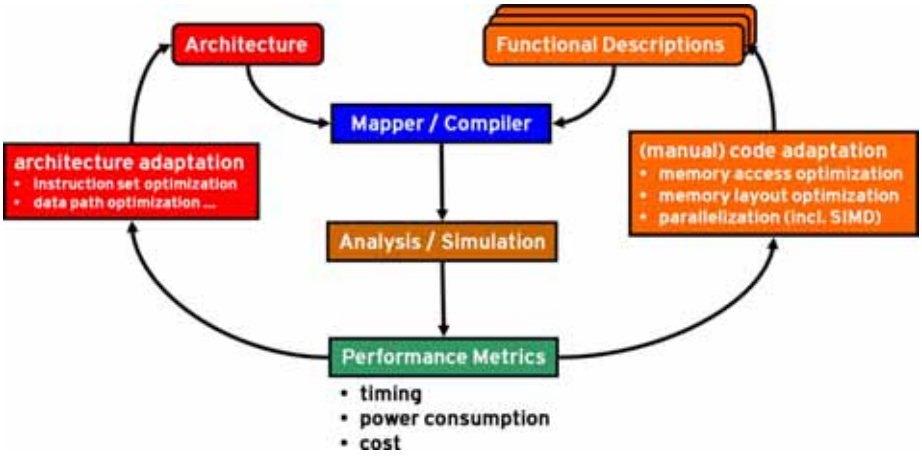



Fig. 1.2: Y-model known from HW/SW co-design

As in the generic embedded system design flow discussed above, the first step in the Y-model design flow consists in specifying platform-independent models for the intended functionality. Based on these models, object code is compiled and mapped on the target architecture. The resulting intermediate implementation is then tested and evaluated with respect to timing, power consumption, cost, etc., using simulation and analysis. Based on these metrics the designer decides about architecture and/or code adaptations. This process is iteratively repeated until a satisfactory design is found. Obviously, to evaluate a large number of different architectures, and thus to potentially obtain a better final implementation, it is desirable to achieve short turn-around times for one iteration.

The risk that is linked to the design flow according to the Y-model is relatively small, since the designer can react in each iteration to performance problems and solve them.

### 1.1.3 The V-model

The Y-model defines a very efficient design flow for hardware-software co-design. However, to be fully applicable an important prerequisite is that one design team controls most of the design parameters. Therefore, it only partly fits design tasks with shared responsibilities requiring sub-system integration. This is, for instance, an important issue in the automotive industry. Different sub-systems are independently developed and

delivered by multiple external suppliers, and the OEM[1] has to integrate these into the car under a huge amount of constraints, including performance, safety, reliability, and consumer demands. In order to overcome this huge integration problem the so-called V-model [75] is used in the automotive industry. Figure 1.3 shows a basic version.

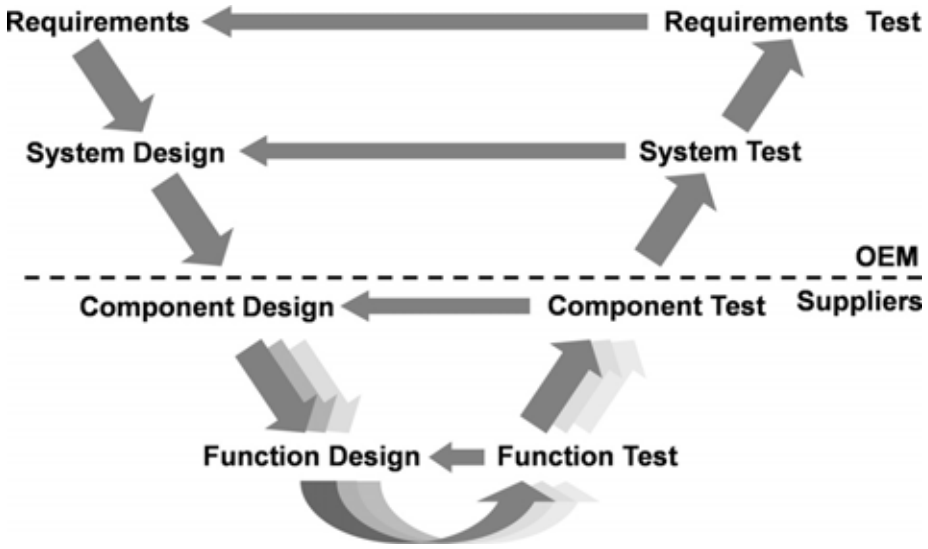



Fig. 1.3: V-model utilized in the automotive industry [75]

The V-model is based on the traditional top-down system engineering approach. First, the requirements imposed on the overall system are specified. Based on these requirements the OEM performs system design. This consists in the definition of the overall structure of the system functionalities and their interactions. Afterwards, the system is partitioned into several components, which are independently designed and tested by external suppliers according to given specifications. Once component design is finished the OEM's task is to integrate the delivered components into the final system, which mainly consists of network integration and a large amount of testing.

However, besides this idealized design flow an automotive system is usually not developed from scratch. In most cases an existing board net is taken as baseline for development. In other words, by reusing existing components time, work, and money can be saved. The V-model is, therefore, often supplemented by bottom-up methods. This does not compromise the V-model, since design has still to go through all the stages.

The basic V-model shown in Figure 1.3 also contains iterative refinements. However, these iteration are performed relatively late on proto-

[1] Original Equipment Manufacturer

types or real implementations. Consequently, iterative refinements are far more time intensive compared to the Y-model, and thus very expensive. Complex OEM-supplier dependencies additionally complicate design iterations.

In order to circumvent this problem the V-model was extended by the concept of *virtual design* (compare e.g. [34] and [104]). Figure 1.4 shows the extended V-model.

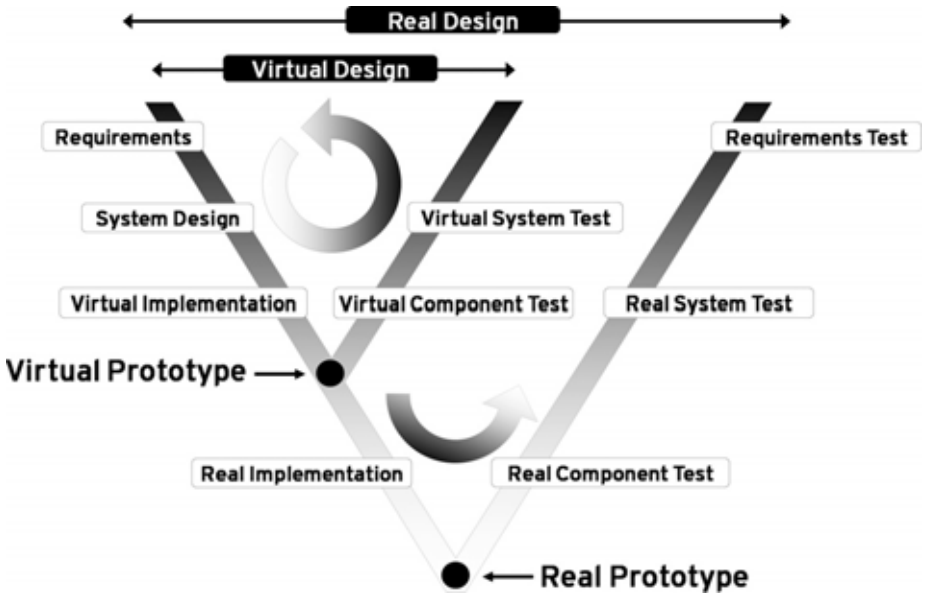



Fig. 1.4: *V*-model extended with the concept of virtual design [34, 104]

The extended V-model differs from the basic V-model in that it contains a second (smaller) V that is used to iteratively refine a virtual system model. Design iterations based on such abstract virtual models are usually far less time consuming and allow, therefore, to efficiently explore the design space. This is of great help for the system architect to choose an optimal system architecture (i.e. topology, number of nodes, number of buses, etc.) as well as an efficient function mapping.

## 1.2 Increasing design efficiency through reuse and modularity

In the development process of complex distributed embedded systems, reuse is recognized as key factor to meet growing productivity demands and cost pressure. In the ideal case whole product families and variants are based on the same set of reusable components allowing the designers to concentrate on basic differences between the products.

One important trend to achieve a high level of reusability is the so-called *platform-based design*. Nowadays, embedded system architectures are usually not designed from scratch. Instead so-called platforms are

used. Platforms are programmable MpSoCs (multi processor system-on-chip) consisting of (multiple) cores, co-processors, specialized hardware components, buses, bridges, interfaces, etc. Platforms are often tailored for specific application domains. Examples are the Nexperia platform for multimedia and mobile digital audio applications from NXP [78], or the Tricore TC1796 platform from Infineon [110] used in the automotive domain (Figure 1.5).

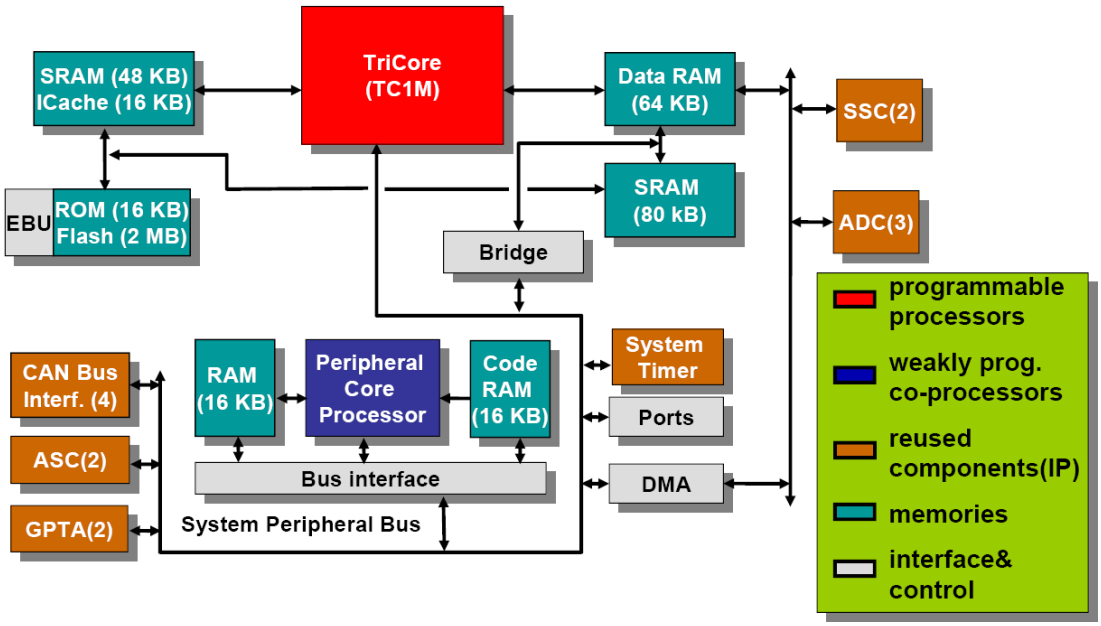



Fig. 1.5: Block diagram of the Infineon Tricore TC1796 micro-controller for automotive applications [110]

The platform-based design style is a so-called meet-in-the-middle approach. It combines the power of top-down methods with the efficiency of bottom-up styles [101]. Platform-based design can drastically reduce time to market while decreasing development and production costs [19]. ST Microelectronics estimates that each platform can lead to four or five products per year and, frequently, ten or more products over the lifetime of the platform [19].

In parallel there exist also efforts to standardize important system functionalities to ensure modularity, maintainability, reusability, scalability, and transferability on the software level. The Artist roadmap [14] identifies software as one key factor to successfully integrate components and sub-systems into complete systems. Classical middleware approaches such as CORBA [21] and COM/DCOM [20] are good examples for successful approaches to software modularization. Another prominent example known from the automotive industry is the AUTOSAR initiative [5]. The core idea of AUTOSAR is the definition of a run-time environment (RTE) executing so-called software components that com-

municate over a Virtual Function Bus (VFB). According to this concept, two functionalities can exchange data without knowing the exact communication path by using abstract communication ports of the RTE. Consequently, software can be developed independently of the real ECU topology in the car. The communication paths are defined relatively late in the design process. Obviously, such middleware concepts facilitate software portability and reusability.

## 1.3 Motivation

The platform-based design style and the standardization efforts on the software level help to increase design efficiency at the functional level. Example scenarios include the integration of several functionalities from different suppliers on the same node, or the distribution of (safety-critical) functionalities over several nodes. However, these concepts do not solve another key embedded systems challenge: the control of performance and other non-functional constraints, such as timing, power consumption, or dependability during the design process and over the service life of the product.

For instance, several components are often dependent on each other, and their integration can lead to complex and hard to predict performance degradation effects, which increase the design risk since they are often discovered late during the integration phase. In automotive systems, for example, the active front steering (AFS) interacts with other functionalities like the active roll stabilization (ARS). Also, not every functionality has its own sensors, data like individual wheel speeds are broadcasted over the bus and shared by many functionalities.

It is desirable to control the impact of sub-system integration on non-functional system properties, both during the design flow and during the lifetime of the product. The benefits are clear. The control of non-functional properties helps the system architect, on the one hand, to take the right design decisions before proceeding to implementation, and thus to decrease the design risk. On the other hand, it also assists the designer in conceiving systems with performance head-room for reuse, future extensions, updates, and bug-fixes. However, in modern design flows conformance to non-functional requirements is difficult to ensure, which is mainly due to the increasing size and complexity of modern systems, and the concurrent design [65] between OEM and suppliers involving dozens of parallel activities that need to be coordinated. As a consequence performance verification is still a major issue during design.

One possible approach to simplify sub-system integration is the so-called *conservative design*. Its principle consists in eliminating all coupling effects by strictly separating functionally independent sub-systems

spatially and timely. The separation is achieved by assigning fixed memory spaces as well as static shares of communication and computational resources to each sub-system. Obviously, this strategy eliminates all complex timing effects and solves the integration problem. However, the resulting systems are not very efficient in terms of resource utilization and, thus, system efficiency and cost, since the statically assigned resources are not released for other functionalities in case of disuse. While this might be acceptable for highly safety critical systems, like for instance in avionics, such over-design is not an alternative in most other industrial sectors, including consumer electronics or automotive systems.

A promising starting point to overcome integration challenges while ensuring system efficiency are state-of-the-art performance analysis methodologies that have been proposed in the last decade [79, 77, 115, 84, 43, 113, 50, 46]. The different approaches operate at different levels of abstraction and allow a step-wise refinement of the utilized application and execution platform models. Figure 1.6 shows how performance verification can be applied along the V-model.

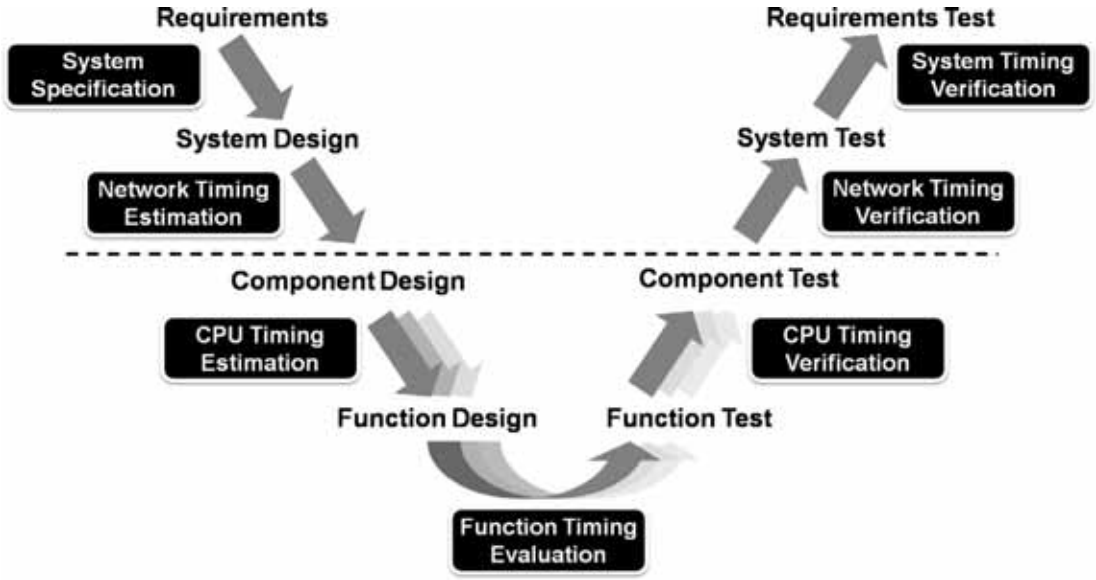



Fig. 1.6: Performance verification along the V-model

During the specification phase system performance is characterized based on data estimates. Even though this information might be coarse-grain and partly incomplete at the beginning, it can be utilized to derive first performance approximations helping the system architect to take architectural or mapping decisions (e.g. number of ECUs, bus bandwidth, etc.). Later, during component design and integration, these estimations can be refined step-by-step to obtain more accurate performance data.

Since the mentioned methods are based on rather abstract performance models and are able to analyze even large systems in a short

time, they are perfectly suited for design space exploration. Design space exploration on top of these methods represents a valuable tool for the system architect to take good design decisions and systematically control system performance throughout the whole design process.

However, even though formal performance analysis methods are capable of deriving accurate performance data and are perfectly suited for design space exploration, their integration into real-world design flows is rather difficult. As a consequence, formal techniques have been mainly (successfully) applied to characterize and solve isolated performance issues like, for instance, timing analysis of single ECUs [97], bus bottleneck detection [98], or bus configuration [16]. A systematic and continuous performance verification flow spanning the whole design process allowing to control and optimize system performance is not trivial to accomplish. There are several reasons for this.

The first reason is that system performance is not composable in the general case. In other words, the system integrator (OEM) cannot automatically conclude that the integrated system satisfies its performance constraints from the fact that all supplied components are compliant to their specifications. The reason are complex performance dependencies that can often only be discovered during integration. One possibility to overcome this problem is to continuously verify and control performance during the design process across all involved design teams. However, in real-world design flows with OEM-supplier dependencies the required information exchange is problematic due to IP (intellectual property) protection issues. In fact, each of the involved design teams controls different parts of the system and is reluctant to share realization details.

There exist first approaches to solve this problem. The authors of [98], for instance, propose that each involved design team individually performs component-level analysis and communicates relevant results to functionally dependent system parts along the supply chain. By iteratively repeating such local analysis steps, OEM-supplier spanning timing analysis can be realized. Note that the performance data that needs to be exchanged mainly describes the dynamic communication behavior of the involved components (e.g. message jitters and frame offsets), and represents uncritical information with respect to IP protection. However, to fully exploit the benefits of formal methods in the context of concurrent design flows, such practical solution approaches for performance verification must be complemented with a flexible design space exploration framework of similar structure that supports iterative partial exploration steps at component level. The introduction of such an exploration framework is one of the aims of this thesis.

Nevertheless, even with systematic and continuous performance verification and exploration flows at hand there is a second problem that complicates performance control during design and in the field: the sensitivity of system performance to modifications of specified system properties, such as execution times, communication volumes, data rates, CPU clock rates, etc. It is, for instance, known that minor local execution time or data rate changes can have drastic impact on performance metrics, such as task response times, buffer sizes, end-to-end latencies, at system level [42, 121]. During design property modifications might occur due to changes or refinements of the specification, software components, or target architecture. During the system's service life similar effects might occur as a result of software updates, bug-fixes, changes in the environment, or the integration of new components. In order to prevent performance degradation effects it is desirable that the system exhibits robustness against such property modifications. Obviously, this kind of robustness can considerably reduce the risk during design, and facilitate maintainability as well as extensibility in the field.

The impact of property modifications on system performance can be characterized with sensitivity analysis [122, 88, 90]. Sensitivity analysis is, therefore, a good starting point for methods increasing system robustness. However, in order to systematically optimize and control system robustness, it must be considered as explicit design goal. The introduction of efficient robustness optimization methods is another main objective of this thesis.

## 1.4 Contributions

In this section we give a brief overview of the major contributions of this thesis. Figure 1.7 shows the proposed modules for design space exploration and robustness optimization and their interdependencies. The proposed modules are highlighted in black, whereas modules that we build upon are highlighted in white.

**Design Space Exploration Framework.** The first contribution of this thesis is a flexible exploration framework that takes into account the iterative nature of performance verification during modern parallel design flows. The proposed framework allows performing partial explorations and provides the possibility to dynamically extend and restrict the search space without loosing previously obtained results. It can, therefore, be iteratively applied to successively explore given systems taking into account data availability and responsibilities along the supply chain. This flexibility ensures that design space exploration can be systematically performed along the whole design flow. Thereby, the in-

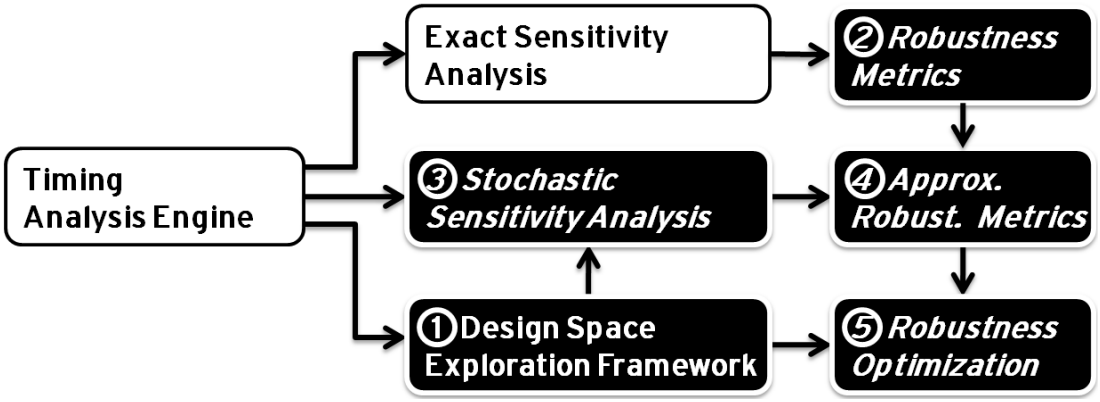



Fig. 1.7: Overview: Proposed Design Space Exploration and Robustness Optimization Framework

volved design team can refocus exploration on the available search space at any design stage. For instance, during specification nearly all system parameters are free and can still be modified, including the application mapping. Later, parts of the mapping might be fixed or priorities of messages and frames that are exchanged over the bus are partially assigned. Note that incremental design space exploration is also very efficient in terms of performance. In case of search space modifications, exploration does not need to start from the beginning. Good results from previous runs can be used as (heuristic) starting point to achieve quicker convergence towards the solution space.

**Robustness Metrics.** Apart from its application to performance and timing optimization in modern parallel design flows, the proposed design space exploration framework is also used to realize efficient robustness optimization methods, the second contribution of this thesis.

Expressive robustness metrics for different assumptions and design scenarios are defined. The proposed metrics are based on sensitivity analysis. Thereby, robustness metrics for system properties with and without performance dependencies are distinguished. In the case of independent properties, the value of one property does not have any influence on the admissible values of other properties. In the case of dependent properties, the modification of one property value leads to restrictions for dependent properties, i.e. their flexibility with respect to modifications decreases.

**Stochastic Sensitivity Analysis.** Especially the robustness metrics for the dependent case are computationally expensive, since multi-dimensional sensitivity analysis is necessary to capture interdependencies between involved properties. In order to circumvent this complexity

problem, a scalable stochastic sensitivity analysis approach is developed that is capable of efficiently approximating robustness characteristics by formulating sensitivity analysis as multi-criterion optimization problem. This represents the third contribution of this thesis.

**Approximation of the Robustness Metrics and Robustness Optimization.** As the fourth contribution of this thesis, it is shown how the stochastic sensitivity analysis can be used to efficiently approximate the proposed robustness metrics with upper and lower robustness bounds. Furthermore, methods are developed that utilize the derived bounds to significantly speed up robustness optimization by quickly identifying promising system parameter configurations, whose in-depth robustness evaluation can be performed subsequently to the optimization process.

**Combined Performance Analysis.** The design space exploration framework and robustness optimization techniques presented in this thesis are orthogonal with respect to the underlying performance analysis engine. However, in [83] several state-of-the-art performance analysis approaches were compared, and it was pointed out, that already for small systems the results yielded by the different approaches exhibit remarkable differences in terms of accuracy. As a consequence, it can be stated that the expressiveness of the techniques proposed in this thesis highly depends on the choice of the right abstraction, i.e. analysis engine, for the considered system.

In the last part of this thesis we, therefore, want to contribute in solving the problem of providing accurate and expressive performance predictions for general distributed real-time systems. Rather than searching for a unified more complex analysis model, we propose to exploit the properties of the so-called *compositional performance analysis methodology* to couple modular performance analysis techniques. In so doing, the advantages offered by the individual models of computation can easily be integrated into a cross-domain analysis that allows modeling and analyzing each system component with the best fitting method.

## 1.5 Overview

This thesis is structured as follows:

- **Chapter 2** gives an overview of formal methods for performance and timing verification. We introduce a state-of-the-art performance model that is used by modern performance analysis methodologies. Based on this performance model, we discuss in detail main require-

ments for an efficient iterative design space exploration framework and system robustness optimization techniques, the two main contributions of this thesis.

- **Chapter 3** introduces the basic concepts of the proposed design space exploration framework. This includes the compositional search space encoding scheme, the integration of systematic techniques for component interaction optimization, the exploration loop adaptively controlling the search process, and the user-controlled exploration methodology. Furthermore, we discuss realization details for the application of the framework to timing and performance optimization. Note that the framework is additionally used to cover stochastic sensitivity analysis and robustness optimization (Chapter 4). Finally, we present a heuristic extension for automated search space modification, and show its application to the optimization of priority assignments on priority scheduled system components.

- **Chapter 4** discusses robustness optimization techniques. We introduce robustness metrics for different application scenarios and assumptions, and show how these can be efficiently integrated into design space exploration. In particular, we discuss how the more complex robustness metrics can be approximated using a scalable stochastic sensitivity analysis.

- **Chapter 5** describes the compositional performance analysis methodology. We shortly present the performance analysis tools SymTA/S and MPA that are based on this methodology, and discuss how both approaches can be coupled by exploiting their common formal basis. By means of a complicated example we demonstrate the seamless analysis interaction and accuracy benefits over using the individual tools alone.

- **Chapter 6** concludes this thesis by summarizing the contributions.

Chapter 2

# PERFORMANCE MODEL FOR EXPLORATION AND ROBUSTNESS OPTIMIZATION

In this chapter we discuss the application of formal methods during the design of complex embedded systems. In particular, we introduce a state-of-the-art performance model that is used by modern formal performance analysis methodologies. Examples for performance metrics that can be determined by such methods include CPU and network loads, task and message latencies, and memory consumption.

Based on the introduced model we discuss the main requirements for design space exploration and robustness optimization, the two main topics that we address in this thesis. Note that the technical realization details are discussed in the subsequent chapters.

## 2.1 Formal methods for performance verification

State of practice in performance verification is cycle-true simulation of functions and target architecture, as well as test using prototype hardware. However, with increasing system complexity these approaches are more and more challenged in delivering reliable performance information, since the identification and coverage of performance corner-cases in the test-bench is unrealistic for sufficiently large distributed systems. One of the main reason is that system-level corner cases cannot be easily derived or constructed from component corner-cases. This is mainly due to transient run-time effects with hard to predict anomalous behavior of the target architecture. Another drawback of these standard methods is that they cannot be used for early design space exploration, since they are very time consuming and need executable code, which is only available at late design stages. Consequently, alternative methods are needed if we want to systematically control system performance along the design flow.

A promising alternative is formal performance verification. The concept of virtual design discussed in Section 1.1.3 is the first step to introduce and utilize formal methods during embedded systems design. Short iterations on virtual system models allow to perform early design space exploration and can, therefore, help to prevent or solve performance problems. However, methods that have found their way into industrial practice are still rather simplistic and not systematic enough to really control and optimize system performance across the entire design flow and in the field.

In the automotive industry, for instance, one popular analytical models that is used for bus dimensioning is the so-called *bus load model*. The model consists in calculating the average network load, also called utilization, by multiplying the frequency of each frame with its length (including protocol overhead), building the sum over all frames, and dividing the result by the network throughput. The OEM can extract the necessary information from the so-called K-matrix (communication matrix) that summarizes all relevant information and communication dependencies in the board network of a car. In order to ensure predictability the OEM then defines a maximum bus utilization limit that must not be exceeded. Clearly, low bus utilization means (on average) less traffic on the bus and, consequently, also (on average) shorter transmission times. The other way around, increasing the average load results in better resource utilization and is, therefore, interesting in terms of cost efficiency. For this reason, the maximum bus utilization limit varies among the OEMs between 30% and more than 60%[1]. However, the bus load model should be used carefully, since even at low utilizations it cannot be theoretically proven that all deadlines are met or that no buffer overflows.

A more systematic approach to communication verification and bus dimensioning that has found its way into industrial practice is based on formal scheduling analysis methods from the real-time community. Tindell et al., for instance, showed how classical research into fixed priority pre-emptive scheduling could be adapted for scheduling messages on the CAN bus [119, 118]. Even though it is now known that the original CAN analysis from Tindell is flawed and may yield optimistic worst-case response times for some messages [22], the proposed techniques were successfully used in the configuration and analysis of the CAN buses for the Volvo S80 [16], and led to the development of a commercial tool by Volcano Communications Technologies [76].

[1] According to personal communication.

However, with the advent of more sophisticated embedded systems containing complex bridged networks, local analysis techniques that only look at single buses or processors are not sufficient anymore. In order to adequately verify and control system performance, methods that systematically discover system-level corner-cases and anomalous behavior of the target platform are needed. In the last decade several promising analysis methodologies capable of analyzing entire complex embedded systems were proposed. Examples include so-called holistic approaches [115, 43, 84] tailored for specific system architectures and application models, and flexible compositional approaches such as SymTA/S [50, 87] and Real-Time Calculus (RTC) [17, 126].

In the next section the performance model that is used by these system-level analysis methodologies is introduced. This model represents an adequate abstraction to address performance problems and will be used as basis of the methods proposed in this thesis. Note that since all proposed methods are orthogonal with respect to the underlying performance analysis engine, it is assumed that a black-box analysis engine is available delivering the necessary performance data based on the introduced performance model. Nevertheless, in Chapter 5 the coupling of SymTA/S and RTC is discussed. In this context the basic formal concepts of these approaches will be explained.

## 2.2 State-of-the-art system performance model

An embedded system consists of hardware and software components interacting with each other to realize a set of functionalities. For performance verification purposes embedded systems can be modeled with a high level of abstraction. The smallest unit modeling performance characteristics of software components is called *task*. Equivalently, messages that are sent by tasks over some communication infrastructure are modeled by so-called *communication channels*. The hardware platform is modeled by *computational* and *communication resources*. Tasks are mapped on computational resources in order to execute, whereas communication channels are associated to communication resources for message transmission. For the sake of brevity, we refer to computation and communication resources in the following as CPU and bus, respectively.

Usually, multiple tasks or communication channels share the same resource. As a consequence, two or more tasks/channels may request the resource at the same time. In order to arbitrate request conflicts, each resource is associated with a *scheduler* that grants the resource to competing tasks/channels according to some scheduling policy. Other active tasks/channels have to wait.

Tasks are activated and executed due to activating events. Activating events can be generated in a multitude of ways, including timer expiration, external or internal interrupts, and task chaining according to the global application structure.

In task chaining, events can be communicated between functionally dependent tasks in different ways. Classically, it is assumed that each task has one input FIFO. A task reads activating events from this FIFO and writes events into input FIFOs of dependent tasks. Thereby, it is assumed that activating events are removed from the input FIFO at the end of execution. Besides this basic model, some approaches have recently been extended to allow for more complex semantics involving several inputs. Examples are AND- and OR-activations [54], or even arbitrary boolean activations [41].

Another possibility is communication over registers [133]. In this case, events from the sending task are stored in a register. The receiving task (periodically) polls this register and reads the event that is currently stored. Obviously, such kind of communication poses interesting questions concerning data age and system reaction time, since events can get lost or be read several times. For instance, in case that the polling interval is shorter than the time between successive write accesses to the register, the receiving task might read the same event multiple times, and thus might operate on old data (over sampling). Furthermore, the receiving task can miss some events, i.e. if the polling interval is chosen to large (under sampling).

Note that in both cases mentioned above, tasks might read activating events at any time during execution. Events are, therefore, assumed to be available during the whole execution.

System performance is influenced on the one hand by system parameters that can be assigned by the designer, and on the other hand by intrinsic platform- and application specific system properties. In the following we give a precise characterization of these parameters and properties.

### 2.2.1 System parameters

**Def. 2.2.1 (System parameter)** A system parameter represents a configurable characteristic of a system component.

The system parameters are, by their definition, all parameters of hardware and software components that can be configured by the designer. This includes the parameters of the utilized scheduling algorithms, like:

- task priority assignment in case of priority based scheduling,

- time slot and turn length definition as well as time slot sequence in case of time-driven scheduling
- task dispatching order in case of static scheduling

In some cases the timing of messages sent over buses might be configurable to some extend. The CAN protocol [15, 1], for instance, allows the specification of *offsets* for individual frames. In practice, such offsets are used to balance the bus load by defining phase relations between frames sent by different CPUs. As an effect of this load balancing, collisions between two or more concurrently transmitted frames might be prevented leading to potentially shorter transmission times.

Additionally, it might be possible for the designer to manipulate the timing properties of buffering components. For instance, messages can be deliberately delayed in a buffer to reduce transient load peaks [96]. This mechanism is called *traffic shaping*. Thereby, the effects of traffic shaping depends on the maximum time a message might be delayed. In this thesis we consider this *traffic shaping delay* as an additional system parameter.

Obviously, system parameters do not contain any stand-alone information about system performance. However, they strongly influence it and represent, therefore, the search space for performance optimization methods and design space exploration.

### 2.2.2 System performance properties

In order to apply performance analysis on the abstract task and resource model introduced above, certain performance-related system properties need to be specified.

**Def. 2.2.2 (System performance property)** System performance properties represent intrinsic characteristics of components directly or indirectly influencing system performance.

System performance properties are classified, based on their scope, in different categories. In this thesis we consider:

**I. Platform related performance properties characterizing the hardware architecture.**

- The *CPU speed* represents the basic performance characteristic of a processor and is expressed in cycles per second.
- The *bus throughput* represents the basic performance characteristic of a bus and is expressed in bytes per second.

**II. Timing properties of tasks and communication channels.**

1. Timing properties of tasks, determined by the software implementation and by the actual configuration of the hardware architecture.

   - The *worst-case execution time (WCET)* or *worst-case execution demand* is the maximum time a task executes assuming exclusive access to a specific CPU at nominal speed.
   - The *best-case execution time (BCET)* or *best-case execution demand* is the minimum time a task executes assuming exclusive access to a specific CPU at nominal speed.

2. Timing properties of the communication channels, determined by the communication volume and by the performance of the communication link.

   - The *worst-case communication time (WCCT)* or *worst-case communication demand* is the maximum time a communication channel requires to transfer a message over a specific bus assuming exclusive access at nominal throughput.
   - The *best-case communication time (BCCT)* or *best-case communication demand* is the minimum time a communication channel requires to transfer a message over a specific bus assuming exclusive access at nominal throughput.

The above mentioned best-case and worst-case bounds represent the basic concept for modeling timing properties of tasks and channels. Note that for a more fine granular description of the timing behavior, and thus the calculation of more realistic response times, many approaches have extended this basic interval model to cover data dependent task behavior [54, 127]. For instance, the execution demand of data processing tasks is typically influenced by size and type of the processed data packets (e.g. MPEG-2 video streams [55]).

**III. Timing properties of the activation model.** These properties describe the arrival of workload, i.e. activating events, at the task inputs. Instead of considering each activation individually, as simulation does, formal performance verification abstracts from individual activating events to *event streams*. Note that different methods utilize different stream models to describe the timing of activating events. Generally, event streams can be described using the upper and lower event arrival curves $\eta^+$ and $\eta^-$.

**Def. 2.2.3 (Upper Event Arrival Curve)** The upper event arrival curve $\eta^+(\Delta t)$ specifies the maximum number of events that occur in the event stream during any time interval of length $\Delta t$.

**Def. 2.2.4 (Lower Event Arrival Curve)** The lower event arrival curve $\eta^-(\Delta t)$ specifies the minimum number of events that occur in the event stream during any time interval of length $\Delta t$.

Note that arrival curves describe the properties of single event streams. However, to realistically model complex systems and to increase analysis precision, some approaches contain additional concepts to model time correlations between several event streams. One possibility is the usage of so-called *relative offsets* describing earliest event arrival times relative to given timing references (i.e. the periodical arrival of an external event at the system input). Examples for approaches exploiting such information can be found in [47, 48, 95, 81].

One popular and computationally efficient abstraction for representing event streams are so-called *standard event models* [96]. Standard event models capture key properties of event streams using three parameters: activation period $P$, activation jitter $J$, and minimum distance $d$. *Periodic* event models have one parameter $P$ stating that each event exactly arrives periodically every $P$ time units. This simple model can be extended with the notion of jitter, leading to *periodic with jitter* event models that are described by two parameters $P$ and $J$. Events generally occur periodically, but can jitter around their exact position within the jitter interval $J$. If the jitter is larger than the period, then two or more events can occur at the same time, leading to bursts. To describe *bursty* event models, *periodic with jitter* event models can be extended with the $d$ parameter capturing the minimum distance between successive events within bursts.

Figure 2.1 illustrates the upper event arrival curve $\eta^+$ of a bursty event stream. Basically, it is split into two different regions. Small time intervals are dominated by bursty behavior, where the system load is only limited by the minimum event distance $d^-$. Larger observation intervals reveal the periodic nature of the event stream.

Both regions, i.e. the periodic region and the region dominated by the burst, can be separately described with equations. The $\eta^+$ function of the stream, illustrated by the black curve in Figure 2.1, is the minimum of them:

$$\eta^+_{\text{in}}(\Delta t) = \min\left(\left\lceil \frac{\Delta t}{d^-} \right\rceil, \left\lceil \frac{\Delta t + J}{P} \right\rceil\right). \tag{2.1}$$

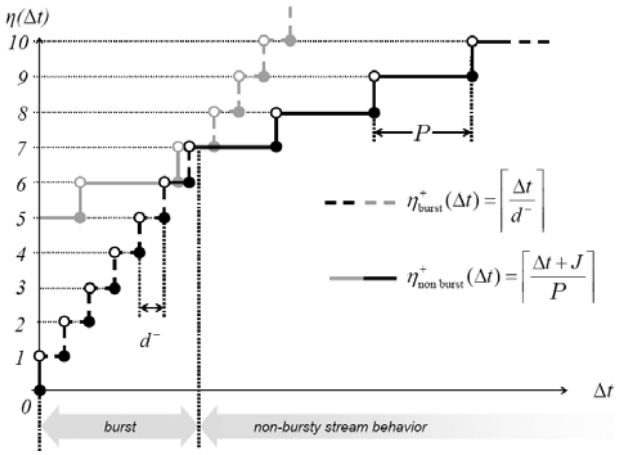



Fig. 2.1: Event arrival curve of a bursty event stream

For more details how $\eta^+$ and $\eta^-$ are defined for standard event models please refer to [96].

### 2.2.3 Performance metrics and constraints

Based on the presented model, formal performance verification methodologies like SymTA/S [50, 87] and Real-Time Calculus (RTC) [17, 126] can derive performance metrics for a given system including:

- best-case and worst-case task response times
- best-case and worst-case message transmission times
- buffer sizes (e.g. activation backlog)
- end-to-end latencies (delay of a stimulus along a task chain)
- power consumption
- jitter of stimuli

In order to function correctly an embedded system has often to fulfill constraints with respect to these performance properties. In this thesis we consider the following constraints:

- End-to-end deadlines (maximum delay of a stimulus along a task chain)
- Maximum jitter constraints (e.g. performance contracts at component outputs)
- Maximum buffer sizes (e.g. global or local buffer budget)
- Maximum power consumption

## 2.3 Example system

In this section we introduce a small example system based on the performance model discussed in the previous section. The setup is shown in Figure 2.2. The system contains four different CPUs connected by a bus. Three applications are mapped on the architecture. A video application (solid chain) gathers data from a camera controlled by the micro controller uC, performs preprocessing on the DSP, and post-processing on the PPC core. The second application (dashed chain) reads data from a sensor, which is first processed on the ARM core and then forwarded to the PPC core, which, in turn, controls an actor. The third application (dotted chain) is a streaming application that runs on the ARM processor core and uses the DSP for data processing.

All three applications have constrained end-to-end latencies which need to be satisfied for the system to function correctly (Table 2.1c). The video application must produce new frames in time, the steering application has to react timely to new sensor input.

The CPUs ($PPC$, $uC$, $DSP$, and $ARM$) are all scheduled according to the static priority preemptive policy, and the interconnecting bus is arbitrated by the CAN protocol. Core communication and core execution times as well as priorities of all tasks and exchanged messages are given in Tables 2.1a and 2.1b, respectively. The periods of incoming data at the system inputs ($Cam$, $Sens$, and $S_{in}$) are specified in Table 2.1d.

The load situation on the four involved CPUs and the interconnecting bus is shown in Table 2.1e. As can be observed, the worst-case system load is rather moderate, except for the video stream processing DSP running at over 90% of its maximum capacity.

## 2.4 Design Space Exploration

During design each implementation decision influences system performance, and needs, therefore, to be constantly evaluated and compared against performance constraints and optimization goals. Ideally, many functionally equivalent implementation alternatives are analyzed during design to identify an optimal solution. This process is called design space exploration. Design space exploration represent a major tool to control and optimize system performance during embedded systems design.

The introduction of an efficient design space exploration framework that fits into real-world design flows is one main goal of this thesis. However, note that the methods presented in this thesis are mainly tailored and applicable to the platform-based design style. For this reason architecture exploration techniques are not considered. Consequently, the main aspect of this thesis, in terms of design space exploration, lies on

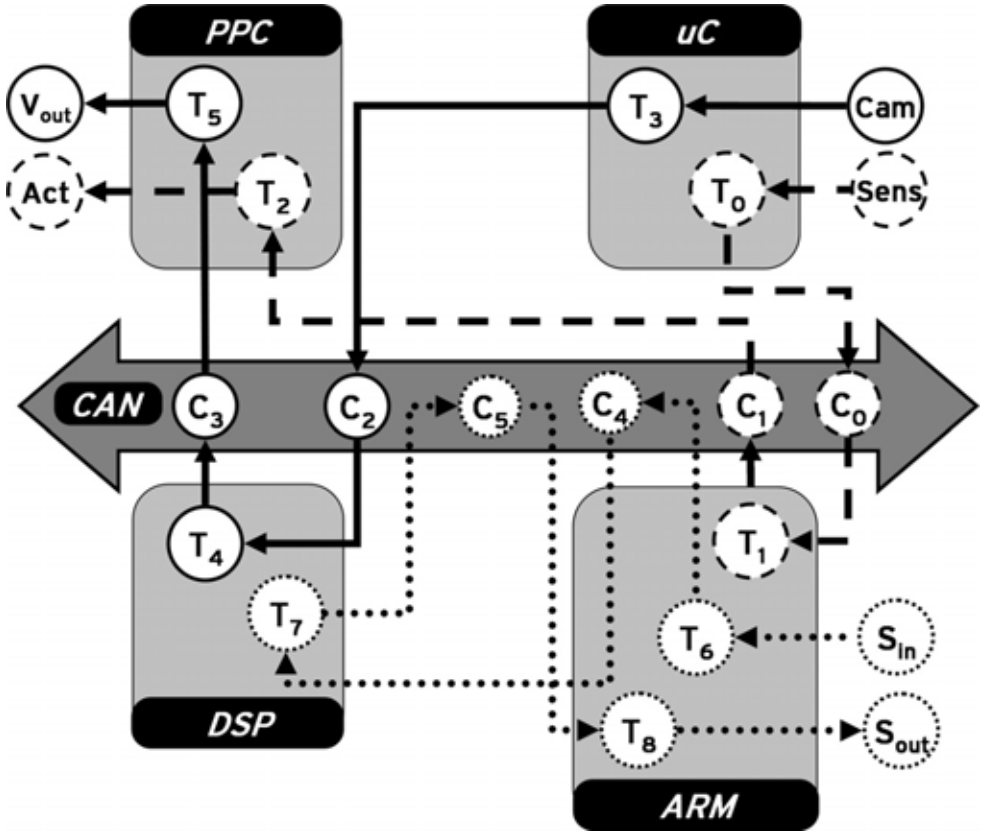



Fig. 2.2: Example system

platform configuration, including the assignment of scheduling parameters, etc. This search space can be efficiently extended by introducing application mapping as additional degree of freedom. However, since the underlying search space does not influence the applicability of the proposed methods, this issue will not be further discussed.

One important aspect that is considered in this thesis is the fact that nowadays distributed embedded systems are often not designed by a sole manufacturer. In the automotive industry, for instance, different subsystems, or even subsystem parts, representing different functionalities of the car, e.g. ABS, ESP, ASR, etc., are designed and delivered by external suppliers. The car manufacturer's (OEM) task is to integrate the different subsystems into the global system, which consists to a major part in network integration.

In order to handle complex OEM-supplier dependencies, automotive systems are developed according to the V-model. Obviously, two of the main challenges for an efficient design space exploration methodology in this context are 1) to handle the complexity and heterogeneity of modern system components, as also has been pointed out by the Artist roadmap [14], and 2) to support the parallel design style involving multiple design teams.

| *Channel* | *CCT* | *Priority* |
|---|---|---|
| C0 | [10.4, 12.4] | 5 |
| C1 | [12, 14.4] | 3 |
| C2 | [20, 26.4] | 2 |
| C3 | [15.2, 22.4] | 6 |
| C4 | [10.4, 14.4] | 1 |
| C5 | [15.2, 26.4] | 4 |

(a) Core communication times (CCT)

| *Task* | *CET* | *Priority* |
|---|---|---|
| T5 | [30.9, 43.1] | 2 |
| T2 | [15.8, 27.4] | 1 |
| T3 | [36.9, 46.3] | 2 |
| T0 | [20.1, 48.5] | 1 |
| T4 | [13, 86] | 2 |
| T7 | [40.3, 44.5] | 1 |
| T1 | [27.1, 154.9] | 2 |
| T6 | [14.2, 63.6] | 1 |
| T8 | [11.5, 291.2] | 3 |

(b) Core execution times (CET)

| *Path* | *Deadline* |
|---|---|
| $Sens \rightarrow Act$ | 850 |
| $Cam \rightarrow V_{out}$ | 1000 |
| $S_{in} \rightarrow S_{out}$ | 1000 |

(c) End-to-end constraints

| *Input* | *Period* $\mathcal{P}$ |
|---|---|
| $Sens$ | 450 |
| $Cam$ | 100 |
| $S_{in}$ | 900 |

(d) Input event models

| *Resource* | *Load* |
|---|---|
| $PPC$ | 49.19% |
| $uC$ | 57.08% |
| $DSP$ | 90.94% |
| $ARM$ | 73.84% |
| $BUS$ | 59.29% |

(e) Average worst-case resource loads

Table 2.1: System parameters, performance related system properties, end-to-end timing constraints, and worst-case resource loads

The remainder of this section is structured as follows. First, we discuss the applicability of previous design space exploration approaches to modern parallel design flows with shared responsibilities (Section 2.4.1). Based on this discussion, we then motivate requirements for the design space exploration framework that we introduce in this thesis (Section 2.4.2). Finally, we explain several practical application scenarios (Section 2.4.3). Note that the concrete technical realization of the design space exploration framework is discussed in Chapter 3.

### 2.4.1 Exploration methods

Many previous approaches to design space exploration perform a fully-automatic closed search over a very huge design space including many parameters, e.g. [25]. However, as already mentioned, in industrial practice it is rarely the case that a single design team is in control of all system parameters. A global closed exploration is, therefore, only partly applicable, and thus of limited use. This fact has only found little attention in many previous exploration approaches. Instead, many contributions

mainly attempt to solve the complexity problem that comes along with the huge considered design space.

Obviously, for real-world sized embedded systems exhaustive search is infeasible. Therefore, many approaches try to reduce the search space to limit the computational effort. Commonly applied techniques, for instance, incorporate (problem specific) local search techniques for one or several parameters [107, 132], or exploit (assumed) parameter independencies to cluster the search space into several parts that are explored separately [37, 82]. However, the necessary parameter dependency model highly depends on the application structure and the target platform, and might be difficult to determine for large embedded systems with complex timing dependencies. Also, system parameters might only offer few independencies prohibiting an efficient clustering of the search space. In such a case, exhaustive exploration becomes infeasible and needs to be replaced by more sophisticated methods.

Another problem that reduces the applicability of these exploration techniques in the context of parallel design flows, is that independent parameter sets possibly span several domains of responsibility. From a structural point of view hierarchical exploration schemes seem, therefore, more appropriate. The main idea of these approaches is to hierarchically combine local optima obtained through sub-component exploration to (hopefully) good global solutions [2, 108]. Even though such hierarchical exploration schemes better fit the structure of complex design flows with OEM-supplier dependencies, they have the major drawback of not sufficiently considering component and performance dependencies, which especially occur in complex distributed embedded systems. In such systems, optimal sub-component configurations rarely compose to optimal system configurations.

Another alternative that has been proposed is the manual restriction of the search to promising design space regions [45]. For such approaches to be applicable, very good knowledge of the system and its performance dependencies is necessary. Even though, such knowledge might be available for specific aspects in a design, it is arguable that these techniques alone are sufficient to efficiently explore complex distributed embedded systems. Nevertheless, the possibility to manually restrict or expand the search space to focus the exploration effort is definitely an important aspect that shall be considered by an efficient exploration framework. However, to fully benefit from manual search space restrictions and expansions they must be applicable in a dynamic manner. Only then, is it possible to perform several successive exploration runs with search space modification in every step. This concept enables the designer to iden-


Fig. 2.3: Black-box performance analysis

tify step-by-step interesting design sub-spaces, worthy to be searched in-depth or even completely.

As counterpart to full automation, also manual exploration approaches have been investigated [36]. During manual exploration the main challenge consists in efficiently choosing the right design points based on performance estimations. Obviously, in the context of concurrent design flows such an approach is difficult to apply. However, one possibility is to exploit knowledge from previous designs (if available) as starting point for design space exploration. Then, however, the question remains how to manually proceed to the next design point. Therefore, manual exploration alone does not seem to be sufficient to cover complex distributed systems. Nevertheless, the integration of known design points into automated exploration seems promising, and shall be supported by an efficient exploration framework.

Another property common to many exploration approaches and optimization methods is their specialization to specific application and execution platform models. Examples include classical results for uniprocessor scheduling, like for instance the rate-monotonic priority assignment strategy assuming independent tasks with deadlines equal to the period [71]. Also many priority assignment algorithms for the multiprocessor case are specific to restricted models, usually assuming independent tasks and homogeneous processors [3, 73]. Even though, such tailored methods can be efficiently applied under specific circumstances to determine optimized system configurations, they are only partly applicable in situations where the application or execution platform models slightly deviate from the initial assumptions.

Certainly, exploration and optimization approaches shall be based on state-of-the-art performance verification techniques. However, they must not be intrinsically tied to specific models, since these might be extended or modified due to new research results. The separation of performance analysis and exploration strategy can be achieved best by considering performance analysis as a black box, delivering required system performance properties for given system parameter configurations (see Figure 2.3).

In the past years many performance analysis approaches have been proposed, e.g. [79, 131, 77, 115, 84, 43, 113, 50, 46][1]. Each of these methods has demonstrated high modeling accuracy and simplicity for certain areas but less applicability for others. In order to increase the expressiveness of analysis results it is, thus, desirable that each system part is analyzed by the method with the best fitting model of computation. Of course, mixed performance analysis requires coupling mechanisms between the involved methods. Several analysis and simulation models have recently been coupled. Examples include interfaces between SymTA/S and RTC-MPA [61], between SymTA/S and synchronous data-flow graphs (SDF) based analysis techniques [103], or between RTC-MPA and the MPARM simulator [62]. In each of these works, the advantage of coupling different techniques in terms of analysis accuracy and expressiveness was recognized and pointed out using expressive examples. Design space exploration in general should also exploit these advantages and be applicable across analysis borders.

### 2.4.2 Requirements for design space exploration

Based on the above discussed observations we now formulate key requirements for the design space exploration framework developed in this thesis:

1. The design space exploration framework shall fit modern design flows and designer needs. This particularly imposes following requirements:
   - The framework shall support partial explorations in order to account for iterative refinements between several involved design teams (e.g. OEM-supplier dependencies) and incremental system specifications.
   - The search space shall be composable and configurable in a flexible and fine granular manner.
   - The framework shall allow reusing partial solutions from previous explorations.
2. The design space exploration framework shall be independent of the underlying performance analysis engine. Note that in this thesis we base our experiments and case studies on state-of-the-art analysis engines like SymTA/S [87] and RTC-MPA [126], but we do not want to rely on their specific properties. Therefore, we impose following requirements:

[1] Some of these approaches with relevance to this work will be discussed later.

- The analysis engine used by the framework for evaluation purposes shall be replaceable.
- The framework shall be flexible enough to allow domain spanning design space explorations across several analysis engines.

Another important requirement that we impose on the design space exploration framework is that it shall support Pareto-optimization of several concurrent optimization objectives. The information that is yielded by Pareto-optimization is very valuable for the designer, since it enables her to exactly evaluate and compare different solution alternatives in a way that cannot be provided by uni-objective approaches. Consider, for instance, the solutions shown in Figure 2.4 representing different trade-offs between system performance and cost as they might be obtained by some Pareto-optimization.

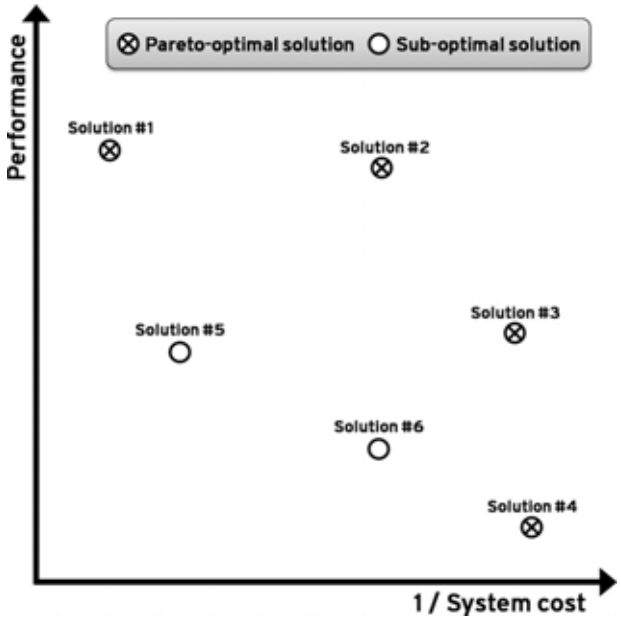



Fig. 2.4: Design trade-offs: performance vs. cost

First of all, Pareto-optimization might yield several optimal solutions that cannot directly be compared. This is the case for the solutions 1, 2, 3, and 4. Optimal in the Pareto-sense means that improvement in one optimization criterion automatically leads to deterioration in other optimization criteria. For instance, if system performance shall be improved and, therefore, e.g. solution 2 is adopted rather than solution 3, then also system cost is increased.

In Pareto-optimization, it is to the designer which solution is chosen from the set of Pareto-optimal trade-offs. However, the knowledge of all possibilities is very valuable for this choice. Consider, for instance, solutions 1 and 2. Both solutions are optimal as explained above. However,

if we take a closer look at the differences, we observe that solution 1 offers a small performance increase compared to solutions 2 at the price of considerably increased overall system cost. With this knowledge at hand the designer would likely opt for solution 2. Similar reflections hold if solutions 3 and 4 are compared. Uni-objective optimization possibly yields solutions 1 or 4, neglecting that there exist nearby design points representing much more interesting trade-offs.

### 2.4.3 Application scenario for iterative design space exploration

In the automotive industry it is common practice that the OEM delegates the development of complex system features to different independently working suppliers. Thereby, the more complex system features, like for instance ABS, ESP, etc., exchange messages with multiple sensors and actuators connected and controlled by several distributed ECUs over a shared network with dynamic behavior (e.g. priority-driven CAN protocol).

Of course, suppliers responsible for the development and implementation of such functionalities are not aware of the globally generated network load. For performance verification it is, therefore, often assumed that different components are mutually independent. In this simplified case the delay of exchanged messages can easily be determined and taken into account for performance verification. However, in the final system several independently developed distributed functionalities are integrated onto the same communication infrastructure. Obviously, the assumption of independency does not hold in this case, since messages from different functionalities interfere on the communication infrastructure leading to dynamic blocking effects and, thus, increased transmission delays.

In order to smoothen such integration effects the OEM usually limits the maximum admissible network load. The idea behind this strategy is that low network load implies less exchanged messages and frames[1], (hopefully) translating into less collisions and shorter transmission delays. However, the main drawback of this approach is that limiting the network load is inefficient in terms of system cost and total system communication delay.

An alternative to control integration effects is systematic optimization of the communication infrastructure, whose optimization potential

[1]In the CAN [15, 1] terminology data packets that are exchanged between tasks are called messages. In the communication layer of the CAN protocol several of these messages from different tasks are packed into frames for transmission over the bus.

is usually not fully exploited. The main reason for this is that it is difficult to perform optimization along the supply chain: suppliers are responsible for the configuration of their ECUs, whereas the OEM controls the configuration of the communication infrastructure.

As already discussed above, isolated exploration and optimization of single components is of limited use, since system level performance interdependencies are not captured. Also closed global exploration is not feasible, since neither OEM nor single suppliers control all necessary model parameters, and are reluctant to share design details due to IP-protection issues. As a consequence, system optimization and exploration can be tackled best in an iterative process. In the following we shortly discuss, as one example, how system communication behavior can be iteratively explored and optimized. In this specific example OEM and suppliers only need to exchange few uncritical communication related performance data: message jitters and frame offsets.

Figure 2.5 visualizes an iterative exploration process between OEM and suppliers including message jitter and frame offset optimization. Note that a similar optimization scenario is proposed in [98].

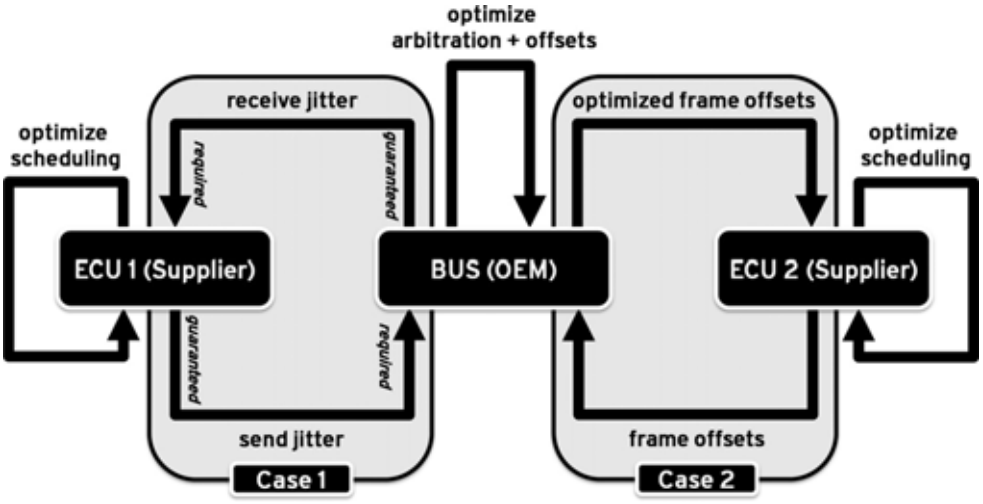



Fig. 2.5: Optimization of the communication infrastructure in a parallel design flow between OEM and several suppliers

**Case 1: Message jitter optimization** The OEM can optimize the bus arbitration (e.g. priorities, time slots, etc.) such that large send jitters, leading to transient load peaks, can be tolerated without resulting in message loss (depending on the application, the transmission delay of a message must usually be significantly smaller than its period). The maximum sustainable send jitters for the individual messages can then be communicated to the suppliers as target requirements for local ECU optimization. The other way around, the OEM can optimize the bus to

ensure optimal message receive jitter properties that might be crucial e.g. for timing sensitive control algorithms. Of course, both scenarios are interdependent, and can thus be addressed best in an iterative exploration approach.

**Case 2: Frame offset optimization.** Frame offsets are mostly ECU specific, and thus unknown to the OEM in early design phases. At the same time system performance is strongly influenced by these offsets, since they can be effectively used for load balancing on the communication infrastructure [98]. Obviously, each supplier can only decide in a very limited scope on reasonable frame offsets. Therefore, the OEM as integrator should be in charge to globally explore and optimize frame offsets.

The following iterative frame offset optimization process is possible. In the first step, all suppliers separately optimize frame offsets that are relevant for their ECUs. In the second step, the OEM as network integrator collects all frame offsets and performs global optimization taking into account specific supplier requirements (e.g. receive message jitters). The optimized frame offsets are then communicated back to the suppliers. Finally, in the third step, each supplier performs local ECU optimization (e.g. scheduling parameters) taking into account the optimized timing properties of incoming messages.

Obviously, this frame offset optimization process is interdependent with message jitter optimization (case 1). Optimized frame offsets can lead to looser jitter requirements or tighter jitter guarantees. Frame offset optimization needs, therefore, to be successively refined once more detailed information about communication timing is available.

## 2.5 Design Robustness

In the field of embedded system design, robustness is usually associated with reliability and resilience. Therefore, many approaches to fault tolerance against transient and permanent faults with different assumptions and for different system architectures can be found in literature, e.g. [67, 38, 86]. These approaches increase system robustness against effects of external interferences (radiation, heat, etc.) or partial system failure and are, therefore, crucial for safety critical systems.

In this thesis a different notion of robustness for embedded systems is introduced and discussed: robustness to variations of system properties. Informally, a system is called robust if it can sustain system property modifications without severe consequences for system performance and integrity. In contrast to fault tolerance requiring the implementation of specific methods such as replication or re-execution mechanisms [53]

to ensure robustness against faults, robustness to property variations is a meta problem that does not directly arise from the expected and specified functional system behavior. It rather represents an intrinsic system property that depends on the system organization (architecture, application mapping, etc.) and its configuration (scheduling, etc.).

Accounting for property variations early during design is key, since even small modifications in systems with complex performance dependencies can have drastic non-intuitive impact on the overall system behavior, and might lead to severe performance degradation effects [90]. Since performance evaluation and exploration do not cover these effects, it is clear that they are insufficient to systematically control performance along the design flow and during system lifetime. Therefore, explicit robustness evaluation and optimization techniques that build on top of performance evaluation and exploration are needed. They enable the designer to introduce robustness at critical positions in the design, and thus help to avoid critical performance pitfalls.

The remainder of this section is structured as follows. First, we demonstrate by means of small but realistic examples that marginal property variations can have severe consequences for system performance (Section 2.5.1). Afterwards, we give a short survey of related work, and discuss how robustness can adequately be evaluated for complex distributed embedded systems (Section 2.5.2). We then explain in detail different situations where system robustness to property variations is highly desirable and crucial (Section 2.5.3). Finally, we motivate assumptions and design scenarios that we want to cover with the robustness evaluation and optimization techniques presented in this thesis (Section 2.5.4). Note that formal definitions and the concrete technical realization are discussed in Chapter 4.

### 2.5.1 Effects of system property variations

System properties variations put at risk system performance, since they invalidate the assumption under which the system was originally dimensioned and configured. That variations of system properties can have severe influences on system performance that cannot be locally predicted shall be demonstrated by means of several small experiments on the example system introduced in Section 2.3. In these experiments we monitor the influence of several system properties on the end-to-end latencies of the constrained sub-applications.

Note that the deadlines in the given examples are synthetic and intentionally placed at unfavorable locations. Consequently, the intention of the examples is not to claim that deadlines are always jeopardized by drastic system performance degradation effects, but rather to show

that such situations might occur and are hard to discover due to abrupt stepwise latency increases (compare Figures 2.6, 2.7, 2.8, and 2.9) that cannot be predicted locally.

**Worst-Case Execution Time** In the first experiment we are interested in the impact of worst-case execution time (WCET) variations of the task $T8$. Figure 2.6 shows the results. The WCET of $T8$ does not have any influence on the latencies of $Sens \rightarrow Act$ and $Cam \rightarrow V_{out}$, since $T8$ has the lowest priority on $ARM$. However, the situation looks different for $S_{in} \rightarrow S_{out}$. A small increase of the originally assumed WCET by less than 1% (to 294.1 time units) leads to an end-to-end latency increase of 7.4% (from 898.9 to 965.4 time units). Afterwards, the end-to-end latency of $S_{in} \rightarrow S_{out}$ slowly increases, until it makes another unexpected jump at 320.3 time units leading to deadline violation and, thus, system failure. At this point an overall WCET increase of $T8$ by approximately 10% lead to an end-to-end latency increase for $S_{in} \rightarrow S_{out}$ by more than 27.5% (from 898.9 to 1146.5 time units).

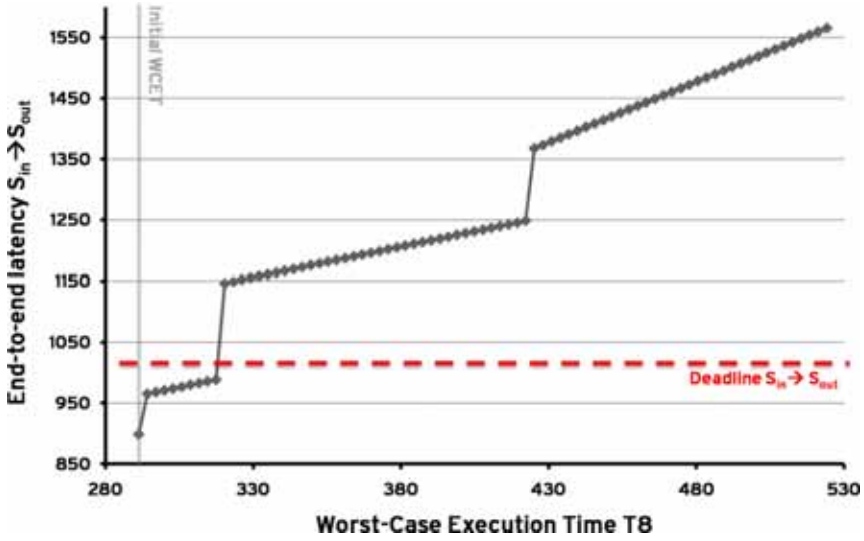



Fig. 2.6: Impact of WCET variations on the performance of the system described in Figure 2.2

**Period and Jitter** In the second experiment we investigate variations of timing properties at system input $Sens$. Figures 2.7 and 2.8 visualize the impact of period and jitter variations, respectively.

Reducing the workload generated through $Sens$ by increasing its activation period has only little positive impact on the system's timing behavior. However, the given system is very sensitive to decreasing the

period separating two consecutive events[1]. Already a minor decrease from 450 to 435 time units (3.3%) leads to an end-to-end latency increase for $S_{in} \rightarrow S_{out}$ from 898.9 to 1037.3 time units and, thus, to deadline violation.

The presence of jitter at the input *Sens* also leads to severe system performance degradation. The reason is that jitter induces transient load peaks in the worst-case. The most affected sub-application is $S_{in} \rightarrow S_{out}$. Already a small input jitter of 30 time units, corresponding to 6.7% of the period, leads to an abrupt increase of the end-to-end latency to 1037.3 time units, which exceeds the imposed deadline. $Cam \rightarrow V_{out}$ seems to be less sensitive at first glance. Up to a jitter of 350 time units the end-to-end latency only slightly increases from 810.8 to 873.2 time units (7.7%). However, increasing the jitter by another 10 time units results in a sudden jump of the end-to-end latency to 1035.1 time units.

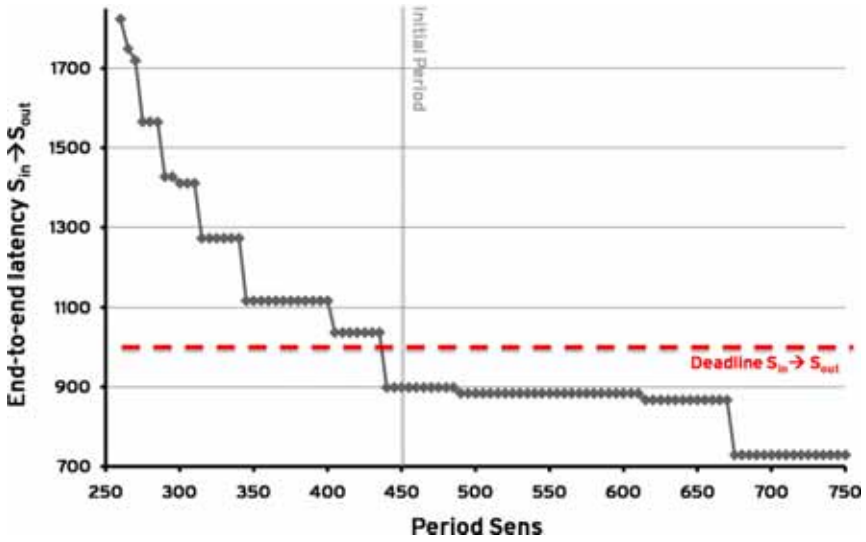



Fig. 2.7: Impact of period variations on the performance of the system described in Figure 2.2

**Resource Speed** In the last experiment we examine the impact of service capacity variations of the *ARM* processor. Figure 2.9 shows the end-to-end latency of $S_{in} \rightarrow S_{out}$ as a function of the clock rate. We observe that a small service capacity decrease of 5% results in an end-to-end latency increase from 898.8 to 1163.9 time units (29.5%) exceeding by far the imposed deadline.

[1]Shorter periods are often used to increase system reaction time.

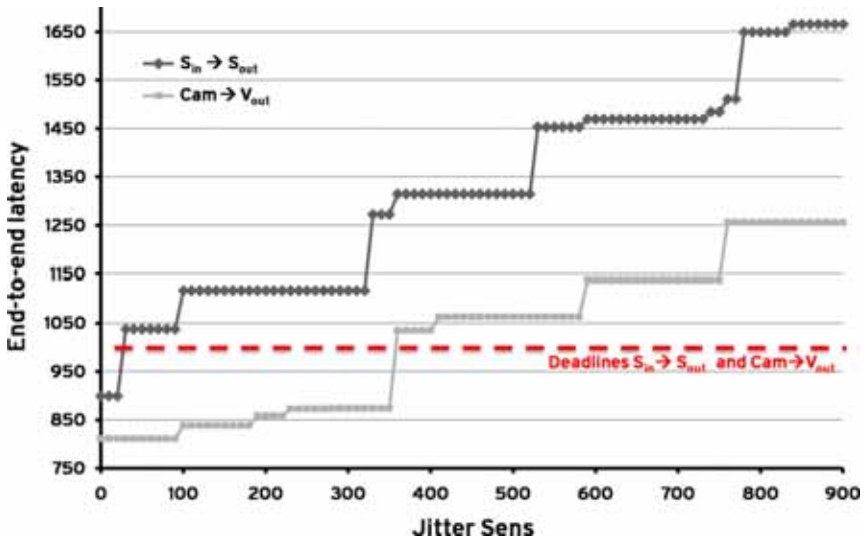



Fig. 2.8: Impact of jitter variations on the performance of the system described in Figure 2.2

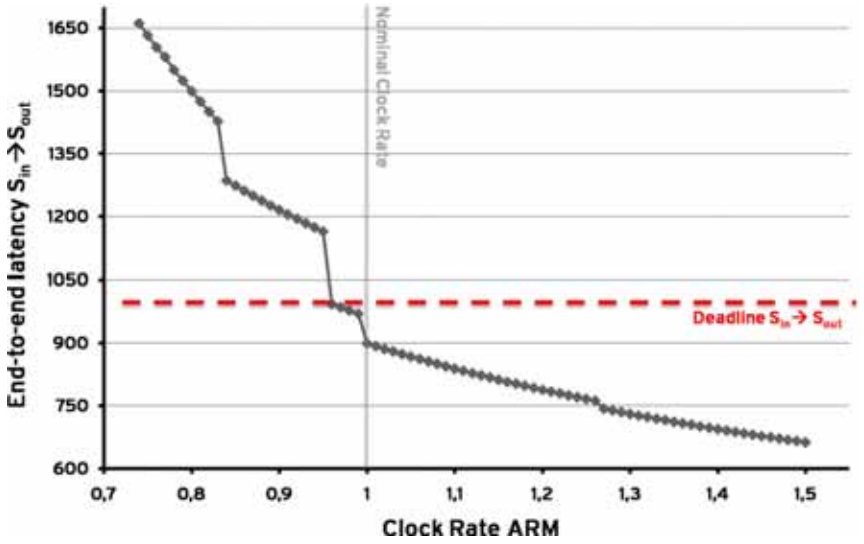



Fig. 2.9: Impact of service capacity variations on the performance of the system described in Figure 2.2

### 2.5.2 Evaluating design robustness

Determining design robustness in complex distributed embedded systems is far more complicated than determining the remaining service capacity that is locally available on computational resources. One reason is that the influence of property variations on global constraints, such as end-to-end deadlines, is not easy to capture. Additionally, complex backward pressure effects further complicate this problem.

Many previous approaches to robustness optimization assume simplistic application models that do not cover these effects, and are, therefore, only partly applicable to complex distributed embedded systems. For instance, in [57] and [39] the authors address the allocation problem for

tasks with environment-dependent workload requirements. Accordingly, the work aims at finding task allocations leading to systems that are robust to workload variations. The considered application model assumes homogeneous multiprocessor systems with rate monotonic scheduling (RMS) and independent strictly periodic tasks with deadlines equal to the periods. In this context, system robustness can be easily derived and optimized using well known utilization bounds for the assumed simple application model. In the above mentioned work the utilization bound for one-processor RMS from Liu and Layland [71], and for n-processor RMS from Baker et al. [80] are used.

A more realistic approach to system robustness optimization applicable to more complex situations is presented in [7]. The authors consider several predefined modification scenarios, e.g. changed execution demands or additional tasks, that a given system shall be able to accommodate with no or little reconfiguration effort. The approach, while technically sound, is only partially appropriate to systematically evaluate and optimize system robustness. The reason is that, in general, property variations can only hardly be anticipated and exactly quantified. In this context each considered scenario only represents one possible design point out of many, which is not sufficient if we consider the fragility of system performance [90]. Consequently, the approach is mainly applicable to minimize the system reconfiguration effort for clearly defined change scenarios between different system states. This is, for instance, the case for system mode transitions [27].

In order to systematically assess robustness with respect to unforeseeable system property variations sensitivity analysis is needed. State-of-the-art sensitivity analysis approaches [93, 91] capture global effects of property variations, and are capable of determining system feasibility limits. Typical application scenarios for sensitivity analysis include the determination of maximum sustainable data rates, or minimum required bus throughputs. Consequently, sensitivity analysis represents a valuable tool for system dimensioning [92].

Sensitivity analysis has already been successfully used for the evaluation and optimization of specific system robustness aspects. In [74], for instance, the authors present a sensitivity analysis technique calculating maximum input rates that can be processed by stream processing architectures without violating on-chip buffer constraints. The authors propose to integrate this technique into automated design space exploration to find architectures with optimal stream processing capabilities, i.e. high robustness against input rate increases.

In this thesis we systematically utilize sensitivity analysis for general robustness evaluation and optimization purposes. More precisely, in-

stead of consuming available slack for system dimensioning, and thus cost minimization, slack is distributed such that the system's capability of sustaining property variations is maximized. Using sensitivity analysis as basis for robustness evaluation and optimization has two important advantages compared to previous approaches:

1. State-of-the-art modular sensitivity analysis techniques capture complex global effects of local system property variations. This ensures the applicability of the proposed robustness evaluation and optimization techniques to realistic performance models, and increases the expressiveness of the results.

2. Rather than providing the system behavior for some isolated discrete design points, sensitivity analysis characterizes continuous design sub-spaces with identical system states. It covers, thus, all possible system property variation scenarios.

### 2.5.3 Use cases for design robustness

In the following we discuss situations and scenarios where robustness of hardware and run-time system against property variations is expected and crucial to efficiently design complex embedded systems.

**Unknown quality of performance data** First of all, robustness is desirable to account for data quality issues in early design phases, where data that is required for performance analysis (e.g. task execution times, data rates, etc.) is often estimated or based on measurements. As a result of the unknown input data quality, also the expressiveness and accuracy of performance analysis results are unknown. Since even small deviations from estimated property values can have severe consequences for the final system performance, it is obvious that robustness against property variations leverages the applicability of formal analysis techniques during design. Clearly, the design risk can be considerably reduced by systematically optimizing the system for robustness.

**Maintainability and extensibility** Secondly, robustness is important to ensure system maintainability and extensibility. Since major system changes in reaction to property variations are usually not possible during late design phases or in the field, it is important to choose system architectures and configurations offering sufficient robustness for future modification and extensions as early as possible. For instance, the huge number of feature combinations in modern embedded systems has lead to the problem of product and software variants. Using robustness optimization techniques systems can be designed, at the outset, to ac-

commodate additional features and changes. Obviously, this can reduce the design effort for the supplier. Other situation where robustness can increase system maintainability and extensibility include late feature requests, product and software updates (e.g. new firmware), bug-fixes, and environmental changes.

**Reusability and modularity** Finally, robustness is crucial to ensure component reusability and modularity. Even though these issues can be solved on the functional level by applying middleware concepts, they are still problematic from the performance verification point of view (see Section 1.3). The reason is that system performance is not composable, prohibiting the straightforward combination of individually correct components in a cut-and-paste manner to whole systems. In this context, robustness to property variations can facilitate the reuse of components across product generations and families, or simplify platform porting.

### 2.5.4 Design scenarios and assumption for robustness evaluation and optimization

In the past years, system robustness to changes of performance properties was usually addressed intuitively and manually during the design flow. The main reason for this informal approach was the lack of reliable methods capable of assessing performance characteristics of complex distributed embedded systems. Consequently, it is not surprising that only little related work can be found in the literature of embedded systems design. However, with the advent of sophisticated performance analysis methodologies, such as SymTA/S [50] or RTC-MPA [126], the situation has changed. Based on the performance analysis results delivered by these methods it has become possible to define expressive and reliable quantitative robustness metrics that can be adequately used to improve the design process. In this thesis, a formal basis for robustness optimization is proposed covering the following design scenarios:

**Static design robustness** The first considered design scenario assumes that system parameters are fixed early during design and cannot be modified later (e.g. at late design stages or in the field) to compensate for system property modifications. This scenario shall be called *static design robustness (SDR)*.

The SDR metric shall express the robustness of parameter configurations with respect to the simultaneous modification of several given system properties. Since the exact extent of system property variations can generally not be anticipated, it is desirable that as many as possible modification scenarios can be sustained by the system. This shall be

transparently expressed by the SDR metric: the more different modification scenarios represent feasible system states for a specific parameter configuration, the higher the corresponding SDR value. Additionally, the SDR metric shall include the possibility to attach different relevance levels to the included system properties through weighting. Criteria for the designer to assign these weights include the estimated probability of future changes and the impact on the overall system performance. Note that SDR optimization yields a single parameter configuration possessing the highest robustness potential for the considered system properties.

**Dynamic design robustness** The SDR metric assumes static systems with fixed parameter configurations. However, in this thesis also the influence of dynamic system behavior on robustness shall be evaluated. In other words, potential designer or system counteractions shall be included into robustness evaluation. This scenario shall be called *dynamic design robustness (DDR).*

The DDR metric shall express the robustness of given systems with respect to the simultaneous modifications of several system properties that can be achieved through reconfiguration. Consequently, it is relevant for the design scenario where parameters can be (dynamically) modified during design or in the field. Obviously, the DDR metric depends on the set of possible parameter configurations $\mathcal{C}$ ("counteractions") that can be adopted through reconfiguration. For instance, it might be possible to react to property variation by scheduling parameters adaptation (e.g. adaptive scheduling strategies [72], network management techniques [31], etc.) or application remapping.

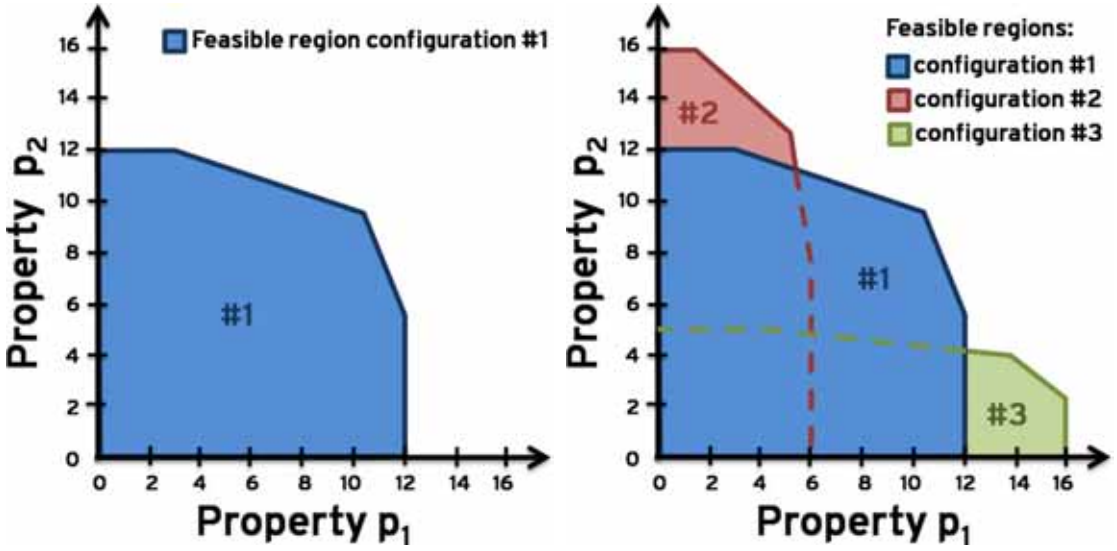
Application scenarios for the DDR metric include the evaluation of dynamic systems and, more generally, the assessment of the design risk connected to specific components. More precisely, already early during design the DDR metric can be used to determine bounds for property values of specific components ensuring their correct functioning in the global context. This information effectively facilitates feasibility and requirements analysis and greatly assists the designer in pointing out critical components requiring special focus during specification and implementation. Another use case concerns reconfigurable systems. The DDR metric can be used to maximize the dynamic robustness headroom for crucial components. Obviously, by early choosing a system architecture offering high DDR for crucial system parts, the designer can significantly increase system stability and maintainability.

Like the SDR metric, also the DDR metric shall include the possibility to attach different relevance levels to the included properties through weighting. Note that DDR optimization yields multiple param-

eter configurations, each possessing partially disjoint robustness properties. For instance, one parameter configuration might exhibit high robustness for some system properties, whereas different parameter configurations might offer more robustness for other system properties. In case of system property variations, an appropriate feasible parameter configuration can be adopted. For adaptations during system runtime, pre-calculated parameter configurations can be stored in a lookup table from which appropriate operating points can be dynamically chosen.

**Robustness gain through reconfiguration** Based on the notions of static and dynamic design robustness, metrics expressing the robustness increase that can be achieved through reconfigurability shall be derived. Given such metrics, the benefit (in terms of system robustness) of designing reconfigurable components is explicitly measurable. Consequently, it can help system architects to decide whether or not investing engineering effort into creating reconfiguration mechanisms is worthwhile. More generally, by adjusting the available reconfiguration space $\mathcal{C}$ these metrics can be used to identify critical components for which reconfigurability is particularly advantageous, i.e. leading to significant robustness gain. In this manner, engineering effort for conceiving robust systems can be efficiently focused.

**Conceptual difference between SDR and DDR** Figures 2.10a and 2.10b visualize the conceptual difference between the notions of static and dynamic design robustness by means of a small example. Figure 2.10a shows the feasible region of two properties $p_1$ and $p_2$, i.e. the region containing all feasible property value combinations, of a given parameter configuration. This corresponds to the static robustness, where a single parameter configuration with high robustness needs to be chosen. Figure 2.10b visualizes the dynamic robustness. In the considered case there exist two additional parameter configurations in the underlying reconfiguration space with interesting robustness properties. Both new parameter configurations contain feasible regions that are not covered by the first parameter configuration. The union of all three feasible regions corresponds to the dynamic robustness.




(a) Static robustness - feasible region of a single system configuration

(b) Dynamic robustness - combined feasible regions of several system configurations

Fig. 2.10: Conceptual difference between static and dynamic design robustness for two considered system properties subject to maximization

Chapter 3

# ITERATIVE DESIGN SPACE EXPLORATION FRAMEWORK

In this chapter we introduce an iterative and composable design space exploration framework that can be easily extended and configured to cover a large variety of optimization tasks in the context of distributed embedded real-time systems. The main goal of the framework is to enable systematic design space exploration of performance criteria along modern concurrent design flows spanning several domains of responsibility (see discussion in Section 2.4). However, the presented framework will also be used in this thesis to cover multi-dimensional sensitivity analysis and robustness optimization (see Chapter 4).

The remainder of this chapter is structured as follows. First, it is motivated why we utilize multi-objective evolutionary algorithms as basis for the proposed design space exploration framework (Section 3.1). Afterwards, the basic concepts of the design space exploration framework are discussed. This includes the compositional encoding scheme that is used to compose and manipulate the search space at component level (Section 3.2), the integration of systematic techniques for component interaction optimization (Section 3.3), the exploration loop steering and adaptively controlling the exploration process (Section 3.4), as well as the user-controlled exploration methodology (Section 3.5). Then, realization details for covering timing and performance optimization with the exploration framework are discussed (Sections 3.6 and 3.7), and demonstrated by means of a small but realistic case study (Section 3.8). Finally, a framework extension for automated search space modification is presented (Section 3.9), and applied to the optimization of priority assignments on priority scheduled system components (Section 3.10).

## 3.1 Multi-objective evolutionary algorithms

Evolutionary algorithms (EA) [6] are modern optimization tools that can be used to efficiently solve a large variety of optimization problems. In the following some basic notions that are often utilized in the context of EAs are explained:

- *Population*: Set of individuals that are simultaneously considered by the evolutionary algorithm and that interact during reproduction.
- *Individual*: Structure representing one candidate solution annotated with fitness values.
- *Chromosome*: Chain of elements encoding the values of a solution. Common structures include binary strings and real numbers. Usually, individuals consists of a single chromosome.
- *Gene*: One element of a chromosome.
- *Fitness*: A set of real values indicating the quality of an individual as a solution to the given optimization problem.
- *Generation*: One iteration of the optimization method.
- *Parents*: Individuals that are used during the reproduction process, i.e. the creation of new candidate solutions.
- *Offsprings*: Individuals created through reproduction of parent individuals.
- *Variation / Reproduction*: Process of creating offspring individuals from parent individuals through crossover and mutation.

The optimization principle used by evolutionary algorithms is inspired by the concept of natural selection and evolution, and can be explained in simplified form as follows. At the beginning of the optimization process a (random) population of individuals, each representing a unique candidate solution, is generated. The individuals are evaluated with respect to their quality as solution to the given optimization problem. Based on the calculated fitness values, specific individuals are chosen as parents for reproduction, and hence get the chance to pass their genes to offsprings that are added to the population. Other parents, usually those with poor fitness values, are removed from the population. The whole process is repeated with the modified population. Using this principle, the quality of obtained solutions is increased gradually from generation to generation.

Many real-world problems consist of the simultaneous optimization of several conflicting optimization objectives. One example in the context of embedded system design is the aim of simultaneously minimizing cost while maximizing performance. Obviously, in such cases there exist no single optima but several so-called *Pareto-optimal solutions* representing trade-offs between several optimization objectives that cannot be compared to each other without providing further preference information (e.g. weighting). More precisely, given a set $V$ of $k$-dimensional vectors $v \in \mathbb{R}^k$, the vector $v \in V$ *Pareto-dominates* the vector $w \in V$ iff for all elements $0 \leq i < k$ it holds that $v_i \leq w_i$ and for at least one element $l$ it holds that $v_l < w_l$. A vector is called *Pareto-optimal* iff it is not Pareto-dominated by any other vector in $V$.

The determination (or approximation) of the Pareto-optimal solution set for a given optimization problem is called *multi-objective optimization*. Accordingly, evolutionary algorithms that are capable of performing multi-objective optimization are called *multi-objective evolutionary algorithms (MOEA)* [23]. Note that the discussion of MOEAs is not subject of this thesis. They are rather used as tools to solve the optimization problems that are formulated in this thesis. For this reason, we utilize the PISA framework (platform- and programming-language-independent interface for search algorithms) [12] making state-of-the-art MOEAs (e.g. SPEA2 [136], IBEA [135], NSGA2 [24], etc.) readily available.

There are several reasons why MOEAs have been chosen to solve the optimization problems formulated in this thesis.

1. MOEAs are very flexible and allow the realization of a composable encoding scheme that enables (1) the dynamic control of the search process and (2) the straight-forward extension to cover new components and optimization parameters (see Section 3.2). Many other optimization techniques including (meta) heuristics do not offer such flexibility.

2. MOEAs are independent of the underlying fitness landscape. This is of particular importance for the optimization problems addressed in this thesis. The reason is that the underlying performance analysis engines (that are considered as black-boxes in this thesis) are still subject to research, and thus improvement. In particular, the scope of the performance models is constantly extended and more and more dynamic effects are considered. Examples are intra- and inter-context information [81, 55, 47], scenario aware analyses [49], mixed synchronous and asynchronous time-table activations, complex

activation semantics [41, 54], hierarchical event models [100], shared memory accesses [102], etc.

Many optimization techniques, particularly heuristics, must be explicitly tailored to specific performance models, and hence need to be adapted in case of model extensions or analysis improvements. By contrast, optimization approaches based on MOEAs automatically adapt to the underlying performance model, and thus directly profit from extension and improvements without further adaptations.

3. As opposed to many other optimization techniques, MOEAs are capable of performing Pareto-optimization. As discussed in Section 2.4.2, Pareto-optimization is very valuable for the designer, since it allows determining and comparing different design trade-offs in a way that cannot be provided by uni-objective approaches. The capability of performing Pareto-optimization was, therefore, identified as one of the key requirements for the proposed design space exploration framework.

## 3.2 Compositional search space encoding

Figure 3.1 shows the compositional search space encoding concept that is used for the proposed exploration framework. The system is seen component wise for design space exploration. Modifiable parameters of different physical or logical components are encoded separately by specialized *chromosomes*. Thereby, each chromosome consists of three parts:

1. The encoding of the underlying problem parameters using arbitrarily complex data structures (e.g. permutations for priority optimization, real numbers for TDMA slot optimization, etc.).

2. Problem-aware variation operators guiding the search process for the system part represented by the chromosome (i.e. crossover and mutation).

3. Chromosome specific parameter restrictions (e.g. fixed TDMA turn-lengths, etc.) and repair algorithms enforcing these restrictions on the underlying data structures[1].

Chromosomes represent encodings and local exploration strategies for specific underlying components. Additionally, they carry variation

[1]The variation operators used in the framework do not directly respect parameter restrictions. Conformity is reached after variation by applying repair algorithms.

operators necessary for recombination with other chromosomes of the same types. Chromosomes are coupled through so-called *exploration interfaces* to the underlying performance models that are responsible for translating the encoded information into concrete parameter assignments. This concept simplifies the applicability of the exploration framework across different analysis engines and performance models.

For system exploration, several chromosomes can be combined to compose the desired search space. In the example given in Figure 3.1, for instance, four chromosomes are used to define the search space. Three chromosomes are connected to the resources $CPU1$, $CPU2$, and $BUS$ and represent scheduling parameter assignments for mapped tasks and communication channels, respectively. The fourth chromosome models a traffic shaper optimizing the communication behavior of $T4$ over the $BUS$ (see Section 3.3).

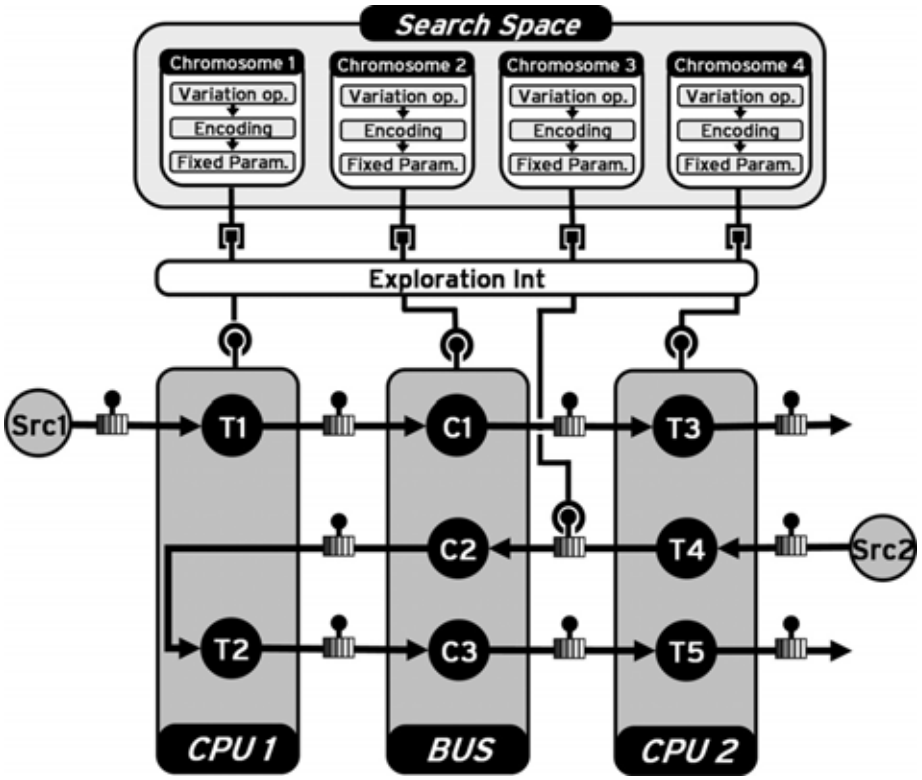



Fig. 3.1: Compositional search space encoding scheme

We have chosen to split the overall system exploration into several entities, i.e. chromosomes, controlling the exploration on local components for several reasons:

1. The effort of checking generated design alternatives for validity is reduced. The reason is that constructively correct encodings for small subsets of design parameters are much easier to establish. Based on valid component encodings, many design alternatives that are invalid

due to local properties and restrictions are not constructed. This considerably reduces the set of possible invalid design alternatives, and thus greatly improves the exploration process.

2. Due to the compositional structure, the exploration framework can be easily extended with new chromosomes to cover additional model parameters and system components. This is important, since the underlying formal performance analysis engines are constantly refined and extended to handle new scheduling policies and bus protocols such as e.g. Flexray [85].

   Furthermore, the flexibility of the compositional encoding scheme allows applying the exploration framework to a large variety of optimization problems in the context of distributed embedded real-time systems. For example, using chromosomes varying system properties rather than system parameters, the exploration framework can be used to efficiently cover multi-dimensional sensitivity analysis and system robustness optimization (see Chapter 4).

3. Modeling sub-optimization problems by separate chromosomes allows integrating problem-specific knowledge into local search strategies, which increases overall exploration efficiency. Examples for problem-aware encodings and variation operators can be found in Section 3.6.2 (TDMA chromosome) and Section 4.3 (multi-dimensional sensitivity analysis).

4. The compositional encoding scheme permits fine-granular search space definition at component level. Additionally, chromosomes can be dynamically added and removed from the search space without loosing previously obtained results. These features enable user-controlled iterative design space exploration with dynamic search space modifications that perfectly suits modern design flows with shared responsibilities and successive model refinements (see discussion in Section 2.4). Furthermore, user-controlled design space exploration can be exploited to manually guide the search process to promising search space regions containing interesting design alternatives. Note that user-controlled design space exploration is discussed in Section 3.5.

5. The compositional structure of the framework enables design space exploration across several analysis domains. More precisely, search space definition through chromosome composition is transparent to the fact that different system parts are analyzed by different analysis engines.

## 3.3 Component interaction optimization

A key property that is exploited by the proposed design space exploration framework is the adaptation of event timing in streams connecting functionally dependent components (expressed through the adaptation of an event model [99]).

Generally, there are several reasons to do this. It may be that schedulers or scheduling analyses for particular components require certain event stream properties. For example, rate-monotonic scheduling [71] requires strictly periodic task activations. Alternatively, IP components may require specific input event stream properties. External system outputs may also impose event model constraints, e.g. bounded minimum distances $d^-$ between output events or maximum jitters. Such constraints may be, for instance, the result of performance contracts with external subsystems [116].

Event stream adaptation can also be done for the sole purpose of *traffic shaping* [99]. Traffic shaping can be used to reduce transient load peaks resulting in more regular system behavior. In this thesis two types of event adaptation functions (EAF) are discussed: *periodic* EAFs, producing periodic event streams from *periodic with jitter* input event streams, and $d^-$-EAFs enforcing a configurable minimum distance between output events.

Since the manipulation of event streams via event stream adaptation weakens performance dependencies between connected components its applicability to system optimization is very promising. An extreme measure would be to use *periodic* EAFs on all event stream in a system, enforcing strictly periodic interactions between all components. Clearly, this completely decouples all performance dependencies, reducing the global optimization problem to several local component optimizations. However, since *periodic* EAFs induce high latencies on the underlying event streams and require large buffers, such a measure would surely lead to unacceptably high end-to-end latencies and exorbitant buffering costs in the resulting system. Therefore, mainly $d^-$-EAF are useful for system optimization and exploration. Compared to *periodic* EAFs providing full synchronization, they allow to trade the grade of component performance decoupling and peak load reduction versus increased delays and buffer sizes along the shaped event stream.

Note that there exist additional traffic shaper types that can be used for component interaction optimization with the method proposed in this thesis. One example are so-called *greedy shapers* [124]. However, this thesis focuses on traffic shaping with $d^-$-EAFs. Therefore, these approaches are not further discussed in the following.

In the next section the concept of traffic shaping with $d^-$-EAFs proposed by Richter [96] for standard event models (see Section 2.2) is explained. Compared to full synchronization, $d^-$-EAFs provide promising peak load reduction and load balancing capabilities with smaller buffers and delays. Larger $d^-$ values result in more balanced system load and better schedulability, while they increase delays and buffering requirements along task chains (or paths). In Section 3.3.2, the optimization potential of traffic shaping will be demonstrated by means of a small but realistic example. Note that the concrete realization of the chromosome that is used to integrate traffic shaping into design space exploration is discussed in Section 3.6.3.

### 3.3.1 Traffic shaping with $d^-$-EAFs

As discussed in Section 2.2.2, a bursty event stream is defined by three parameters, an average period $P$, a jitter $J$, and a minimum event distance $d^-$ during bursts. Figure 2.1 gave an example for a bursty event stream, and in Equation 2.1 an upper bound on arriving events in a given time interval $\Delta t$ was derived.

Additional bounds on minimum event distances can be enforced using time-out buffers, called *traffic shapers*, that are inserted in the system between two functionally dependent components. The time-out mechanism buffers arriving events such that two successive events are not released earlier in time than $d^-_{\text{time out}}$.

According to the extended real-time calculus approach of Thiele et al. [111], the shaper defines a sporadic upper-bound *service curve* [112]:

$$\eta^+_{\text{time out}}(\Delta t) = \left\lceil \frac{\Delta t}{d^-_{\text{time out}}} \right\rceil$$

The shapers output arrival curve can be calculated from both, input arrival curve $\eta^+_{\text{in}}(\Delta t)$ and shaper service curve $\eta^+_{\text{time out}}(\Delta t)$. In case of traffic shapers the real-time calculus equations can be simplified to

$$\begin{aligned} \eta^+_{\text{shaped}}(\Delta t) &= \min\left(\eta^+_{\text{timeout}}(\Delta t), \eta^+_{\text{in}}(\Delta t)\right) \\ &= \min\left(\left\lceil \frac{\Delta t}{d^-_{\text{timeout}}} \right\rceil, \left\lceil \frac{\Delta t}{d^-_{\text{in}}} \right\rceil, \left\lceil \frac{\Delta t + J}{P} \right\rceil\right). \end{aligned}$$

Obviously, the larger value of $d^-_{\text{in}}$ and $d^-_{\text{time out}}$ dominates the other, allowing to further simplify the $\eta^+_{\text{shaped}}(\Delta t)$ function. In case of $d^-_{\text{time out}} \leq d^-_{\text{in}}$, the shaper does not represent an additional constraint. In other words, the shaper is "inactive", no events are buffered and the out-

put arrival curve equals the input arrival curve. Clearly, the case of $d^-_{\text{time out}} > d^-_{\text{in}}$ is more interesting. Input events are buffered and the shaper "flattens" the burst slope of the output arrival curve according to $d^-_{\text{time out}}$:

$$\eta^+_{\text{shaped}}(\Delta t) = \min\left(\left\lceil \frac{\Delta t}{d^-_{\text{time out}}} \right\rceil, \left\lceil \frac{\Delta t + J}{P} \right\rceil\right)$$

Figure 3.2 illustrates this behavior. The arrival curve with minimum distance $d^-_{\text{in}}$ covers the service curve defined by $d^-_{\text{time out}}$. The block arrows indicate buffering.

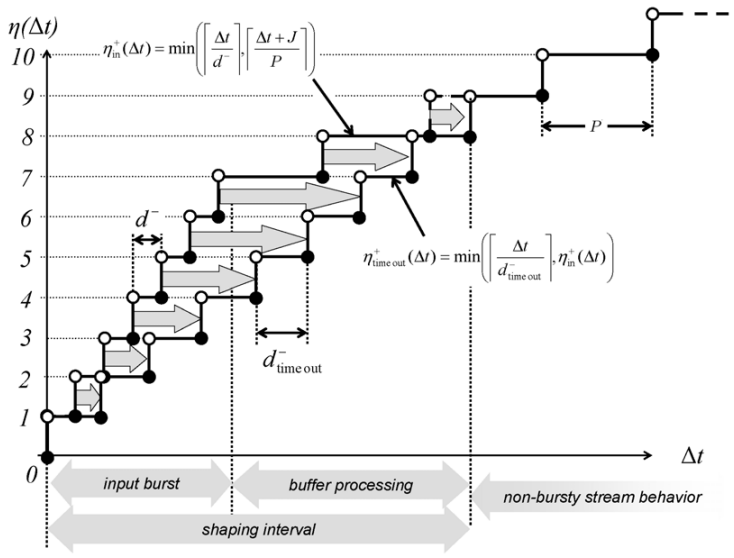



Fig. 3.2: Event arrival curve of output event stream

The vertical distance between arrival and service curve captures the so-called backlog [112], i.e. the number of buffered events at a given point in time:

$$backlog(\Delta t) = \eta^+_{\text{in}}(\Delta t) - \eta^+_{\text{time out}}(\Delta t)$$

Correspondingly, the horizontal distance between the curves, i.e. the arrow lengths in Figure 3.2, represents the buffering delay. The calculations are slightly more sophisticated than for the *backlog*, and can be best formalized using the function $\delta^-(n)$ describing the minimum distance between $n$ successive events in a given stream [96][1]. For the bursty arrival curve and the sporadic service curve, the corresponding $\delta^-(n)$ functions are defined as follows:

[1]Roughly speaking, $\delta^-(n)$ is the inverse of $\eta^+(\Delta t)$.

$\delta^-_{\text{in}}(n) = \max\big((n-1)d^-_{\text{in}}, (n-1)P - J\big)$ and $\delta^-_{\text{time out}}(n) = (n-1)d^-_{\text{time out}}$

Hence, the buffering delay is given by:

$$delay(n) = \delta^-_{\text{time out}}(n) - \delta^-_{\text{in}}(n)$$

Note that the sought-after maxima $backlog_{\max} = \max_{\Delta t>0} backlog(\Delta t)$ and $delay_{\max} = \max_{n\geq 2} delay(n)$ can be calculated through linearization of the discrete $\eta^+$ and $\delta^-$ functions. Details can be found in [96]. For the purposes of this thesis, the following qualitative explanation shall be sufficient. It should not surprise that the worst-case buffering and delay situations appear at the end of the input burst. At that time, *the most events* are stored "waiting" for being processed until the buffer is empty and the behavior returns to "non-bursty". And clearly the last event of the input burst has to *wait longest.*

### 3.3.2 Example

To illustrate the benefit of controlling component interaction with traffic shaping, the example system shown in Figure 3.1 is considered. The system consists of two CPUs connected via a BUS, all scheduled according to the static priority preemptive policy. Best-case and worst-case execution times as well as external activating event models are given in Tables 3.1a and 3.1b, respectively. In order to function correctly, the system has to satisfy the path latency constraints listed in Table 3.2.

It can be easily verified that there exist no priority assignments leading to feasible systems satisfying the imposed latency constraints. However, the situation looks different if traffic shaping is added to the search space.

Let us, for instance, consider the following priority assignment:

- $CPU1$: $T2 > T1$
- $BUS$: $C2 > C3 > C1$
- $CPU2$: $T4 > T5 > T3$

Without traffic shaping this parameter configuration leads to constraint violations for both paths: 1630 time units for $Src1 \rightarrow T3$ and 680 time units for $Src2 \rightarrow T5$. However, system behavior can be tremendously improved by performing traffic shaping with $d^-$-EAFs.

Let us, for instance, take a look at $C1$'s worst-case response time with and without traffic shaping at the output of $T4$. Figure 3.3a visualizes the worst-case scheduling scenario of $C1$ without traffic shaping in the system. Figure 3.3b shows the improved worst-case scheduling scenario

| computation task | core execution time |
|---|---|
| *T1* | [20,20] |
| *T2* | [40,40] |
| *T3* | [30,30] |
| *T4* | [25,25] |
| *T5* | [25,25] |
| **communication task** | **core communication time** |
| *C1* | [10,10] |
| *C2* | [20,20] |
| *C3* | [15,15] |

(a) Core execution and communication times

| input | event model |
|---|---|
| $Src1$ | periodic, $\mathcal{P}_{Src1} = 100$ |
| $Src2$ | periodic, $\mathcal{P}_{Src2} = 100$, $\mathcal{J}_{Src2} = 400$ |

(b) Input event models

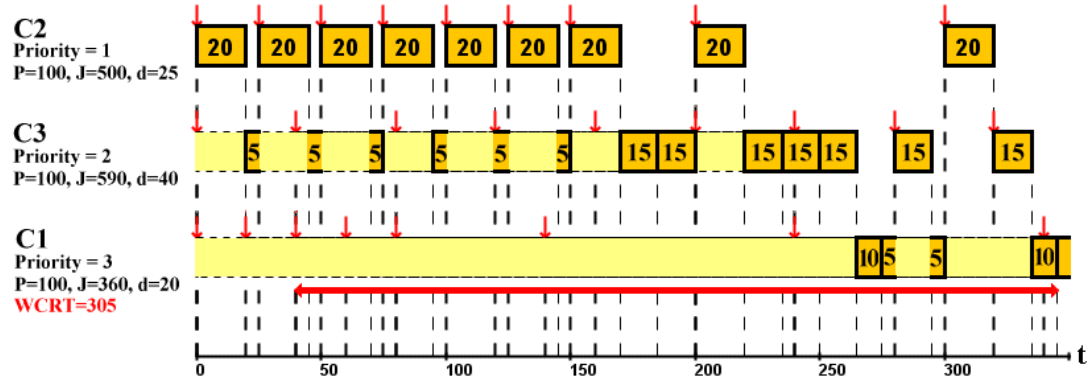
Table 3.1: System parameters for the system depicted in Figure 3.1

| constraint # | path | maximum latency |
|---|---|---|
| 1 | *Src1* $\rightarrow$ *T3* | 800 |
| 2 | *Src2* $\rightarrow$ *T5* | 600 |

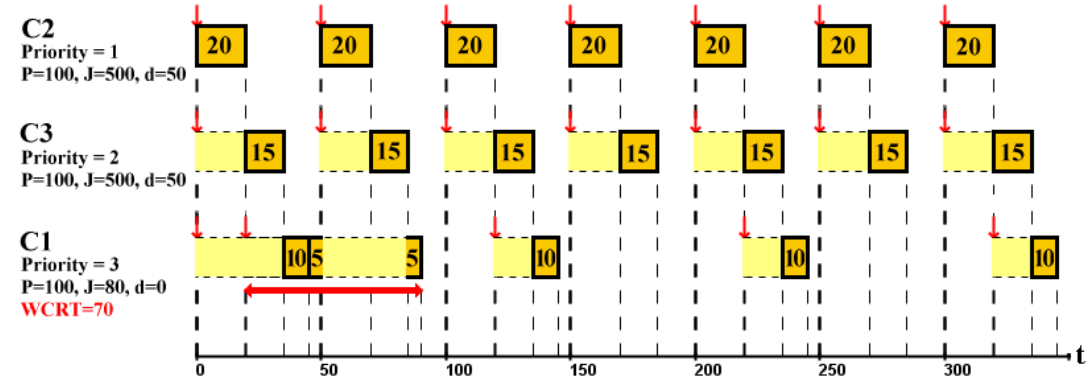
Table 3.2: Path latency constraints for the system depicted in Figure 3.1

of $C1$ with a $d^-$-EAF at the output of $T4$ extending the minimum distance of successive events from 25 to 50 time units. Note that the given activating event models for $C1$, $C2$, and $C3$ are intermediate analysis results that are obtained using the SymTA/S analysis engine [87].

We observe that the inserted $d^-$-EAF leads to the reduction of $C1$'s worst-case response time from 305 to 70 time units. This is due to two effects. First of all, the $d^-$-EAF relaxes the activation burst of $C2$, leading to more freedom for the lower priority tasks $C1$ and $C3$ to execute. This results in less preemption, and thus earlier completion of $C1$ and $C3$. Secondly, the activation jitters of $C1$ and $C3$ are reduced. The reason is that the positive shaping effects on the $BUS$ are propagated through improved output timing behavior, i.e. less response time jitter, to the neighboring components. In the considered case, for instance, the activation jitter of $T2$ on $CPU1$ is reduced. The lower priority task $T1$ is profiting from this in terms of a shorter worst-case response time, and produces, in turn, less output jitter. In the considered case the output




(a) Without traffic shaping




(b) With $d^-$-EAF at the output of $T4$

Fig. 3.3: Worst-case scheduling scenarios for task $C1$

jitter of $T1$, and thus the input jitter of $C1$, is reduced from 360 to 80 time units due to the shaping effects at the output of $T4$.

Figure 3.4 visualizes the global impact of traffic shaping at the output of $T4$ on the constrained end-to-end paths $Src1 \rightarrow T3$ and $Src2 \rightarrow T5$. Possible $d^-$ values lie between 25 and 100 time units, given by the best-case execution time and the activating period of $T4$, respectively.

We observe that the latency of the path $Src1 \rightarrow T3$ falls with growing $d^-$. This is not surprising, since all tasks along the path have the lowest priority on their resources, and are thus highly profiting from the inserted traffic shaper. For the path $Src2 \rightarrow T5$ the situation looks different. First, the end-to-end latency falls, reaching a minimum for $d^-$ values between 40 and 45 time units. Afterwards, the end-to-end latency increases again. The reason for this behavior is, that traffic shaping does not only improve system timing behavior by relaxing peak loads, but also introduces buffering delays. However, up to a $d^-$ value of 69 time units the positive shaping effects dominate the buffering delay, leading to smaller latencies compared to the original parameter configuration without traffic modulation. Altogether, $d^-$-EAFs enforcing $d^-$ values

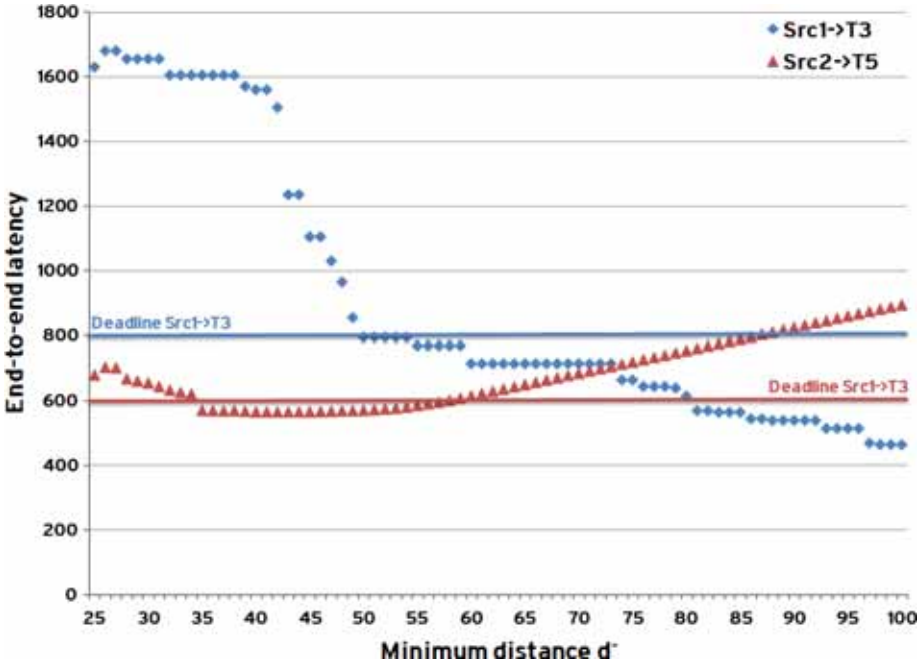



Fig. 3.4: Example illustrating the influence of traffic shaping on system performance

between 50 and 57 time units lead to feasible parameter configurations satisfying the given end-to-end latency constraints.

## 3.4 Design space exploration loop

Figure 3.5 visualizes the design space exploration loop performed by the proposed framework. The *Optimization Controller* is the central element. It is connected to the *Analysis Engine* and to the *Evolutionary Optimizer*. The *Analysis Engine* checks the validity of given individuals and provides data for the *Objectives* to calculate the fitness values subject to optimization.

The *Evolutionary Optimizer* is responsible for the problem-independent part of the optimization problem, i.e. elimination of poor individuals and selection of interesting individuals for variation. Currently, SPEA2 (Strength Pareto Evolutionary Algorithm 2) [136], IBEA (Indicator-Based Evolutionary Algorithm) [135], and FEMO (Fair Evolutionary Multi-objective Optimizer) [64] are used for this part. They are coupled via the PISA interface (Platform and Programming Language Independent Interface for Search Algorithms) [12] with the exploration framework. Note that the selection and elimination strategy, i.e. the strategy to walk through the search space and to approximate the Pareto-optimal solution set in case of multi-criterion optimization, depends on the utilized optimizer. FEMO, for instance, eliminates all Pareto-dominated individuals after each processed generation and pursuits a fair sampling

strategy, i.e. each parent individual participates in the creation of the same number of offsprings. This leads to a uniform search in the neighborhood of elitist individuals.

The problem-specific part of the optimization problem is coded in the chromosomes and their variation operators, i.e. crossover and mutation. Note that specific chromosomes and optimization objectives for timing and performance exploration of distributed embedded systems are discussed in Sections 3.6 and 3.7, respectively.

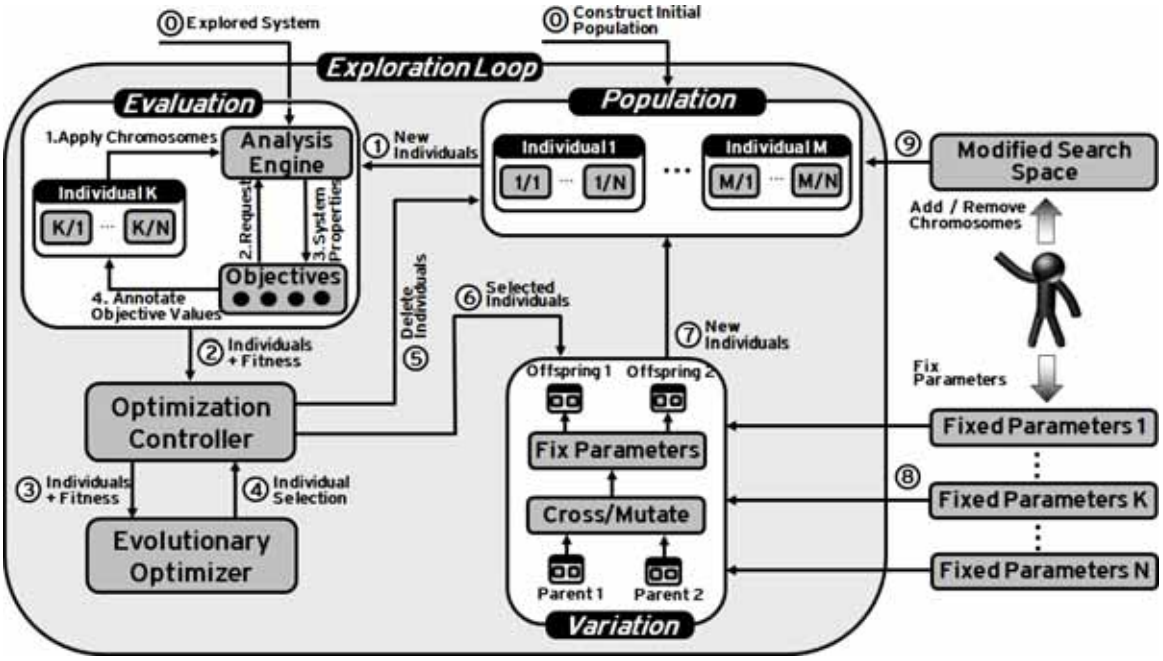



Fig. 3.5: Design space exploration loop

Before exploration can be started *Chromosomes* representing the desired search space as well as *Objectives* subject to optimization have to be selected and configured. Chromosomes representing the search space are included into evolutionary exploration, while all other parameters remain immutable.

After specification of the optimization task, the *Analysis Engine* is initialized with the immutable part of the search space, and the selected chromosomes are used as blueprints to create the initial population (step 0). In other words, each individual consists of specific chromosome instances (phenotypes). Note that each chromosome is responsible for creating initial values for the parameters it represents. By default, chromosomes create random initial parameters to distribute the initial individuals uniformly in the search space. However, sometimes it may be favorable to initialize chromosomes with heuristically determined parameters.

Afterwards, for each non-evaluated individual in the population the following is done (step 1):

- Step 1.1: The individual's chromosome instances are applied to the *Analysis Engine* and the system is analyzed.
- Step 1.2 + 1.3: Each *Objective* requests necessary system properties to calculate its fitness value.
- Step 1.4: The individual is annotated with the calculated fitness values.

Once all individuals have been analyzed they are communicated along with their fitness values to the *Optimization Controller* (step 2) that forwards this information to the *Evolutionary Optimizer* (step 3). Based on the fitness values the *Evolutionary Optimizer* creates two lists, a list of individuals selected for deletion and a list of individuals selected for variation, and sends them back to the *Optimization Controller* (step 4). Afterwards the *Optimization Controller* manipulates the population. First, individuals selected for deletion are removed from the population (step 5). Second, individuals selected for variation are used to create new individuals through recombination (i.e. mutation and crossover, step 6). Finally, all created individuals are added to the population (step 7). This completes the processing of one generation, and the whole loop begins again.

Note that after each completed generation the user has the possibility to modify the search space (steps 8 and 9). This subject is discussed in the following section.

## 3.5 User-controlled design space exploration

In Section 3.3 it was discussed how component interactions and dependencies can be optimized using traffic shaping. Traffic shaping improves system behavior by weakening performance dependencies between components and reducing the global impact of transient load peaks. Consequently, the system becomes more predictable, and thus easier to optimize and explore. However, traffic shaping might not be sufficient as control mechanism for efficient design space exploration. Especially when facing large systems with many parameters, design space exploration can hardly cover the complete search space in adequate time, even with efficient stochastic search techniques. Consequently, it is crucial to find appropriate sub-search spaces containing good solutions.

As already discussed in Section 2.4.1, the idea of restricting the search space to speed up exploration is nothing new, and some previous approaches contain mechanisms to do so. Common techniques try to

automatically partition the search space into (independent) parts and perform hierarchical exploration, i.e. local exploration on single components and subsequent recombination of locally Pareto-optimal solutions to hopefully good global solutions [37, 2]. Limitations of such approaches include that the search space might contain only few independencies, and that it might be difficult to identify these without further aid. More precisely, to safely identify parameter independencies in complex systems exploration techniques are needed. This directly leads to a circular reasoning, exploration needs information that shall be provided by exploration. Consequently, dynamic parameter dependencies are often heuristically ignored [108], which can easily lead to the incapacity of the underlying exploration algorithm to find good solutions to the optimization problem.

The design space exploration framework presented in this chapter pursues another strategy to increase exploration efficiency. Instead of performing closed automated exploration over all system parameters or taking a-priori heuristic assumption about the search space structure, the control over the search process is transferred to the user. Thereby, the exploration concept consists of performing several successive partial explorations with search space modifications after every step in reaction to previously obtained results. Using this iterative exploration approach, interesting design sub-spaces, worthy to be searched in-depth or even completely, can be identified efficiently. Note that apart from efficiency reasons, another important motivation to integrate mechanisms for user-controlled partial explorations into the proposed framework is that it shall be applicable to modern parallel design flows with shared responsibilities (e.g. OEM-supplier dependencies) and iterative refinements (see discussion in Section 2.4.3).

There are two possibilities to dynamically modify the search space during exploration:

1. Adding or removing restrictions on parameters represented by chromosomes already included into the search space. This possibility corresponds to step 8 in Figure 3.5.

2. Extending (restricting) the search space by adding (removing) chromosomes to (from) the search space. This possibility corresponds to step 9 in Figure 3.5.

An important precondition to enable iterative design space exploration is that search space modifications during exploration can be performed without losing results from previous iterations. Therefore, the following operations are performed in case of dynamic search space modifications:

First, exploration is paused once the processing of the current generation has been finished. Afterwards, all individuals contained in the population are adapted to account for the search space modification. Depending on the type of modification different measures must be taken:

1. Adding parameter restrictions to chromosomes already included in the search space: the parameter restrictions are added to the corresponding chromosomes and the repair algorithms are applied. All individuals with at least one modified chromosome are marked as *not analyzed* and their fitness values are set to *undefined*.

2. Removing parameter restrictions from chromosomes already included in the search space: the parameter restrictions are removed from the corresponding chromosomes. No further adaptations are necessary.

3. Adding new chromosomes to the search space: chromosome instances are added to each individual in the population. By default, the new chromosome instances are initialized with (valid) random parameters. All individuals are marked as *not analyzed* and their fitness values are set to *undefined*.

4. Removing chromosomes from the search space: for each removed chromosome one specific instance must be chosen as common basis for further exploration. The concrete parameter values represented by the selected chromosome instances are applied to the analysis engine and added to the immutable search space part. All instances of the removed chromosomes are deleted from the individuals in the population. All individuals are marked as *not analyzed* and their fitness values are set to *undefined*.

Once the search space modification has been completed, all individuals that are marked as not analyzed are evaluated and their fitness values are recalculated. Afterwards, the exploration is resumed with the modified search space.

Figure 3.6 shows an example exploration with two search space modifications.

The search space of the example exploration consists of nine chromosomes $C1$ to $C9$. However, the first exploration step is performed considering only $C2$, $C5$, and $C8$. Let us assume that after some generations the designer observes that the quality of obtained results is unsatisfactory and that fitness improvement is stagnating. In such a case, it makes sense to extend the search space. Accordingly, the designer adds additional parameters represented by $C1$, $C3$, and $C7$ to the search space, and starts a second exploration step. After exploring the

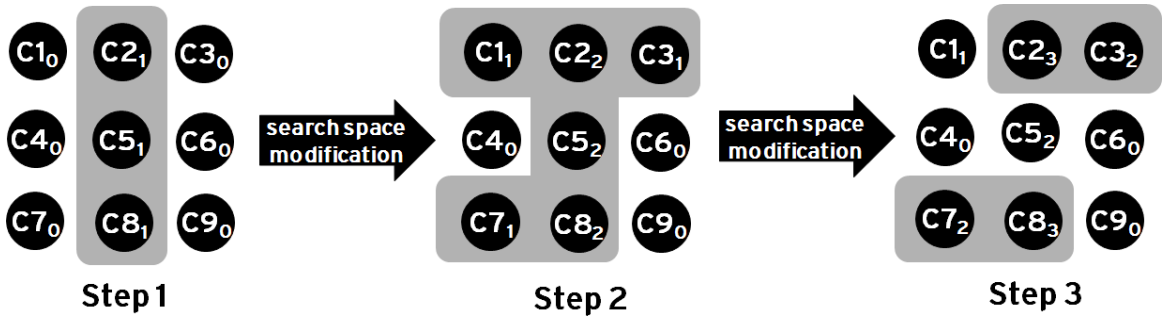



Fig. 3.6: User-controlled design space exploration including nine chromosomes with search space modifications. Increasing indexes indicate potential modifications, i.e. the discovery of different chromosome instances as solution alternatives, between two exploration steps.

extended search space for some time, the designer examines the obtained results and observes that many obtained design alternatives show only little differences with respect to the search space part represented by $C1$ and $C5$. Consequently, the designer removes $C1$ and $C5$ from the search space and starts a third, more focused, exploration step.

Of course, the search space modifications performed in the above described exploration example are heuristic and based on partial knowledge of the fitness landscape. However, search space restrictions can always be reversed if exploration results are unsatisfactory.

Manual search space manipulation on a coarse-grain level, i.e. based on chromosomes representing parameters of whole system components, is practical and feasible. However, more fine-grain search space manipulation is tedious and can only hardly be performed manually. Fine-grain in this context means defining parameter restrictions for specific chromosomes like, for instance, priority precedence relations on priority-scheduled resources. For this reason, an extension of the proposed exploration framework for automated search space modification will be presented in Section 3.9. The application of this extension to automated search space modification for priority scheduled resources is discussed in Section 3.10.

## 3.6 Chromosomes for timing and performance exploration

In this section we introduce chromosomes that can be used in conjunction with the proposed exploration framework to optimize timing and performance of distributed embedded real-time systems. Three exemplary chromosomes for different search space parts are discussed: the priority chromosome for priority optimization on priority scheduled resources (Section 3.6.1), the TDMA chromosome for time slot and turn-

length optimization on TDMA scheduled resources (Section 3.6.2), and the shaper chromosome for optimizing interactions and performance dependencies between functionally dependent system components using $d^-$-EAFs (Section 3.6.3). For details about the underlying performance model refer to Section 2.2.

Note that timing and performance optimization represent only one application domain for the proposed exploration framework. In Chapter 4 its application to multi-dimensional sensitivity analysis and robustness optimization will be discussed.

## 3.6.1 Priority chromosome

In this section we introduce search space encoding (Section 3.6.1.1) and exploration methods (Sections 3.6.1.3 and 3.6.1.2) that are used by our framework for optimizing priority assignments on priority scheduled resources in complex distributed embedded systems.

### 3.6.1.1 Search space encoding

Priority assignments optimization is a discrete permutation problem. In the context of evolutionary optimization such permutation problems are well studied, and thus efficient coding techniques and variation operators achieving good optimization results are known (for a small overview see [129]).

There exist several exact models of evolutionary algorithms based on binary string representations. However, the proposed chromosome encodes priority assignments directly as permutation. Accordingly, the priority assignment on a resource is encoded as a list containing the relative priority ordering (in descending order) of the mapped tasks (Definition 3.6.1).

**Def. 3.6.1 (Priority Assignment $V_\mathcal{R}$)** The priorities of $n$ tasks $t_1, \ldots, t_n$ mapped on resource $\mathcal{R}$ are encoded as $n$-dimensional vector $V_\mathcal{R}$, where $V_\mathcal{R}[i] \in [1, n]$ corresponds to the index of the task with the $i$-th highest priority.
The function $Prio(i, V_\mathcal{R})$ returns the priority of $t_i$ in the priority assignment $V_\mathcal{R}$, with 1 corresponding to the highest priority.

Example: the priority vector $V_\mathcal{R} = (2, 3, 1, 4)$ represents the following priority assignment: $t_2 > t_3 > t_1 > t_4$. Correspondingly, the priority of $t_1$ is $Prio(1, V_\mathcal{R}) = 3$.

Starting from a random set of priority assignments, the ordering problem given by the permutation encoding is solved by using a simple mu-

tation operator and several crossover operators from literature. These operators are introduced in the following sections.

#### 3.6.1.2 Crossover operators

In this section two standard crossover operators used by evolutionary algorithms for solving scheduling related permutation problems are introduced: *Uniform Order-Based Crossover (UX)* [109], and *Precedence Preservative Crossover (PPX)* [10]. Both crossover operators are applied with the same probability during exploration. In experimental results these operators turned out to be effective in solving priority assignment problems in complex distributed embedded systems. Note that this is not self-evident, since ordering problems can be deceptive and might mislead evolutionary algorithms. For a discussion of how ordering problems can be deceptive refer to [58].

**Uniform Order-Based Crossover (UX).** *UX* [109] combines order information from two parent priority assignments to create one offspring. The offspring is created in two steps. First, tasks at randomly selected crossover points are directly copied from the first parent. Second, the remaining tasks are filled into the empty positions in order of appearance in the second parent. Example:

| | | | | | | |
|---|---|---|---|---|---|---|
| Parent 1 | : 2 | 1 | 5 | 4 | 3 | 6 |
| Parent 2 | : 3 | 6 | 2 | 5 | 4 | 1 |
| Cross Pts | : * | | * | | * | |
| Offspring | : 2 | 6 | 5 | 4 | 3 | 1 |

*UX* is considered the best crossover operator for scheduling problems. However, it might destroy precedence relation between task priorities that are common in both parents. In the given example, for instance, both parents assign a higher priority to $t_3$ compared to $t_6$. In the generated offspring, however, this relation is destroyed. Since precedence relations represent very important characteristics of the search space structure for priority optimization problems, a second crossover operator preserving precedence relation is utilized by the proposed priority chromosome.

**Precedence Preserving Crossover (PPX).** In contrast to *UX*, *PPX* [10] maintains all common priority precedence relations during crossover. It uses a binary crossover mask to select the order in which tasks are drawn from the parents priority assignments to create the offspring. 0 and 1 indicate that the first and second parent are selected, respectively.

In each drawing step the priority assignment of the selected parent is scanned linearly for the first task which has not yet be drawn. This task is appended to the offspring's priority assignment.

| | | | | | | |
|---|---|---|---|---|---|---|
| Parent 1 | : 2 | 1 | 5 | 4 | 3 | 6 |
| Parent 2 | : 3 | 6 | 2 | 5 | 4 | 1 |
| Mask | : 1 | 0 | 0 | 1 | 1 | 0 |
| Offspring | : 3 | 2 | 1 | 6 | 5 | 4 |

*PPX* ensures that all precedence relations between priorities in the generated offspring come from the same parent. Consequently, precedence relations which are common to both parents are preserved in the offspring's priority assignment.

#### 3.6.1.3 Mutation operator

The crossover operators presented in the previous section implement strategies to preserve interesting commonalities of the parent individuals during variation. In other words, they filter out interesting search space structures and lead to the convergence of the evolutionary search.

Of course, it is possible that the variety of the population is insufficient to find good solutions only by using crossover operators. Additionally, the exploration may get stuck in local optima, without the possibility to reach globally better solutions. Therefore, mutation operators are used to enable the evolutionary algorithm to break out of local optima, and to reach parts of the search space not yet explored.

The mutation operator used by the proposed priority chromosome switches the priorities of two randomly selected tasks.

### 3.6.2 TDMA chromosome

The search space of all time slot assignments for tasks mapped on TDMA scheduled resources is very large, even if the arithmetic precision is limited. Turn-length variation, which is often necessary to find good solutions, adds another search dimension. Since it is unrealistic to try all possible time slot assignments and turn-lengths, a good strategy to walk through the search space is indispensable.

In the following arithmetic real-coded variation operators tailored for time slot and turn optimization on TDMA scheduled resources are introduced. Thereby, the optimization strategy is split into two aspects: optimizing the admitted loads of the mapped tasks as well as optimizing the turn-length. Both factors together define the quality of a TDMA time slot assignment.

According to the two problem aspects four variation operators are proposed. One crossover and one mutation operator varying only admitted

loads while ensuring that turn-lengths remain constant, as well as one crossover and one mutation operator varying only turn-lengths without modifying admitted loads. The crossover and mutation operators are described in Sections 3.6.2.3 and 3.6.2.4, respectively.

#### 3.6.2.1 Search space encoding

One approach frequently used for continuous optimization problems like TDMA time slot optimization in the context of evolutionary algorithms, is discretizing the desired search space into a power of 2 and using a binary string representation with binary variation operators, like i.e. single-point crossover.

However, for the realization of the TDMA chromosome it was decided to use a real number encoding of the problem variables and arithmetic variation operators to guide the search space. This is suitable for the given problem because it gives much more control over the generated alternatives, and allows to implement problem-aware variation operators guaranteeing the validity of generated time slot assignments.

In the following it is assumed that the time slot assignments for a TDMA scheduled resource $R$ with the mapped tasks $T_0, \ldots, T_{k-1}$ is subject to optimization. Thereby, the time-slots of the involved tasks are represented by the real numbers $slot_0, \ldots, slot_{k-1}$. Accordingly, the TDMA turn-length is implicitly given by the time slots sum: $turn = \sum_{i=0}^{k-1} slot_i$. Note that during exploration the maximum arithmetic precision is bounded (which is not reflected in the given algorithms). Typical search precisions range between allowing only integer values and rounding to two digits after the decimal point.

#### 3.6.2.2 Creation of the initial population

For the creation of the initial population an initial TDMA turn-length $turn_{init}$ is specified. Note that choosing a sub-optimal initial turn-length for the initial population is uncritical, since the proposed variation operators are capable of adapting the turn-length in the course of optimization. Nevertheless, by choosing a good initial turn-length the chromosome converges faster towards the solution space.

In the following we refer to a specific time slot assignment as *individual* and to the set of individuals maintained during evolutionary optimization as *population*.

Let the worst-case execution time of $T_i$ be denoted by $WCET_i$, and its average arrival rate (i.e. for instance the period in case of periodic tasks) by $rate_i$. In order to create only valid (i.e. resource not overloaded, etc.) initial individuals, it has to be ensured that the maximum load $load_{max;i}$ of each task $T_i$ does not exceed its admitted load $load_{adm;i}$:

$$load_{adm;i} \geq load_{max;i} \quad \Leftrightarrow \quad \frac{slot_i}{turn_{init}} \geq \frac{WCET_i}{rate_i}$$
$$\Leftrightarrow \quad slot_i \geq \frac{WCET_i}{rate_i} * turn_{init}$$

This implies for $T_i$ a minimum time slot

$$slot_{min;i} = \frac{WCET_i}{rate_i} * turn_{init}.$$

Algorithm 1 is used to create the initial population. It uniformly generates individuals in the search space of all valid time slot assignments with turn-length $turn_{init}$. To do so, it randomly distributes the initial turn to the tasks $T_0, \ldots, T_{k-1}$, while respecting the above mentioned minimum time slot lengths to prevent the creation of invalid individuals. Note that the algorithm assumes that the maximum load of the scheduled task set does not exceed 100%. Accordingly, a trivial load check needs to be performed before the TDMA time slot optimization is started.

**Algorithm 1** Create valid initial individual

**Require:** initial turn-length $turn_{init}$ and minimum time slots $slot_{min;0}, \ldots, slot_{min;k-1}$ for the tasks $T_0, \ldots, T_{k-1}$

**Ensure:** random valid time slot assignment $slot_0, \ldots, slot_{k-1}$ for the tasks $T_0, \ldots, T_{k-1}$

1: $free \leftarrow turn_{init}$
2: $set \leftarrow \{0, 1, \ldots, k-1\}$
3: **while** $(set \neq \emptyset)$ **do**
4: choose random $r \in set$
5: $set \leftarrow set \setminus r$
6: **if** $(set = \emptyset)$ **then**
7: $slot_r = free$
8: **else**
9: $slot_{max} \leftarrow free - \sum_{x \in set} slot_{min;x}$
10: $slot_r \leftarrow \text{RANDOM}(slot_{min;r}, slot_{max})$
11: $free \leftarrow free - slot_r$
12: **end if**
13: **end while**

#### 3.6.2.3 Crossover operators

In this section crossover operators for optimizing TDMA time slot assignments are described. They implement a heuristic binary search related optimization strategy to ensure convergence of the search process towards solutions lying "between" individuals considered by the evolutionary algorithm.

**Admitted load crossover.** Algorithm 2 describes the crossover operator varying only admitted loads while ensuring that turn-lengths remain constant. It takes two parent individuals as input and creates two offsprings.

First, the admitted task loads of the generated offsprings are placed evenly (i.e. at $\frac{1}{3}$ and $\frac{2}{3}$) in the admitted load intervals defined by the two parents. Afterwards, the time slots of offsprings 1 and 2 are calculated to fit the turn-lengths imposed by parent 1 and 2, respectively. Note that the admitted load crossover operator automatically generates offsprings that are valid (in terms of the minimum admitted load condition) if the parent individuals are valid.

This is easy to understand. Each time slot calculated for a given task $T_i$ in any offspring $o$ corresponds to an admitted load $load_{adm;o;i}$ that is larger or equal than the minimum admitted load

$$\min(load_{adm;p_1;i}, load_{adm;p_2;i})$$

defined by the corresponding parents $p_1$ and $p_2$. Consequently, since all parents are assumed to be valid, also all generated offsprings automatically satisfy the minimum admitted load condition.

Figure 3.7a gives an example for the admitted load crossover operator. The two offsprings are created by modifying the time slots of the three considered tasks $T0$, $T1$, and $T2$ according to the above described algorithm. Note that the time slot of task $T2$ is not modified for both offsprings. The reason is that $T2$ possesses an admitted load of 30% in both parents.

**Turn crossover.** The crossover operator varying the turn-length is described by Algorithm 3. Given two parent individuals it first calculates the average turn-length. Afterwards, offspring 1 and 2 are created by adapting the time slots of parent 1 and 2 to fit the average turn-length without modifying the admitted loads.

Since the turn crossover operator does not modify the admitted loads of the involved tasks, the validity of offsprings that are generated from

valid parents is automatically ensured. Figure 3.7b visualizes the turn crossover operator by means of an example.

**Algorithm 2** Admitted load crossover

**Require:** time slots $slot_{p_1;0}, \ldots, slot_{p_1;k-1}$ of parent $p_1$ and time slots $slot_{p_2;0}, \ldots, slot_{p_2;k-1}$ of parent $p_2$

**Ensure:** time slots $slot_{o_1;0}, \ldots, slot_{o_1;k-1}$ of offspring $o_1$ and time slots $slot_{o_2;0}, \ldots, slot_{o_2;k-1}$ of offspring $o_2$

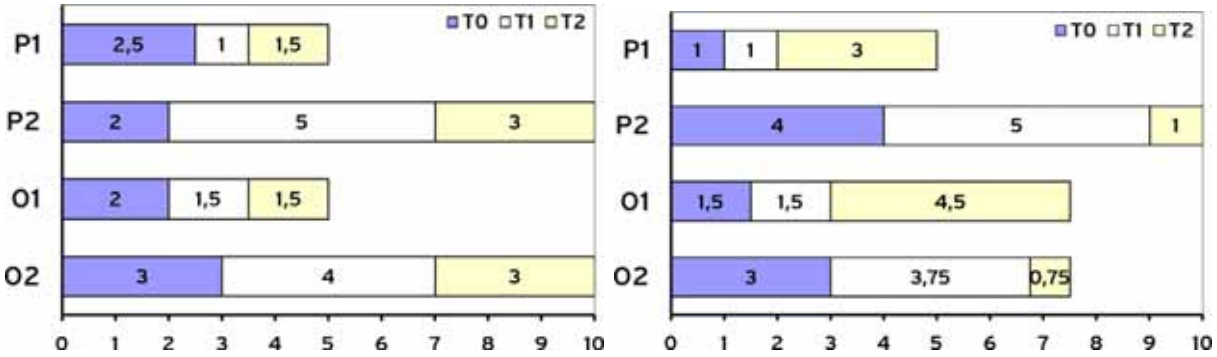
1: $turn_{p_1} \leftarrow \sum_{i=0}^{k-1} slot_{p_1;i}$
2: $turn_{p_2} \leftarrow \sum_{i=0}^{k-1} slot_{p_2;i}$
3: **for** i $\leftarrow$ 0 **to** k-1 **do**
4: $\quad load_{adm;p_1;i} \leftarrow slot_{p_1;i}/turn_{p_1}$
5: $\quad load_{adm;p_2;i} \leftarrow slot_{p_2;i}/turn_{p_2}$
6: $\quad difference \leftarrow \mid load_{adm;p_1;i} - load_{adm;p_2;i} \mid$
7: $\quad$ **if** $load_{adm;p_1;i} < load_{adm;p_2;i}$ **then**
8: $\quad\quad slot_{o_1;i} \leftarrow (load_{adm;p_1;i} + difference/3) * turn_{p_1}$
9: $\quad\quad slot_{o_2;i} \leftarrow (load_{adm;p_2;i} - difference/3) * turn_{p_2}$
10: $\quad$ **else**
11: $\quad\quad slot_{o_1;i} \leftarrow (load_{adm;p_1;i} - difference/3) * turn_{p_1}$
12: $\quad\quad slot_{o_2;i} \leftarrow (load_{adm;p_2;i} + difference/3) * turn_{p_2}$
13: $\quad$ **end if**
14: **end for**

**Algorithm 3** Turn crossover

**Require:** time slots $slot_{p_1;0}, \ldots, slot_{p_1;k-1}$ of parent $p_1$ and time slots $slot_{p_2;0}, \ldots, slot_{p_2;k-1}$ of parent $p_2$

**Ensure:** time slots $slot_{o_1;0}, \ldots, slot_{o_1;k-1}$ of offspring $o_1$ and time slots $slot_{o_2;0}, \ldots, slot_{o_2;k-1}$ of offspring $o_2$

1: $turn_{p_1} \leftarrow \sum_{i=0}^{k-1} slot_{p_1;i}$
2: $turn_{p_2} \leftarrow \sum_{i=0}^{k-1} slot_{p_2;i}$
3: $turn_{new} = (turn_{p_1} + turn_{p_2})/2;$
4: **for** i $\leftarrow$ 0 **to** k-1 **do**
5: $\quad slot_{o_1;i} = slot_{p_1;i}/turn_{p_1} * turn_{new};$
6: $\quad slot_{o_2;i} = slot_{p_2;i}/turn_{p_2} * turn_{new};$
7: **end for**




(a) Admitted load crossover (b) Turn crossover

Fig. 3.7: Crossover operators of the TDMA chromosome

#### 3.6.2.4 Mutation operators

The crossover operators presented in the previous section lead to the discovery of (locally) optimal solutions lying "between" individuals considered during evolutionary exploration. Of course, it is possible that the variety of the initial population is insufficient to find good solutions only by using these crossover operators. Additionally, the exploration may get stuck in local optima. For this reason, two mutation operators enabling the evolutionary search to break out of local optima and to reach new parts of the search space are introduced.

**Mutate admitted load.** The mutation operator varying only admitted loads while ensuring that turn-lengths remain constant is described by Algorithm 4. It takes one parent individual as input and creates one offspring. After initialization, two tasks are randomly chosen. The first task gives a part of its disposable time slot, i.e. the time slot part it can dispense without overloading the resource, to the second task. Thereby, the percentage of the disposable time slot that is dispensed is randomly chosen in the interval $]0, d_{max} \leq 1]$, where $d_{max}$ is configurable. Figure 3.8a visualizes the admitted load mutation operator by means of an example.

**Mutate turn.** Algorithm 5 describes the mutation operator varying the turn-length. First, the target turn-length is chosen by increasing or decreasing the turn-length of the parent individual by a percentage randomly chosen in the interval $]0, d_{max} \leq 1]$, where $d_{max}$ is configurable. The offspring's time slots are then calculated to fit the target turn-length without altering the admitted loads imposed by the parent's time slot assignment. Figure 3.8b visualizes the turn mutation operator for a turn-length reduction of 20%.

**Algorithm 4** Mutate admitted load

**Require:** time slots $slot_{p;0}, \ldots, slot_{p;k-1}$ of parent $p$ and maximum % of disposable time slot dispensed $d_{max}$
**Ensure:** time slots $slot_{o;0}, \ldots, slot_{o;k-1}$ of offspring $o$

$turn_p \leftarrow \sum_{i=0}^{k-1} slot_{p;i}$
**for** i ← 0 **to** k-1 **do**
    $slot_{o;i} \leftarrow slot_{p;i}$
**end for**
**repeat**
    $r_1 \leftarrow \text{RANDOM}(0, k-1)$
    $r_2 \leftarrow \text{RANDOM}(0, k-1)$
**until** $r_1 \neq r_2$
$slot_{disposable} = slot_{p;r_1} - load_{max;r_1} * turn_p$
$d_{applied} \leftarrow \text{RANDOM}(0, d_{max})$
$slot_{o;r_1} = slot_{o;r_1} - d_{applied} * slot_{disposable}$
$slot_{o;r_2} = slot_{o;r_2} + d_{applied} * slot_{disposable}$

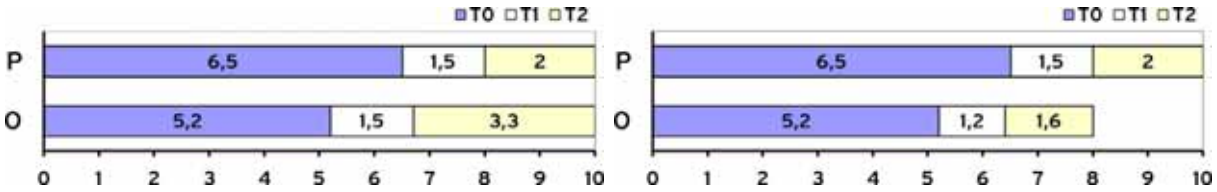
**Algorithm 5** Mutate turn

**Require:** time slots $slot_{p;0}, \ldots, slot_{p;k-1}$ of parent $p$ and max. % by which turn is cut or extended $d_{max}$
**Ensure:** time slots $slot_{o;0}, \ldots, slot_{o;k-1}$ of offspring $o$

$turn_p \leftarrow \sum_{i=0}^{k-1} slot_{p;i}$
$bool \leftarrow \text{RANDOM}(true, false)$
$d_{applied} \leftarrow \text{RANDOM}(0, d_{max})$
**if** *bool* **then**
    $turn_{new} \leftarrow turn_p + d_{applied} * turn_p$
**else**
    $turn_{new} \leftarrow turn_p - d_{applied} * turn_p$
**end if**
**for** i ← 0 **to** k-1 **do**
    $slot_{o;i} \leftarrow slot_{p;i}/turn_p * turn_{new}$;
**end for**

### 3.6.3 Traffic shaping chromosome

Traffic shaping chromosomes represent $d^-$-EAFs performing traffic shaping on event streams connecting functionally dependent components. Like demonstrated in Section 3.3.2, traffic shaping can be efficiently used to control component dependencies and interactions, and




(a) Admitted load mutation (b) Turn mutation

Fig. 3.8: Mutation operators of the TDMA chromosome

hence represents a valuable tool for performance optimization in the context of complex distributed embedded systems.

#### 3.6.3.1 Search space encoding

The traffic shaping chromosome is realized using a real number representation of the minimum distance $d^-$ and arithmetic real-coded variation operators. During exploration the range of allowed $d^-$ values is bounded by a configurable interval $[d_{min}, d_{max}]$. Additionally, the arithmetic precision can be configured. Typical search precisions range between allowing only integer values and rounding to two digits after the decimal point.

For convenience reasons, the search interval bounds can be configured using metrics that are more comprehensible than the rather abstract $d^-$ values:

- Possibility 1: Specifying the maximum buffering delay that may be produced by the traffic shaper is more comprehensible for timing constrained systems.
- Possibility 2: Specifying the maximum buffer size of the traffic shaper is more comprehensible if global buffer budgets have to be respected.

Note that specified maximum buffering delays and maximum buffer sizes can be easily translated into corresponding $d^-$ values for the search intervals that are internally used for optimization (see Section 3.3.1).

#### 3.6.3.2 Variation operators

The search strategy of the variation operators is similar to that of the TDMA chromosome, with the difference that only one problem dimension needs to be considered. Starting from random $d^-$ values, the search is conducted by one crossover and one mutation operator.

The crossover operator takes as input two parent individuals and creates two offsprings. The $d^-$ values of the first and the second offspring are placed at $\frac{1}{3}$ and $\frac{2}{3}$, respectively, in the $d^-$ interval defined by the

two parents. This strategy leads to the discovery of of (locally) optimal $d^-$ values lying "between" individuals considered during evolutionary exploration.

The purpose of the mutation operator is to prevent the search getting stuck in local optima and to reach parts of the search space that are inaccessible by only using the crossover operator. It creates one offspring by increasing or decreasing the $d^-$ value of the parent individual by a percentage randomly chosen in the interval $]0, percentage_{max}]$, where $percentage_{max}$ is configurable.

## 3.7 Optimization objectives for timing and performance exploration

In this section several example metrics for optimizing timing and performance of distributed hard real-time systems are proposed (Section 3.7.1). Additionally, the concept of fitness landscape partitioning is discussed (Section 3.7.2).

### 3.7.1 Example metrics

In the following example metrics that can be used by the proposed exploration framework for timing and performance optimization are introduced using the following notations:

| | |
|---|---|
| $R_i$ - | worst-case response time of the i-th task or<br>worst-case end-to-end latency along the i-th path |
| $D_i$ - | deadline of the i-th task or path |
| $\omega_i$ - | constant weight $> 0$ of the i-th task or path |
| $k$ - | number of tasks or<br>number of constrained tasks/paths in the system |

Note that in the following timing properties are used as an example. However, corresponding metrics can easily be derived to optimize jitters, buffer requirements, power consumption, etc.

A basic metric for expressing the quality of given individuals with respect to timing is the weighted sum of completion times:

$$\sum_{i=1}^{k} \omega_i * R_i$$

Even though this metric can be used to minimize task response times or end-to-end latencies along paths, its practical relevance for systems with timing constraints is limited. In such cases, metrics taking deadlines into account are more appropriate.

The lateness of a task (a path) is defined as the amount of time by that it misses its deadline. Consequently, negative values denote that

the task (the path) completes before the expiration of its deadline. In the case of constrained systems, lateness can be used to define expressive global metrics for the timing properties of given individuals. Following example metric can be used to minimize the (weighted) average lateness:

$$\sum_{i=1}^{k} \omega_i * (R_i - D_i)$$

The given metric expresses the average timing behavior of individuals with respect to timing constraints. However, since met deadlines compensate linearly for missed deadlines, the metric might mislead the evolutionary search, and hence hinder the discovery of individuals fulfilling all timing constraints. Consequently, metrics with higher penalties for missed deadline and less rewards for met deadlines can be more appropriate for systems with hard real-time constraints. Following metric penalizes violated deadlines exponentially, and can be used to optimize systems with hard real-time constraints:

$$\sum_{i=1}^{k} c_i^{R_i - D_i},\ c_i > 1 \text{ constant}$$

A further refinement can be obtained by expressing lateness *relatively* to the deadline. This is more appropriate for systems containing timing constraints with significant differences in absolute values. Following metric penalizes relative deadline misses exponentially:

$$\sum_{i=1}^{k} c_i^{\frac{R_i - D_i}{D_i} * 100},\ c_i > 1 \text{ constant}$$

The above discussed metrics integrate timing properties of given individuals into single values. Using these metrics is appropriate to quickly find good functioning system parameter configurations. However, integrated metrics do not provide information about trade-offs between several timing properties. For this purpose the proposed metrics need to be applied on subsets of tasks or paths (including single ones). Together with the Pareto-optimization capabilities of the proposed exploration framework, system optimization can be carried out considering multiple sets of significant timing properties to discover interesting design trade-offs.

Note that the discussed metrics only represent examples of how timing properties and constraints can be used for optimization purposes. Clearly, depending on the use-case different metrics might be more ap-

propriate. Examples include metrics based on mean operators (arithmetic, geometric, harmonic, etc.).

### 3.7.2 Partitioning of the fitness landscape

The primary criterion when optimizing systems with hard-real time constraints is feasibility, i.e. adherence to performance constraints. Only after this superordinate categorization has been accomplished a more fine-granular differentiation with respect to given optimization objectives makes sense. Accordingly, it must be ensured that feasible individuals are always assigned better fitness values than infeasible individuals.

However, using the metrics proposed in the previous section allowing the integration of two or more constraints into single fitness values, this might not always be guaranteed, even if exponential penalties are applied for constraint violations. For instance, infeasible individuals marginally violating single constraints might be assigned better fitness values than feasible individuals marginally fulfilling all constraints.

This issue is visualized in Figure 3.9a for an optimization objective subject to minimization. As can be seen, the fitness landscape of the underlying optimization problem does not precisely separate feasible from infeasible individuals, i.e. there exist infeasible individuals that are assigned better fitness values than feasible individuals. Obviously, during exploration such inconsistent fitness landscape might mislead the evolutionary search.

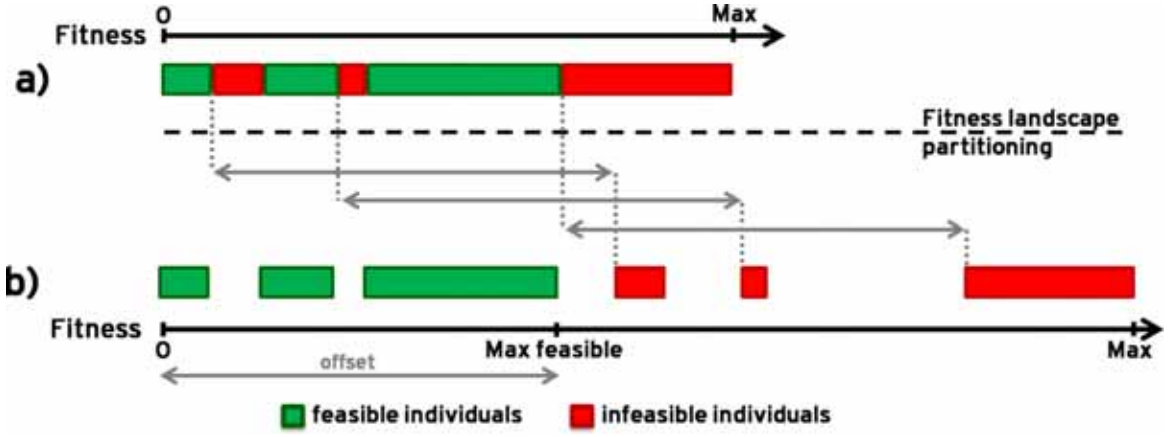



Fig. 3.9: Partitioning of the fitness landscape for exploring and optimizing hard real-time systems. Green and red sections indicate regions of the fitness landscape containing feasible and infeasible individuals, respectively.

In order to safely separate feasible and infeasible individuals the fitness landscape must be partitioned as visualized in Figure 3.9b. Fitness landscape partitioning consists in penalizing infeasible individuals with

an *offset* ensuring that feasible individuals are always assigned better, i.e. in this case smaller, fitness values. Note that in Figure 3.9b it was assumed that the maximum feasible fitness value is known. Obviously, this value can be safely chosen as offset. However, if the maximum feasible fitness cannot be determined, a sufficiently large offset needs to be chosen manually to safely distinguish feasible from infeasible individuals.

## 3.8 Case study

In this section the applicability of the proposed framework and user-controlled design space exploration is demonstrated by means of a small but realistic example. The studied multi-processor system is presented in Section 3.8.1, and the design space exploration process is discussed in Section 3.8.2.

### 3.8.1 Multi-processor platform example

Figure 3.10 visualized a multi-processor platform consisting of a micro-controller ($uC$), a RISC[1] CPU ($RISC$), and dedicated hardware ($HW$), all connected via a bus ($BUS$). The $HW$ acts as interface to an external physical system. It runs one task ($sys\_if$) that issues actuator commands to the physical system and collects routine sensor readings. $sys\_if$ is controlled by the controller task $ctrl$ that evaluates sensor data and calculates necessary actuator commands. $ctrl$ is activated by a periodic timer ($tmr$) and by the arrival of new sensor data (AND-activation in a cycle).

The physical system is additionally monitored by 3 smart sensors ($s_1$ - $s_3$) sporadically producing data as reaction to irregular system events. This data is registered by the OR-activated monitor task ($mon$) on the $uC$ that decides how to update the control algorithm. This information is sent to the task $upd$ on the $RISC\ CPU$ writing the updated controller parameters into shared memory.

The $RISC\ CPU$ additionally executes the task ($fltr$) that filters the data stream arriving at input $sig\_in$, and sends the processed data via output $sig\_out$. All communication (with the exception of shared-memory on the $RISC\ CPU$) is carried out by the communication tasks $c1$ - $c5$ over the on-chip $BUS$.

Computation and communication tasks have the core execution and communication times listed in Table 3.3. The event models at the system inputs are specified in Table 3.4. In order to function correctly, the system has to satisfy the path latency constraints and the maximum

[1] Reduced Instruction Set Computing.

jitter constraint at *sig_out* listed in Tables 3.5a and 3.5b, respectively. In the following it is assumed that the *RISC CPU* as well as the *BUS* are scheduled according to the static priority preemptive policy.

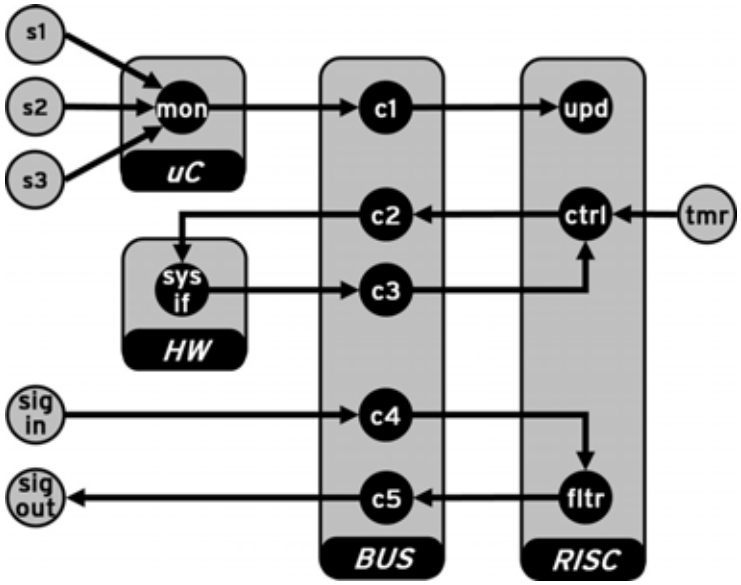



Fig. 3.10: Multi-processor example system

| computation task | core execution time |
|---|---|
| *mon* | [10,12] |
| *sys_if* | [15,15] |
| *fltr* | [12,15] |
| *upd* | [5,5] |
| *ctrl* | [20,23] |
| **communication task** | **core communication time** |
| *c1* | [4,4] |
| *c2* | [4,4] |
| *c3* | [4,4] |
| *c4* | [8,8] |
| *c5* | [4,4] |

Table 3.3: Core execution and communication times

| input | event model |
|---|---|
| $s_1$ | sporadic, $\mathcal{P}_{s_1} = 1000$ |
| $s_2$ | sporadic, $\mathcal{P}_{s_2} = 750$ |
| $s_3$ | sporadic, $\mathcal{P}_{s_3} = 600$ |
| *sig_in* | periodic, $\mathcal{P}_{in} = 60$ |
| *tmr* | periodic, $\mathcal{P}_{tmr} = 70$ |

Table 3.4: Input event models

| constraint # | path | maximum latency |
|---|---|---|
| 1 | $s_i \rightarrow upd$ | 70 |
| 2 | $sig\_in \rightarrow sig\_out$ | 60 |
| 3 | cycle (e.g. $ctrl \rightarrow ctrl$) | 140 |

(a) Path latency constraints

| constraint # | output | event model jitter |
|---|---|---|
| 4 | $sig\_out$ | $\mathcal{J}_{sig\_out,max} = 22$ |

(b) Maximum jitter constraint

Table 3.5: System timing constraints

### 3.8.2 Exploring the example system

In this section, the given multi-processor example is explored in several exploration steps. First, local optimization of the *BUS* is performed altering only the priorities of the communication channels. Afterwards, the search space is extended by allowing traffic shaping with $d^-$-EAFs at reasonable positions in the system. During these two exploration steps, the following priority assignment for the tasks running on the *RISC CPU* is assumed: $upd > fltr > ctrl$. In the final exploration step, the priority assignment on the *RISC CPU* is included into the search space. Optimization objectives during all exploration steps are the minimization of the path latencies ($s_i \rightarrow upd$ and $sig\_in \rightarrow sig\_out$), the cycle latency ($ctrl \rightarrow ctrl$), and the output jitter at $sig\_out$ ($\mathcal{J}_{sig\_out}$).

**Optimizing the BUS.** The first exploration step consists in local optimization of the *BUS*. Although there are only five communication channels on the *BUS*, it is not intuitive which priority assignments lead to systems meeting all timing constraints. Local exploration of the *BUS* will provide a first feeling about the systems behavior, and thus a deeper understanding of its performance dependencies. Table 3.6 shows the obtained solutions.

| # | *BUS* tasks | *RISC* tasks | $s_i \rightarrow upd$ | $sig\_in \rightarrow sig\_out$ | $ctrl \rightarrow ctrl$ | $\mathcal{J}_{sig\_out}$ |
|---|---|---|---|---|---|---|
| 1 | *c5,c4,c1,c2,c3* | *upd,fltr,ctrl* | 55 | 42 | 120 | 18 |
| 2 | *c5,c4,c2,c1,c3* | *upd,fltr,ctrl* | 59 | 42 | 112 | 18 |
| 3 | *c5,c2,c4,c1,c3* | *upd,fltr,ctrl* | 59 | 46 | 108 | 22 |
| 4 | *c4,c5,c2,c3,c1* | *upd,fltr,ctrl* | 63 | 42 | 96 | 18 |
| 5 | *c5,c2,c4,c3,c1* | *upd,fltr,ctrl* | 63 | 46 | 92 | 22 |

Table 3.6: First exploration step: Pareto-optimal solutions obtained through local optimization on the *BUS*

As can be seen, there exist five Pareto-optimal priority assignments for the communication channels on the *BUS* representing different trade-offs between the explored optimization objectives. Furthermore, it can be observed that the channels $c4$ and $c5$ have high priorities, whereas channel $c3$ has the lowest or second lowest priority in all obtained solutions.

**Traffic shaping.** In the second exploration step the optimization potential of selective traffic shaping (see Section 3.3) in the given system is investigated. More precisely, the search-space is extended by a $d^-$-EAF manipulating the minimum distance of successive data packets sent by *mon* over the *BUS*. It makes sense to perform traffic shaping at this location, since the OR-activation of *mon* leads in the worst-case to bursts at its output. That is, if all three sensors trigger at the same time, *mon* will send three packets over the *BUS* with a minimum distance of 10 time units, which corresponds to its best-case core execution time. This transient load peak affects the overall system performance in a negative way. However, a $d^-$-EAF can increase the minimum distance $d^-$ between successive data packets to weaken the global impact of this burst. Exploration over $d^-$ values enforced by the inserted $d^-$-EAF is subject of this exploration step.

In the previous exploration step it was observed that in all obtained solutions the communication channel $c3$ was assigned the lowest or second lowest priority, and even in the latter case the cycle constraint ($ctrl \rightarrow ctrl$) was easily met. This knowledge is exploited in the second exploration step to narrow the search space by fixing the priority of $c3$ to the lowest on the *BUS*. This reduces the number of possible priority assignments from $5! = 120$ to $4! = 24$.

Table 3.7 lists the additional Pareto-optimal solutions obtained using a $d^-$-EAF at the output of *mon* extending the minimum distance of successive data packets to integer values between 11 and 25 time units. Note that feasible parameter configurations that are Pareto-dominated by the results obtained in the first exploration step are not listed.

| # | *BUS* tasks | *RISC* tasks | $d^-$ | $s_i \rightarrow upd$ | $sig\_in \rightarrow sig\_out$ | $ctrl \rightarrow ctrl$ | $\mathcal{J}_{sig\_out}$ |
|---|---|---|---|---|---|---|---|
| 6 | *c1,c5,c4,c2,c3* | *upd,fltr,ctrl* | 13 | 51 | 45 | 120 | 21 |
| 7 | *c1,c5,c4,c2,c3* | *upd,fltr,ctrl* | 14 | 53 | 45 | 116 | 21 |
| 8 | *c1,c5,c4,c2,c3* | *upd,fltr,ctrl* | 16 | 57 | 45 | 112 | 21 |
| 9 | *c5,c1,c4,c2,c3* | *upd,fltr,ctrl* | 17 | 63 | 41 | 112 | 17 |
| 10 | *c1,c5,c4,c2,c3* | *upd,fltr,ctrl* | 20 | 65 | 40 | 104 | 16 |

Table 3.7: Second exploration step: Additional Pareto-optimal solutions obtained by performing traffic shaping at the output of *mon*

As can be observed, traffic shaping at the output of *mon* leads to several new interesting solutions. New priority assignments on the *BUS* are found that, combined with certain $d^-$ values enforced by the inserted $d^-$-EAF, lead to better timing properties for the constrained paths $s_i \rightarrow upd$ and $sig\_in \rightarrow sig\_out$ as well as the jitter constraint $\mathcal{J}_{sig\_out}$.

Only two different priority assignments on the *BUS*, $c5 > c1 > c4 > c2 > c3$ and $c1 > c5 > c4 > c2 > c3$, occur in the solutions listed in Table 3.7. In order to better understand the global impact of growing $d^-$ values enforced by the inserted $d^-$-EAF, a closer look is taken at these two priority assignments. Tables 3.8a and 3.8b visualize the results. Rows containing Pareto-optimal solutions are emphasized.

| $d^-$ | $s_i \rightarrow upd$ | $sig\_in \rightarrow sig\_out$ | $ctrl \rightarrow ctrl$ | $\mathcal{J}_{sig\_out}$ |
|---|---|---|---|---|
| 10 | 49 | 50 | 162 | 26 |
| 11 | 51 | 50 | 162 | 26 |
| 12 | 53 | 46 | 120 | 22 |
| 13 | 55 | 46 | 120 | 22 |
| 14 | 57 | 46 | 116 | 22 |
| 15 | 59 | 46 | 116 | 22 |
| 16 | 61 | 46 | 112 | 22 |
| **17** | **63** | **41** | **112** | **17** |
| 18 | 65 | 41 | 112 | 17 |
| 19 | 67 | 41 | 112 | 17 |
| 20 | 69 | 41 | 104 | 17 |

(a) BUS priorities: $c5 > c1 > c4 > c2 > c3$

| $d^-$ | $s_i \rightarrow upd$ | $sig\_in \rightarrow sig\_out$ | $ctrl \rightarrow ctrl$ | $\mathcal{J}_{sig\_out}$ |
|---|---|---|---|---|
| 10 | 45 | 54 | 162 | 30 |
| 11 | 47 | 54 | 162 | 30 |
| 12 | 49 | 50 | 120 | 26 |
| **13** | **51** | **45** | **120** | **21** |
| **14** | **53** | **45** | **116** | **21** |
| 15 | 55 | 45 | 116 | 21 |
| **16** | **57** | **45** | **112** | **21** |
| 17 | 59 | 45 | 112 | 21 |
| 18 | 61 | 45 | 112 | 21 |
| 19 | 63 | 45 | 112 | 21 |
| **20** | **65** | **40** | **104** | **16** |
| 21 | 67 | 40 | 104 | 16 |
| 22 | 69 | 40 | 104 | 16 |

(b) BUS priorities: $c1 > c5 > c4 > c2 > c3$

Table 3.8: System performance with traffic shaping at the output of *mon*

As can be observed, the end-to-end latency of the constrained path $s_i \rightarrow upd$ is increasing with growing minimum distance $d^-$. This is not surprising, since the inserted $d^-$-EAF is introducing buffering delays for the data packets transmitted by *mon*. The maximum $d^-$ values not leading to path constraint violations for the priority assignments $c5 > c1 > c4 > c2 > c3$ and $c1 > c5 > c4 > c2 > c3$ are 20 and 22, respectively. However, while the $d^-$-EAF leads to increased end-to-end latencies for the path $s_i \rightarrow upd$, the rest of the system is profiting from the weakened burst.

**Including the RISC CPU.** In the third and last exploration step the priority assignment on the *RISC CPU* is included into the search space. Since it was observed in the previous exploration steps that the cycle constraint $ctrl \rightarrow ctrl$ is uncritical, the two lowest priorities on the *BUS* are statically assigned to the communication channels $c2$ and $c3$. Furthermore, the priority of the task *ctrl* is fixed to the lowest on the *RISC CPU*.

Table 3.9 summarizes the new Pareto-optimal parameter configurations found. Note that parameter configurations that are Pareto-dominated by results obtained in the previous exploration steps are not listed.

| # | *BUS* Tasks | *RISC* Tasks | $d^-$ | $s_i \rightarrow upd$ | $sig\_in \rightarrow sig\_out$ | $ctrl \rightarrow ctrl$ | $\mathcal{J}_{sig\_out}$ |
|---|---|---|---|---|---|---|---|
| 11 | *c5,c4,c1,c2,c3* | *fltr,upd,ctrl* | 10 | 70 | 27 | 120 | 3 |
| 12 | *c1,c5,c4,c2,c3* | *fltr,upd,ctrl* | 12 | 64 | 35 | 120 | 11 |
| 13 | *c5,c1,c4,c2,c3* | *fltr,upd,ctrl* | 12 | 68 | 31 | 120 | 7 |
| 14 | *c1,c5,c4,c2,c3* | *fltr,upd,ctrl* | 14 | 68 | 35 | 116 | 11 |

Table 3.9: Third exploration step: Additional Pareto-optimal solutions obtained by including the $RISC\,CPU$ into the search space

The obtained solutions represent new interesting design trade-offs with low jitters at *sig_out*. This quality did not exist in any of the previously obtained system parameter configurations. However, the low jitter at *sig_out* is bought with high end-to-end latencies along the constrained path $s_i \rightarrow upd$.

## 3.9 Extension for automated search space modification

In Section 3.5 it was discussed how dynamic search space modifications can be performed without losing previously obtained results. Furthermore, it was explained and demonstrated how this can be exploited to realize an iterative user-controlled exploration approach. However, analyzing exploration results manually at fine-grain level to decide about

search space modifications is time consuming and tedious, especially for large systems with many parameters.

For this reason, an extension for automated search space modification is developed in this section. The introduced extension allows to integrate chromosome specific automated search space modification strategies into the proposed design space exploration framework. During exploration these strategies dynamically analyze the search space structure, extract advantageous parameter dependencies, and restrict or widen the search space accordingly.

### 3.9.1 Integration into the exploration framework

The function of the extension is visualized in Figure 3.11. It is embedded in the exploration process after the variation process, and modifies the search space exactly once after completion of each generation.

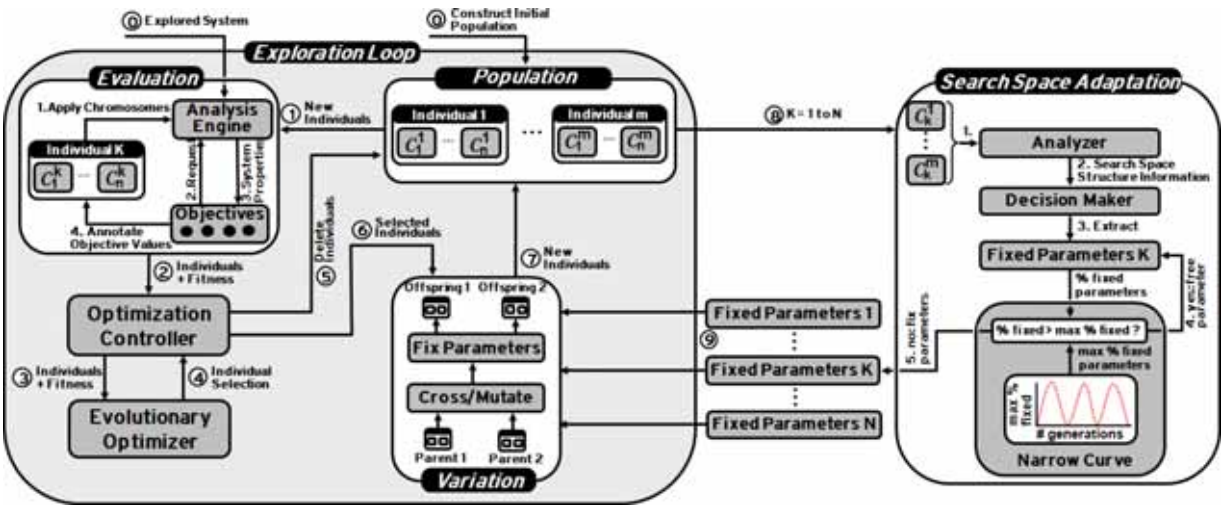



Fig. 3.11: Design space exploration loop with extension for automated search space modification

Like already mentioned, automated search space modification strategies are chromosome dependent. For instance, heuristic strategies finding interesting priority assignments for priority scheduled components are fundamentally different from those assigning reasonable time slots in time-triggered domains. Therefore, automated search space modification (step 8) is applied separately for different chromosomes. More precisely, if the population consists of the individuals $[i_1, \ldots, i_m]$, where individual $i_k$, $1 \leq k \leq m$, consists of $n$ chromosome instances $[c_1^k, \ldots, c_n^k]$, then step 8 is performed once for each set of chromosome instances $[c_k^1, \ldots, c_k^m]$, $1 \leq k \leq n$.

### 3.9.2 Concept

Concrete search space modifications strategies consist of three different parts: the *Analyzer*, the *Decision Maker*, and the *Narrow Curve*.

The *Analyzer* analyzes the structure of the search space covered by given chromosome instance sets during exploration (step 8.1). It extracts information that is useful to evaluate the quality of specific system parameter values. As mentioned above, this search space structure analysis is performed chromosome-wise on the population.

The *Decision Maker* takes as input the search space structure information extracted by the *Analyzer* (step 8.2). It is responsible for interpreting this information and makes concrete decisions on promising search space modifications (step 8.3).

*Narrow Curves* bound the maximum level of influence taken by search space modification strategies on the exploration process. More precisely, they define when and to what extend the search space might be restricted during exploration. It is important to bound the level of intervention, since heavy search space restrictions early in the exploration process might destroy diversity and lead to the convergence of the search towards local optima. *Narrow Curves* are defined as follows:

**Def. 3.9.1 (Narrow Curve)** A narrow curve is a function $\mathcal{N} : \mathbb{N} \times \mathbb{N} \times [0,1] \rightarrow [0,1]$. Its parameters are (1) the index of the currently processed generation $g_c$, (2) the total number of generations $g_t$ processed during evolutionary exploration, and (3) the target percentage $p_t$ denoting the maximum search space reduction allowed at arbitrary times during exploration. $\mathcal{N}(g_c, g_t, p_t)$ corresponds to the maximum percentage by which the search space may be restricted in generation $g_c$.

In case that the *Decision Maker* restricts the search space too heavily, the utilized *Narrow Curve* frees parameters until the imposed maximum percentage criterion is satisfied (step 8.4). Afterwards, the search space modification decisions are applied (step 8.5), and will be enforced during the subsequent variation process (step 9). Note that search space modification decisions are only taken into account during the variation process of a single generation. For the subsequent generation new search space adaptations are calculated.

In the next section, concrete *Analyzer*, *Decision Maker*, and *Narrow Curve* algorithms for automated search space modification on priority scheduled resources are introduced and evaluated.

## 3.10 Automated search space modification for priority chromosomes

In this section an automated search space modification strategy for priority chromosomes is presented. Three essential parts will be discussed: the *Analyzer* (Section 3.10.1), the *Decision Maker* (Section 3.10.2), and the *Narrow Curves* (Section 3.10.3). For a better overview, the information flow between the involved algorithms is visualized in Figure 3.12.

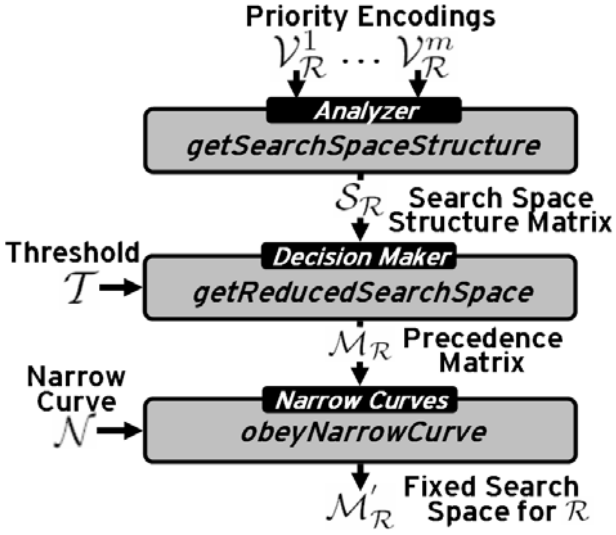



Fig. 3.12: Search space modification for priority chromosomes

In Section 3.10.4 it will be shown that the proposed automated search space modification strategy considerably improves exploration speed and quality of obtained results compared to "plain" exploration.

### 3.10.1 Analyzer

The analyzer used by the proposed automated search space modification strategy for priority chromosomes analyzes precedence relations between priorities of tasks mapped on the same resources. The algorithm used to extract that information is presented in Section 3.10.1.2. It uses so-called *precedence matrices* which are introduced in Section 3.10.1.1.

#### 3.10.1.1 Priority precedence matrices

For a given priority assignment $\mathcal{V}_{\mathcal{R}}$ (compare Definition 3.6.1) of tasks mapped on the resource $\mathcal{R}$, the *priority precedence matrix* $\mathcal{M}_{\mathcal{R}}$ is defined as follows:

**Def. 3.10.1 (Priority Precedence Matrix $\mathcal{M}_\mathcal{R}$)** For a resource $\mathcal{R}$ with $n$ mapped tasks $t_1, \ldots, t_n$ and the priority assignment $\mathcal{V}_\mathcal{R}$, the priority precedence matrix $\mathcal{M}_\mathcal{R}$ is of the following form:

$$\mathcal{M}_\mathcal{R} = \begin{pmatrix} \Phi_{11} & \cdots & \Phi_{1n} \\ \vdots & \ddots & \vdots \\ \Phi_{n1} & \cdots & \Phi_{nn} \end{pmatrix}$$

where $\Phi_{ij}$ $(= \mathcal{M}_\mathcal{R}[i][j])$ is defined as follows:

$$\Phi_{ij} = \begin{cases} 0 & : \quad i = j \\ 1 & : \quad Prio(i, \mathcal{V}_\mathcal{R}) < Prio(j, \mathcal{V}_\mathcal{R}) \\ -1 & : \quad \text{otherwise} \end{cases}$$

Note that $\Phi_{ij} = 1$ $(\Phi_{ij} = -1)$ denotes that $t_i$ has a higher (lower) priority than $t_j$. The precedence matrix for a given priority assignment can be derived with a simple algorithm that will be referred to as *getPriorityPrecedenceMatrix* in the following.

#### 3.10.1.2 Search space structure

For each priority scheduled resource $\mathcal{R}$ included into automated search space modification a so-called *search space structure matrix* $\mathcal{S}_\mathcal{R}$ is introduced:

**Def. 3.10.2 (Search space structure matrix $\mathcal{S}_\mathcal{R}$)** Given a population $\mathcal{P}$ with individuals $\mathcal{I}_1, \ldots, \mathcal{I}_m$ and a resource $\mathcal{R}$ with the mapped tasks $t_1, \ldots, t_n$.
The search space structure matrix $\mathcal{S}_\mathcal{R}$ integrates precedence information of all specific priority assignments $\mathcal{V}^1_\mathcal{R}, \ldots, \mathcal{V}^m_\mathcal{R}$ of $t_1, \ldots, t_n$ belonging to the individuals $\mathcal{I}_1, \ldots, \mathcal{I}_m$, and is defined as follows:

$$\mathcal{S}_\mathcal{R} = \begin{pmatrix} \Psi_{11} & \cdots & \Psi_{1n} \\ \vdots & \ddots & \vdots \\ \Psi_{n1} & \cdots & \Psi_{nn} \end{pmatrix}$$

where $\Psi_{ij} \in [0, 1]$ $(= \mathcal{S}_\mathcal{R}[i][j])$ denotes the percentage of $t_i$ having a higher priority than $t_j$ in the priority assignments $\mathcal{V}^1_\mathcal{R}, \ldots, \mathcal{V}^m_\mathcal{R}$.

The straightforward Algorithm 6 calculates the search space structure matrix for a given resource and population.

**Algorithm 6** getSearchSpaceStructure

**Require:** Priority assignments $\mathcal{V}_\mathcal{R}^1, \ldots, \mathcal{V}_\mathcal{R}^m$ for resource $\mathcal{R}$ with mapped tasks $t_1, \ldots, t_n$ of all individuals $\mathcal{I}_1, \ldots, \mathcal{I}_m$ in current population $\mathcal{P}$
**Ensure:** Search space structure matrix $\mathcal{S}_\mathcal{R}$

```
1: for all (i, j) with i, j ∈ [1, n] do
2:     S_R[i][j] ← 0
3: end for
4: for k ← 1 to m do
5:     M_R ← getPriorityPrecedenceMatrix(V_R^k)
6:     for all (i, j) with i, j ∈ [1, n] do
7:         if M_R[i][j] ← 1 then
8:             S_R[i][j] ← S_R[i][j] + 1
9:         end if
10:    end for
11: end for
12: for all (i, j) with i, j ∈ [1, n] do
13:    S_R[i][j] ← S_R[i][j] ÷ m
14: end for
```

### 3.10.2 Decision maker

The *Decision Maker* uses the search space structure information extracted by the *Analyzer* (Algorithm 6) to decide about search space modifications.

Algorithm 7 is used to calculate priority precedence relations that shall be fixed during the subsequent variation process. Note that during variation first the standard variation operators of the priority chromosome are applied. Afterwards, repair algorithms are used to enforce the priority precedence relations fixed by the *Decision Maker*.

All priority precedence relations occurring more often than the given threshold $\mathcal{T}$ are considered as promising and are, therefore, fixed by Algorithm 7. Note that under certain circumstances the calculated precedence matrix $\mathcal{M}_\mathcal{R}$ describing parameter restrictions can contain inconsistencies, i.e. cyclic priority precedence relations between tasks leading to infeasible priority assignments.

For example

$$\mathcal{M}_\mathcal{R} = \begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix}$$

**Algorithm 7** getReducedSearchSpace

**Require:** Search space structure matrix $\mathcal{S}_\mathcal{R}$ for resource $\mathcal{R}$ with mapped tasks $t_1, \ldots, t_n$, threshold $\mathcal{T} \in ]0.5, 1]$
**Ensure:** Precedence matrix $\mathcal{M}_\mathcal{R}$ representing parameter restrictions for $\mathcal{R}$

```
1: for all (i, j) with i, j ∈ [1, n] do
2:     M_R[i][j] ← 0
3: end for
4: for all (i, j) with i, j ∈ [1, n] do
5:     if S_R[i][j] > T then
6:         M_R[i][j] ← 1
7:         M_R[j][i] ← −1
8:     end if
9: end for
```

represents the following cyclic priority precedence relation: $Prio(1, V_\mathcal{R}) > Prio(2, V_\mathcal{R}) > Prio(3, V_\mathcal{R}) > Prio(1, V_\mathcal{R})$.

Theorem 3.10.1 states under which conditions a precedence matrix $\mathcal{M}_\mathcal{R}$ calculated according to Algorithm 7 is guaranteed to be free of such cyclic priority precedence relations.

**Theorem 3.10.1 (Consistency of $\mathcal{M}_\mathcal{R}$)** Assuming that the search space structure matrix $\mathcal{S}_\mathcal{R}$ was generated based on valid priority assignments, then the priority precedence matrix $\mathcal{M}_\mathcal{R}$ calculated by Algorithm 7 does not contain cyclic priority precedence relations of the length $n$ or shorter if the threshold $\mathcal{T}$ is chosen greater or equal than $\frac{n-1}{n}$. Mathematically spoken, if $\mathcal{T} \geq \frac{n-1}{n}$ then $\forall k \in [2, n]$ and $\forall (i_1, \ldots, i_k) \in [1, n]^k$ with $i_x \neq i_y \Leftrightarrow x \neq y$ it holds that

$$\mathcal{M}_\mathcal{R}[i_1][i_2] = \mathcal{M}_\mathcal{R}[i_2][i_3] = \ldots = \mathcal{M}_\mathcal{R}[i_{k-1}][i_k] = 1$$
$$\Rightarrow \mathcal{M}_\mathcal{R}[i_k][i_1] \neq 1$$

Proof: Proof through contradiction. It is assumed that $\mathcal{M}_\mathcal{R}$ contains a cyclic priority precedence relation chain of length $n$:

$$\mathcal{M}_\mathcal{R}[i_1][i_2] = \mathcal{M}_\mathcal{R}[i_2][i_3] = \ldots = \mathcal{M}_\mathcal{R}[i_{n-1}][i_n] = \mathcal{M}_\mathcal{R}[i_n][i_1] = 1$$

According to the assumption it holds for $i \in \{1, \ldots, n\}$:

$$\Psi_{i,i+1} \geq \frac{n-1}{n}$$

Consequently, the share of priority assignments for that $\mathcal{M}_{\mathcal{R}}[i][i+1] = 1$ does not hold can be bounded as follows:

$$\Psi_{i+1,i} < 1 - \frac{n-1}{n} = \frac{1}{n}$$

Accordingly, the maximum share of priority assignments for that at least one precedence relations of the assumed cyclic chain does not hold is bounded by the following inequality:

$$\Psi_{n,1} + \sum_{i=1}^{n-1} \Psi_{i+1,i} < \sum_{i=1}^{n} \frac{1}{n} = n * \frac{1}{n} = 1$$

Hence, there exist at least one priority assignment that satisfies the assumed cyclic precedence relation. The assumption that the $\mathcal{S}_{\mathcal{R}}$ was generated based on valid priority assignments is violated. □

For instance, Theorem 3.10.1 states that if the threshold $\mathcal{T} = 0.5$ is chosen then direct cyclic priority precedence relations between two arbitrary processes cannot occur in $\mathcal{M}_{\mathcal{R}}$.

Note that for the experiments in Section 3.10.4 $\mathcal{T} = 0.8$ was chosen, ensuring that $\mathcal{M}_{\mathcal{R}}$ is free of cyclic priority precedence relations up to the length of 5. Even though the systems used for the experiments contained resources with up to 10 mapped processes, i.e. theoretically allowing cyclic priority precedence relations with lengths superior to 5, no inconsistent priority matrices were generated by Algorithm 7 during the performed experiments.

### 3.10.3 Narrow curves

As already mentioned, *Narrow Curves* are utilized to bound the influence of search space modification strategies on the design space exploration process. In this section two different *Narrow Curves* are presented: the *Alternating Linear Narrow Curve* (Section 3.10.3.1) and the *Sinus Narrow Curve* (Section 3.10.3.2). Their effect on the search process is evaluated and discussed in Section 3.10.4.

In case that the *Decision Maker* fixed more priority precedence relations than allowed by the utilized *Narrow Curve*, some entries of the priority precedence matrix $\mathcal{M}_{\mathcal{R}}$ need to be removed. A simple algorithm freeing excess precedence relations is presented in Section 3.10.3.3.

#### 3.10.3.1 Alternating linear narrow curve $\mathcal{N}_{alt}$

The maximum allowed percentage by which the search space may be restricted alternates between being set to 0 during $z$ generations, and increasing linearly during $l$ generations:

$$\mathcal{N}_{alt}^{z,l}(g_c, g_t, p_t) = \begin{cases} 0 : (g_c - 1)\%(z+l) < z \\ \frac{n(g_c)}{n(g_t)} \times p_t : \text{otherwise} \end{cases}$$

where $n(x) = l * \left\lfloor \frac{x}{z+l} \right\rfloor + \max(0, x\%(z+l) - z)$.

The basic idea behind this alternating behavior of $\mathcal{N}_{alt}$ is to prevent exploration from converging to quickly towards local optima by allowing the reinsertion of diversity into the population during the periods where the search space modification is disabled.

Figure 3.13a visualizes $\mathcal{N}_{alt}$ for $z = l = 5$ and $p_t = 0.8$.

#### 3.10.3.2 Sinus narrow curve $\mathcal{N}_{sin}$

The maximum allowed percentage by which the search space may be restricted, increases and decreases according to the $2\pi$ periodic sinus function. The additional parameter *passes* indicates how often the sinus slope from 0 to $\pi$ is traversed during exploration:

$$\mathcal{N}_{sin}^{passes}(g_c, g_t, p_t) = \left| sin\left( \pi \times \frac{g_c}{\left\lfloor \frac{g_t}{passes} \right\rfloor} \right) \right| \times p_t.$$

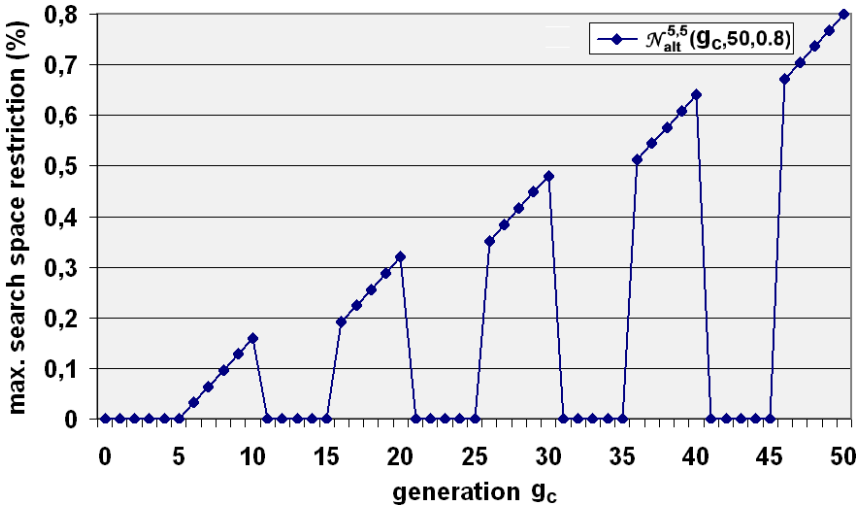
$\mathcal{N}_{sin}$ also alternates between periods allowing more and periods allowing less search space restrictions. However, overall $\mathcal{N}_{sin}$ pursues a much more aggressive search space restriction policy compared to $\mathcal{N}_{alt}$, and thus intervenes more into the "standard" evolutionary exploration process.

Figure 3.13b visualizes $\mathcal{N}_{sin}$ for $passes = 3$ and $p_t = 0.8$.

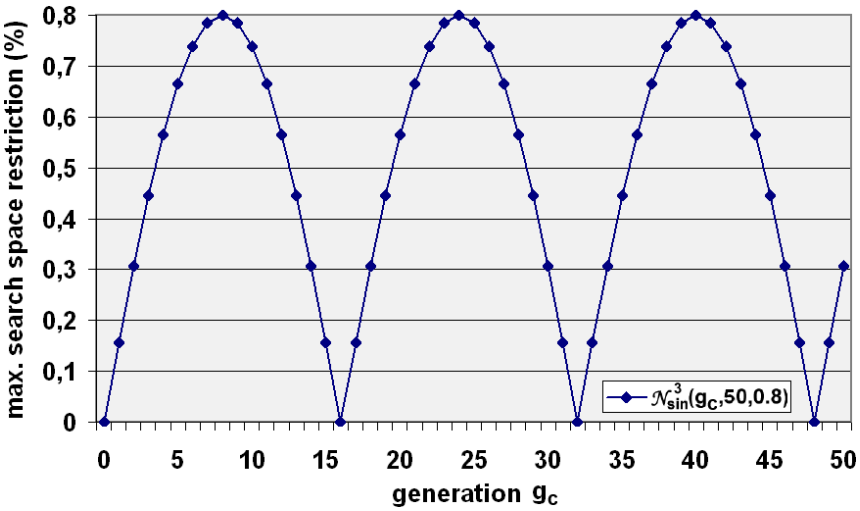
#### 3.10.3.3 Obeying the narrow curve

Algorithm 8 takes as input the priority precedence matrix $\mathcal{M}_{\mathcal{R}}$ calculated by the *Decision Maker* (Algorithm 7). If necessary it randomly removes priority precedence relations to ensure that the percentage search space reduction satisfies the given *Narrow Curve* $\mathcal{N}$.

In the first line, the number of priority precedence relations corresponding to completely fixed priority assignments is determined. Accordingly, the number of allowed priority precedence relations is calculated in line 2. In lines $3 - 8$ the actual number of fixed priority




(a) $\mathcal{N}_{alt}$ with $z = l = 5$ and $p_t = 0.8$




(b) $\mathcal{N}_{sin}$ with $passes = 3$ and $p_t = 0.8$

Fig. 3.13: Examples for an alternating linear and a sinus narrow curve

precedence relations contained in $\mathcal{M}_{\mathcal{R}}$ is determined, and in lines $9-16$ excess priority precedence relations are randomly removed.

### 3.10.4 Evaluation

In this section the effectiveness of the proposed priority search space modification strategy for priority chromosomes is evaluated. For this purpose, the exploration success rates, i.e. the average time needed to find the first feasible system parameter configuration (in terms of processed generations), of standard exploration (see Section 3.6.1) and exploration with automated search space modification are compared. For the experiments with automated search space modification two different narrow curves are evaluated, $\mathcal{N}_{alt}^{5,5}$ and $\mathcal{N}_{sin}^{5}$ with target percentage $p_t = 80\%$, and the threshold $\mathcal{T} = 80\%$ is applied for the *Decision Maker*.

**Algorithm 8** obeyNarrowCurve

**Require:** Precedence matrix $\mathcal{M}_\mathcal{R}$ representing parameter restrictions for resource $\mathcal{R}$ with mapped tasks $t_1, \ldots, t_n$, narrow curve $\mathcal{N}$, current generation $g_c$, total number of generations $g_t$, target percentage $p_t$

**Ensure:** Reduced precedence matrix $\mathcal{M}'_\mathcal{R}$ with at most $\mathcal{N}(g_c, g_t, p_t)$ percent fixed parameters

1: $param_{total} \leftarrow n^2 \div 2 - n$
2: $param_{allowed} \leftarrow \lfloor param_{total} \times \mathcal{N}(g_c, g_t, p_t) \rfloor$
3: $param_{fix} \leftarrow 0$
4: **for all** $(i, j)$ with $i, j \in [1, n]$ **do**
5:  **if** $\mathcal{M}_\mathcal{R}[i][j] = 1$ **then**
6:  $\text{param}_{fix} \leftarrow \text{param}_{fix} + 1$
7:  **end if**
8: **end for**
9: **while** $param_{fix} > param_{allowed}$ **do**
10:  choose random $i, j \in [1, n]$
11:  **if** $\mathcal{M}_\mathcal{R}[i][j] = 1 \vee \mathcal{M}_\mathcal{R}[i][j] = -1$ **then**
12:  $param_{fix} \leftarrow param_{fix} - 1$
13:  $\mathcal{M}_\mathcal{R}[i][j] \leftarrow 0$
14:  $\mathcal{M}_\mathcal{R}[j][i] \leftarrow 0$
15:  **end if**
16: **end while**

For evaluation randomly generated distributed systems containing 50 tasks with complex functional dependencies mapped on 5 to 10 priority scheduled resources are used. The systems are constrained with maximum end-to-end latencies (deadlines) along several task chains. Note that all experiments are performed twice, once for the generated systems with original constraints and once for the same systems with constraints tightened by 10%.

To guide exploration towards feasible system parameter configurations the following scheduling metric is used as optimization objective (compare Section 3.7).

**Def. 3.10.3 (Scheduling metric)** For a given system $\mathcal{S}$ with parameter configuration $c$ and constrained task chains $\mathcal{C}_1, \ldots, \mathcal{C}_n$, the scheduling index is denoted as $\mathcal{I}_{\mathcal{S},c}$ and defined as follows:

$$\mathcal{I}_{\mathcal{S},c} = \sum_{i=1}^{n} 1.1^{100 \times \frac{R_i - D_i}{D_i}}$$

where $R_i$ and $D_i$ denote the end-to-end latency and the deadline of the task chain $\mathcal{C}_i$, respectively.

**Original deadlines.** The results obtained for systems with original constraints using a sample size of 200 and confidence level 95% are visualized in Figure 3.14a. Note that for this experiment confidence level 95% means that, by repetition, in 95% of the cases the average exploration success rates obtained by 200 exploration runs (samples) will lie inside the confidence intervals indicated by the thin black lines drawn at the top of the bars, and in 5% of the cases they will lie outside.

As can be seen, exploration with automated search space modification largely outperforms standard exploration. The difference is especially remarkable during the first 15 generations. At this point exploration with automated search space modification using $\mathcal{N}_{sin}^{5}$ found feasible system parameter configurations in nearly 90% of all runs (confidence interval size 4.25%), whereas standard exploration yields an exploration success rate of only 34% (confidence interval size 6.58%).

Furthermore, it can be observed that automated search space modification with $\mathcal{N}_{sin}^{5}$ yields significant better exploration success rates compared to $\mathcal{N}_{alt}^{5,5}$ in the first 18 generations. However, after that point the obtained results converge to an exploration success rate of 98.5% after 30 generations (confidence interval size 1.69%).

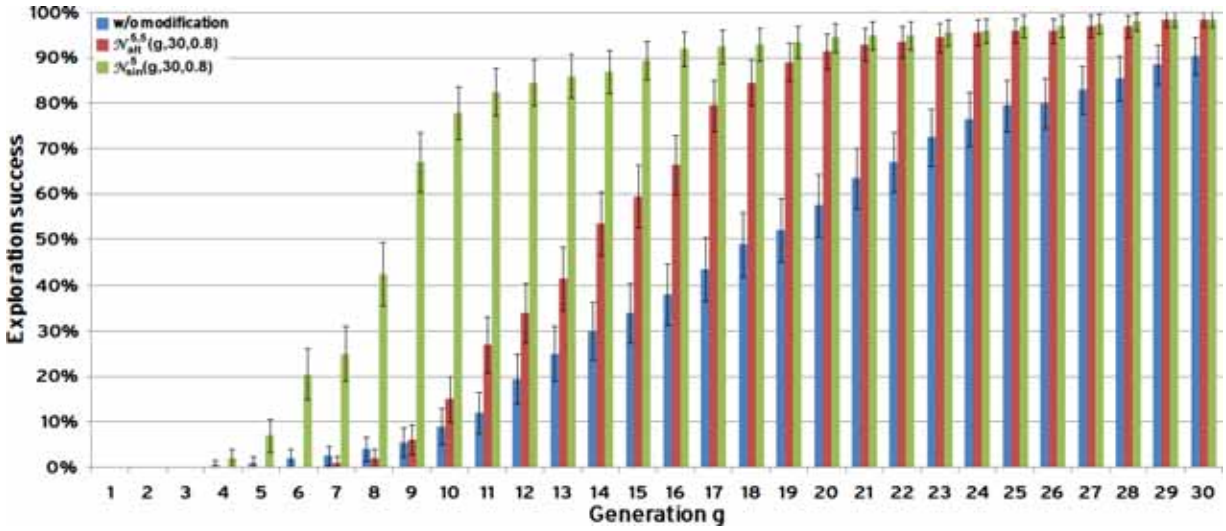
**Deadlines tightened by 10%.** In the second experiment, the same systems as in the first experiment are used. The only difference is that all end-to-end latency constraints are tightened by 10%. Obviously, this measure renders the optimization task more difficult.

The results using a sample size of 200 and confidence level 95% are visualized in Figure 3.14b.

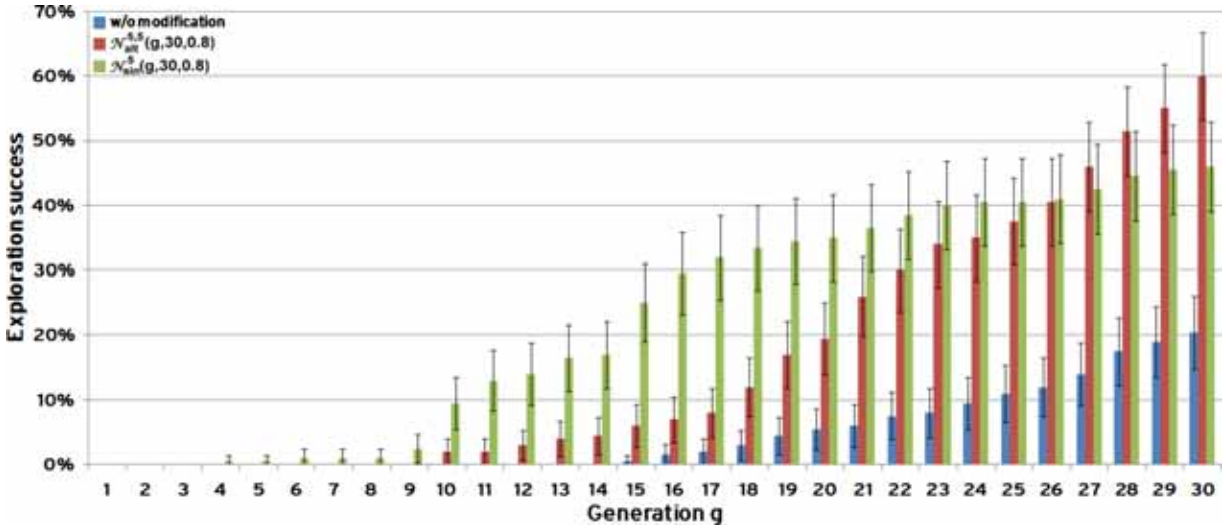
As can be seen, the difference between standard exploration and exploration with automated search space modification is even more significant than in the first experiment. Standard exploration does not succeed in finding feasible system parameter configurations until generation 15 (exploration success rate 0.5% with confidence interval size 1%). By then, exploration with automated search space modification using $\mathcal{N}_{sin}^{5}$

already exhibits an exploration success rate of 25% (confidence interval size 6%).

The difference between the two evaluated narrow curves $\mathcal{N}_{alt}^{5,5}$ and $\mathcal{N}_{sin}^{5}$ is particularly interesting. Like in the first experiment, $\mathcal{N}_{sin}^{5}$ leads to significantly better exploration success rates during the first generations. However, later during exploration $\mathcal{N}_{alt}^{5,5}$ outperforms $\mathcal{N}_{sin}^{5}$, and ultimately yields a significantly better exploration success rate after 30 generations (60% with confidence interval size 6.8% vs. 46% with confidence interval size 6.9%).




(a) Systems with original constraints (200 samples, confidence level of 95%)




(b) Same systems with constraints tightened by 10% (200 samples, confidence level of 95%)

Fig. 3.14: Exploration success with and without automated search space modification using different narrow curves. One generation corresponds to the evaluation of 25 individuals.

**Conclusion.** The conducted experiments show that the proposed automated search space modification approach for priority chromosomes significantly increases exploration efficiency. Thereby, the efficiency gain increases with the difficulty of the optimization task. Hence, automated search space modification during exploration of complex distributed embedded systems is feasible and effective.

Additionally, the experiments indicate that the choice of the utilized narrow curve can have major influence on exploration efficiency with automated search space modification. It is, for instance, reflected in the obtained results that $\mathcal{N}_{sin}$ pursues a much more aggressive search space restriction policy compared to $\mathcal{N}_{alt}$. At first, $\mathcal{N}_{sin}$ leads to significant better exploration success rates. However, for difficult optimization tasks $\mathcal{N}_{alt}$ turns out to be more efficient in the long run. The reason is that the aggressive search space restriction policy of $\mathcal{N}_{sin}$ might destroy diversity in the population early during exploration leading to convergence in local optima. Contrarily, $\mathcal{N}_{alt}$ is more unhurried in restricting the search space, resulting in less exploitation of search space restrictions in the beginning but longer lasting and more sustainable efficiency improvements.

Chapter 4

# DESIGN ROBUSTNESS OPTIMIZATION

In this chapter the robustness optimization problem that was motivated in Section 2.5 is addressed. Remember, that in this thesis robustness is not meant, as it is commonly assumed, in the sense of reliability and fault tolerance. Informally, a system is called *robust* if it can sustain modifications of system properties, such as worst-case execution times, data rates, etc., without experiencing severe system performance degradation, e.g. drastically increased response times or heavily reduced throughput.

Consequently, the robustness optimization task for a given system consists in finding parameter configurations that maximize system robustness with respect to modifications of a given system property set. To solve this problem, expressive robustness metrics as well as efficient robustness optimization techniques are needed.

Figure 4.1 visualizes the global robustness evaluation and optimization approach that is pursued in this chapter.

The robustness optimization problem for given metrics is solved using an iterative exploration approach. First, the *optimizer* generates a set of candidate parameter configurations. Afterwards, the *robustness evaluator* determines robustness characteristics for each parameter configuration that are used by *robustness metric calculation* for robustness assessment. Finally, the results are communicated to the *optimizer*, where they are used to decide about new candidate parameter configurations. This process is iteratively repeated until satisfactory parameter configurations are found. Note that the exploration framework presented in Chapter 3 is used for this part of the robustness optimization approach.

Some of the robustness metrics that are discussed in this chapter are computationally expensive, and can, therefore, not directly be inte-

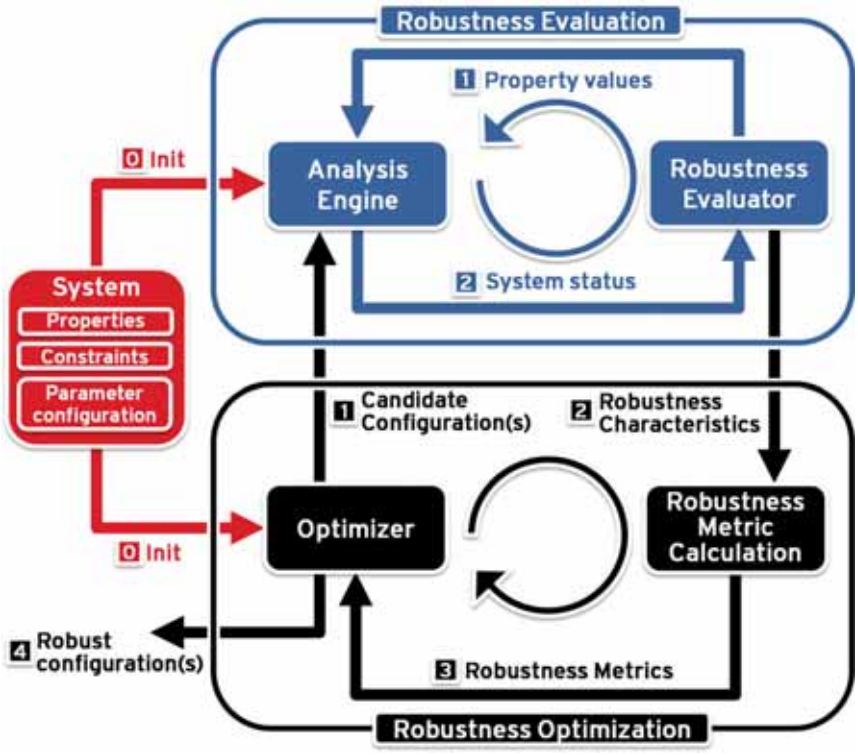



Fig. 4.1: Robustness evaluation and optimization

grated into the proposed robustness optimization loop. Consequently, the concerned robustness metrics must be approximated to be efficiently explored. In order to do so, robustness evaluation can be formulated as optimization problem that is solved using stochastic techniques. This approach allows to efficiently approximate the sought-after robustness metrics with conservative upper and lower bounds.

This possibility is reflected in Figure 4.1. The *robustness evaluator* iteratively issues requests to the *analysis engine* to determine robustness characteristics that are necessary for robustness metric calculation. Depending on the underlying metric this consists either in the exact determination or in the approximation of required robustness characteristics. Correspondingly, *robustness metric calculation* then derives either exact robustness metric values or approximating upper and lower robustness bounds.

The remainder of this chapter is organized as follows. In Section 4.1, basic techniques and notions that are needed to define and evaluate expressive robustness metrics are introduced: sensitivity analysis and hypervolume calculation. Afterwards, in Section 4.2, robustness metrics for the application scenarios and assumptions discussed in Section 2.5.4 are introduced. In the last part of this chapter, Section 4.4, it is shown how the proposed metrics can be efficiently integrated into robustness optimization. Thereby, the more complex robustness metrics are ap-

proximated using a scalable stochastic sensitivity analysis method that is presented in Section 4.3.

## 4.1 Preliminaries

In this section the formal fundament that is needed to define and evaluate expressive robustness metrics is introduced. First, *sensitivity analysis* is discussed and formally defined. As motivated in Section 2.5.2, the robustness optimization approach proposed in this chapter is based on sensitivity analysis. Secondly, the notion of *hypervolume* is introduced. The hypervolume metric is used to assign scalar values to robustness characteristics derived by sensitivity analysis in the multi-dimensional case.

### 4.1.1 Sensitivity analysis

In this section sensitivity analysis of embedded systems is discussed and formally defined. Note that below given definitions are valid for sensitivity analysis in monotone search spaces. For the performance model used in this thesis (see Section 2.2), most system properties influence system performance monotonically, e.g. worst-case execution times, activation jitters, and activation periods. In order to consider non-monotone system properties, like, for instance, resource speeds, the given definitions need to be extended. One possibility is the decomposition of the search space into piecewise monotone sub-spaces exploiting knowledge about so-called *scheduling anomalies*. Note that for dimensions greater two the decomposition of the search space into monotone sub-spaces can become very complex and, at present, there exists no complete theory covering this issue. Therefore, more research into this subject is necessary to pursue this approach. This work is out of the scope of this thesis, and will, therefore, not be further discussed. However, it shall be noted that a good starting point is the work in [90] covering the detection of scheduling anomalies for the two-dimensional case.

Another alternative to cover non-monotone search spaces is to prevent the appearance of scheduling anomalies. In the utilized performance model scheduling anomalies occur due to modifications of task best-case execution times (BCET). There are two reason for that. First, so-called *load anomalies* may occur due to BCET modifications. For instance, if a data processing task executes more rapidly (i.e. with a lower BCET) it potentially generates higher transient load peaks on the communication infrastructure (e.g. in the case of bursts) with negative effects on overall system performance. Second, BCETs directly influence so-called *task offsets* describing earliest task activation times with respect to timing

references. Modern analysis approaches, that are also used for experiments in this thesis, exploit these offsets to rule out overly conservative analysis assumptions (e.g. all tasks are activated at the same time), and thus calculate tighter response times [47, 48, 95, 81, 117]. Obviously, response times very much depend on the relative phasing between tasks. Certain offset combinations lead to specific task phasing situations resulting in shorter response times. However, this behavior is not linear and exhibits anomalous behavior, since increasing as well as decreasing relative phases between tasks might lead to either better or worse response times [55]. To conclude, if the BCETs of all tasks is set to zero, then scheduling anomalies are prevented at the cost of more conservative performance analysis results.

Sensitivity analysis systematically captures the impact of system property variations on system feasibility.

**Def. 4.1.1 (System feasibility)** A system $\mathcal{S}$ with parameter configuration $c$ is *feasible* if all imposed performance constraints, including maximum end-to-end latencies, maximum buffer sizes, etc., are satisfied.

System feasibility is influenced, on the one hand, by property values of system components (e.g. worst-case execution times, input data rates, etc.), and, on the other hand, by the system's parameter configuration (e.g. scheduling parameter assignments, etc.). Obviously, only certain system property value combinations lead to feasible systems for a given fixed parameter configuration. For instance, increasing worst-case execution times might lead to deadline violations or resource overload.

Sensitivity analyses calculate the exact boundaries between feasible and infeasible system states with respect to modifications of a large variety of system properties, including worst-case execution times, communication volumes, input data rates, processor and communication link performance, etc.

There is a large body of work addressing this issue. Classical approaches (e.g. [71, 69, 68, 63, 59, 106, 105, 4]) consider very restricted application and execution platform models (e.g. single components, fully periodic systems with deadlines equal to the period, etc.), and derive so-called *feasibility tests* or *schedulability tests* to characterize system feasibility. Using these tests, the border between feasible and infeasible system states can be determined based on quantitative evaluations of resource utilizations (*utilization bound*). More recent approaches include system properties variations into schedulability considerations [122, 88]. However, also these techniques, the first that are called sensitivity analyses, are based on simplistic application and execution platform models.

Consequently, the results yielded by these methods are of limited use for robustness evaluation and characterization of larger heterogeneous embedded systems.

To be relevant and expressive, sensitivity analyses must be independent of the underlying performance model. Modern sensitivity analysis approaches [93, 91] fulfill this requirement, and can be applied to arbitrary state-of-the-art formal analysis engines, e.g. [50, 17], which increases their applicability to real-world design flows of complex embedded systems.

The following formal definitions are based on an abstract system evaluation function capable of verifying system feasibility.

**Def. 4.1.2 (System evaluation function)** Let $\mathcal{S}_c$ denote the system $\mathcal{S}$ with parameter configuration $c$ and $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of system properties. The system evaluation function $\text{EVAL}_{\mathcal{S}_c}$ checks whether or not $\mathcal{S}_c$ is *feasible* if the property values $\vec{v} = (v_1, \ldots, v_n)$ are applied to the properties $\mathcal{P}$.

$$\text{EVAL}_{\mathcal{S}_c}(\mathcal{P},\vec{v}) = \begin{cases} \text{true, if } \mathcal{S}_c \text{ is feasible with values } \vec{v} \text{ for the} \\ \qquad \text{properties } \mathcal{P} \\ \text{false, otherwise.} \end{cases}$$

**Note:** *The abstract system evaluation function can be implemented using an arbitrary state-of-the-art analysis engine. Examples include SymTA/S [50, 96] and RTC-MPA [17, 123].*

**One-dimensional sensitivity analysis.** Given an abstract system evaluation function, minimum and maximum feasible values for arbitrary system properties can be calculated using, for instance, the approach in [93]. Calculating these extreme values for single system properties is called *one-dimensional sensitivity analysis*. For sensitivity analysis two different system property types are distinguished:

**Def. 4.1.3 (System property type)** Let $\mathcal{S}_c$ denote the system $\mathcal{S}$ with parameter configuration $c$. The type of a given system property $p$ subject to sensitivity analysis is categorized as follows:

1. $p$ is *subject to minimization* if a decrease of its initial value $v(p)$ potentially leads to the deterioration of the system performance. Following properties hold for $p$:
   - $\text{EVAL}_{\mathcal{S}_c}(p, v(p)) = true \Rightarrow \forall x > v(p) : \text{EVAL}_{\mathcal{S}_c}(p, x) = true$
   - $\text{EVAL}_{\mathcal{S}_c}(p, v(p)) = false \ \wedge \ \exists x : \text{EVAL}_{\mathcal{S}_c}(p, x) = true \Rightarrow x > v(p)$
2. $p$ is *subject to maximization* if an increase of its initial value $v(p)$ potentially leads to the deterioration of the system performance. Following properties hold for $p$:
   - $\text{EVAL}_{\mathcal{S}_c}(p, v(p)) = true \Rightarrow \forall x < v(p) : \text{EVAL}_{\mathcal{S}_c}(p, x) = true$
   - $\text{EVAL}_{\mathcal{S}_c}(p, v(p)) = false \ \wedge \ \exists x : \text{EVAL}_{\mathcal{S}_c}(p, x) = true \Rightarrow x < v(p)$

Examples for system properties subject to maximization are worst-case execution times and activation jitters. Examples for system properties subject to minimization are activation periods and resource speeds. For the two different system property types, one-dimensional sensitivity analysis can be formalized according to Definition 4.1.4.

**Def. 4.1.4 (Initial and extreme property values)** Let $\mathcal{S}_c$ denote the system $\mathcal{S}$ with parameter configuration $c$. For a given system property $p$ the initial property value is denoted as $v(p)$. The extreme property value for $p$ is denoted as $v^+_{\mathcal{S}_c}(p)$ and defined as follows:

1. If $p$ is subject to minimization:

$$v^+_{\mathcal{S}_c}(p) = \min \{x \in \mathbb{R}_+ \mid \text{EVAL}_{\mathcal{S}_c}(p, x) = true\}$$

2. Otherwise, if $p$ is subject to maximization:

$$v^+_{\mathcal{S}_c}(p) = \max \{x \in \mathbb{R}_+ \mid \text{EVAL}_{\mathcal{S}_c}(p, x) = true\}$$

**Note:**
- *$v^+_{\mathcal{S}_c}(p)$ can be calculated using sensitivity analysis algorithms, e.g. [93, 122, 11].*
- *Sensitivity analysis can be applied to feasible and infeasible systems. Accordingly, $v^+_{\mathcal{S}_c}(p)$ can be greater (smaller) than $v(p)$ for system properties subject to minimization (maximization). However, for the robustness metrics proposed in this chapter sensitivity analysis is always performed on feasible systems, since infeasible systems can obviously not be associated with any robustness properties.*

**Multi-dimensional sensitivity analysis.** To extend sensitivity analysis to the multi-dimensional case, i.e. taking into account interdependencies between multiple system properties, one-dimensional sensitivity analysis can be used to delimit the space containing the sought-after boundary between feasible and infeasible system property value combinations:

**Def. 4.1.5 (Bounding hypercube)** Let $\mathcal{S}_c$ denote the system $\mathcal{S}$ with parameter configuration $c$. For a given set of system properties $\mathcal{P} = \{p_1, \ldots, p_n\}$ the $n$-dimensional bounding hypercube $\mathcal{H}_{\mathcal{S}_c}(\mathcal{P}) \subset \mathbb{R}^n_+$ is defined as follows:

$$\begin{aligned}\mathcal{H}_{\mathcal{S}_c}(\mathcal{P}) = &\left[\min\left(v(p_1), v^+_{\mathcal{S}_c}(p_1)\right), \max\left(v(p_1), v^+_{\mathcal{S}_c}(p_1)\right)\right] \\ &\times \ldots \times \\ &\left[\min\left(v(p_n), v^+_{\mathcal{S}_c}(p_n)\right), \max\left(v(p_n), v^+_{\mathcal{S}_c}(p_n)\right)\right]\end{aligned}$$

The exact boundary between feasible and infeasible system property value combination in the multi-dimensional case is called *sensitivity front*. It can be formally characterized using the notion of Pareto-optimality:

**Def. 4.1.6 (Pareto-optimality)** Given a set $V$ of $n$-dimensional vectors in $\mathbb{R}^n$, the vector $\vec{v} = (v_1, \ldots, v_n) \in V$ Pareto-dominates the vector $\vec{w} = (w_1, \ldots, w_n) \in V$ iff for all elements $1 \leq i \leq n$ we have

1. minimization problem: $v_i \leq w_i$ and for at least one element $l$ we have $v_l < w_l$.
2. maximization problem: $v_i \geq w_i$ and for at least one element $l$ we have $v_l > w_l$.

A vector is called *Pareto-optimal* iff it is not Pareto-dominated by any other vector in $V$.

**Notations:**

- $\vec{v} \triangleright_+ \vec{w}$ denotes $\vec{v}$ Pareto-dominates $\vec{w}$ in the sense of a maximization problem.
- $\vec{v} \triangleright_- \vec{w}$ denotes $\vec{v}$ Pareto-dominates $\vec{w}$ in the sense of a minimization problem.
- $\Omega^+(V)$ represents the set of vectors in $V$ which are Pareto-optimal in the sense of a maximization problem.
- $\Omega^-(V)$ represents the set of vectors in $V$ which are Pareto-optimal in the sense of a minimization problem.

**Note:**

- *Given the notion of Pareto-optimality a simple algorithm can be derived checking if a given $n$-dimensional vector $v$ is Pareto-optimal with respect to a collection of reference vectors $V$. In the following, such an algorithm is referred to as* DOMINATEDBY$(v, V, max)$*, where $max \in \{true, false\}$ indicates whether Pareto-optimality is meant in the sense of a maximization or minimization problem.*
- *Checking Pareto-optimality is of linear computational complexity with respect to the cardinality of the reference vector set $V$.*

The formal characterization of the sensitivity front is given in Definition 4.1.7, which generalizes Definition 4.1.4.

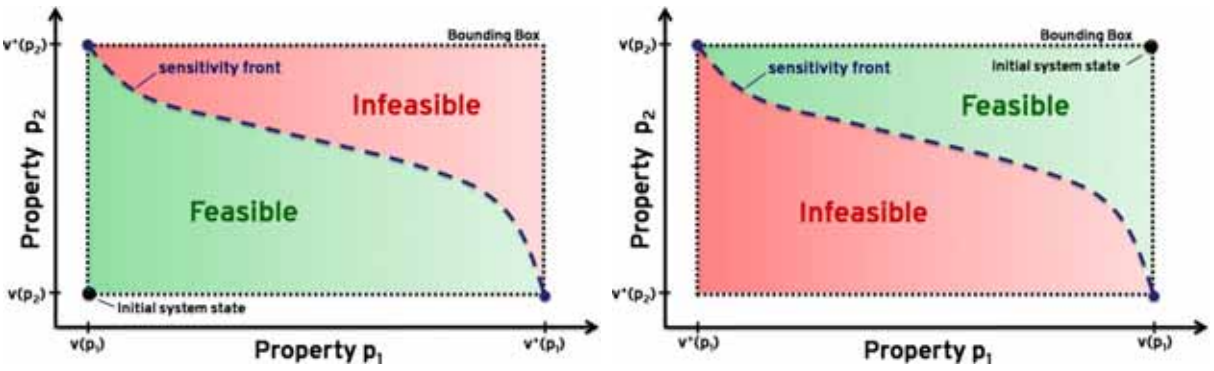
**Def. 4.1.7 (Sensitivity front)** Let $\mathcal{S}_c$ denote the system $\mathcal{S}$ with parameter configuration $c$. For a given set of system properties subject to maximization $\mathcal{P} = \{p_1, \ldots, p_n\}$ the *sensitivity front* is defined as the set of all feasible property value combinations which are not Pareto-dominated (in the sense of a maximization problem) by any other working property value combination:

$$\mathcal{F}^{sens}_{\mathcal{S}_c}(\mathcal{P}) = \{\vec{x} \in \mathcal{H}_{\mathcal{S}_c}(\mathcal{P}) \,|\, \text{EVAL}_{\mathcal{S}_c}(\mathcal{P},\vec{x}) \wedge \nexists \vec{y} \in \mathcal{H}_{\mathcal{S}_c}(\mathcal{P}) : (\text{EVAL}_{\mathcal{S}_c}(\mathcal{P},\vec{y}) \wedge \vec{y} \triangleright_+ \vec{x})\}$$

**Note:**

- *For system properties subject to minimization Pareto-domination must be considered in the sense of a minimization problem.*
- *In case the involved system properties are of different types, the Pareto-dominance relation must be adapted to reflect the mixed minimization/maximization problem.*
- *In case $\mathcal{S}_c$ is not feasible with the initial values for the considered properties $\mathcal{P}$ the resulting sensitivity front might be equal to the empty set.*

The sensitivity front separates the continuous spaces of system property value combinations leading to feasible and infeasible systems. Figure 4.2 visualizes two-dimensional example sensitivity fronts for system properties subject to maximization and minimization.




(a) Properties subject to maximization (b) Properties subject to minimization

Fig. 4.2: Two-dimensional example sensitivity fronts

### 4.1.2 Hypervolume

Some of the introduced robustness metrics in this chapter are based on the notion of *hypervolume*. Hypervolume measures the portion of space covered, in the sense of Pareto-dominance (see Definition 4.1.6), by a given vector set.

As can be observed in Figure 4.2, feasible regions are obtained if multi-dimensional sensitivity analysis is applied to two or more system properties. Hypervolume is meaningful as starting point for robustness metrics in the multi-dimensional case, since it is sensitive to improvement in space covered by such feasible regions. More precisely, feasible regions including (i.e. dominating) other feasible regions lead to strictly greater hypervolume values.

First,*absolute* hypervolume (Section 4.1.2.1) is introduced. Afterwards, the *weighted percentage* hypervolume is derived (Section 4.1.2.2).

#### 4.1.2.1 Absolute hypervolume

According to Definition 4.1.8 two different types of hypervolumes are distinguished: the inner and the outer hypervolume.

**Def. 4.1.8 (Absolute inner and outer hypervolumes)** Let $\mathcal{H} = \left[\underline{b}_1,\overline{b}_1\right] \times \ldots \times \left[\underline{b}_n,\overline{b}_n\right] \subset \mathbb{R}^n_+$ denote a $n$-dimensional hypercube, and $\mathcal{V} = \{\vec{v}_1, \ldots, \vec{v}_m\}$ a set of $n$-dimensional vectors with $\forall_i \vec{v}_i \in \mathcal{H}$.

1. The inner hypervolume of $\mathcal{V}$ in $\mathcal{H}$ is denoted as $\lambda^-_{\mathcal{H}}(\mathcal{V})$ and is defined as the volume of the space in $\mathcal{H}$ containing all vectors $\vec{w}$ which are Pareto-dominated by at least one vector $\vec{v} \in \mathcal{V}$:

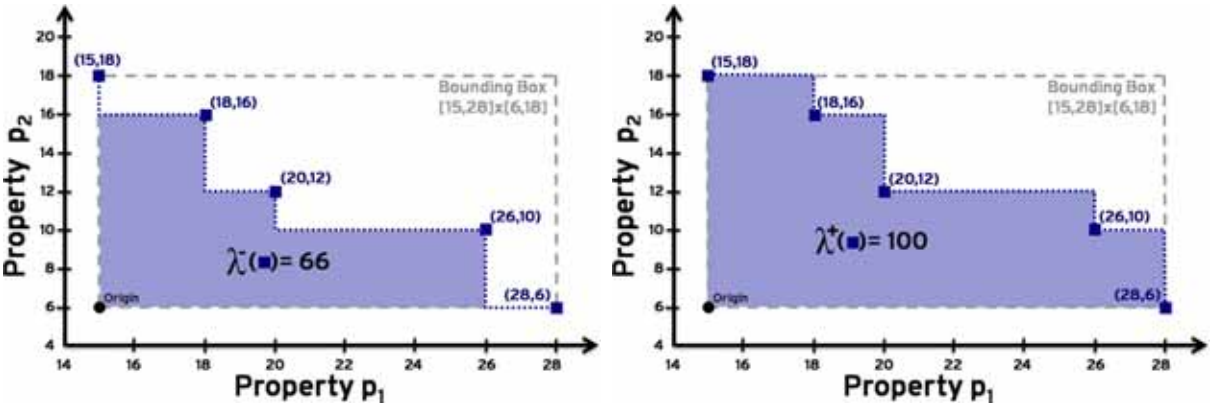
$$\lambda^-_{\mathcal{H}}(\mathcal{V}) = \mathrm{vol}\left\{\vec{w} \in \mathcal{H} \mid \exists \vec{v} \in \mathcal{V} : \vec{v} \triangleright_+ \vec{w}\right\}$$

2. The outer hypervolume of $\mathcal{V}$ in $\mathcal{H}$ is denoted as $\lambda^+_{\mathcal{H}}(\mathcal{V})$ and is defined as the difference between the volume of the hypercube $\mathcal{H}$ and the volume of the space in $\mathcal{H}$ containing all vectors $w$ Pareto-dominating at least one vector $v \in \mathcal{V}$:

$$\lambda^+_{\mathcal{H}}(\mathcal{V}) = \mathrm{vol}(\mathcal{H}) - \mathrm{vol}\left\{\vec{w} \in \mathcal{H} \mid \exists \vec{v} \in \mathcal{V} : \vec{w} \triangleright_+ \vec{v}\right\}$$

**Note:** *Formal definitions of the abstract vol operator based on the volume of polytopes or hypercubes can be found in [137] and [32], respectively.*

Figures 4.3a and 4.3b visualize the difference between inner and outer hypervolume in the two-dimensional case: the inner hypervolume corresponds to the space covered by the lower step function, whereas the upper hypervolume corresponds to the space covered by the upper step function.




(a) Absolute inner hypervolume (b) Absolute outer hypervolume

Fig. 4.3: Absolute inner and outer hypervolumes

The inner hypervolume is usually used as measure to compare the efficiency of evolutionary multi-objective algorithms [32, 138]. However, recently it was also used as selection criterion for multi-objective search [28, 135]. Especially for the second point it is required that hypervolume can be calculated efficiently. Therefore, several efficient algorithms for hypervolume calculation were proposed in the last years [128, 35, 134].

Given an algorithm for calculating the *inner hypervolume* $\lambda^-(\mathcal{V})$, the *outer hypervolume* $\lambda^+(\mathcal{V})$ can be calculated according to Algorithm 9.

First, the hypercube bounding $\mathcal{V}$ is calculated (lines $1-8$). Afterwards, the origin of the vectors in $\mathcal{V}$ is translated to the extreme point of the bounding hypercube (lines $9-13$). Note that the inner hypervolume of the translated vector set corresponds to the space containing all vectors dominating at least one vector in the initial set $\mathcal{V}$. Finally, $\lambda^+(\mathcal{V})$ is calculated by subtracting the inner hypervolume of the translated vector set from the hypervolume of the bounding hypercube (lines $14-18$).

#### 4.1.2.2 Weighted percentage hypervolume

In this section the so-called *weighted percentage hypervolume*, which is based on the notion of absolute hypervolume, is introduced.

For the weighted percentage hypervolume the coordinates of the considered vectors are translated to the percentage increase with respect to

**Algorithm 9** $\lambda^+(\mathcal{V})$

**Require:** Set of $n$ dimensional Pareto-optimal vectors $\mathcal{V}$
**Ensure:** Outer hypervolume $\lambda^+$ of $\mathcal{V}$

```
for i ← 1 to n do
    min[i] ← ∞
    max[i] ← −∞
    for all v ∈ V do
        min[i] ← min(min[i], v[i])
        max[i] ← max(max[i], v[i])
    end for
end for
for all v ∈ V do
    for i ← 1 to n do
        v[i] ← max[i] − v[i]
    end for
end for
λ+_tmp ← 1
for i ← 1 to n do
    λ+_tmp ← λ+_tmp × (max[i] − min[i])
end for
λ+(V) ← λ+_tmp − λ−(V)
```

the origin of the bounding hypercube. This is necessary for obtaining expressive and comparable results in case that coordinate values in different dimensions are of different orders of magnitude. Additionally, it is possible to attach importance levels to each dimension, i.e. the space covered in one dimension might be considered more important than that covered in other dimensions.

**Def. 4.1.9 (Weighted percentage hypervolumes)** Let $\mathcal{H} = \left[\underline{b}_1,\overline{b}_1\right] \times \ldots \times \left[\underline{b}_n,\overline{b}_n\right] \subset \mathbb{R}^n_+$ denote a $n$-dimensional hypercube, and $\mathcal{V} = \{\vec{v}_1, \ldots, \vec{v}_m\}$ a set of $n$-dimensional vectors with $\forall_i \vec{v}_i = (v_{i1}, \ldots, v_{in}) \in \mathcal{H}$. Given a set of weights $\mathcal{W} = \{w_1, \ldots, w_n\}$ with $\forall_i w_i \geq 1$, the inner and outer weighted percentage hypervolumes of $\mathcal{V}$ are defined as follows:

$$\tilde{\lambda}^-_{\mathcal{H}}(\mathcal{V},\mathcal{W}) = \lambda^-_{\tilde{\mathcal{H}}_{\mathcal{W}}}(\tilde{\mathcal{V}}_{\mathcal{W}}) \quad \text{and} \quad \tilde{\lambda}^+_{\mathcal{H}}(\mathcal{V},\mathcal{W}) = \lambda^+_{\tilde{\mathcal{H}}_{\mathcal{W}}}(\tilde{\mathcal{V}}_{\mathcal{W}})$$

Thereby:

- $\tilde{\mathcal{H}}_{\mathcal{W}} = \left[0, f_1(\frac{\bar{b}_1 - \underline{b}_1}{\underline{b}_1})\right] \times \ldots \times \left[0, f_n(\frac{\bar{b}_n - \underline{b}_n}{\underline{b}_n})\right]$
- $\tilde{\mathcal{V}}_{\mathcal{W}} = \{\vec{v}_1^*, \ldots, \vec{v}_m^*\}$, with $\vec{v}_i^* = \left(f_1(\frac{v_{i1} - \underline{b}_1}{\underline{b}_1}), \ldots, f_n(\frac{v_{in} - \underline{b}_n}{\underline{b}_n})\right)$
- $f_i(x) = x \times \left(x \times \frac{w_i - 1}{10} + 1\right)$

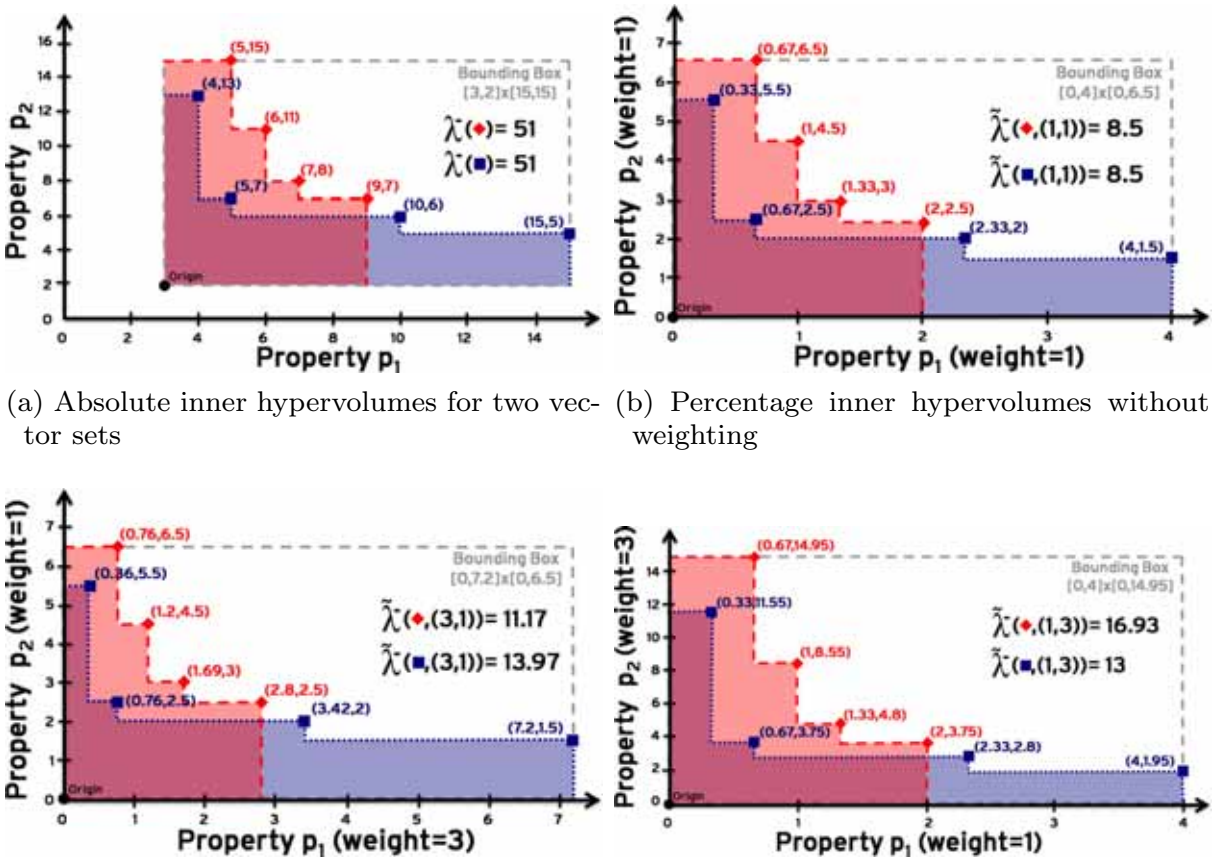
**Note:** *The $f$ function can be replaced by other weighting functions. However, an adequate weighting function for $\tilde{\lambda}^-$ and $\tilde{\lambda}^+$ must grow faster than linear. The reason is that linear functions do not change the proportion between two compared hypervolumes even if they cover different amounts of space in different dimensions.*

Figure 4.4a visualizes the absolute inner hypervolumes for two example vector sets. The corresponding inner percentage hypervolumes without weighting are visualized in Figure 4.4b. As can be observed, both vector sets cover exactly the same amount of space. However, the vector set represented by the squares covers more space in x-dimension, whereas the vector set represented by the diamonds covers more space in y-dimension. This is reflected by the metric if weighting is applied. Figure 4.4c visualizes the percentage hypervolumes with weight 3 for the x-dimension. As expected the vector set represented by the squares is assigned a higher weighted percentage hypervolume compared to the vector set represented by the diamonds. The other way around, if the y-dimension is weighted 3 it can be observed in Figure 4.4d that the vector set represented by the diamonds results in a higher weighted percentage hypervolume.

## 4.2 Robustness metrics

In this section meaningful robustness metrics that can be included into design space exploration to conceive robust systems are introduced. The proposed metrics are based on sensitivity analysis techniques (see Section 4.1.1). Basically, two different robustness metrics types are distinguished:

1. *Independent system properties*: In Section 4.2.1 robustness metrics for system properties that are independent with respect to system performance are proposed. Thereby, independent denotes that the modification of one system property does not influence the permissible modification space of other system properties. The proposed metrics are based on one-dimensional sensitivity analysis.




(a) Absolute inner hypervolumes for two vector sets

(b) Percentage inner hypervolumes without weighting

(c) Percentage inner hypervolumes with weight 3 for x-dimension and no weighting for y-dimension

(d) Percentage inner hypervolumes with no weighting for x-dimension and weight 3 for y-dimension

Fig. 4.4: Examples for weighted percentage hypervolume

2. *Dependent system properties*: In Section 4.2.2 robustness metrics for system properties that are dependent with respect to system performance are proposed. Thereby, dependent denotes that the modification of one system property decreases the maximum permissible modification space for dependent system properties. In order to be meaningful, the robustness metrics must include dependencies between involved properties. Obviously, slack values determined by one-dimensional sensitivity analysis are not sufficient to characterize system robustness in this case. The proposed metrics are, therefore, based on multi-dimensional sensitivity analysis.

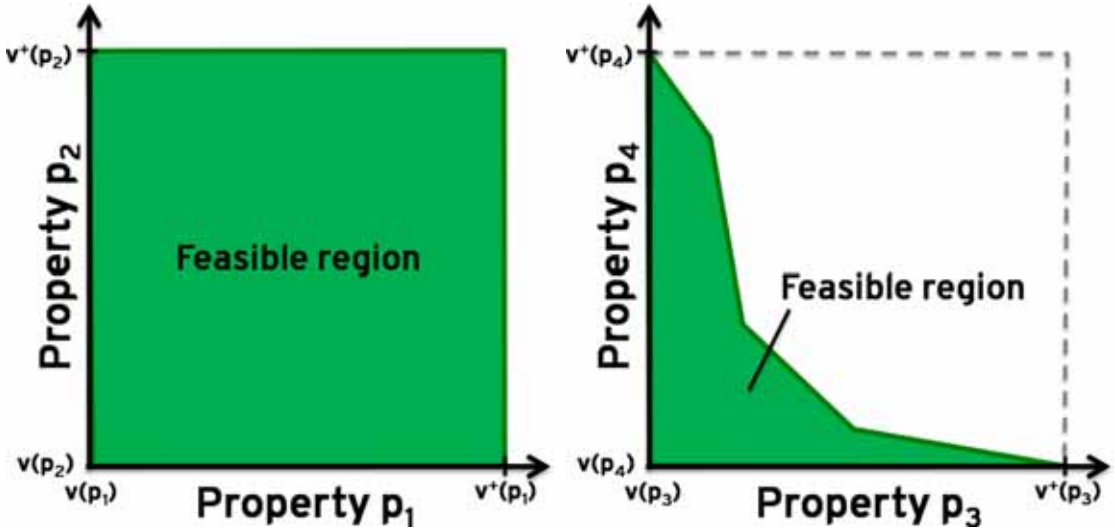
Figures 4.5a and 4.5b visualize the difference between independent and dependent system properties in the two-dimensional case. If one-dimensional sensitivity analysis is applied, the same slack values are obtained for the given system property pairs $(p_1, p_2)$ and $(p_3, p_4)$. Consequently, their robustness would be considered equal. However, intuitively it is clear that the robustness of $(p_1, p_2)$ is higher, since the feasible

region that can be determined using two-dimensional sensitivity analysis covers far more valid system property value combinations.

In some situations, e.g. in the case of indirect dependencies, it might not be possible for the designer to a-priori determine whether or not the considered system properties exhibit dependencies with respect to system performance. This fact might only be discovered during robustness analysis. Therefore, a tool that implements the proposed robustness optimization methods should not a-priori separate both cases, but rather allow the designer to dynamically change her initial assumptions. For instance, she might start assuming dependent behavior and later, as more information become available, possibly switch to the metrics tailored for the independent case.

Furthermore, the proposed robustness metrics for dependent system properties can be combined with those for independent system properties. For instance, several sets of dependent system properties might exhibit no interdependencies. In such a case it makes sense to individually assess the robustness of the dependent system properties sets, and to combine the results using the metrics for independent system properties.

Note that in both cases, i.e. for independent and dependent system properties, robustness metrics covering static as well as dynamic system behavior are discussed (compare Section 2.5.4). Additionally, metrics quantifying the benefit of conceiving reconfigurable system components are derived (Section 4.2.3).




(a) Independent properties $p_1$ and $p_2$ (b) Dependent properties $p_3$ and $p_4$

Fig. 4.5: Example feasible regions for independent and dependent system properties subject to maximization (two-dimensional case)

### 4.2.1 Independent system properties

The robustness metrics presented in this section are tailored for independent system properties. This comes with the advantage that the proposed metrics can be computed very efficiently, since only one-dimensional sensitivity analysis needs to be conducted for each considered system property. However, the downside is that the metrics cannot be applied to situations where system properties influence each other.

The proposed metrics allow to flexibly weight and adjust the impact of the involved system properties. In addition to classical weighting mechanisms, the concepts of the *generalized mean* are integrated into the proposed metrics. Furthermore, rather than using absolute slack values, the proposed metrics are based on the notion of *percentage slack* describing the permissible modification space of system properties relative to their initial values in percent.

**Def. 4.2.1 (Percentage slack)** Let $\mathcal{S}_c$ denote the system $\mathcal{S}$ with parameter configuration $c$. The percentage slack of a given system property $p$ is defined as follows:

$$\text{pslack}_{\mathcal{S}_c}(p) = \frac{\mid v^+_{\mathcal{S}_c}(p) - v(p) \mid}{v(p)} \times 100$$

**Static design robustness.** The SDR metric is given in Definition 4.2.2. The influence of the involved system properties $p_1, \ldots, p_n$ can be configured by adjusting the corresponding weights $w_1, \ldots, w_n$. Additionally, the SDR metric is strongly influenced by the parameter $k$. Note that the way in which $k$ can be utilized to adapt the behavior of the SDR metric to different use-cases is discussed below.

**Def. 4.2.2 (Static design robustness)** Let $\mathcal{S}_c$ denote the system $\mathcal{S}$ with parameter configuration $c$ and $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of system properties. Given a set of weights $\mathcal{W} = \{w_1, \ldots, w_n\}$ with $\forall_i w_i > 0$ and $w = \sum_{i=1}^{n} w_i$, and a real number $k$, the static design robustness of $\mathcal{S}_c$ with respect to $\mathcal{P}$ is defined as follows:

$$\text{SDR}_{\mathcal{S}_c}(\mathcal{P}, \mathcal{W}, k) = \begin{cases} \sqrt[w]{\prod_{i=1}^{n} \text{pslack}_{\mathcal{S}_c}(p_i)^{w_i}} & , k = 0 \\ \sqrt[k]{\frac{1}{w} \times \sum_{i=1}^{n} \left( w_i \times \text{pslack}_{\mathcal{S}_c}(p_i)^k \right)} & , \text{otherwise} \end{cases}$$

**Dynamic design robustness.** The DDR metric for a single system property is given in Definition 4.2.3. It is defined as the percentage slack of the parameter configuration (included in the set of possible parameter configurations $\mathcal{C}$) that allows the maximum variation for the considered system property.

**Def. 4.2.3 (Dynamic design robustness)** The dynamic design robustness of a given system $S$ with respect to a system property $p$ considering a set of possible parameter configurations $\mathcal{C}$ (reconfiguration space) is defined as follows:

$$\mathrm{DDR}_{\mathcal{S},\mathcal{C}}(p) = \max_{c \in \mathcal{C}} \mathrm{pslack}_{\mathcal{S}_c}(p)$$

The dynamic design robustness metric integrating the robustness potential of several system properties that can be achieved through reconfigurability is given in Definition 4.2.4.

**Def. 4.2.4 (Aggregated dynamic design robustness)** Let $\mathcal{S}$ denote a given system, and $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of system properties. Given a set of possible parameter configurations $\mathcal{C}$ for $\mathcal{S}$ (reconfiguration space), a set of weights $\mathcal{W} = \{w_1, \ldots, w_n\}$ with $\forall_i w_i > 0$ and $w = \sum_{i=1}^{n} w_i$, and a real number $k$, the aggregated dynamic design robustness of $S$ with respect to $\mathcal{P}$ is defined as follows:

$$\mathrm{ADDR}_{\mathcal{S},\mathcal{C}}(\mathcal{P},\mathcal{W},k) = \begin{cases} \sqrt[w]{\prod_{i=1}^{n} \mathrm{DDR}_{\mathcal{S},\mathcal{C}}(p_i)^{w_i}} & , k = 0 \\ \sqrt[k]{\frac{1}{w} \times \sum_{i=1}^{n} \left(w_i \times \mathrm{DDR}_{\mathcal{S},\mathcal{C}}(p_i)^k\right)} & , \text{otherwise} \end{cases}$$

The proposed ADDR metric is based on the DDR values of the involved system properties, and includes classical weighting mechanisms. Furthermore, the ADDR metric is influenced by the $k$ parameter in the same manner as the SDR metric.

**Influence of the parameter $k$.** The SDR and ADDR metrics can be configured to favor system properties with either low or high slack values by modifying the parameter $k$. Figure 4.6 visualizes the impact of $k$ for two hypothetical system properties (both weighted 1) with slack values $v_1$ and $v_2$.

It can be observed that decreasing and increasing values for $k$ increase the impact of system properties with small and large slacks, respectively. In particular, low $k$ values lead to decreased metric values for parameter

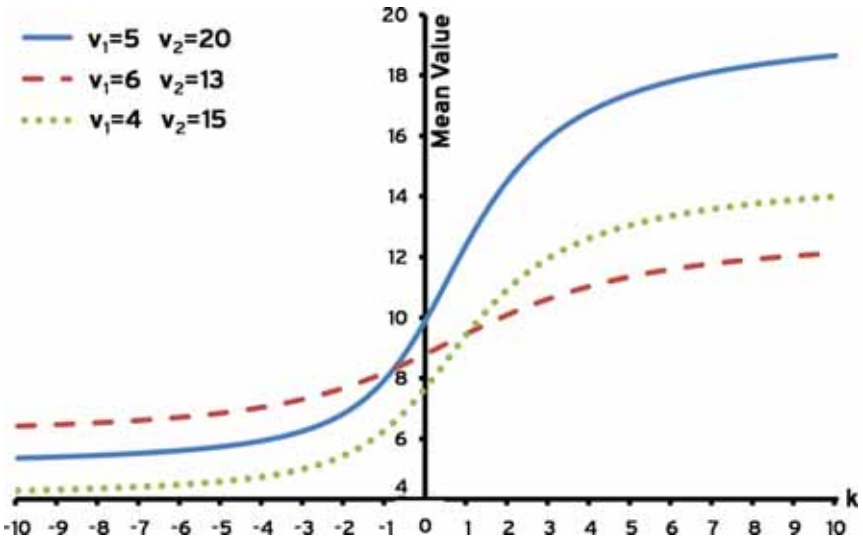



Fig. 4.6: Influence of the parameter $k$ on the metric values

configurations with unbalanced robustness properties, i.e. parameter configurations buying large slack for single system properties with small slacks for other system properties. In other words, with low $k$ values the metrics tend to maximize slacks of system properties with low robustness potentials, leading to a more balanced distribution of globally available performance head-room. Contrarily, choosing high $k$ values leads to slack maximization for the most robust system properties.

Special cases for the choice of $k$ (not including the effects of weighting):

- $k \to -\infty$: minimum of the involved slacks
- $k = 0$: geometric mean of the involved slacks
- $k = 1$: arithmetic mean of the involved slacks
- $k \to \infty$: maximum of the involved slacks

### 4.2.2 Dependent system properties

The robustness metrics discussed in the previous section quantify the robustness of several system properties. However, since the underlying sensitivity analysis is one-dimensional, possible performance interdependencies are ignored. In case that the considered system properties are independent or influence each other only marginally those metrics are meaningful. However, in case of dependent system properties it is necessary to include their performance dependencies into the robustness metrics.

In this section robustness metrics capable of capturing performance dependencies between considered system properties are introduced. The proposed metrics are based on multi-dimensional sensitivity analysis

(Section 4.1.1), and the notion of weighted percentage hypervolume (Section 4.1.2.2).

**Static design robustness.** The static design robustness in the case of dependent system properties is formalized by Definition 4.2.5.

**Def. 4.2.5 (Static design robustness for dependent system properties)** Let $\mathcal{S}_c$ denote the system $\mathcal{S}$ with parameter configuration $c$ and $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of dependent system properties. Given a set of weights $\mathcal{W} = \{w_1, \ldots, w_n\}$ with $\forall_i w_i \geq 1$, the static design robustness of $\mathcal{S}_c$ with respect to $\mathcal{P}$ is defined as follows:

- If the properties $\mathcal{P}$ are subject to maximization

$$\mathrm{SDR}^{\mathrm{dep}}_{\mathcal{S}_c}(\mathcal{P}, \mathcal{W}) = \tilde{\lambda}^{-}_{\mathcal{H}_{\mathcal{S}_c}(\mathcal{P})}(\mathcal{F}^{sens}_{\mathcal{S}_c}(\mathcal{P}), \mathcal{W})$$

- Otherwise, if the properties $\mathcal{P}$ are subject to minimization

$$\mathrm{SDR}^{\mathrm{dep}}_{\mathcal{S}_c}(\mathcal{P}, \mathcal{W}) = \tilde{\lambda}^{-}_{\mathcal{H}_{\mathcal{S}_c}(\mathcal{P})}\left(\left\{(\bar{b}_1, \ldots, \bar{b}_n)\right\}, \mathcal{W}\right) - \tilde{\lambda}^{+}_{\mathcal{H}_{\mathcal{S}_c}(\mathcal{P})}(\mathcal{F}^{sens}_{\mathcal{S}_c}(\mathcal{P}), \mathcal{W})$$[1]

Depending on the considered system properties type, either the inner or the outer weighted percentage hypervolume is applied to the sensitivity front of the given parameter configuration to quantify its robustness.

Consider, for instance, the two-dimensional situation depicted in Figure 4.7. On the left-hand side the points defining the sensitivity front for two system properties $p_1$ and $p_2$ are visualized. In case that both system properties are subject to maximization, the region that can safely be included into robustness evaluation is equal to the region that is Pareto-dominated by the sensitivity front in the sense of a maximization problem. In the shown two-dimensional case, this corresponds to the region below the lower step function, and more generally, in the n-dimensional case, to the inner hypervolume covered by the sensitivity front inside the bounding hypercube.

In the contrary case, i.e. if $p_1$ and $p_2$ are subject to minimization, the region that can be safely considered for robustness assessment is equal to the region that is Pareto-dominated by the sensitivity front in the sense of a minimization problem. In the two-dimensional case, this region corresponds to the intersection between the bounding hypercube and the region below the upper step function. In the general case, the

[1] $\bar{b}_1, \ldots, \bar{b}_n$ denote the extreme points of the bounding hypercube $\mathcal{H}_{\mathcal{S}_c}(\mathcal{P})$

outer hypervolume covered by the sensitivity front must be subtracted from the hypervolume of the bounding hypercube.

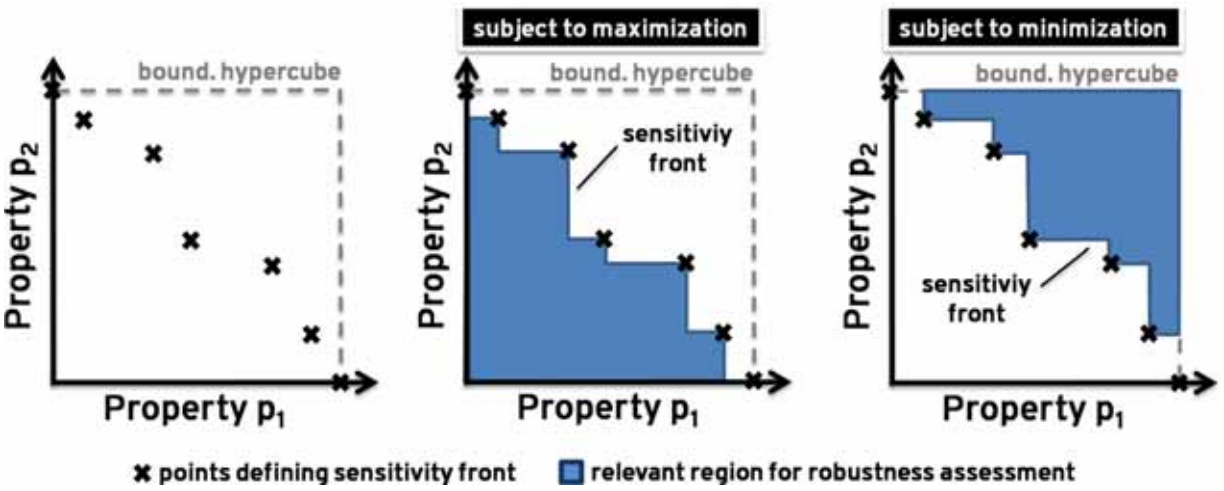



Fig. 4.7: Static design robustness (SDR) for dependent system properties in the two-dimensional case

**Dynamic design robustness.** In the dynamic case the robustness properties of several parameter configuration, representing the possible reconfiguration space, have to be included into the robustness metric. The so-called *dynamic sensitivity front* integrates the multi-dimensional sensitivity fronts of several parameter configurations. Its formal characterization is given in Definition 4.2.6.

**Def. 4.2.6 (Dynamic sensitivity front)** Let $\mathcal{S}$ denote a given system, and $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of system properties. Given a set of possible parameter configurations $\mathcal{C}$ for $\mathcal{S}$ (reconfiguration space), the dynamic sensitivity front is defined as follows[1]:

- If the properties $\mathcal{P}$ are subject to maximization

$$\tilde{\mathcal{F}}^{sens}_{\mathcal{S},\mathcal{C}}(\mathcal{P}) = \Omega^{+}(\bigcup_{c \in \mathcal{C}} \mathcal{F}^{sens}_{\mathcal{S}_c}(\mathcal{P}))$$

- Otherwise, if the properties $\mathcal{P}$ are subject to minimization

$$\tilde{\mathcal{F}}^{sens}_{\mathcal{S},\mathcal{C}}(\mathcal{P}) = \Omega^{-}(\bigcup_{c \in \mathcal{C}} \mathcal{F}^{sens}_{\mathcal{S}_c}(\mathcal{P}))$$

[1]For the definition of $\Omega^{+}$ and $\Omega^{-}$ see Definition 4.1.6.

- The dynamic sensitivity front is contained in the *dynamic bounding hypercube* that is defined as follows:

$$\tilde{\mathcal{H}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}) = \bigcup_{c \in \mathcal{C}} \mathcal{H}_{\mathcal{S}_c}(\mathcal{P})$$

An interpretation of the dynamic sensitivity front is discussed in the example below.

Based on the dynamic sensitivity front, dynamic design robustness can be formalized according to Definition 4.2.7.

**Def. 4.2.7 (Dynamic design robustness for dependent system properties)** Let $\mathcal{S}$ denote a given system, and $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of system properties. Given a set of possible parameter configurations $\mathcal{C}$ for $\mathcal{S}$ (reconfiguration space), and a set of weights $\mathcal{W} = \{w_1, \ldots, w_n\}$ with $\forall_i w_i \geq 1$, the dynamic design robustness of $\mathcal{S}$ with respect to $\mathcal{P}$ is defined as follows:

- If the properties $\mathcal{P}$ are subject to maximization

$$\mathrm{DDR}^{\mathrm{dep}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}, \mathcal{W}) = \tilde{\lambda}^{-}_{\tilde{\mathcal{H}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})}(\tilde{\mathcal{F}}^{sens}_{\mathcal{S},\mathcal{C}}(\mathcal{P}), \mathcal{W})$$

- Otherwise, if the properties $\mathcal{P}$ are subject to minimization

$$\mathrm{DDR}^{\mathrm{dep}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}, \mathcal{W}) = \tilde{\lambda}^{-}_{\tilde{\mathcal{H}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})}\left(\left\{(\bar{b}_1, \ldots, \bar{b}_n)\right\}, \mathcal{W}\right) - \tilde{\lambda}^{+}_{\tilde{\mathcal{H}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})}(\tilde{\mathcal{F}}^{sens}_{\mathcal{S},\mathcal{C}}(\mathcal{P}), \mathcal{W})$$ [1]

An example of how the dynamic design robustness for dependent system properties is evaluated in the two-dimensional case is illustrated in Figure 4.8.

On the left-hand side, two sets of points defining the sensitivity fronts of $p_1$ and $p_2$ for two different parameter configurations $c_1$ and $c_2$ are visualized. The dynamic bounding hypercube is obtained by building the union of the hypercubes bounding the two given sensitivity fronts.

In case that $p_1$ and $p_2$ are subject to maximization, the dynamic sensitivity front is defined by the Pareto-optimal points (in the sense of a maximization problem) contained in the union set of the two given sensitivity fronts. The dynamic sensitivity front can be interpreted as follows: for each point (i.e. property value combination of $p_1$ and $p_2$) that is Pareto-dominated by the dynamic sensitivity front in the sense of a maximization problem, there exist at least one feasible parameter configuration. Hence, the region covered by the dynamic sensitivity

[1] $\bar{b}_1, \ldots, \bar{b}_n$ denote the extreme points of the dynamic bounding hypercube $\tilde{\mathcal{H}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})$

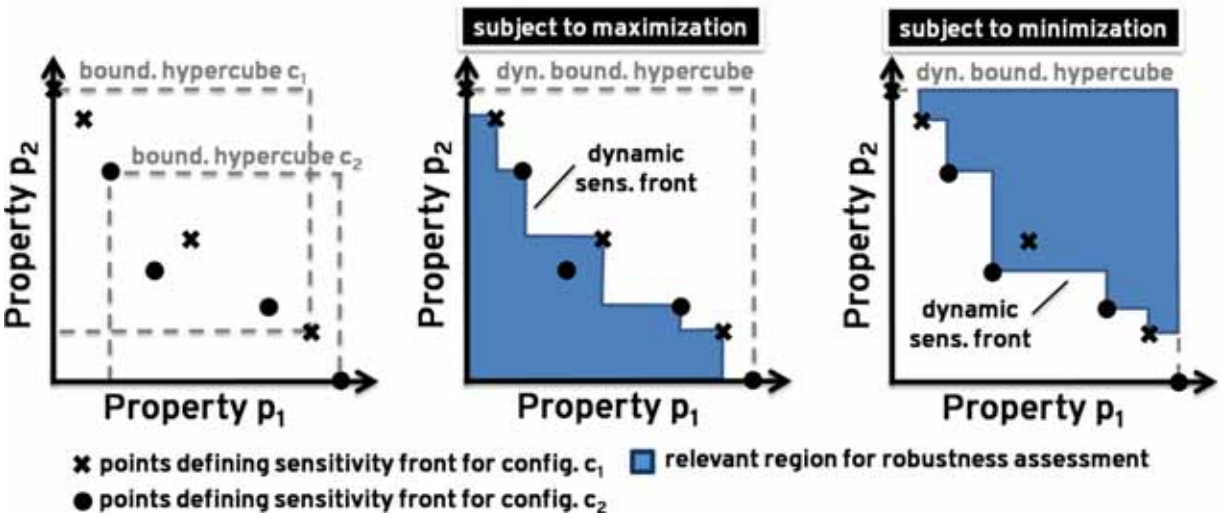



Fig. 4.8: Dynamic design robustness (DDR) for dependent system properties in the two-dimensional case including two parameter configurations $c_1$ and $c_2$

front can be safely included into robustness evaluation. In the shown two-dimensional case, this corresponds to the region below the lower step function, and more generally, in the n-dimensional case, to the inner hypervolume covered by the dynamic sensitivity front inside the dynamic bounding hypercube.

In the opposite case, i.e. if $p_1$ and $p_2$ are subject to minimization, the dynamic sensitivity front is defined by those points of the two given sensitivity fronts that are Pareto-optimal in the sense of a minimization problem. Accordingly, there exist at least one feasible parameter configuration for each point (i.e. property value combination of $p_1$ and $p_2$) that is Pareto-dominated by the dynamic sensitivity front in the sense of a minimization problem. In the two-dimensional case, this region corresponds to the intersection between the dynamic bounding hypercube and the region below the upper step function. In the general case, the outer hypervolume covered by the dynamic sensitivity front must be subtracted from the hypervolume of the dynamic bounding hypercube.

### 4.2.3 Robustness gain through reconfigurability

In this section metrics are derived that quantify the robustness gain achievable through system reconfigurability or dynamic system behavior compared to the static case, where parameters remain fixed during system lifetime. Given these metrics, the benefit of designing reconfigurable system components is explicitly measurable. Consequently, they can help system architects to decide if it is worthwhile investing engineering effort in creating reconfiguration mechanisms. More generally,

by adjusting the reconfiguration space $\mathcal{C}$, system components for which reconfigurability is particularly advantageous, i.e. leading to significant robustness gains, can be identified. Obviously, by this means the engineering effort to conceive robust systems can be efficiently focused.

Like in the previous sections the cases of independent and dependent system properties are distinguished.

**Independent system properties.** Given the formal definitions of static design robustness (Definition 4.2.2) and aggregated dynamic design robustness (Definition 4.2.4) for independent system properties, the robustness gain through reconfigurability is defined as follows.

**Def. 4.2.8 (Robustness gain for independent system properties)** Let $\mathcal{S}$ denote a given system, and $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of independent system properties. Given a set of possible parameter configurations $\mathcal{C}$ for $\mathcal{S}$ (reconfiguration space), a set of weights $\mathcal{W} = \{w_1, \ldots, w_n\}$ with $\forall_i w_i > 0$, and a real number $k$, the robustness gain through system reconfigurability is defined as follows:

$$\mathcal{G}_{\mathcal{S},\mathcal{C}}(\mathcal{P}, \mathcal{W}, k) = \mathrm{ADDR}_{\mathcal{S},\mathcal{C}}(\mathcal{P}, \mathcal{W}, k) - \max_{c \in \mathcal{C}} \{\mathrm{SDR}_{\mathcal{S}_c}(\mathcal{P}, \mathcal{W}, k)\}$$

**Dependent system properties.** For dependent system properties the formal characterization of the robustness gain that can be achieved through system reconfigurability is given in Definition 4.2.9. It is based on the formal definitions of static design robustness (Definition 4.2.5) and dynamic design robustness (Definition 4.2.7) for dependent system properties.

**Def. 4.2.9 (Robustness gain for dependent system properties)** Let system $\mathcal{S}$ denote a given system, and $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of dependent system properties. Given a set of possible parameter configurations $\mathcal{C}$ for $\mathcal{S}$ (reconfiguration space), and a set of weights $\mathcal{W} = \{w_1, \ldots, w_n\}$ with $\forall_i w_i \geq 1$, the robustness gain through system reconfigurability is defined as follows:

$$\mathcal{G}^{\mathrm{dep}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}, \mathcal{W}) = \mathrm{DDR}^{\mathrm{dep}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}, \mathcal{W}) - \max_{c \in \mathcal{C}} \{\mathrm{SDR}^{\mathrm{dep}}_{\mathcal{S}_c}(\mathcal{P}, \mathcal{W})\}$$

## 4.3 Stochastic multi-dimensional sensitivity analysis

In this section a method for multi-dimensional sensitivity analysis based on stochastic exploration techniques is introduced. The main reason why this method is developed as an alternative to existing exact analysis approaches [93, 91] is that its precision is continuously scalable. More precisely, the method yields upper and lower bounds for the sought-after sensitivity front that are iteratively refined during exploration. Note that for the determination of exact sensitivity fronts the presented stochastic approach does not have advantages in terms of computational complexity compared to exact approaches [93, 91]. However, in many cases upper and lower bounds that are sufficient for robustness exploration are easier to determine than exact sensitivity fronts. In Section 4.4.2 it will be shown how these bounds can be used to efficiently approximate and explore the robustness metrics for dependent system properties presented in Section 4.2.2.

Besides its applicability for robustness optimization, sensitivity analysis can be used for further problems in the field of embedded system design (e.g. system dimensioning). For a detailed discussion of sensitivity analysis refer to [89].

Note that the definitions and algorithms given in this section assume that the considered system properties are subject to maximization. However, the method can be modified in a straight-forward manner to also cover system properties that are subject to minimization. Additionally, it is important to mention that the proposed method assumes monotone search spaces. This is not a limitation of the approach. However, in order to cover non-monotone search spaces further measures have to be taken (for a discussion see Section 4.1.1).

### 4.3.1 Analysis idea

In this section multi-dimensional sensitivity analysis is formulated as multi-objective optimization problem, which is solved using the exploration framework and evolutionary search techniques presented in Chapter 3. Consequently, the analysis is based on a stochastic process, hence it is called *stochastic* multi-dimensional sensitivity analysis.

Classical applications for design space exploration in the field of embedded systems design assume the variation of parameter configurations, including scheduling, mapping, etc., to optimize criteria such as timing, power consumption, and buffer sizes. To cover multi-dimensional sensitivity analysis, design space exploration needs to be used differently. Rather than varying parameter configurations during exploration, sys-

tem properties subject to sensitivity analysis are modified, i.e. worst-case core execution times, CPU clock rates, input data rates, etc. Thereby, the optimization objectives consist in, depending on the considered system property types, either the maximization or the minimization of the property values under the condition that the system stays feasible, i.e. all system constraints (e.g. end-to-end deadlines) must be satisfied.

For instance, in the case of three-dimensional $WCET$ sensitivity analysis for three tasks, the search space is composed of their $WCET$ assignments, and the optimization objectives consist in the simultaneous maximization of the latter. If Pareto-optimization is applied to this multi-objective optimization problem, the obtained Pareto-front corresponds to the sought-after sensitivity front representing the boundary between feasible and non-feasible system property combinations.

It is important to mention, that the sensitivity front coverage is controlled by problem independent selector algorithms [12]. For all experiments conducted in this thesis SPEA2 [136] is used as selector. Another promising selector for stochastic multi-dimensional sensitivity analysis is IBEA [135].

Generally, it is possible to use different methods to quickly find points on the sensitivity front (e.g. gradient methods, binary search, etc.). However, in this case the problem of efficiently covering the whole sensitivity front still needs to be addressed. The efficient approximation of Pareto-sets is essential part of modern MOEAs like SPEA2 and IBEA, that dynamically focus the exploration effort on insufficiently covered parts of the sensitivity front using various metrics. SPEA2, for instance, ensures diversified approximation of the sensitivity front through Pareto-dominance-based selection and density approximation. Note that the same approximation quality is usually not reached using simpler strategies (e.g. equidistant sampling), since parts of the sensitivity front with high variance might require more sampling points than, for instance, constant regions.

### 4.3.2 Search space encoding

A system property value combination considered during stochastic multi-dimensional sensitivity analysis is encoded as vector containing one real number entry for each considered property. In the following we refer to such a vector as *individual*.

For instance, in the case of a three-dimensional sensitivity analysis for the system properties $p_1$, $p_2$ and $p_3$, an individual $A$ is represented as three dimensional vector, i.e. $A = (a_1, a_2, a_3)$.

In the following it is assumed that stochastic multi-dimensional sensitivity analysis is performed for $n$ system properties $\mathcal{P} = \{p_1, \ldots, p_n\}$ on a system $\mathcal{S}$ with fixed parameter configuration $c$ (denoted as $\mathcal{S}_c$).

### 4.3.3 Initial population

Algorithm 10 describes the creation of the initial population. In the first part (lines 1 to 4) one-dimensional sensitivity analysis [93] is used to calculate the available slack for each considered system property. The one-dimensional slack values represent the extreme points of the sought-after sensitivity front, and thus describe the bounding hypercube containing all valid system property value combinations. This information is used throughout the whole exploration to considerably reduce the generation of invalid individuals .

In the second part of the algorithm (lines 5 to 10) the rest of the initial population is randomly generated. Thereby, the individuals are uniformly distributed in the search space bounded by the hypercube that was calculated in the first part of the algorithm.

Figure 4.9 shows an example Pareto-front after the creation of the initial population. Note that this example is used throughout this section to visualize the basic concepts of the stochastic multi-dimensional sensitivity analysis, including the search space bounding strategy (Section 4.3.4), and different variation operators (Sections 4.3.5 and 4.3.6).

**Algorithm 10** Initial population

**Require:** individual to be analyzed $(p_1, \ldots, p_n)$, the initial property values $v(p_1), \ldots, v(p_n)$, the initial population size $\alpha > n$
**Ensure:** Initial population $\mathcal{I}$

1: **for** i $\leftarrow$ 1 **to** n **do**
2: $\quad v^+_{\mathcal{S}_c}(p_i) \leftarrow$ COMPUTESLACK$(p_i)$
3: $\quad \mathcal{I} \leftarrow \left(v(p_1), \ldots, v^+_{\mathcal{S}_c}(p_i), \ldots, v(p_n)\right)$
4: **end for**
5: **while** $(|\mathcal{I}| < \alpha)$ **do**
6: $\quad$ **for** i $\leftarrow$ 1 **to** n **do**
7: $\quad\quad r_i \leftarrow$ RANDOM$(\min(v(p_i), v^+_{\mathcal{S}_c}(p_i)), \max(v(p_i), v^+_{\mathcal{S}_c}(p_i)))$
8: $\quad$ **end for**
9: $\quad \mathcal{I} \leftarrow (r_1, \ldots, r_n)$
10: **end while**

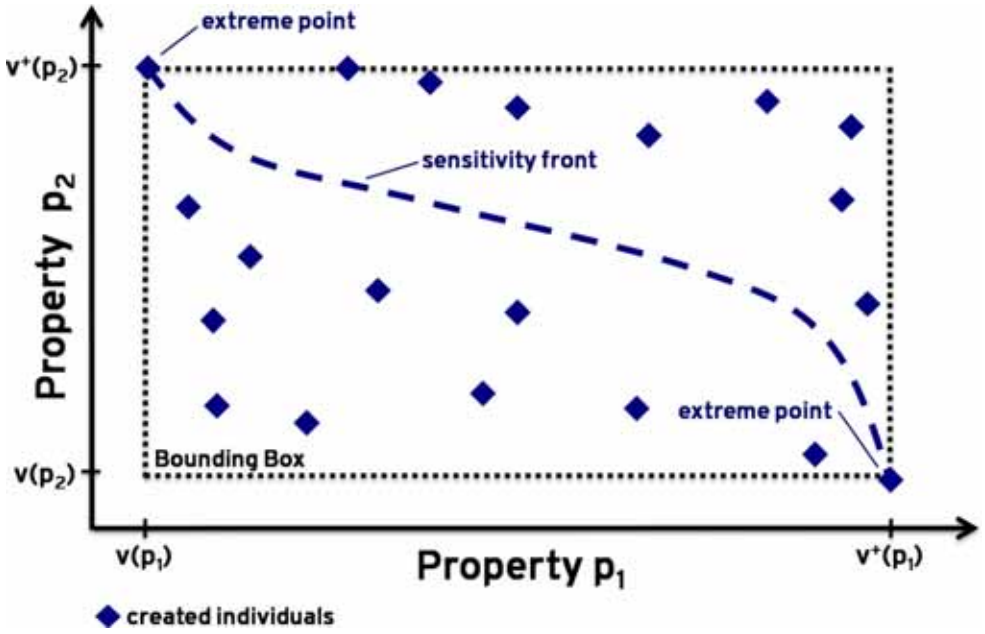



Fig. 4.9: Initial Population

### 4.3.4 Bounding the search space

In this section the search space bounding strategy that is used to efficiently focus the exploration effort, and thus improving analysis speed, is presented. During stochastic sensitivity analysis so-called *bounding Pareto-fronts* are maintained and updated (Definition 4.3.1). Thereby, it is ensured during the whole approximation process that the sought-after exact sensitivity front lies between the *bounding working Pareto-front* and the *bounding non-working Pareto-front*. This property is exploited by the variation operators presented in the following sections to prevent exploration from generating and evaluating individuals not directly improving approximation quality.

**Def. 4.3.1 (Bounding Pareto-fronts)** Let $\mathcal{S}_c$ denote the system $\mathcal{S}$ with parameter configuration $c$. For a given set of system properties subject to maximization $\mathcal{P} = \{p_1, \ldots, p_n\}$ the bounding working and non-working Pareto-fronts are defined as follows:

1. The bounding working Pareto-front $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ is defined as a set of vectors $\vec{f}^{w}_1, \ldots, \vec{f}^{w}_n$ with the following properties:

   (a) $\forall_i \vec{f}^{w}_i \in \mathcal{H}_{\mathcal{S}_c}(\mathcal{P})$

   (b) $\Omega^{+}(\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})) = \mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$

   (c) $\forall_i \exists \vec{x} \in \mathcal{F}^{sens}_{\mathcal{S}_c}(\mathcal{P}) : \vec{x} \triangleright_{+} \vec{f}^{w}_i$

2. The bounding non-working Pareto-front $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$ is defined as a set of vectors $\vec{f}^{nw}_1, \ldots, \vec{f}^{nw}_n$ with the following properties:

   (a) $\forall_i \vec{f}^{nw}_i \in \mathcal{H}_{\mathcal{S}_c}(\mathcal{P})$

   (b) $\Omega^{-}(\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})) = \mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$

   (c) $\forall_i \exists \vec{x} \in \mathcal{F}^{sens}_{\mathcal{S}_c}(\mathcal{P}) : \vec{x} \triangleright_{-} \vec{f}^{nw}_i$

The sets $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ and $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$ are dynamically updated during exploration after each processed generation, i.e. directly after offspring evaluation. More precisely, for each feasible (infeasible) individual $i$ created during variation it is checked whether it is Pareto-optimal with respect to the individuals contained in $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ ($\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$). If this is the case, $i$ is added to $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ ($\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$) and all individuals Pareto-dominated by $i$ in the sense of a maximization (minimization) problem are removed.

The region that is contained between the bounding Pareto-fronts $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ and $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$ is called *relevant region*. It is easy to understand, that the sought-after exact sensitivity front lies inside the relevant region. Consider, for instance, the situation for two system properties subject to maximization visualized in Figure 4.10.

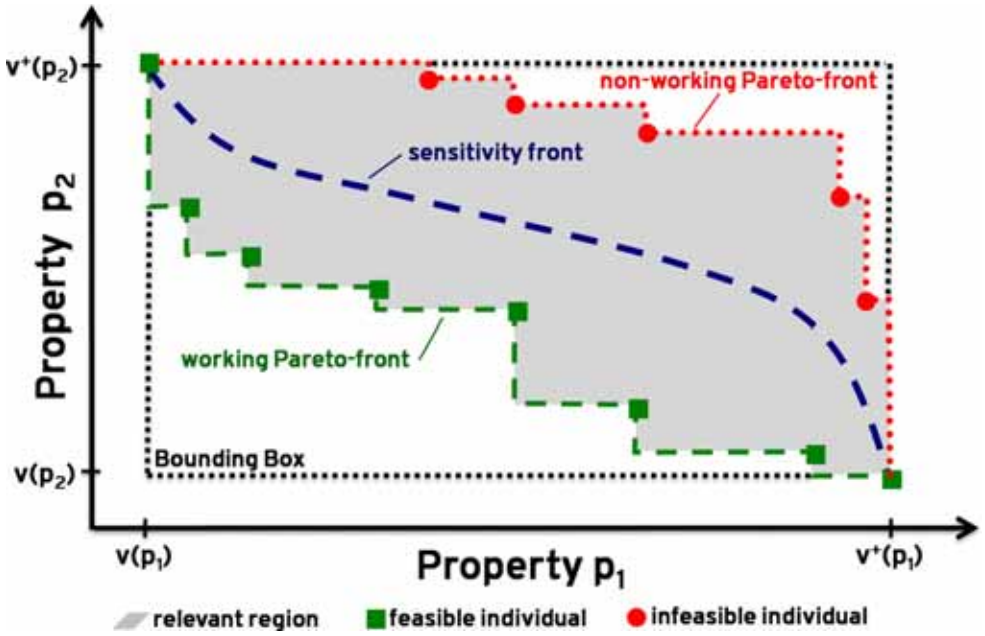



Fig. 4.10: Relevant region for two system properties subject to maximization

An individual lying below the bounding working Pareto-front cannot be part of the sensitivity front, since at least one individual on the bounding working Pareto-front has higher, and thus better, values for all considered system properties. Also, an individual lying above the bounding non-working Pareto-front cannot be part of the sensitivity

front, since there exist at least one infeasible individual on the bounding non-working Pareto-front, which has smaller values for all considered system properties. Consequently, the individual in question is infeasible as well.

Note that if the stochastic sensitivity analysis runs long enough, the bounding Pareto-fronts $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ and $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$ ultimately converge into the sought-after sensitivity front. The convergence behavior of the stochastic sensitivity analysis approach is discussed in Section 4.3.8.

The variation operators guiding the exploration process, which are presented in the following sections, use Algorithm 11 to ensure that generated offsprings are placed inside the relevant region, and thus directly improve approximation quality. Obviously, this leads to decreased exploration time for the same approximation quality. Note that by configuring the input variable *max*, Algorithm 11 can be used either for system properties subject to minimization ($max = false$) or maximization ($max = true$).

**Algorithm 11** isInRR

**Require:** $k$-dimensional vector $v$, bounding working Pareto-front $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$, bounding non-working Pareto-front $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$, type of considered properties $max \in \{true, false\}$

**Ensure:** *true*: if $v$ lies inside the relevant region, *false*: otherwise

1: $workingOK \leftarrow !\textsc{dominatedBy}(v, \mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P}), max)$
2: $nonWorkingOK \leftarrow !\textsc{dominatedBy}(v, \mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P}), !max)$
3: **return** $workingOK$ && $nonWorkingOK$

In practice, the variation operators are repeated until the generated offspring is situated inside the relevant region. This approach was chosen to preserve the randomized nature of the variation operators. Note that the process is stopped after a maximum number of tries if no valid offspring is generated. In this case the last generated offspring is returned. Of course, using this best effort strategy it is not guaranteed that all generated offsprings are situated inside the relevant region. However, generated invalid offspring do not need to be evaluated[1], which increases exploration speed compared to approaches that do not exploit knowledge about bounding Pareto-fronts (see Section 4.3.8).

In the following sections crossover and mutation operators responsible for improving the approximation quality of the sought-after sensitivity

[1]System evaluation has a much higher computational complexity compared to the proposed variation operators consisting of few basic arithmetic operations.

front are presented. For a better comprehension, the different variation operators are applied subsequently on the initial population that was determined in Section 4.3.3 for the example sensitivity front visualized in Figure 4.9). This process is visualized in Figures 4.12, 4.13, and 4.14. Note that the initial situation, i.e. after the creation of the random initial population, is visualized in Figure 4.12a, whereas the final approximation, i.e. the resulting bounding Pareto-fronts after the variation process, is visualized in Figure 4.14b.

### 4.3.5 Crossover operators

In this section, two crossover operators that are used to locally refine the approximation of the sought-after sensitivity front are presented. The first operator is based on standard randomized techniques and generates uniformly distributed offsprings in the neighborhood of the parent individuals. The second operator is adapted to the problem structure, and hence heuristic in nature. Note that the benefit (in terms of approximation quality and convergence speed) of using the proposed problem-dependent heuristic variation operators is discussed in Section 4.3.8.

**Random crossover.** The random crossover operator takes as input two parent individuals $A_1$ and $A_2$ and creates two offspring individuals $B_1$ and $B_2$. Random crossover is a problem-independent variation operator pursuing a very simple recombination strategy: the two parent individuals span a hypercube in which the two offspring individuals are randomly placed. This simple strategy leads to the local refinement of the sensitivity front approximation in the parent individuals' neighborhood.

A pseudo-code representation of the random crossover operator is given in Algorithm 12. Note that random crossover uses Algorithm 11 to ensure that the generated offspring individuals are placed inside the relevant region (line 8). More precisely, the crossover process is repeated until either both offspring individuals are situated inside the relevant region or the maximum number of attempts $iter_{max}$ is reached.

Figure 4.13a visualizes the functionality of the random crossover operator based on the example used throughout this section.

**Generalized mean crossover.** In contrast to the generic random crossover operator, the generalized mean crossover operator is more focused and tailored to approximate various possible characteristics of the sought-after multi-dimensional sensitivity front. It is, therefore, heuristic in nature. The generalized mean crossover operator takes as input two parent individuals $A_1$ and $A_2$ from which it generates two offspring

**Algorithm 12** Random crossover

**Require:** parent individuals $A_1 = (a_{11}, \ldots, a_{1n})$ and $A_2 = (a_{21}, \ldots, a_{2n})$, type of considered properties $max$ (boolean), bounding working Pareto-front $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$, bounding non-working Pareto-front $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$, maximum number of attempts to reach relevant region $iter_{max}$

**Ensure:** offspring individuals $B_1 = (b_{11}, \ldots, b_{1n})$ and $B_2 = (b_{21}, \ldots, b_{2n})$

1: **for** i $\leftarrow$ 1 **to** 2 **do**
2: $iter \leftarrow 0$
3: **repeat**
4: $iter \leftarrow iter + 1$
5: **for** j $\leftarrow$ 1 **to** n **do**
6: $b_{ij} \leftarrow \textsc{random}(\min(a_{1j}, a_{2j}), \max(a_{1j}, a_{2j}))$
7: **end for**
8: **until** $\textsc{isInRR}(B_i, \mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P}), \mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P}), max)$ **or** $iter > iter_{max}$
9: **end for**

individuals $B_1$ and $B_2$ by using the generalized mean function (Definition 4.3.2).

**Def. 4.3.2 (Generalized mean)** For positive numbers $x_1, \ldots, x_n$ the $k$-th mean is defined as follows:

$$M_k(x_1, \ldots, x_n) = \sqrt[k]{\frac{1}{n} \sum_{i=1}^{n} x_i^k}$$

Special cases: $k \to -\infty$ : $\min(x_1, \ldots, x_n)$; $k = -1$: harmonic mean; $k \to 0$: geometric mean; $k = 1$: arithmetic mean; $k = 2$: quadratic mean; $k \to \infty$ : $\max(x_1, \ldots, x_n)$.

A pseudo-code representation the generalized mean crossover is given in Algorithm 13.

During crossover the generalized mean function is applied coordinate-wise on the vector representations of the parent individuals (line 7). For a better adaptation to the problem structure, the values calculated according to the generalized mean formula are slightly modified (lines 8-11). The aim of this modification is to obtain a $n$-dimensional curve connecting the parent individuals. This can be achieved, for instance, by mirroring the calculated values at the bisector in each dimension for that the first parent has a higher coordinate value compared to the second

**Algorithm 13** Generalized mean crossover

**Require:** parent individuals $A_1 = (a_{11}, \ldots, a_{1n})$ and $A_2 = (a_{21}, \ldots, a_{2n})$, $k_{min}$ and $k_{max}$ with $k_{max} \geq k_{min}$, type of considered properties $max$ (boolean), bounding working Pareto-front $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$, bounding non-working Pareto-front $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$, maximum number of attempts to reach relevant region $iter_{max}$

**Ensure:** offspring individuals $B_1 = (b_{11}, \ldots, b_{1n})$ and $B_2 = (b_{21}, \ldots, b_{2n})$

**for** i $\leftarrow$ 1 **to** 2 **do**
  $k \leftarrow$ RANDOM$(k_{min}, k_{max})$
  $iter \leftarrow 0$
  **repeat**
    $iter \leftarrow iter + 1$
    **for** j $\leftarrow$ 1 **to** n **do**
      $b_{ij} \leftarrow M_k(a_{1j}, a_{2j})$
      **if** $a_{1j} > a_{2j}$ **then**
        $temp \leftarrow \min(a_{1j}, a_{2j}) + \frac{|a_{1j} - a_{2j}|}{2}$
        $b_{ij} \leftarrow temp - (b_{ij} - temp)$
      **end if**
    **end for**
  **until** isInRR$(B_i, \mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P}), \mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P}), max)$ **or** $iter > iter_{max}$
**end for**

parent. Figure 4.11 visualizes the difference between the original and the modified generalized mean function in the two-dimensional case.

The variation strategy of the generalized mean crossover operator is based on the assumption that both parent individuals are situated close to the sought-after sensitivity front. Using the modified generalized mean function a connecting curve between the parent individuals $A$ and $B$ is created that, depending on the chosen $k$ value, approximates different possible sensitivity front characteristics:

- If $k = 1$ is chosen the arithmetic mean between $A$ and $B$ is obtained. This corresponds to a linear characteristic of the sensitivity front, which can, for instance, be observed in the case of load dependent system properties.
- If $k < 1$ is chosen a convex characteristic of the sensitivity front is approximated.
- If $k > 1$ is chosen a concave characteristic of the sensitivity front is approximated.

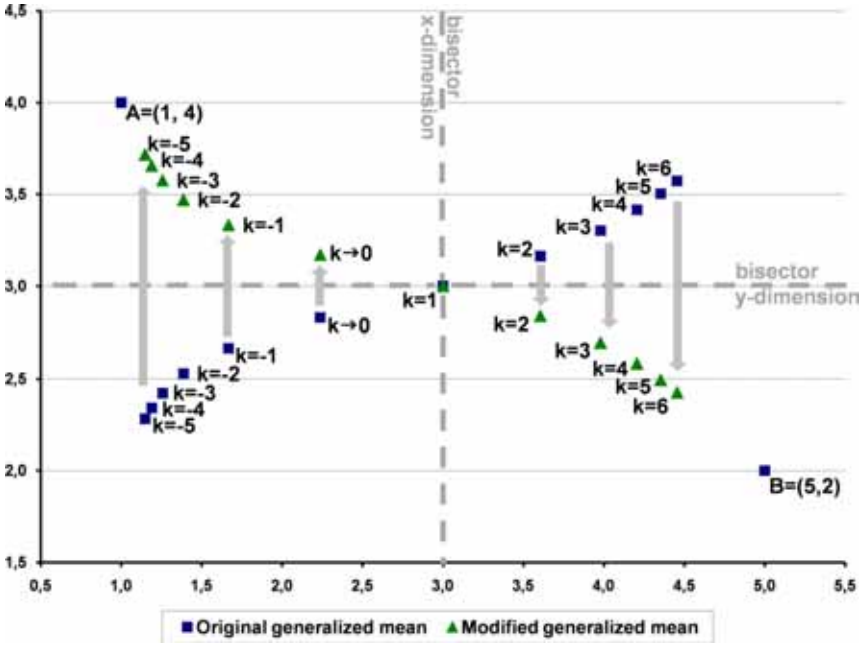



Fig. 4.11: Coordinate-wise generalized mean including the modified version used by Algorithm 13 between two points, A and B, for different k. For the visualized case, the modified generalized mean is obtained by mirroring the original points at the bisector in the y-dimension (A has a higher coordinate value than B). For the x-dimension no modifications have to be performed (A has a lower coordinate value than B).

Like the random crossover operator also the generalized mean crossover operator uses Algorithm 11 to ensure that the generated offspring individuals are situated inside the relevant region (line 13). Figure 4.13b visualizes the functionality of the random crossover operator based on the example utilized throughout this section.

### 4.3.6 Mutation operators

The described crossover operators lead to the local convergence of the bounding Pareto-fronts towards the sought-after sensitivity front. In other words, they approximate the sensitivity front "between" individuals considered during evolutionary exploration. Of course, it is possible that the variety of the initial population is insufficient to cover the whole sensitivity front by only using the proposed crossover operators. Additionally, exploration may get stuck in sub-regions of the front. For these reasons mutation operators are introduced in this section enabling the approximation process to break out of sub-regions and to cover unexplored parts of the sensitivity front.

Two different mutation operators are proposed. The first operator is fully randomized and generates offsprings that are uniformly distributed in the neighborhood of the parent individual. Thereby, its Euclidean

range is dynamically adapted to match the convergence progress of the approximation process. The second operator is heuristic in nature. It exploits knowledge about the bounding Pareto-fronts (see Section 4.3.4) and directly supports their convergence.

**Adaptive range mutation.** The adaptive range mutation operator takes as input one parent individual $A$ and generates one offspring individual $B$. The basic idea, borrowed from standard evolutionary variation operators, is to randomly place the offspring individual in the region surrounding the parent individual. However, due to the nature of the problem formulation and the utilized search space bounding technique (see Section 4.3.4), minimum and maximum Euclidean ranges can be derived that adaptively bound this region according to the global approximation progress. Both ranges are based on distance measures between the bounding working and the bounding non-working Pareto-fronts (Definition 4.3.3).

**Def. 4.3.3 (Average and maximum front distances)** Given the bounding working Pareto-front $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ and the bounding non-working Pareto-front $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$, the average and maximum front distances $\mathcal{D}_{avg}$ and $\mathcal{D}_{max}$ are defined as follows:

$$\mathcal{D}_{avg} = \frac{\sum_{v \in \mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})} \min_{w \in \mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})} \|v - w\|_2}{|\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})|}$$

$$\mathcal{D}_{max} = \max_{v \in \mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})} \left\{ \min_{w \in \mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})} \|v - w\|_2 \right\}$$

Following rules are applied for the generation of the offspring individual:

1. The Euclidean distance between the offspring individual and the parent individual must be greater than $\mathcal{D}_{avg}$. This ensures a minimum improvement of the approximation quality through the offspring individual.
2. The Euclidean distance between the offspring individual and the parent individual must be smaller than $\mathcal{D}_{max}$. This reasonably limits the mutation range dependent on the current global approximation progress.

A pseudo-code representation of the adaptive range mutation operator is given in Algorithm 14.

**Algorithm 14** Adaptive range mutation

**Require:** parent individual $A = (a_1, \ldots, a_n)$, average and maximum front distances $\mathcal{D}_{avg}$ and $\mathcal{D}_{max}$, type of considered properties $max$ (boolean), bounding working Pareto-front $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$, bounding non-working Pareto-front $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$, maximum number of attempts to reach relevant region $iter_{max}$

**Ensure:** offspring individual $B = (b_1, \ldots, b_n)$

1: $iter \leftarrow 0$
2: **repeat**
3: $iter \leftarrow iter + 1$
4: $B \leftarrow A$
5: $dist \leftarrow \textsc{random}(\mathcal{D}_{avg}, \mathcal{D}_{max})$
6: $order[] \leftarrow \textsc{shuffle}(1, \ldots, n)$
7: **for** i $\leftarrow$ 1 **to** n **do**
8: $exact \leftarrow 0$
9: **for** j $\leftarrow$ 1 **to** i-1 **do**
10: $exact \leftarrow exact + (a_{order[j]} - b_{order[j]})^2$
11: **end for**
12: $exact \leftarrow \sqrt{dist^2 - exact}$
13: **if** $i < n$ **then**
14: $x \leftarrow \textsc{random}(0, exact)$
15: **else**
16: $x \leftarrow exact$
17: **end if**
18: $bool \leftarrow \textsc{random}(true, false)$
19: **if** *bool* **then**
20: $b_{order[i]} \leftarrow \min(a_{order[i]} + x, v^{+}_{\mathcal{S}_c}(p_i))$
21: **else**
22: $b_{order[i]} \leftarrow \max(a_{order[i]} - x, v(p_i))$
23: **end if**
24: **end for**
25: **until** $\textsc{isInRR}(B, \mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P}), \mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P}), max)$ **or** $iter > iter_{max}$

According to the above defined rules, the generated offspring individual is placed with minimum distance $\mathcal{D}_{avg}$ inside the n-dimensional sphere defined by $\mathcal{D}_{max}$ around the parent individual. This is achieved with some simple computations.

First, the offspring individual is created as a copy of the parent individual and a random target distance *dist* between offspring and parent individuals is chosen in the interval $[\mathcal{D}_{avg}, \mathcal{D}_{max}]$ (lines 4-5). This target distance is achieved by iteratively adding (subtracting) random values to

(from) the offspring individual's coordinates in random order. Thereby, the maximum absolute difference applicable to an arbitrary coordinate $b_k$ is updated in every iteration by solving the equation $dist = \sqrt{b_1^2, \ldots, b_n^2}$, taking into account already performed modifications (lines 8-12). Afterwards, a random value is chosen between 0 and the calculated maximum value (line 14) and added (subtracted) from (to) the corresponding coordinate (lines 18-23). In the last iteration the remaining coordinate is modified by the maximum admissible value to achieve the desired target distance between offspring and parent individuals (line 16).

Note that in some cases the offspring individual might not be placed correctly inside the n-dimensional sphere defined by $\mathcal{D}_{max}$, but instead on the edge of the bounding hypercube. The reason for this behavior is that the adaptive range mutation operator ensures that the minimum and maximum values for each property, which are calculated during the creation of the initial population, are respected (lines 20 and 22).

Like the previously presented operators, also adaptive range mutation uses Algorithm 11 to ensure that the generated offspring individual is situated inside the relevant region (line 25).

Figure 4.14a visualizes the functionality of the adaptive range mutation operator based on the example utilized throughout this section.

**Front convergence mutation.** The front convergence mutation operator is heuristic in nature, and exploits that the sought-after sensitivity front is contained between the bounding working and the bounding non-working Pareto-fronts (see Section 4.3.4). Like the adaptive range mutation operator it takes as input one parent individual $A$ to create one offspring individual $B$.

The front convergence mutation operator distinguishes two cases to directly increase the convergence speed of the bounding Pareto-fronts:

- If the parent individual is *feasible*, the front convergence mutation operator selects the $k$ closest individuals on the bounding non-working Pareto front $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$.

- If the parent individual is *infeasible*, the front convergence mutation operator selects the $k$ closest individuals on the bounding working Pareto front $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$.

In both cases one of the selected individuals is randomly chosen, and the offspring individual is randomly placed on the straight line connecting the parent individual with the chosen individual. Since in most cases the parent individual is situated close or directly on one of the bounding Pareto-fronts, the created offspring individual is placed with high prob-

ability inside the relevant region, and thus directly supports the local convergence of the two bounding Pareto-fronts.

A pseudo-code representation of the front convergence mutation operator is given by Algorithm 15.

---

**Algorithm 15** Front convergence mutation

---

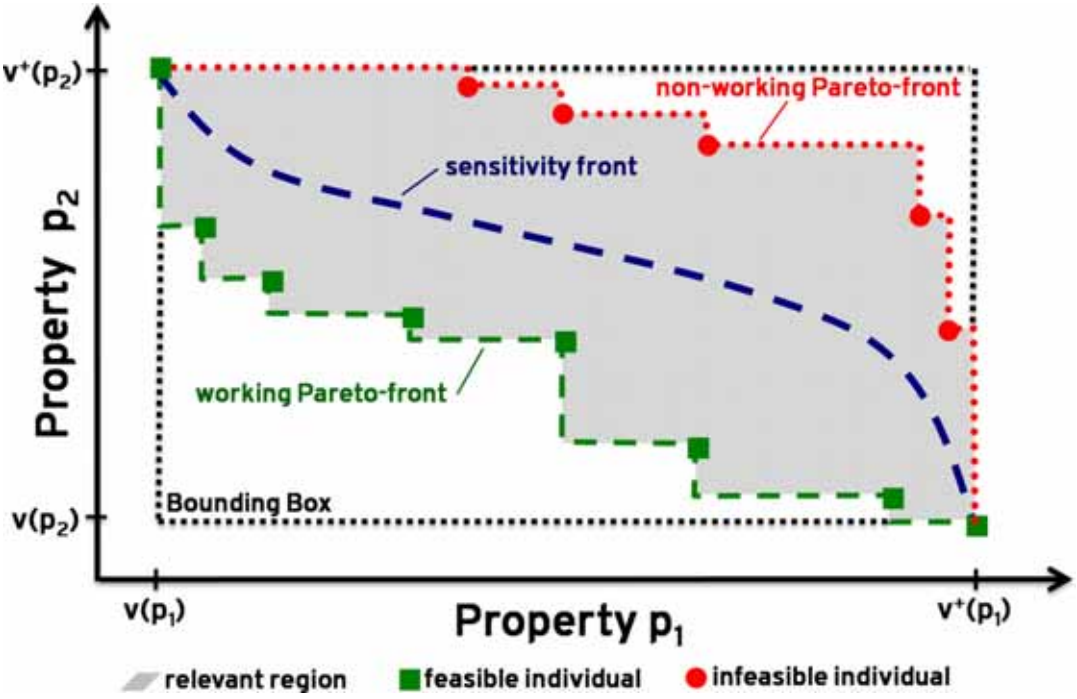
**Require:** parent individual $A = (a_1, \ldots, a_n)$, number of considered closest individuals $k$, type of considered properties $max$ (boolean), bounding working Pareto-front $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$, bounding non-working Pareto-front $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$, maximum number of attempts to reach relevant region $iter_{max}$

**Ensure:** offspring individual $B = (b_1, \ldots, b_n)$

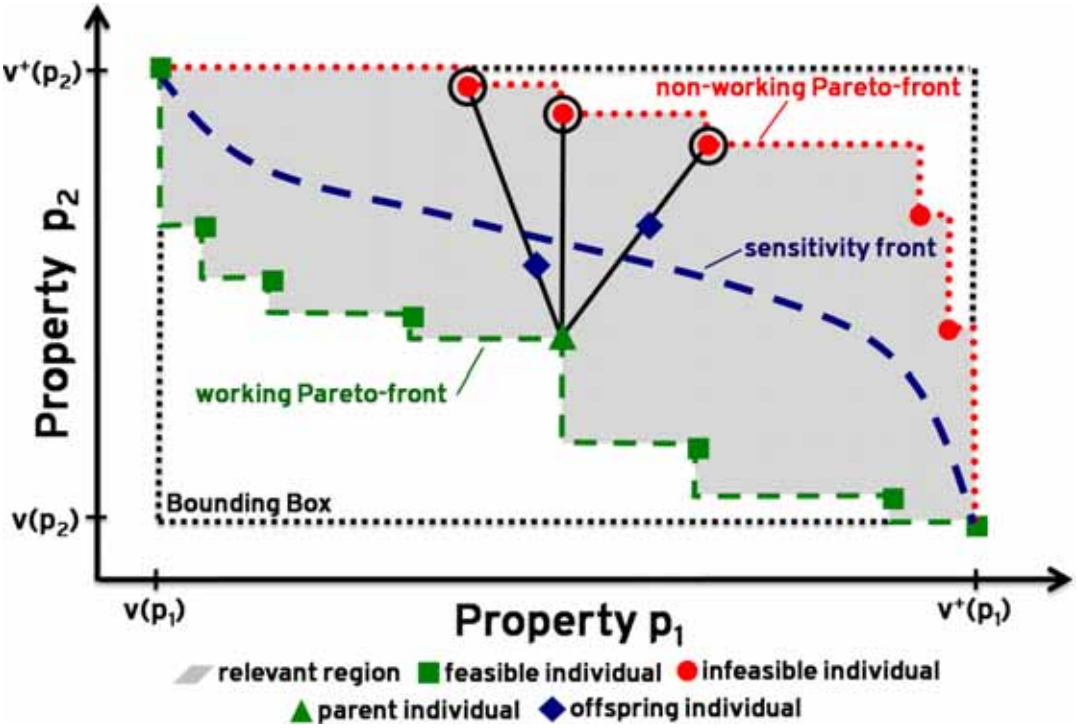
1: **if** $\textsc{eval}_{\mathcal{S}_c}(\mathcal{P}, A)$ **then**
2: $\quad \mathcal{F} \leftarrow \mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$
3: **else**
4: $\quad \mathcal{F} \leftarrow \mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$
5: **end if**
6: $Map_{dist} \leftarrow \emptyset$
7: **for all** $f = (f_1, \ldots, f_n) \in \mathcal{F}$ **do**
8: $\quad d \leftarrow \sqrt{\sum_{j=1}^{n} (f_j - a_j)^2}$
9: $\quad Map_{dist} \leftarrow (d, f)$
10: **end for**
11: $\textsc{sortByKeys}(Map_{dist})$
12: $iter \leftarrow 0$
13: **repeat**
14: $\quad iter \leftarrow iter + 1$
15: $\quad r \leftarrow \textsc{random}(1, k)$
16: $\quad f = (f_1, \ldots, f_n) \leftarrow \textsc{getValues}(Map_{dist})[r]$
17: $\quad factor \leftarrow \textsc{random}(0, 1)$
18: $\quad$ **for** i $\leftarrow$ 1 **to** n **do**
19: $\quad\quad b_i \leftarrow \min(a_i, f_i) + factor \times \mid a_i - f_i \mid$
20: $\quad$ **end for**
21: **until** $\textsc{isInRR}(B, \mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P}), \mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P}), max)$ **or** $iter > iter_{max}$

---

First, the algorithm checks whether or not the parent individual is feasible to select the appropriate target Pareto-front $\mathcal{F}$ (lines 1-5). Note that the computationally expensive evaluation function EVAL (see Definition 4.1.2 in Section 4.1.1) does not need to be executed at this point. The reason is that the parent individual was already evaluated in one of the previous generations. The call of EVAL can, thus, be replaced by a simple lookup in a table containing the status of all already evaluated

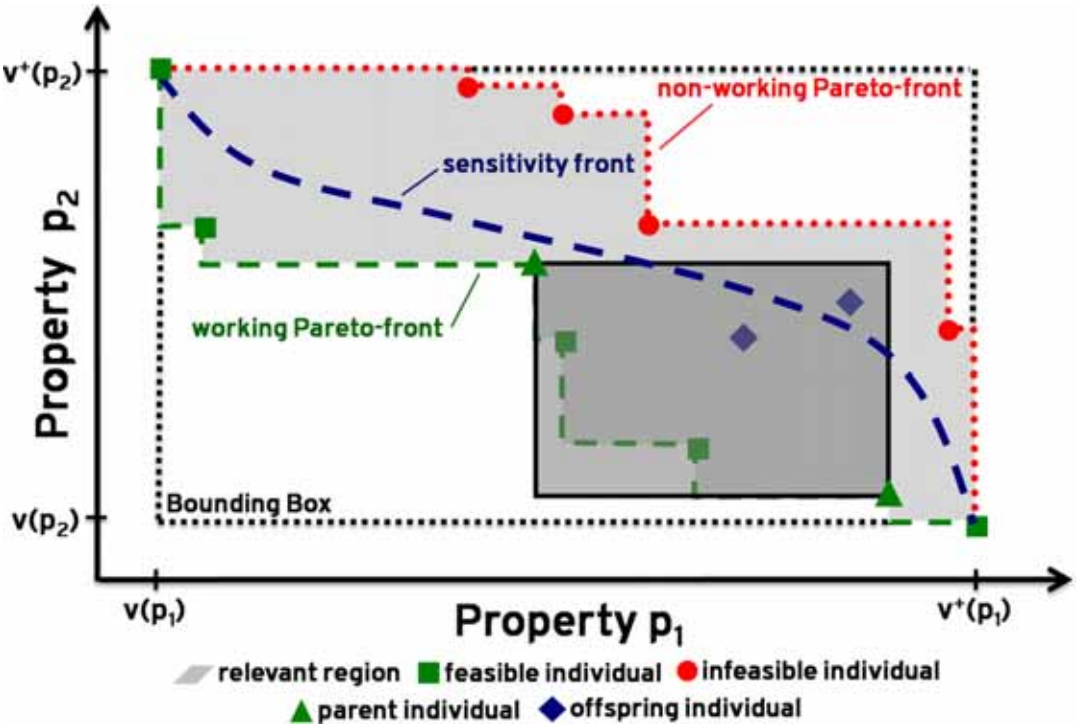



(a) Situation after creation of the initial population for the considered example sensitivity front (compare Figure 4.9 in Section 4.3.3)

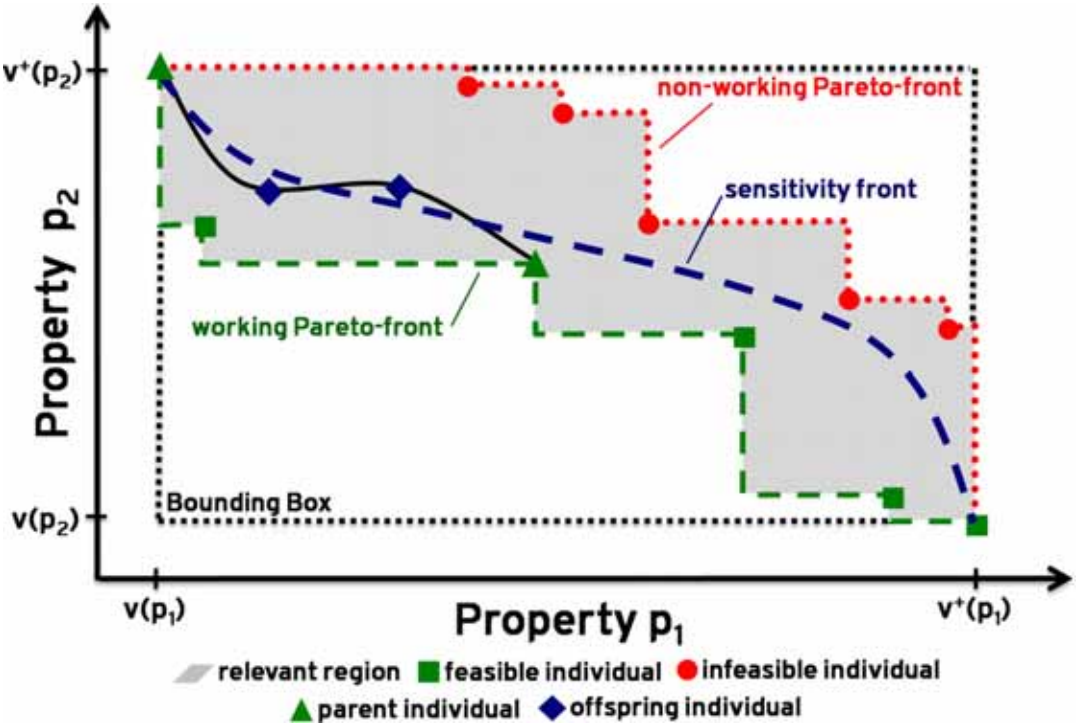



(b) Front convergence mutation applied two times to the same (feasible) parent individual. First, the three (configurable) closest individuals on the non-working Pareto-front are selected. Then, for both applications of the mutation operator, one of the selected individuals is randomly chosen, and the offspring individual is randomly placed on the straight line connecting the parent individual with the chosen individual.

Fig. 4.12: Variation operators of stochastic sensitivity analysis (part 1)

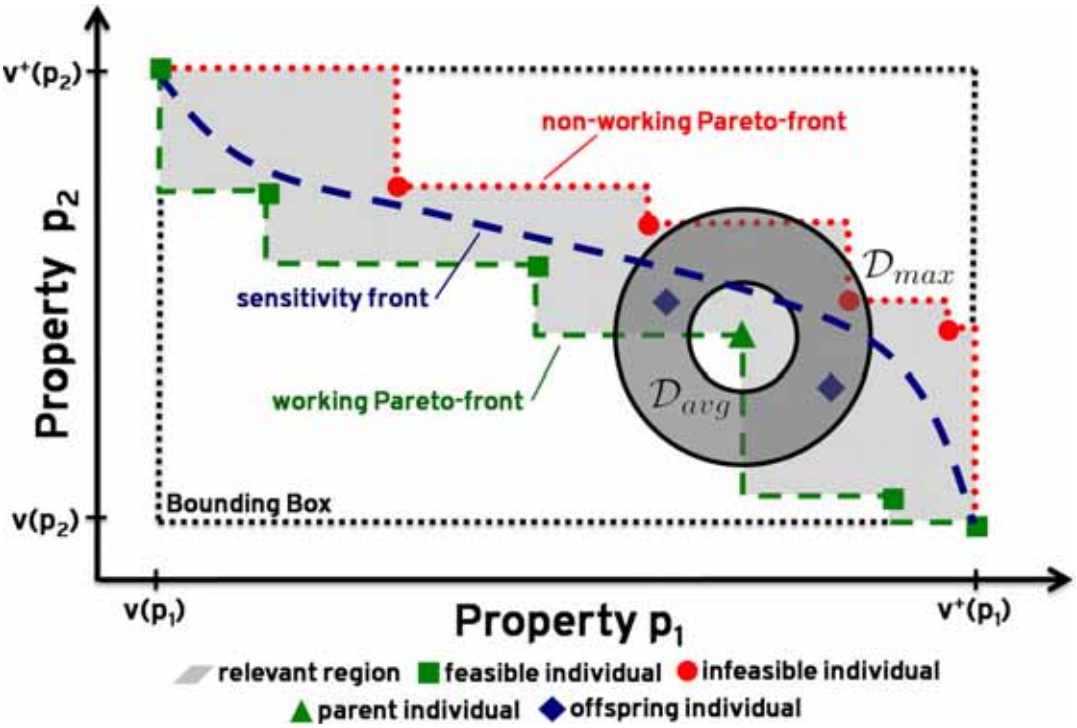



(a) Random crossover. The two offspring individuals are randomly placed in the bounding box (visualized in gray) defined by the two parent individuals.

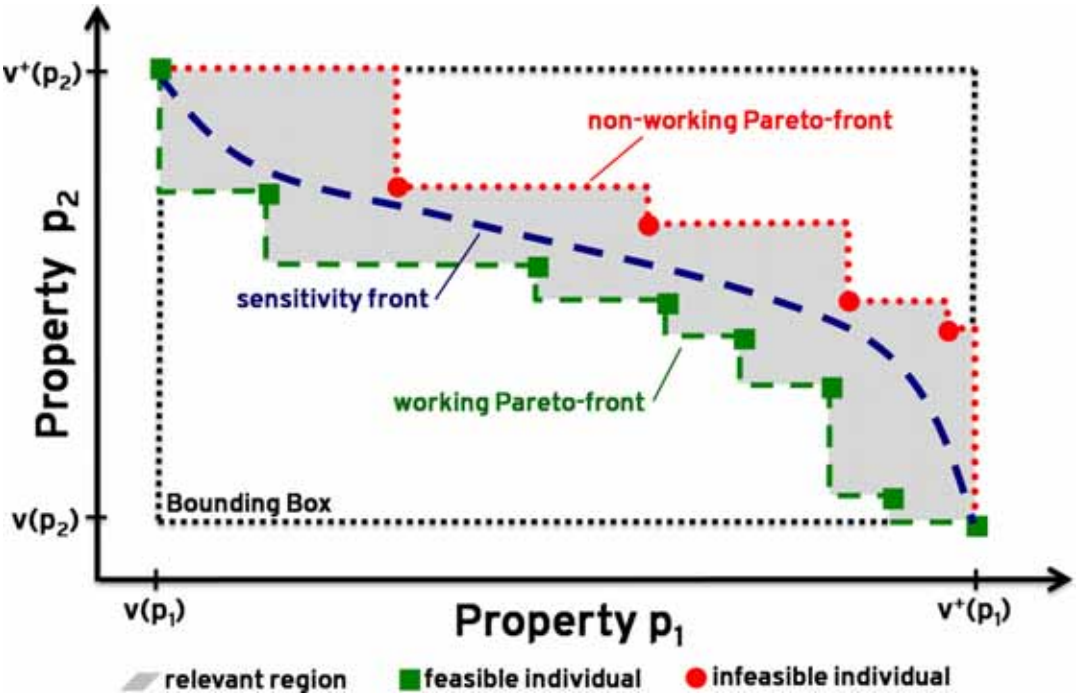



(b) Generalized mean crossover. The black solid line connecting the two parent individuals represents the set of possible offspring individuals. More precisely, each different $k$ (compare Definition 4.3.2) represents a different offspring individual lying on this line.

Fig. 4.13: Variation operators of stochastic sensitivity analysis (part 2)




(a) Adaptive range mutation applied two times to the same parent individual. Offspring individuals are randomly placed inside the area visualized in Gray. Note that this area is delimited by the average and maximum distances ($\mathcal{D}_{avg}$ and $\mathcal{D}_{max}$) between the bounding working and non-working Pareto-fronts (compare Definition 4.3.3).




(b) Final approximation of the example sensitivity front after application of the variation operators as visualized in Figures 4.12b, 4.13a, 4.13b, and 4.14a

Fig. 4.14: Variation operators of stochastic sensitivity analysis (part 3)

individuals. If the parent individual is feasible $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$ is selected as target Pareto-front, in the opposite case $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ is selected.

Afterwards, the Euclidean distances between the parent individual and all individuals contained in $\mathcal{F}$ are calculated and stored in $Map_{dist}$ (lines 6-10). Thereby, the calculated distances are used as keys and the considered individuals represent the mapped values. Note that if several individuals in $\mathcal{F}$ have the same distance to the parent individual, only the last checked individual is stored in $Map_{dist}$.

Finally, $Map_{dist}$ is sorted by keys (line 11), one of the closest $k$ individuals is randomly chosen (lines 15-16), and the offspring individual is randomly placed on the straight line connecting the parent individual with the chosen individual (lines 17-20).

Like the previously presented operators, also front convergence mutation uses Algorithm 11 to ensure that the generated offspring individual is situated inside the relevant region (line 21).

Figure 4.12b visualizes the functionality of the front convergence mutation operator based on the example utilized throughout this chapter.

### 4.3.7 Limiting the search resolution

The sought-after exact sensitivity front is defined in a continuous space. Obviously, to efficiently approximate the sensitivity front with the proposed stochastic technique, the search resolution must be limited. One effect that may occur with unrestricted search resolution is the clustering of generated individuals in a small area of the search space, with only small difference in property values. Obviously, this has negative effects on diversity, and thus approximation quality.

Figure 4.15 visualizes the chosen approach for adjusting the search resolution.

Before stochastic sensitivity analysis is started a search resolution $r$ must be chosen. This resolution defines the number of possible different property values in each dimension. Hence, the number of possible individuals in the search space during n-dimensional sensitivity analysis is equal to $r^n$. Based on the chosen resolution and the search ranges defined by the initial and extreme property values in each dimension (compare Definition 4.1.4 in Section 4.1.1), a particular step width is calculated for each dimension: $s_n = \frac{|v^+(p_n) - v(p_n)|}{r}$. These step widths define the set of valid individuals that may be generated during stochastic sensitivity analysis (marked with dots in Figure 4.15). The proposed variation operators do not directly respect this discretization of the search space. However, after variation, all generated invalid individuals (marked with

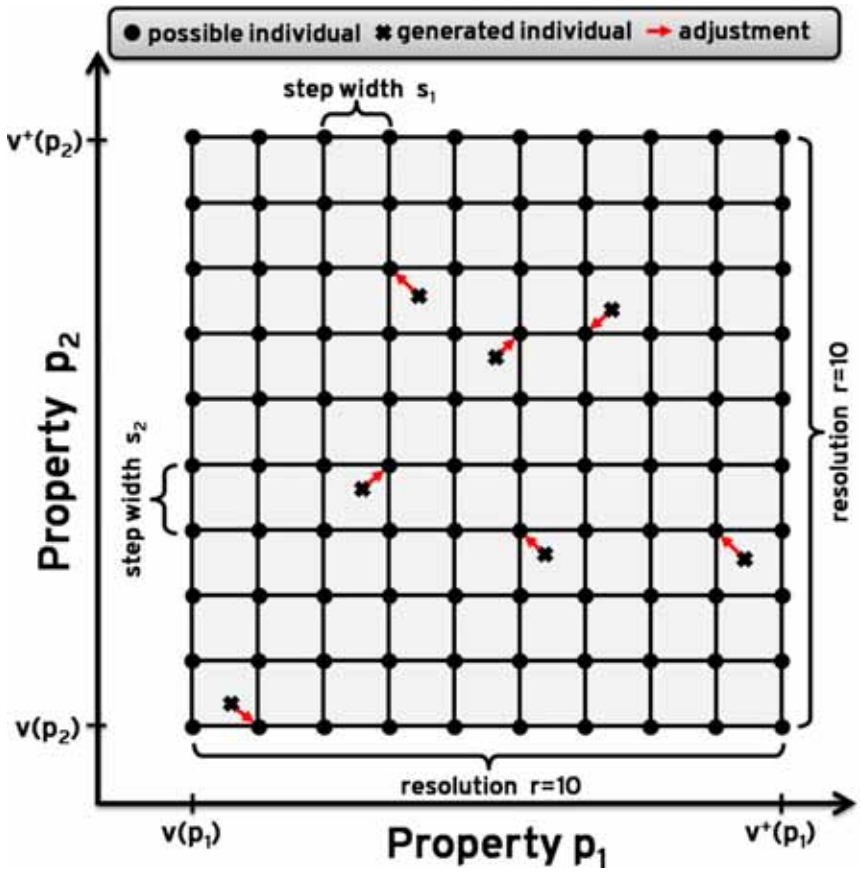



Fig. 4.15: Adjusting the search resolution of the stochastic sensitivity analysis

crosses in Figure 4.15) are adjusted to match the closest valid individual (in the sense of Euclidean distance).

Note that smaller resolutions lead to quicker searches with lower analysis precisions, whereas higher resolutions increase analysis precisions at the expense of higher analysis run-times. Typical search resolutions range from 50 to 500.

### 4.3.8 Approximation quality and convergence behavior

In this section the approximation quality and convergence behavior of the stochastic sensitivity analysis approach are discussed. The goal of this discussion is to get an idea of the required run-time for the approach to calculate sensitivity front approximations (i.e. working and non-working bounding Pareto-fronts) that are sufficiently precise for robustness optimization. In this context *sufficiently precise* also means that it does not make sense to run stochastic sensitivity analysis with high precision during robustness exploration, since in that case the evaluation of each candidate parameter configuration would take too long to efficiently explore the parameter configuration space.

#### 4.3.8.1 Approximation quality measure

The approximation quality measure that is used to evaluate the approximation behavior of the proposed stochastic sensitivity analysis approach is based on the notion of *normalized hypervolume* (Definition 4.3.4).

**Def. 4.3.4 (Normalized inner and outer hypervolumes)** Given a $n$-dimensional hypercube $\mathcal{H} = \left[\underline{b}_1,\bar{b}_1\right] \times \ldots \times \left[\underline{b}_n,\bar{b}_n\right] \subset \mathbb{R}^n_+$ and a set of $n$-dimensional vectors $\mathcal{V} = \{\vec{v}_1, \ldots, \vec{v}_m\}$ with $\forall_i \vec{v}_i \in \mathcal{H}$, the inner and outer normalized hypervolumes of $\mathcal{V}$ in $\mathcal{H}$ are defined as follows:

$$\bar{\lambda}^-_{\mathcal{H}}(\mathcal{V}) = \lambda^-_{\bar{\mathcal{H}}}(\bar{\mathcal{V}}) \quad \text{and} \quad \bar{\lambda}^+_{\mathcal{H}}(\mathcal{V}) = \lambda^+_{\bar{\mathcal{H}}}(\bar{\mathcal{V}}) \text{ , where}$$

$$\bar{\mathcal{H}} = [0,1] \times \ldots \times [0,1] \text{ and } \bar{\mathcal{V}} = \left\{ \frac{\vec{v}_1 - \underline{b}_1}{\bar{b}_1 - \underline{b}_1}, \ldots, \frac{\vec{v}_n - \underline{b}_n}{\bar{b}_n - \underline{b}_n} \right\}$$

Normalized hypervolume can be obtained by first translating the coordinates of the considered points, and then computing the absolute hypervolume as described in Section 4.1.2.1.

The approximation quality at a given moment during stochastic sensitivity analysis is defined as the normalized hypervolume of the space between the bounding Pareto-fronts $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ and $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$.

**Def. 4.3.5 (Approximation quality)** Let $\mathcal{S}_c$ denote the system $\mathcal{S}$ with parameter configuration $c$ and $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of system properties. Given a bounding working Pareto-front $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ and a bounding non-working Pareto-front $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$ obtained at an arbitrary point during exploration, the approximation quality of the sought-after sensitivity front $\mathcal{F}^{sens}_{\mathcal{S}_c}(\mathcal{P})$ is defined as follows:

1. If the properties $\mathcal{P}$ are subject to minimization

$$\mathcal{Q}_{\mathcal{S}_c}(\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P}), \mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})) = \bar{\lambda}^+_{\mathcal{H}_{\mathcal{S}_c}(\mathcal{P})}(\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})) - \bar{\lambda}^-_{\mathcal{H}_{\mathcal{S}_c}(\mathcal{P})}(\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P}))$$

2. If the properties $\mathcal{P}$ are subject to maximization

$$\mathcal{Q}_{\mathcal{S}_c}(\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P}), \mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})) = \bar{\lambda}^+_{\mathcal{H}_{\mathcal{S}_c}(\mathcal{P})}(\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})) - \bar{\lambda}^-_{\mathcal{H}_{\mathcal{S}_c}(\mathcal{P})}(\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P}))$$

**Note:**
- *The normalized hypervolume of the bounding hypercube $\mathcal{H}_{\mathcal{S}_c}(\mathcal{P})$ containing the sought-after sensitivity front is equal to 1. Consequently, the co-domain of the approximation quality measure is equal to the interval [0,1], with lower values corresponding to better approximation qualities.*
- *The approximation quality corresponds to the portion of space that has not yet been categorized. For robustness exploration approximation qualities around 10% are sufficient.*

#### 4.3.8.2 Initial considerations

In order to get a first idea of the approximation behavior of the stochastic sensitivity analysis, two experiments on the example system presented in Section 2.3 are performed. The first experiment consists of the worst-case execution / communication time sensitivity analysis for the task $T2$ and the communication channel $C3$, whereas in the second experiment the jitter sensitivity analysis for the system inputs $Cam$ and $S_{in}$ is performed. In both cases the increasing approximation quality during exploration is investigated, both visually and metrically. The results are shown in Figure 4.16. Due to the search space bounding strategy discussed in Section 4.3.4 not all individuals generated during stochastic sensitivity analysis need to be evaluated. Therefore, the number of generated and evaluated individuals is stated separately in the following. Note that in the first and second experiments the size of one generation is equal to 20 and 10 individuals, respectively.

In the first experiment, the approximation after 5 generations ($\sim$ 100 evaluated individuals) is coarse but nevertheless sufficient for robustness optimization, which is reflected by the approximation quality metric value of 7.8% (Figure 4.16a). 5 generations later ($\sim$ 180 evaluated individuals) the approximation quality has been improved, both visually and metrically (3.9%, Figure 4.16b). Finally, after 15 generations ($\sim$ 270 evaluated individuals) the two bounding Pareto-fronts have nearly converged, and the approximation quality has been increased to 2.2% (Figure 4.16c).

In the second experiment, the stochastic sensitivity analysis converges much faster compared to the first experiment. After 5 generations ($\sim$ 35 evaluated individuals) almost the final approximation quality obtained in the first experiment is reached (2.7%, Figure 4.16d). After 10 generations ($\sim$ 50 evaluated individuals) the two bounding Pareto-fronts have virtually converged, translating into an approximation quality metric value of 1.2% (Figure 4.16e). This value is only slightly improved after 15 generations ($\sim$ 55 evaluated individuals) to 1.1% (Figure 4.16f).

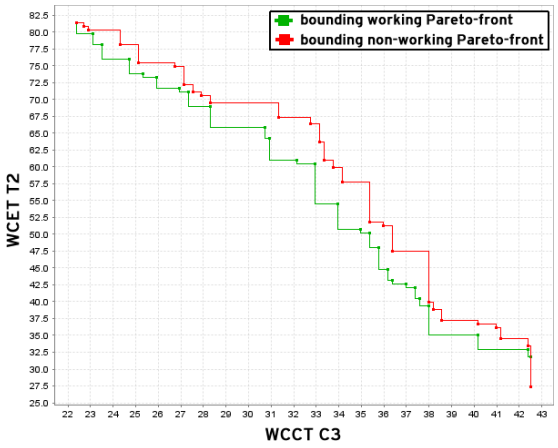
These first experiments indicate that the stochastic sensitivity analysis' approximation speed highly depends on the characteristics of the sought-after sensitivity front. In fact, the differences that were observed can easily be explained. In general, constant sensitivity front regions are far easier to approximate than non-constant regions. The reason is that constant regions can be precisely approximated with only few points, whereas the exact approximation of non-constant regions requires many (potentially infinite) points. For instance, the sensitivity front obtained for the jitter analysis of $Cam$ and $S_{in}$ corresponds to a step function consisting of several constant regions. Obviously, each step can be precisely approximated using a single (!) point. Contrarily, the sensitivity front obtained for the WCET analysis of $T2$ and $C3$ is composed of linear regions. To approximate these linear regions adequately many points are needed, which directly explains the observed differences in approximation quality and speed.

Note that this effect is comparable to the so-called *smart step* technique proposed by Racu et al. in [91, 89] to speed up exact sensitivity analysis for monotone sensitivity front regions. In the two-dimensional case, for instance, the smart step technique exploits that the sensitivity front is constant between two points in one dimension, if they have the same values in the other dimension. Obviously, if two such points are found no further search needs to conducted between them, leading to greatly improved analysis speed.

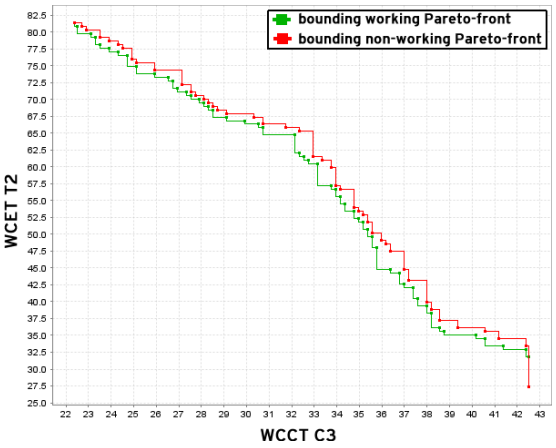
Generally, sensitivity front characteristics can easily be predicted. Sensitivity analyses of system properties like input periods or input jitters usually yield sensitivity fronts consisting of several connected constant regions (e.g. step functions in the two-dimensional case). Contrarily, sensitivity analyses of system properties like worst-case execution times or CPU clock rates usually lead to sensitivity fronts with continuous non-constant regions (e.g. linear dependencies). Input period and jitter sensitivity analysis is, therefore, potentially less time consuming than worst-case execution time and CPU clock rate sensitivity analysis.

#### 4.3.8.3 Convergence behavior

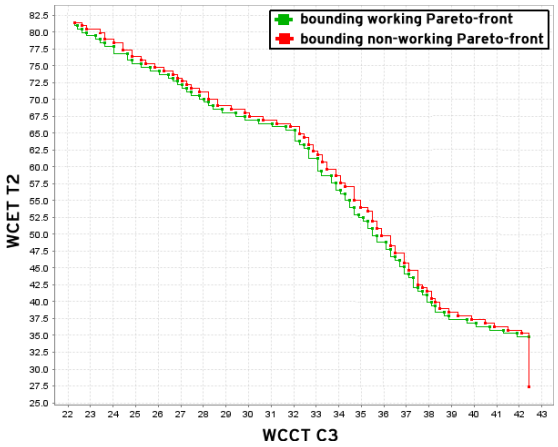
In this section, experiments are performed to characterize the convergence behavior of the proposed stochastic sensitivity analysis. Thereby, according to the discussions in the previous section, two cases are distinguished: sensitivity fronts consisting of constant regions and sensitivity fronts consisting of continuous non-constant regions. Note that the analysis' convergence behavior for sensitivity fronts with mixed characteristics lies between these two investigated cases.



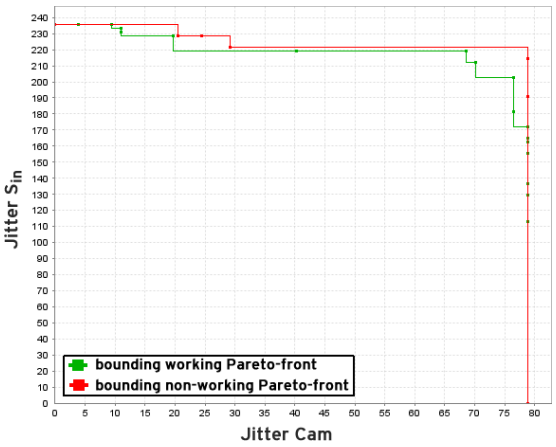

(a) WCCT C3 and WCET T2: after 100 generated individuals, with ∼ 100 evaluations (approximation quality: 7.8%)



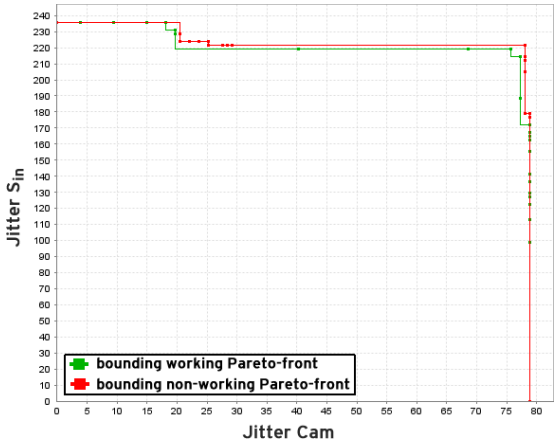

(b) WCCT C3 and WCET T2: after 200 generated individuals, with ∼ 180 evaluations (approximation quality: 3.9%)



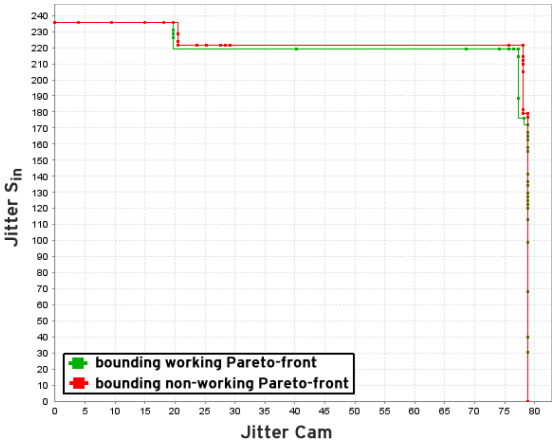

(c) WCCT C3 and WCET T2:after 300 generated individuals, with ∼ 270 evaluations (approximation quality: 2.2%)




(d) Jitter *Cam* and $S_{in}$: after 50 generated individuals, with ∼ 35 evaluations (approximation quality: 2.7%)




(e) Jitter *Cam* and $S_{in}$: after 100 generated individuals, with ∼ 50 evaluations (approximation quality: 1.2%)




(f) Jitter *Cam* and $S_{in}$: after 150 generated individuals, with ∼ 55 evaluations (approximation quality: 1.1%)

Fig. 4.16: Examples visualizing gradually increasing approximation quality during exploration with search resolution 100. The experiments are based on the example system presented in Section 2.3.

Furthermore, the effectiveness of the proposed heuristic variation operators and the search space bounding strategy are evaluated. Therefore, all experiments are performed with three different setups for stochastic sensitivity analysis:

1. Usage of "standard" variation operators: random crossover (Section 4.3.5) and random mutation (similar to adaptive range mutation, Section 4.3.6, with the difference that the mutation range is not adaptively steered by the analysis' convergence status but kept constant at 0.2 times the diagonal length of the bounding hypercube, Definition 4.1.5).

2. Usage of the proposed heuristic variation operators: random and general mean crossover (Section 4.3.5) as well as adaptive range and front convergence mutation (Section 4.3.6).

3. Usage of the proposed heuristic variation operators and search space bounding (Section 4.3.4) with 20 tries to reach the relevant region.

In order to determine and compare the convergence behavior of the stochastic sensitivity analysis for all three setups, the bounding Pareto-fronts are maintained during all experiments. However, only for the experiments with search space bounding they are exploited to increase analysis efficiency.

Note that SPEA2 [136] is used as selector to steer the coverage of the sensitivity front during exploration, and that the search resolution is limited to 100 (compare Section 4.3.7). All experiments are performed using a sample size of 200 and confidence level 99%.

**Two-dimensional case.** The results for the approximation of two-dimensional sensitivity fronts are visualized in Figure 4.17. As can be observed, the efficiency gains obtained by applying heuristic variation operators and search space bounding are highly significant.

For instance, in the case of sensitivity fronts without constant regions (Figure 4.17a), the approximation quality after 7 generations (200 individuals) using standard variation operators is 15.22% (confidence interval 0.4%). In comparison, the approximation quality using heuristic variation operators is 9.47% at this point (confidence interval 0.3%), and with search space bounding even 4.83% (confidence interval 0.1%). Using standard variation operators, a comparable approximation quality is not reached until generation 40.

In the case of sensitivity fronts consisting of constant regions (Figure 4.17b), the stochastic sensitivity analysis converges much faster. This was expected. For instance, using heuristic variation operators and

search space bounding, an approximation quality of approximately 2% is reached after 7 generations (80 individuals). For sensitivity fronts without constant regions a similar approximation quality is only achieved after 16 generations (425 individuals).

**Three-dimensional case.** Also in the three-dimensional case the efficiency gains obtained by applying heuristic variation operators and search space bounding are highly significant. The detailed results are visualized in Figure 4.18.

Furthermore, the results show that three-dimensional sensitivity analysis is computationally more expensive than two-dimensional sensitivity analysis. For instance, in the case of sensitivity fronts without constant regions (Figure 4.18a), an approximation quality of 10%, which is largely sufficient for robustness exploration, is obtained after 9 generation (1000 individuals). In the two-dimensional case the same approximation quality is reached after 4 generations (125 individuals). It can, thus, be concluded that in the three-dimensional case approximately 8 times more individuals need to be considered to obtain approximation qualities that are adequate for robustness exploration.

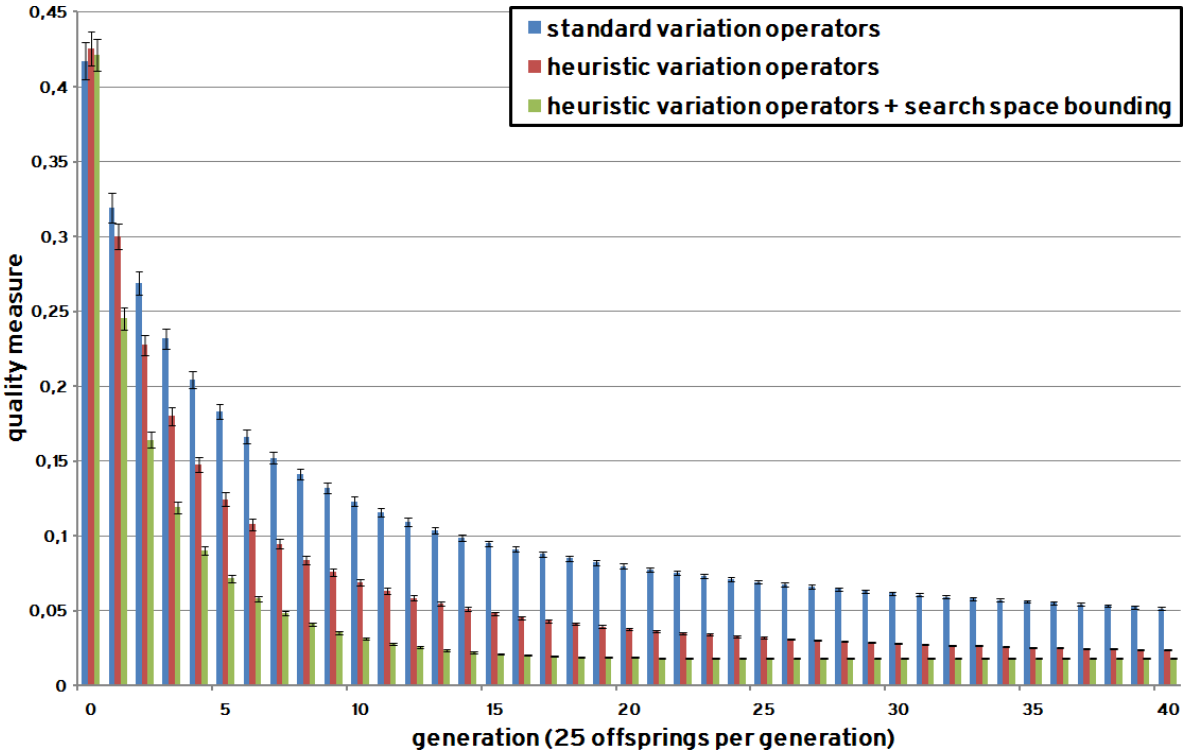
For the approximation of sensitivity fronts consisting of constant regions the difference is higher (Figure 4.18b). In the two-dimensional case an approximation quality below 10% is achieved after 2 generations (30 individuals). In the three-dimensional case 13 times more individuals need to be considered to obtain the same approximation quality (12 generations, 390 individuals).

#### 4.3.8.4 Analysis speed

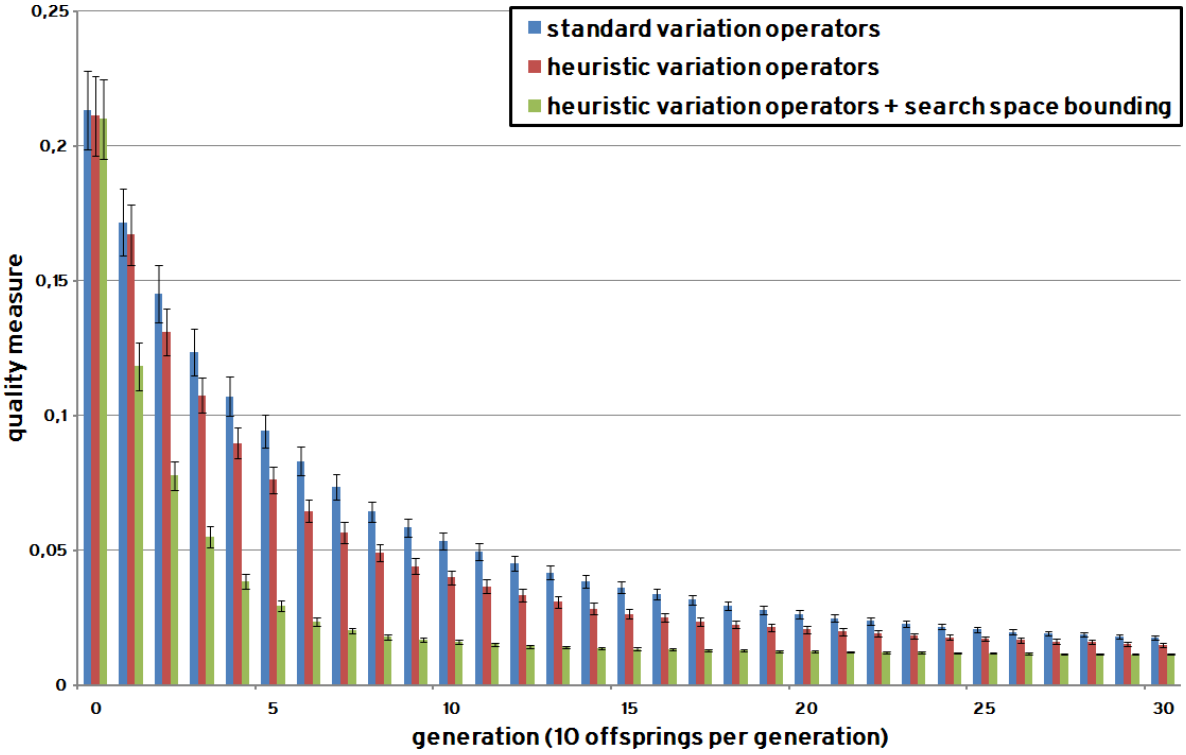
In the previous section the progressively increasing approximation quality was measured as a function of individuals considered during stochastic sensitivity analysis. However, due to the applied search space bounding technique not every generated individual needs to be evaluated. For instance, the variation operators may fail in some situations to generate offsprings that lie inside the relevant region.

In order to evaluate the computational complexity (i.e. the analysis time) of the proposed stochastic sensitivity analysis, the actual number of individuals that were evaluated during the above conducted experiments was measured. The results are visualized in Figure 4.19.

As can be see observed, the number of evaluated individuals in the experiments with search space bounding is considerably lower compared to the experiments without search space bounding. The differences are particularly remarkable at late exploration stages. The reason is that with progressing approximation, i.e. with converging bounding Pareto-

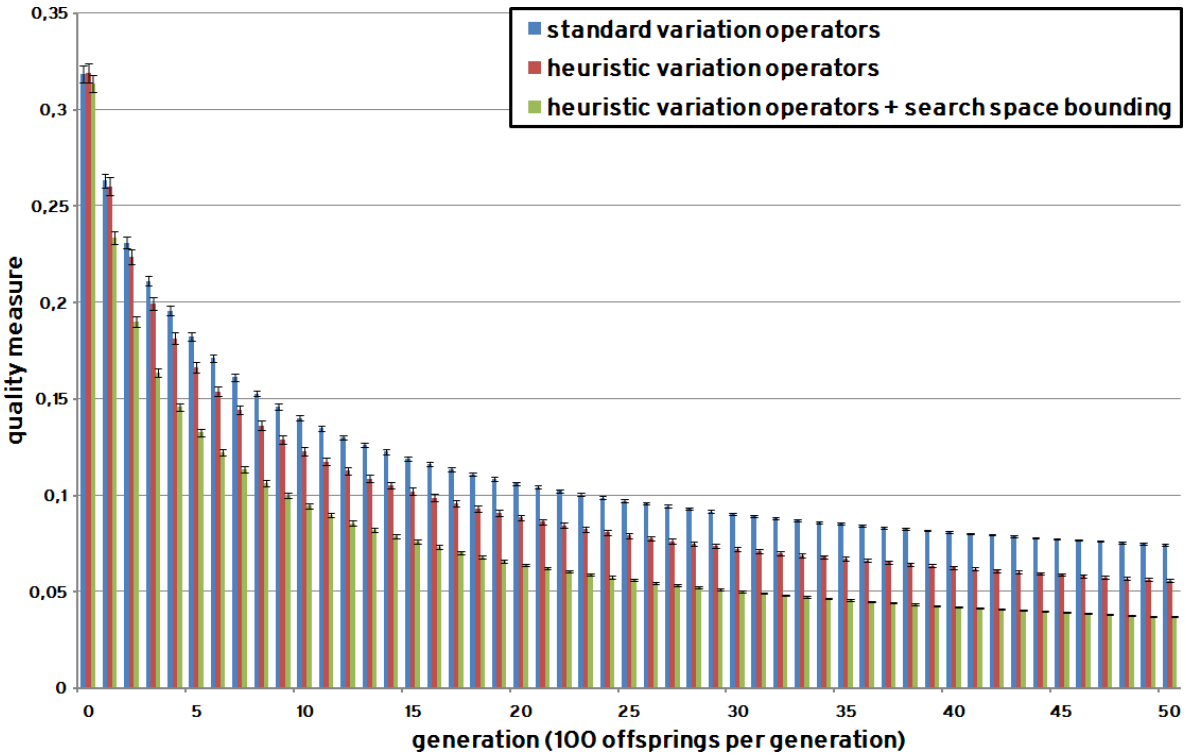



(a) Approximation of sensitivity fronts w/o constant regions (200 samples, confidence level 99%)

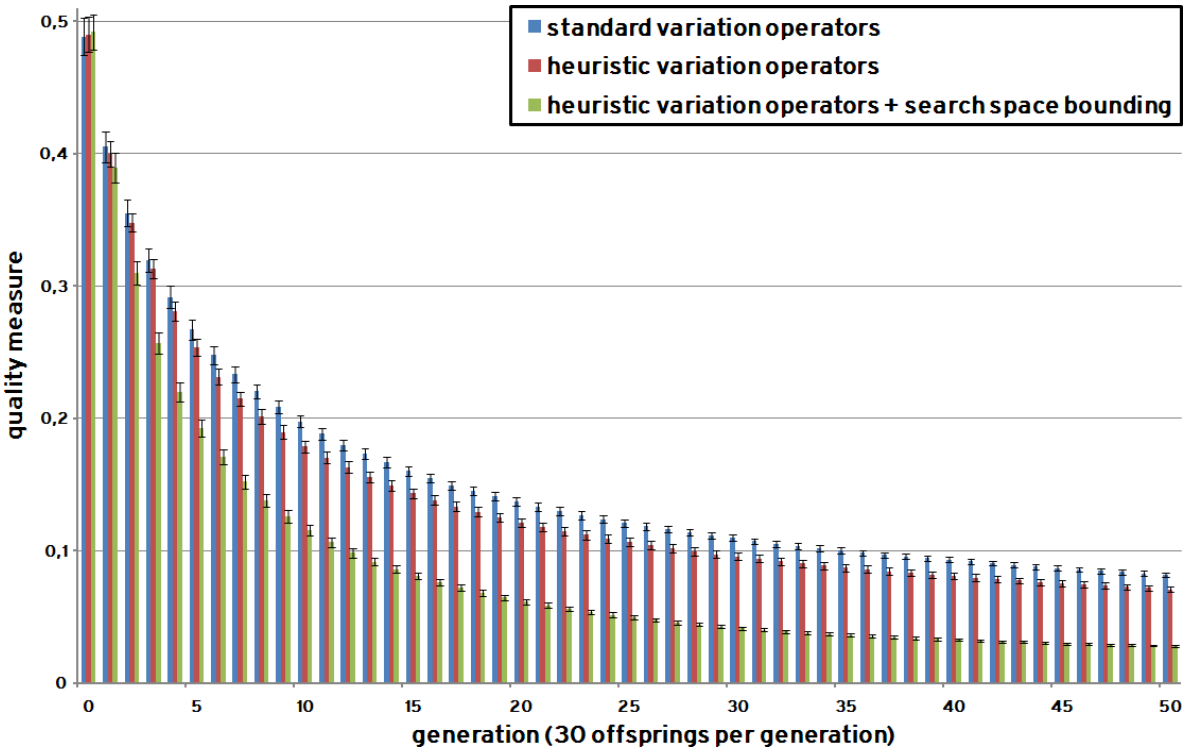



(b) Approximation of sensitivity fronts consisting of several constant regions (200 samples, confidence level 99%)

Fig. 4.17: Convergence behavior and approximation quality of the stochastic sensitivity analysis in the two-dimensional case



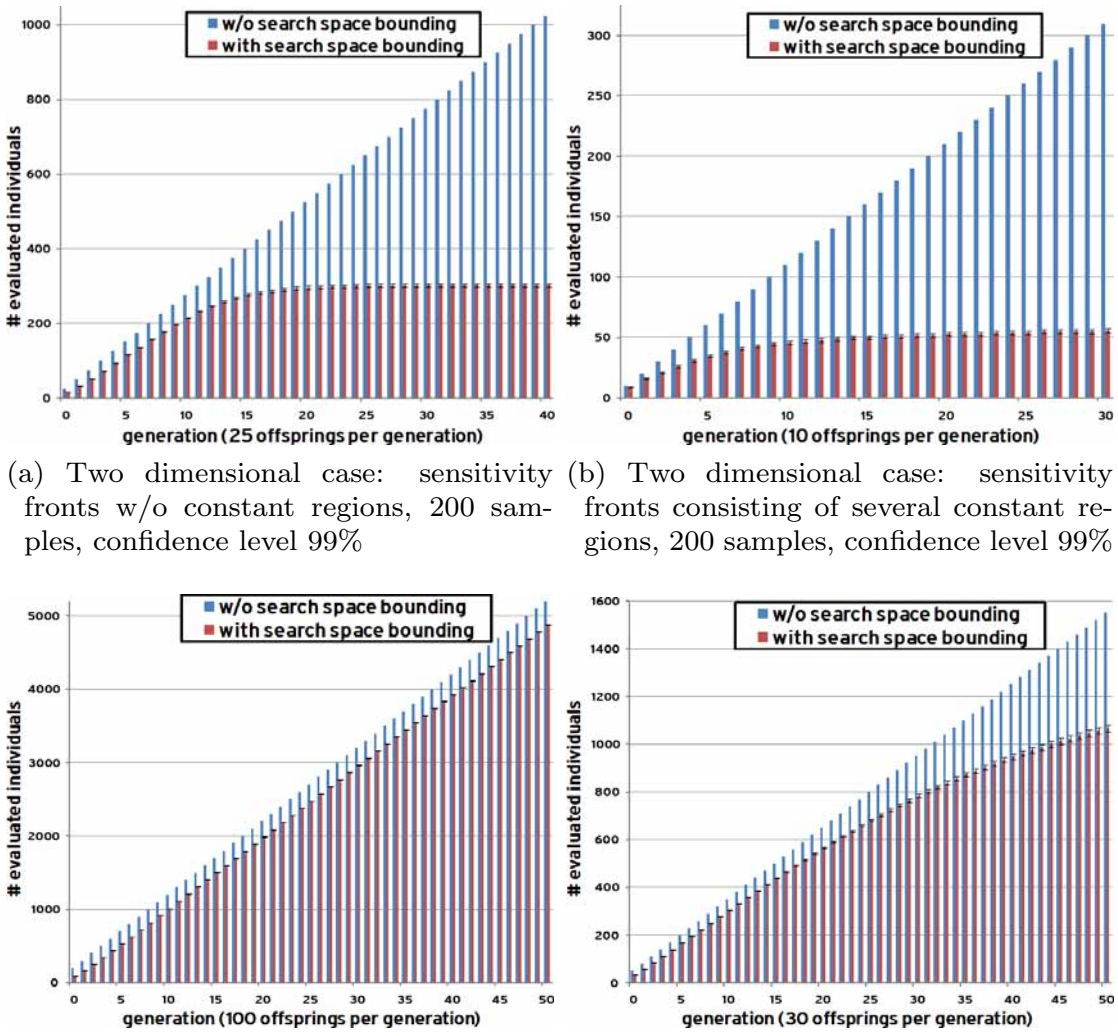

(a) Approximation of sensitivity fronts w/o constant regions (200 samples, confidence level 99%)




(b) Approximation of sensitivity fronts consisting of several constant regions (200 samples, confidence level 99%)

Fig. 4.18: Convergence behavior and approximation quality of the stochastic sensitivity analysis in the three-dimensional case

fronts, it is getting increasingly more difficult for the variation operators to generate individuals that are lying inside the relevant region. For instance, in the case of two-dimensional sensitivity fronts consisting of constant pieces (Figure 4.19b) the bounding Pareto-fronts have nearly converged (in average) after approximately 10 generations. At this point, it is very hard for the variation operators to generate individuals that are lying inside the relevant region (i.e. improve approximation quality). Consequently, most of the generated individuals are lying outside the relevant region, and thus do not need to be analyzed.




(a) Two dimensional case: sensitivity fronts w/o constant regions, 200 samples, confidence level 99%

(b) Two dimensional case: sensitivity fronts consisting of several constant regions, 200 samples, confidence level 99%

(c) Three dimensional case: sensitivity fronts w/o constant regions, 200 samples, confidence level 99%

(d) Three dimensional case: sensitivity fronts consisting of several constant regions, 200 samples, confidence level 99%

Fig. 4.19: Analysis speed-up through search space bounding

## 4.4 Exploring robustness

In this section it is discussed how the robustness metrics presented in Section 4.2 can be efficiently explored. First, the case of independent system properties is addressed (Section 4.4.1). Second, the more involved case of dependent system properties is discussed (Section 4.4.2).

### 4.4.1 Independent system properties

The robustness metrics for independent system properties presented in Section 4.2.1 are based on one-dimensional sensitivity analysis information. Since one-dimensional sensitivity analysis is computationally efficient (e.g. the analysis of Racu et al. based on binary search techniques [93]) the static robustness metric (SDR) can directly be calculated during exploration. In other words, it is proposed to use the exploration framework presented in Chapter 3 to traverse the parameter configuration space, and to integrate the SDR metric directly as optimization objective. Since the exploration framework is capable of performing multi-criterion optimization, SDR can be Pareto-optimized with other optimization objectives, such as timing, power, etc.

The dynamic case is a little bit more involved. To exactly determine the dynamic design robustness $\mathrm{DDR}_{\mathcal{S},\mathcal{C}}(p)$ of a system $\mathcal{S}$ with respect to a given system property $p$, exhaustive search in the given reconfiguration space $\mathcal{C}$ is necessary. Contrary to the static case, where exact metric values can be explicitly calculated, the determination of the dynamic design robustness $\mathrm{DDR}_{\mathcal{S},\mathcal{C}}(p)$ is, therefore, an optimization problem itself.

In order to approximate the dynamic design robustness of $\mathcal{S}$ with respect to $p$, it is proposed to use the exploration framework proposed in Chapter 3 to traverse the reconfiguration space $\mathcal{C}$. Thereby, the robustness of each considered parameter configuration is evaluated using one-dimensional sensitivity analysis. By this means, exploration yields parameter configurations with high robustness for $p$ representing lower bounds for the sought-after dynamic design robustness $\mathrm{DDR}_{\mathcal{S},\mathcal{C}}(p)$. Note that for the approximation of the aggregated dynamic design robustness $\mathrm{ADDR}_{\mathcal{S},\mathcal{C}}(\mathcal{P})$ a separate exploration needs to be performed for each system property in $\mathcal{P}$.

### 4.4.2 Dependent system properties

The static and dynamic robustness metrics for dependent system properties introduced in Section 4.2.2 are based on multi-dimensional sensitivity analysis information. Since the exact calculation of multi-dimensional sensitivity fronts might be computationally expensive, the direct integration of the exact metrics into exploration is of limited prac-

ticability, especially for more than three system properties. In order to circumvent this complexity problem, efficient approximation techniques for the static (Section 4.4.2.1) and the dynamic robustness metrics (Section 4.4.2.2) are proposed in this section.

The proposed methods are based on the stochastic multi-dimensional sensitivity analysis presented in Section 4.3. The main reason for applying this analysis to system robustness optimization is its capability of deriving upper and lower system sensitivity bounds with scalable computational effort. The proposed approximation techniques use these bounds to efficiently approximate the static and dynamic robustness metrics during exploration. These approximations are adequate to guide exploration towards interesting parameter configurations, whose detailed robustness analysis can be performed afterwards.

Note that for simplicity, the given definitions assume that the considered system properties are subject to maximization. However, the exploration techniques can easily be adapted to cover system properties that are subject to minimization.

#### 4.4.2.1 Static case

In the static case the robustness exploration task consists in finding the parameter configuration for a given system $\mathcal{S}$ that maximizes the static design robustness with respect to a set of dependent system properties $\mathcal{P}$ weighted according to $\mathcal{W}$.

To tackle this problem, a nested design space exploration approach is proposed. The outer exploration loop variates free system parameters (e.g. scheduling parameters), whereas the inner exploration loop evaluates the robustness of each candidate parameter configuration $\mathcal{S}_c$ using stochastic multi-dimensional sensitivity analysis (Section 4.3). More precisely, the bounding working Pareto-front $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ and the bounding non-working Pareto-front $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$ are used to derive upper and lower robustness bounds for each evaluated parameter configuration $\mathcal{S}_c$.

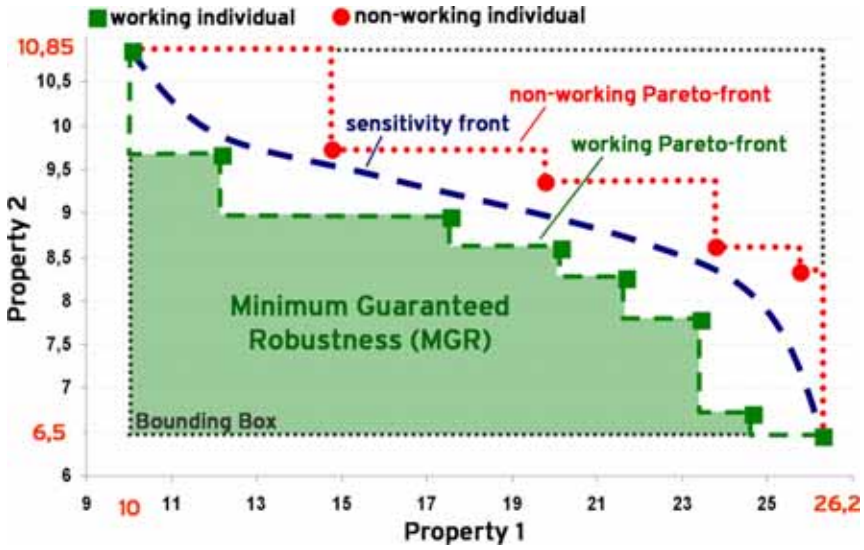
**Def. 4.4.1 (Static Robustness Bounds)** Let $\mathcal{S}_c$ denote the system $\mathcal{S}$ with parameter configuration $c$ and $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of dependent system properties. Given the bounding working and non-working Pareto-fronts $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ and $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$ as well as the set of weights $\mathcal{W} = \{w_1, \ldots, w_n\}$, the static design robustness $\mathrm{SDR}^{\mathrm{dep}}_{\mathcal{S}_c}(\mathcal{P}, \mathcal{W})$ is bounded by the minimum guaranteed robustness $\mathcal{R}^{-}_{\mathcal{S}_c}$ and the maximum possible robustness $\mathcal{R}^{+}_{\mathcal{S}_c}$:

$$\mathcal{R}^{-}_{\mathcal{S}_c}(\mathcal{P},\mathcal{W}) \leq \mathrm{SDR}^{\mathrm{dep}}_{\mathcal{S}_c}(\mathcal{P},\mathcal{W}) < \mathcal{R}^{+}_{\mathcal{S}_c}(\mathcal{P},\mathcal{W}), \text{ where}$$

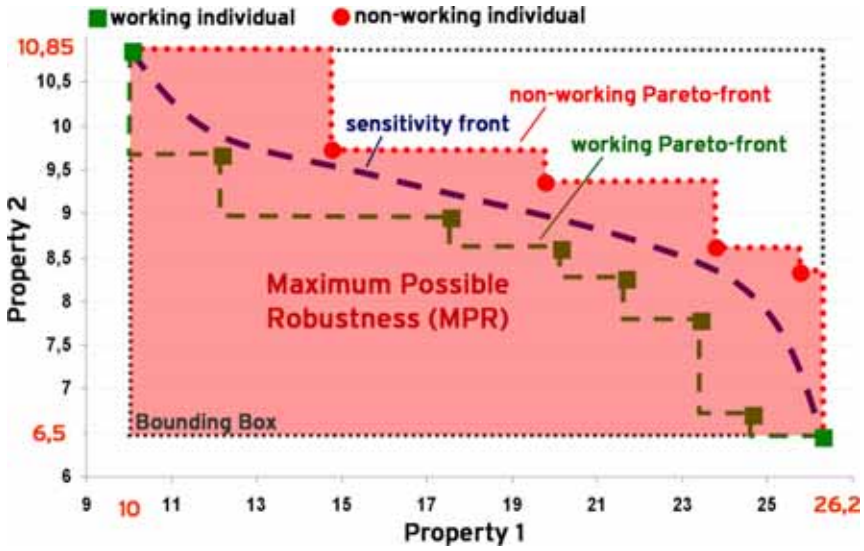
$$\mathcal{R}^{-}_{\mathcal{S}_c}(\mathcal{P},\mathcal{W}) = \tilde{\lambda}^{-}_{\mathcal{H}_{\mathcal{S}_c}(\mathcal{P})}(\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P}),\mathcal{W})$$

$$\mathcal{R}^{+}_{\mathcal{S}_c}(\mathcal{P},\mathcal{W}) = \tilde{\lambda}^{+}_{\mathcal{H}_{\mathcal{S}_c}(\mathcal{P})}(\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P}),\mathcal{W})$$

Figures 4.20a and 4.20b visualize the minimum guaranteed and the maximum possible robustness (without weighting).




(a) Minimum guaranteed robustness $\mathcal{R}^{-}_{\mathcal{S}_c}$




(b) Maximum possible robustness $\mathcal{R}^{+}_{\mathcal{S}_c}$

Fig. 4.20: Static robustness bounds $\mathcal{R}^{-}_{\mathcal{S}_c}$ and $\mathcal{R}^{+}_{\mathcal{S}_c}$ derived from the bounding Pareto-fronts obtained through stochastic sensitivity analysis of two system properties for the parameter configuration $\mathcal{S}_c$

**Exploration control.** In order to find all parameter configurations with high robustness, despite variations in the evaluation results of the underlying stochastic multi-dimensional sensitivity analysis, it is proposed to perform a multi-objective exploration of the minimum guaranteed robustness $\mathcal{R}^-$ and the maximum possible robustness $\mathcal{R}^+$ in the outer exploration loop. In so doing, exploration yields, on the one hand, parameter configurations with large guaranteed robustness and, on the other hand, parameter configurations with possibly large robustness potential, which needs to be confirmed or disapproved by more detailed analysis.

Note that for this specific optimization problem two parameter configurations need to be considered as *non-comparable* if the intervals defined by their minimum guaranteed and maximum possible robustness bounds intersect. Clearly, in this case it cannot be decided without further analysis which parameter configuration possesses the higher real robustness, and hence needs to be considered as *better*. Unfortunately, Pareto-dominance does not exhibit the desired behavior for interval semantics. Consequently, the notion of optimality that is used by the selector algorithm[1] steering the outer exploration loop needs to be modified. More precisely, instead of Pareto-dominance the notion of *interval-dominance* must be used to decide about optimality.

**Def. 4.4.2 (Interval-dominance)** Given two intervals $A, B \subset \mathbb{R}_+$, the interval $A = [a_1, a_2]$ dominates the interval $B = [b_1, b_2]$ (notation $A \rhd B$), iff it holds that:

$$a_1 \geq b_2$$

Figure 4.21 visualizes the differences between the notions of *Pareto-dominance* in the sense of a maximization problem and *interval-dominance*.

Three different situations are distinguished for the intervals defined by the robustness bounds of the given parameter configurations $c_1$ and $c_2$: a) the intervals do not intersect, b) the intervals intersect, and c) one interval includes the other interval. In case a) it can be definitely stated that $c_2$ has a higher robustness than $c_1$. Obviously, interval-dominance reflects this fact. However, also Pareto-dominance considers $c_1$ as sub-optimal, since both, its minimum guaranteed and its maximum possible robustness, are lower. In case b) and c) it cannot be decided whether $c_1$ or $c_2$ have higher robustness, since the intervals de-

[1] We use SPEA2 [136]. Compare Chapter 3.

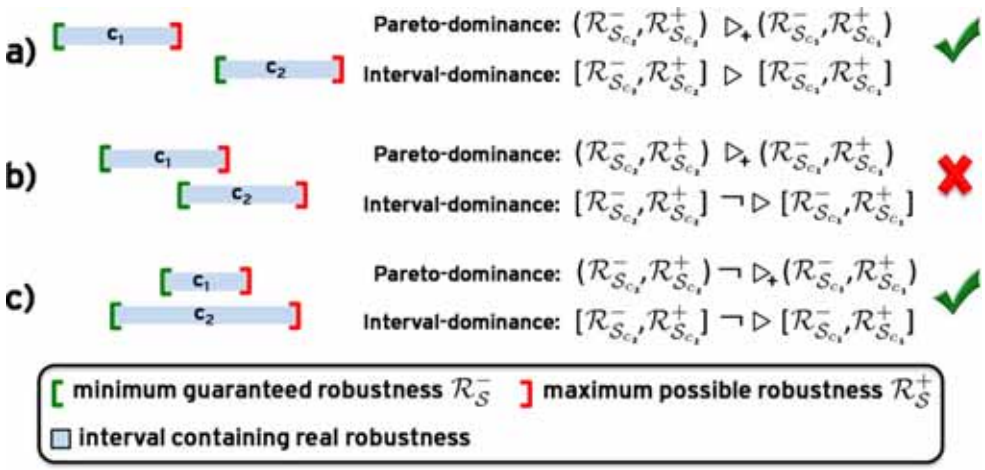



Fig. 4.21: Difference between Pareto-dominance and interval-dominance

fined by the given lower and upper robustness bounds overlap. In case c), Pareto-dominance correctly evaluates the situation, since $c_1$ has a higher minimum guaranteed robustness and $c_2$ has a higher maximum possible robustness. However, in case b) Pareto-dominance leads to a wrong interpretation. Pareto-dominance states that $c_1$ is sub-optimal, even though both parameter configurations cannot be compared.

**Scalability.** Clearly, the real robustness of any evaluated parameter configuration $\mathcal{S}_c$ is guaranteed to be contained in the interval defined by $\mathcal{R}^-_{\mathcal{S}_c}$ and $\mathcal{R}^+_{\mathcal{S}_c}$. Consequently, the precision of the underlying robustness approximation can be safely scaled. At high precisions, exploration takes longer but yields more accurate results. Contrarily, lower precisions allow for better search space coverage at the cost of less precise results. Hence, more sub-optimal parameter configurations might be misinterpreted as potentially interesting. However, despite that, the minimum guaranteed and the maximum possible robustness metrics are very good indicators for identifying interesting parameter configurations during exploration, even with low precision. The detailed analysis of these parameter configurations can be postponed, and performed after exploration.

#### 4.4.2.2 Dynamic case

The optimization task in the dynamic case consists in finding all parameter configurations that exhibit partially disjoint robustness characteristics for the considered dependent system properties, and hence contribute to the sought-after dynamic design robustness. Like in the static case, the robustness of each considered parameter configuration is

evaluated with the stochastic sensitivity analysis proposed in Section 4.3. The main difference to the static case is how exploration is controlled.

During exploration so-called *dynamic bounding Pareto-fronts* are maintained and successively updated:

- The dynamic bounding *working* Pareto-front integrates the information that is contained in the bounding *working* Pareto-fronts of all evaluated parameter configurations, i.e. for each system property value combination below the dynamic bounding working Pareto-front there exists at least one feasible parameter configuration.

- The dynamic bounding *non-working* Pareto-front integrates the information contained in the bounding *non-working* Pareto-fronts of all evaluated parameter configurations, i.e. for each system property value combination below the dynamic bounding non-working Pareto-front there exists at least one parameter configuration for that infeasibility has not yet been proven.

In the following, formal definitions of the dynamic bounding Pareto-fronts are introduced, and lower and upper bounds for the sought-after dynamic design robustness of a given system are derived. Additionally, it is shown how the dynamic bounding Pareto-fronts are successively updated and refined using an iterative exploration approach with adaptive fitness assignment.

The dynamic bounding Pareto-fronts are contained in the *dynamic bounding hypercube* (Definition 4.4.3).

**Def. 4.4.3 (Dynamic bounding hypercube)** Let $\mathcal{S}$ be a given system, $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of system properties, and $\mathcal{C} = \{c_1, \ldots, c_m\}$ a set of evaluated parameter configurations for $\mathcal{S}$. Given the bounding hypercubes $\mathcal{H}_{\mathcal{S}_{c_1}}(\mathcal{P}), \ldots, \mathcal{H}_{\mathcal{S}_{c_m}}(\mathcal{P})$ for the parameter configurations in $\mathcal{C}$ obtained by stochastic sensitivity analysis, the dynamic bounding hypercube is defined as follows:

$$\tilde{\mathcal{H}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}) = \bigcup_{i=1}^{m} \mathcal{H}_{\mathcal{S}_{c_i}}(\mathcal{P})$$

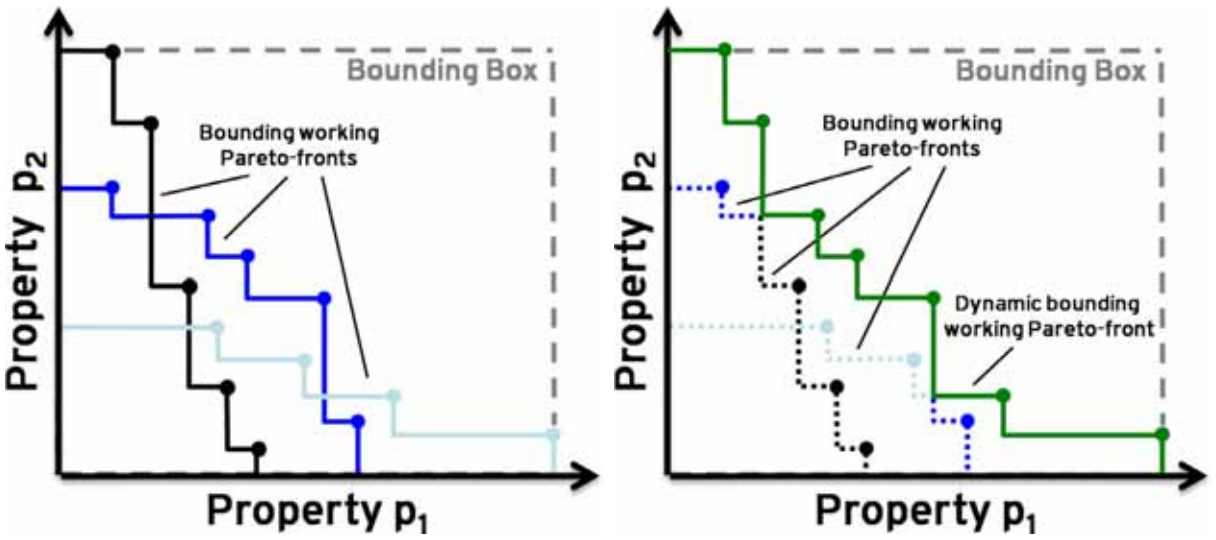
**Dynamic bounding working Pareto-front.** The calculation of the dynamic bounding working Pareto-front is straight-forward. At a given time during exploration, it is defined by the Pareto-optimal (in the sense of a maximization problem) vectors in the union set of the bounding

working Pareto-fronts of all evaluated parameter configurations. Definition 4.4.4 gives a formal characterization.

**Def. 4.4.4 (Dynamic bounding working Pareto-front)** Let $\mathcal{S}$ be a given system, $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of system properties, and $\mathcal{C} = \{c_1, \ldots, c_m\}$ a set of evaluated parameter configurations for $\mathcal{S}$. Given the bounding working Pareto-fronts $\mathcal{F}_w = \{\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_{c_1}}(\mathcal{P}), \ldots, \mathcal{F}^{\mathrm{w}}_{\mathcal{S}_{c_m}}(\mathcal{P})\}$ for the parameter configurations in $\mathcal{C}$ obtained by stochastic sensitivity analysis, the dynamic bounding working Pareto-front is defined as follows:

$$\tilde{\mathcal{F}}^{\mathrm{w}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}) = \Omega^{+}(\bigcup_{i=1}^{m} \mathcal{F}^{\mathrm{w}}_{\mathcal{S}_{c_i}}(\mathcal{P})) = \Omega^{+}(\{\vec{x} \mid \exists \mathcal{F} \in \mathcal{F}_w : \vec{x} \in \mathcal{F}\})$$

Figure 4.22 visualizes how three given bounding working Pareto-fronts are integrated into the dynamic bounding working Pareto-front.




(a) Bounding working Pareto-fronts of three different parameter configurations

(b) Dynamic bounding working Pareto-front integrating the three given bounding working Pareto-fronts

Fig. 4.22: Calculation of the dynamic bounding working Pareto-front

**Dynamic bounding non-working Pareto-front.** The calculation of the dynamic bounding non-working Pareto-front is more involved. Figure 4.23 visualizes how the bounding non-working Pareto-fronts of three given parameter configurations are integrated into the dynamic bounding non-working Pareto-front.

The sought-after Pareto-front must cover all system property value combinations, whose infeasibility has not yet been proven for at least one evaluated parameter configuration. This is true for all value combinations, which are covered by at least one of the involved bounding non-working Pareto-fronts.

For the correct determination of the dynamic bounding non-working Pareto-front the approach applied for the dynamic bounding working Pareto-front is not valid. This is visualized in Figure 4.23.

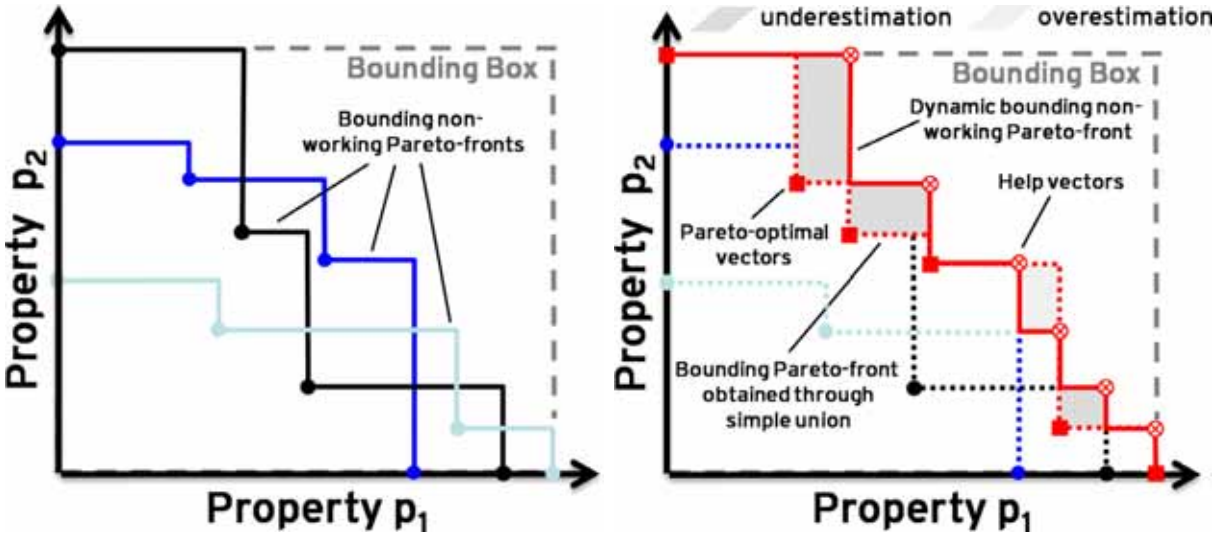
Figure 4.23a shows the bounding non-working Pareto-fronts of three evaluated parameter configurations. Figure 4.23b visualizes two different attempts for deriving the dynamic bounding non-working Pareto-front.

The red dotted line connects all Pareto-optimal vectors (marked with a square) with an upper step function. This corresponds to the Pareto-front that is obtained if the same approach as for the dynamic bounding working Pareto-front is applied. However, a comparison with the real dynamic bounding non-working Pareto-front, visualized by the solid red line, reveals that this algorithm is not correct. In fact, the red dotted line does not cover several regions highlighted in gray, which corresponds to an underestimation, and it incorrectly covers the region hatched in gray, which corresponds to an overestimation of the sought-after dynamic bounding non-working Pareto-front.

As one possibility, the dynamic bounding non-working Pareto-front is correctly characterized by the set of all Pareto-optimal vectors that Pareto-dominate (in the sense of a maximization problem) no vector in at least one of the considered bounding non-working Pareto-fronts. For the example shown in Figure 4.23b these vectors are marked with a cross. As can be seen, they correctly define the limits of the sought-after dynamic bounding non-working Pareto-front. Definition 4.4.5 gives a formal characterization.

**Def. 4.4.5 (Dynamic bounding non-working Pareto-front)** Let $\mathcal{S}$ be a given system, $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of system properties, and $\mathcal{C} = \{c_1, \ldots, c_m\}$ a set of evaluated parameter configurations for $\mathcal{S}$. Given the bounding non-working Pareto-fronts $\mathcal{F}_{nw} = \{\mathcal{F}^{\text{nw}}_{\mathcal{S}_{c_1}}(\mathcal{P}), \ldots, \mathcal{F}^{\text{nw}}_{\mathcal{S}_{c_m}}(\mathcal{P})\}$ for the parameter configurations in $\mathcal{C}$ obtained by stochastic sensitivity analysis, the dynamic bounding non-working Pareto-front is defined as follows:

$$\tilde{\mathcal{F}}^{\text{nw}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}) = \Omega^{+}(\{\vec{f} \in \tilde{\mathcal{H}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}) | \exists \mathcal{F} \in \mathcal{F}_{nw} : (\nexists \vec{x} \in \mathcal{F} : \vec{f} \triangleright_{+} \vec{x})\})$$




(a) Bounding non-working Pareto-fronts of three different parameter configurations

(b) Dynamic bounding non-working Pareto-front integrating the three given bounding non-working Pareto-fronts

Fig. 4.23: Calculation of the dynamic bounding non-working Pareto-front

**Dynamic robustness bounds.** Like in the static case, the dynamic bounding Pareto-fronts $\tilde{\mathcal{F}}^{\mathrm{w}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})$ and $\tilde{\mathcal{F}}^{\mathrm{nw}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})$ can be used to derive bounds for the dynamic design robustness of the analyzed system (Definition 4.4.6).

Definition 4.4.6 formally characterizes the dynamic minimum guaranteed robustness $\tilde{\mathcal{R}}^{-}_{\mathcal{S}}$, representing a lower bound, and the dynamic maximum possible robustness $\tilde{\mathcal{R}}^{+}_{\mathcal{S}}$, representing an upper bound for the sought-after dynamic design robustness.

**Def. 4.4.6 (Dynamic Robustness Bounds)** Let $\mathcal{S}$ denote a given system, $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of system properties, and $\mathcal{C}$ a set of evaluated parameter configurations for $\mathcal{S}$. Given the dynamic bounding working and non-working Pareto-fronts $\tilde{\mathcal{F}}^{\mathrm{w}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})$ and $\tilde{\mathcal{F}}^{\mathrm{nw}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})$, as well as the set of weights $\mathcal{W} = \{w_1, \ldots, w_n\}$, the dynamic design robustness $\mathrm{DDR}^{\mathrm{dep}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}, \mathcal{W})$ is bounded by the dynamic minimum guaranteed robustness $\tilde{\mathcal{R}}^{-}_{\mathcal{S}}$ and the dynamic maximum possible robustness $\tilde{\mathcal{R}}^{+}_{\mathcal{S}}$:

$$\tilde{\mathcal{R}}^{-}_{\mathcal{S}}(\mathcal{P}, \mathcal{W}) \leq \mathrm{DDR}^{\mathrm{dep}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}, \mathcal{W}) < \tilde{\mathcal{R}}^{+}_{\mathcal{S}}(\mathcal{P}, \mathcal{W}), \text{ where}$$

$$\tilde{\mathcal{R}}^{-}_{\mathcal{S}}(\mathcal{P}, \mathcal{W}) = \tilde{\lambda}^{-}_{\tilde{\mathcal{H}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})}(\tilde{\mathcal{F}}^{\mathrm{w}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}), \mathcal{W})$$

$$\tilde{\mathcal{R}}^{+}_{\mathcal{S}}(\mathcal{P}, \mathcal{W}) = \tilde{\lambda}^{-}_{\tilde{\mathcal{H}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})}(\tilde{\mathcal{F}}^{\mathrm{nw}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}), \mathcal{W})$$

Note that $\tilde{\mathcal{R}}_{\mathcal{S}}^{-}$ represents a globally valid lower bound for the sought-after dynamic design robustness of the analyzed system, i.e. its real dynamic design robustness is larger or equal to $\tilde{\mathcal{R}}_{\mathcal{S}}^{-}$. However, $\tilde{\mathcal{R}}_{\mathcal{S}}^{+}$ only represents an upper dynamic robustness bound for the set of evaluated parameter configurations $\mathcal{C}$. Note that this is not a limitation of the approach, since, in general, a globally valid upper bound cannot be determined without evaluating all possible parameter configurations.

**Exploration control.** In the previous paragraphs it was discussed how upper and lower dynamic design robustness bounds can be derived using the stochastic sensitivity analysis presented in Section 4.3. In the remainder of this section it is explained how exploration is controlled.

In order to efficiently approximate the dynamic design robustness an iterative exploration approach is proposed. Basically, each iteration represents an independent exploration that runs for a predefined number of generations. However, the dynamic bounding Pareto-fronts are maintained and updated across all iterations. This iterative exploration approach has been chosen to implement an adaptive fitness assignment strategy. The reason why adaptive exploration control is necessary to efficiently approximate the dynamic design robustness of a given system is discussed in the following.

**Adaptive fitness assignment.** One possibility for controlling the explorative dynamic design robustness approximation consists in using the minimum guaranteed robustness (compare Definition 4.4.1 in Section 4.4.2.1) of the evaluated parameter configurations as fitness values. That this straight-forward method is not necessarily effective shall be demonstrated by means of an example. Figure 4.24 shows intermediate results obtained during dynamic design robustness approximation for two system properties $P_1$ and $P_2$.

It is assumed that already several iterations have been performed. The intermediate dynamic bounding working Pareto-front is shown by the dashed line. In the current iteration, two new parameter configurations, $C_1$ and $C_2$, are evaluated. The corresponding bounding working Pareto-fronts are highlighted in light blue and green, respectively.

As can be seen, $C_1$ has a higher minimum guaranteed robustness than $C_2$. However, the bounding working Pareto-front of $C_1$ is completely covered by the dynamic bounding working Pareto-front. Consequently, $C_1$ does not improve approximation quality. The situation looks different for $C_2$. Even though $C_2$ has a smaller minimum guaranteed robustness than $C_1$, it directly improves the dynamic bounding working Pareto-front

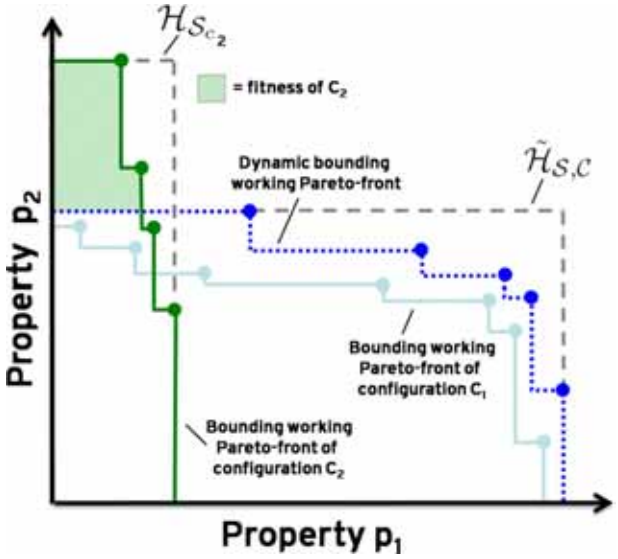



Fig. 4.24: Adaptive fitness assignment strategy during dynamic design robustness approximation

at the depicted state of exploration. It is, thus, preferable to assign a higher fitness value to $C_2$ than to $C_1$.

This small example indicates that plain robustness properties are not adequate to control the explorative approximation of the dynamic design robustness. For this reason, it is proposed to use the approximation improvements achieved by the considered parameter configurations as fitness values. In Figure 4.24 the fitness of $C_2$ corresponding to this strategy is represented by the hatched area. Definition 4.4.7 formalizes the proposed adaptive fitness value.

**Def. 4.4.7 (Adaptive fitness value)** Let $\mathcal{S}$ denote a given system, and $\mathcal{P} = \{p_1, \ldots, p_n\}$ a set of system properties weighted according to $\mathcal{W} = \{w_1, \ldots, w_n\}$. Given the bounding working Pareto-front $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ of the parameter configuration $c$ obtained by stochastic sensitivity analysis, and the current dynamic bounding working Pareto-front $\tilde{\mathcal{F}}^{\mathrm{w}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})$, the fitness of $c$ is defined as follows:

$$\mathrm{fitness}(c) = \tilde{\lambda}^{-}_{\mathcal{H}}(\Omega^{+}(\tilde{\mathcal{F}}^{\mathrm{w}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}) \cup \mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})), \mathcal{W}) - \tilde{\lambda}^{-}_{\tilde{\mathcal{H}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})}(\tilde{\mathcal{F}}^{\mathrm{w}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}), \mathcal{W})$$

$$, \text{ where } \mathcal{H} = \tilde{\mathcal{H}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}) \cup \mathcal{H}_{\mathcal{S}_c}(\mathcal{P})$$

Note that during iterative exploration the dynamic bounding working Pareto-front, which represents the baseline for the adaptive fitness assignment strategy, is updated after each iteration. In this manner, exploration is adaptively controlled and constantly re-focused on search

space areas with the most approximation improvement potential. In other words, parameter configurations that have been evaluated in previous iterations are assigned low fitness values[1] in subsequent iterations, since their robustness characteristics are already included in the dynamic bounding Pareto-fronts.

**Pseudo-code representation.** For a better overview and understanding of the iterative exploration approach that is proposed for approximating the dynamic design robustness of a system $\mathcal{S}$ with respect to a set of dependent system properties $\mathcal{P}$ and a reconfiguration space $\mathcal{C}$, a pseudo-code representation of the specific actions that need to be performed before, during, and after each iteration is given in the following. Note that it is assumed that the exploration runs for a predefined number of iterations. Therefore, no explicit stop condition is specified.

- Initialization of the iterative exploration

  1. Initialize dynamic bounding hypercube
  $$\tilde{\mathcal{H}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}) = \emptyset$$
  2. Initialize dynamic bounding working and non-working Pareto-fronts
  $$\tilde{\mathcal{F}}^{\mathrm{w}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}) = \emptyset \text{ and } \tilde{\mathcal{F}}^{\mathrm{nw}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}) = \emptyset$$
  3. Initialize the sets containing all bounding working and non-working Pareto-fronts of parameter configurations that have been evaluated
  $$\Gamma^{\mathrm{w}} = \emptyset \text{ and } \Gamma^{\mathrm{nw}} = \emptyset$$

- During an exploration iteration: for each evaluated parameter configuration $c$ do the following

  1. Perform a stochastic multi-dimensional sensitivity analysis (Section 4.3) for $c$ to obtain the bounding working and non-working Pareto-fronts $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ and $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$
  2. Add $\mathcal{F}^{\mathrm{w}}_{\mathcal{S}_c}(\mathcal{P})$ to $\Gamma^{\mathrm{w}}$ and $\mathcal{F}^{\mathrm{nw}}_{\mathcal{S}_c}(\mathcal{P})$ to $\Gamma^{\mathrm{nw}}$
  3. Update the dynamic bounding hypercube
  $$\tilde{\mathcal{H}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}) = \tilde{\mathcal{H}}_{\mathcal{S},\mathcal{C}}(\mathcal{P}) \cup \mathcal{H}_{\mathcal{S}_c}(\mathcal{P})$$

[1] The stochastic sensitivity analysis does not yield deterministic results.

4. Use the current dynamic bounding working Pareto-front $\tilde{\mathcal{F}}^{\mathrm{w}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})$ to calculate the fitness of $c$ according to Definition 4.4.7

- After each exploration iteration do the following

1. Update the dynamic bounding working Pareto-front $\tilde{\mathcal{F}}^{\mathrm{w}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})$ by integrating all working Pareto-fronts stored in $\Gamma^{\mathrm{w}}$ according to Definition 4.4.4
2. Update the dynamic bounding non-working Pareto-front $\tilde{\mathcal{F}}^{\mathrm{nw}}_{\mathcal{S},\mathcal{C}}(\mathcal{P})$ by integrating all non-working Pareto-fronts stored in $\Gamma^{\mathrm{nw}}$ according to Definition 4.4.5

**Illustrative example.** Figure 4.25 gives an example for the dynamic robustness approximation approach presented in this section.

Figure 4.25a shows the initial approximations of the dynamic bounding Pareto-fronts. Two new parameter configurations are evaluated in the pending exploration iteration. The approximations of the bounding Pareto-fronts for these two parameter configurations, obtained through stochastic multi-dimensional sensitivity analysis, are visualized in Figures 4.25b and 4.25d, respectively. The derived fitness values are illustrated in Figures 4.25c and 4.25e. Figure 4.25f shows the updated dynamic bounding Pareto-fronts at the end of the performed exploration iteration.

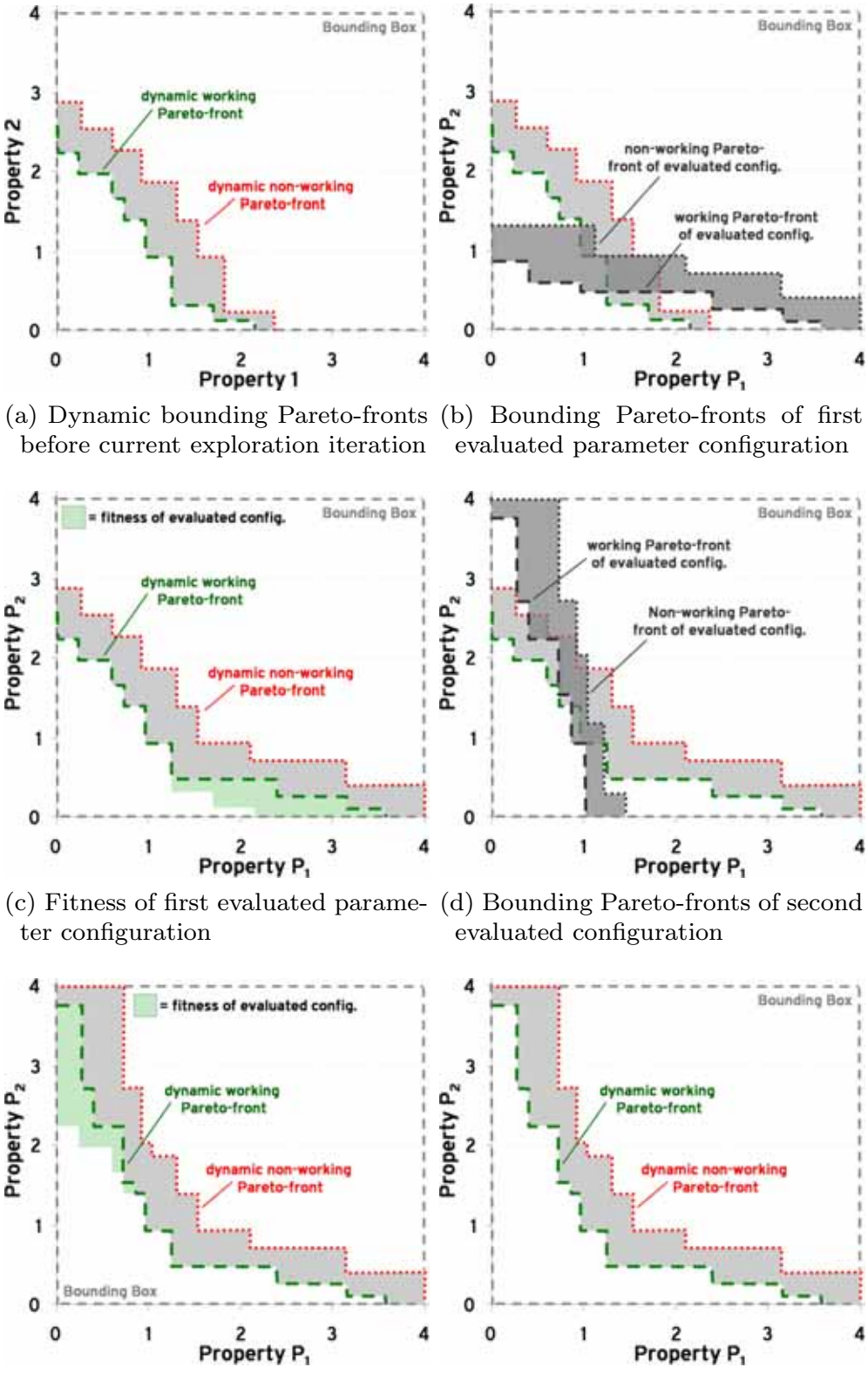
## 4.5 Case study

In this section robustness optimization is performed for the example system introduced in Section 2.3. Thereby, several two- and three-dimensional static and dynamic robustness optimization problems for dependent system properties are investigated.

In all experiments the search space consists of the priority assignments on computational resources (i.e. ARM, DSP, uC, and PPC) as well as on the interconnecting CAN bus. For the dynamic cases it is examined in which parts of the system it is particularly advantageous to introduce reconfiguration mechanisms. Note that for these considerations the configurations with optimal static robustness properties are taken as baseline.

### 4.5.1 Two-dimensional robustness optimization

First, the two-dimensional WCCT robustnesses of the communication channel pairs belonging to the three sub-applications, i.e. $Sens \rightarrow Act$, $Cam \rightarrow V_{out}$, and $S_{in} \rightarrow S_{out}$, are investigated. Additionally, the two-dimensional WCCT/WCET robustness of communication channel $C3$




(a) Dynamic bounding Pareto-fronts before current exploration iteration

(b) Bounding Pareto-fronts of first evaluated parameter configuration

(c) Fitness of first evaluated parameter configuration

(d) Bounding Pareto-fronts of second evaluated configuration

(e) Fitness of second evaluated parameter configuration

(f) Dynamic bounding Pareto-fronts after current exploration iteration

Fig. 4.25: Dynamic design robustness approximation example

and task $T1$ is optimized. For all experiments the involved tasks and communication channels are weighted 1.

The experiments were performed on an Intel Core Duo T2400 (2*1.83 GHz). The outer exploration loop varied scheduling parameters and was executed during 20 generation each consisting of 25 different parameter configurations. In the inner loop the robustness properties of discovered feasible parameter configurations were evaluated using stochastic sensitivity analysis with resolution 200 during 5 generation with 25 individuals each. Thereby, the average analysis time for a single individual was 150 ms. Additionally, two one-dimensional sensitivity analyses needed to be performed for each evaluated parameter configuration to determine the bounding box containing the sought-after sensitivity front. Consequently, the robustness evaluation of a single feasible parameter configuration took approximately 20 seconds (using both cores of the T2400). Note that infeasible parameter configuration do not possess any robustness properties, and thus did not need to be evaluated.

During exploration on average about 100 feasible parameter configurations were found and needed to be evaluated, translating into an overall robustness optimization time of between 30 and 35 minutes. Note that the run-time is independent of whether static or dynamic robustness properties are considered.

In the following the robustness optimization results are discussed. All considered parameter configurations are listed in Table 4.1.

| # | *CAN* | *ARM* | *DSP* | *uC* | *PPC* |
|---|---|---|---|---|---|
| 1 | $C0 > C4 > C5 > C2 > C3 > C1$ | $T6 > T1 > T8$ | $T7 > T4$ | $T3 > T0$ | $T2 > T5$ |
| 2 | $C4 > C5 > C2 > C0 > C1 > C3$ | $T6 > T1 > T8$ | $T7 > T4$ | $T3 > T0$ | $T2 > T5$ |
| 3 | $C4 > C0 > C5 > C2 > C1 > C3$ | $T6 > T1 > T8$ | $T7 > T4$ | $T3 > T0$ | $T2 > T5$ |
| 4 | $C4 > C2 > C5 > C0 > C1 > C3$ | $T6 > T1 > T8$ | $T7 > T4$ | $T3 > T0$ | $T2 > T5$ |
| 5 | $C0 > C5 > C4 > C2 > C1 > C3$ | $T6 > T1 > T8$ | $T7 > T4$ | $T3 > T0$ | $T2 > T5$ |
| 6 | $C4 > C0 > C5 > C2 > C3 > C1$ | $T6 > T1 > T8$ | $T7 > T4$ | $T3 > T0$ | $T2 > T5$ |
| 7 | $C4 > C5 > C0 > C2 > C1 > C3$ | $T6 > T1 > T8$ | $T7 > T4$ | $T0 > T3$ | $T2 > T5$ |
| 8 | $C4 > C5 > C0 > C2 > C1 > C3$ | $T6 > T1 > T8$ | $T7 > T4$ | $T3 > T0$ | $T2 > T5$ |
| 9 | $C0 > C4 > C5 > C2 > C1 > C3$ | $T6 > T1 > T8$ | $T7 > T4$ | $T3 > T0$ | $T2 > T5$ |

Table 4.1: Considered system parameter configurations, i.e. priority assignments, during two-dimensional robustness optimization

**Communication channels C0 and C1.** The robustnesses of the original parameter configuration and the parameter configurations exhibiting optimal SDR properties (configuration #1) are visualized in Figure 4.26a. The robustness of the original parameter configuration is equal to

$$\mathrm{SDR}^{\mathrm{dep}}_{org}(\{C0, C1\}) = 1.0583.$$

Through static robustness optimization it can be increased by more than factor 7:

$$\mathrm{SDR}^{\mathrm{dep}}_{\#1}(\{C0, C1\}) = 7.7244.$$

In the dynamic case, i.e. assuming reconfigurability of the communication channel priorities on the interconnecting CAN bus, two additional configurations (#2 and #3) are found increasing robustness by 10.55% to

$$\mathrm{DDR}^{\mathrm{dep}}_{\{CAN\}}(\{C0, C1\}) = 8.539.$$

The space of feasible property value combinations for the dynamic case is visualized in Figure 4.26b. Additional experiments reveal that no further dynamic robustness improvement can be achieved by adding other resources to the reconfiguration space.

**Communication channels C2 and C3.** The overall WCCT robustness of the communication channels C2 and C3 is far smaller compared to C0 and C1. The original parameter configuration possesses only little robustness:

$$\mathrm{SDR}^{\mathrm{dep}}_{org}(\{C2, C3\}) = 0.1682.$$

By means of static robustness optimization parameter configuration #2 exhibiting optimal SDR properties is discovered:

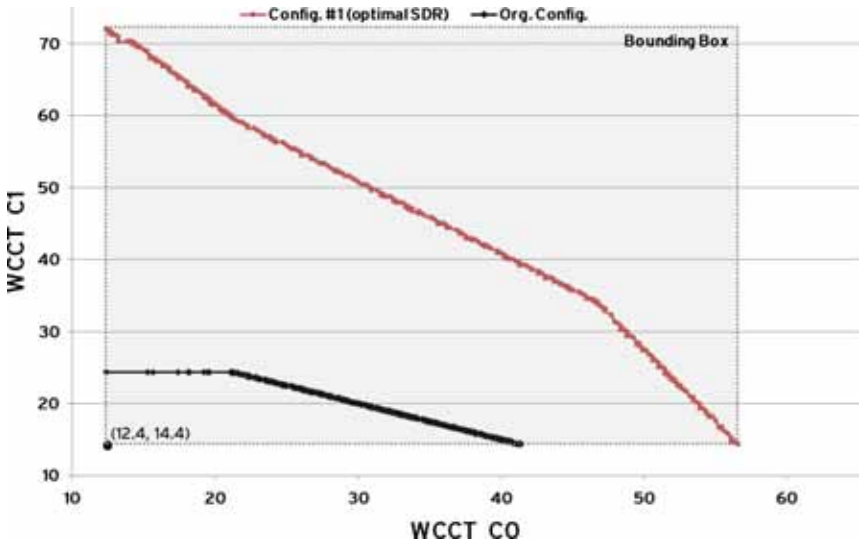
$$\mathrm{SDR}^{\mathrm{dep}}_{\#2}(\{C2, C3\}) = 0.66.$$

This corresponds to a robustness increase of almost 300 % compared to the original configuration. The spaces of feasible property value combinations for the original and the optimized parameter configurations are visualized in Figure 4.27a.

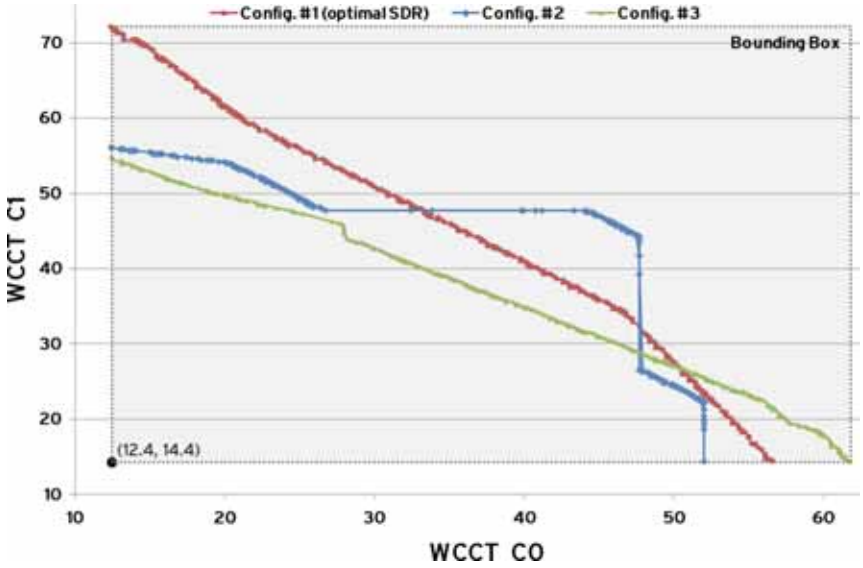
Again, robustness can be further increased by assuming reconfigurability on the CAN bus. As can be seen in Figure 4.27b two additional parameter configurations exist (#4 and #5) increasing robustness by 15.79 % to

$$\mathrm{DDR}^{\mathrm{dep}}_{\{CAN\}}(\{C2, C3\}) = 0.7642.$$

Like in the previous case, reconfigurability on other resources does not lead to an additional robustness increase.




(a) Static design robustness - configuration #1 has optimal SDR properties



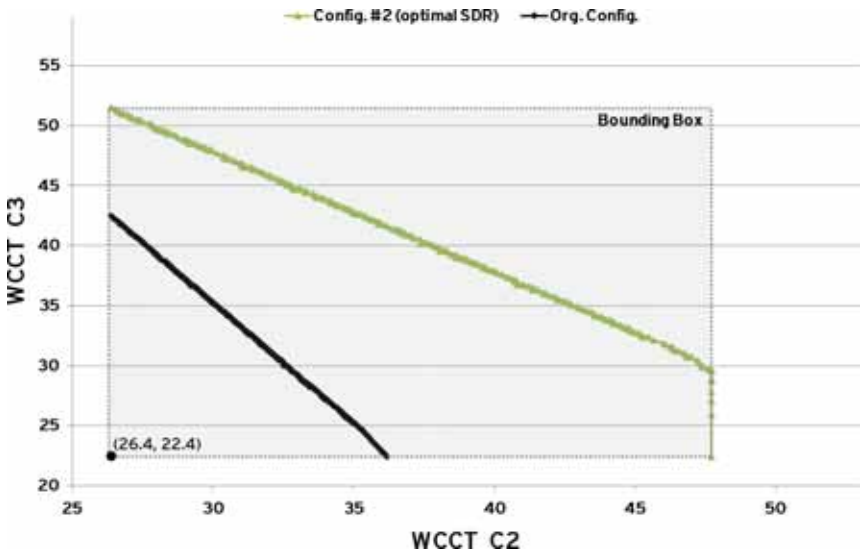

(b) Dynamic design robustness - reconfigurability on CAN

Fig. 4.26: Two-dimensional static and dynamic WCCT robustness of communication channels C0 and C1

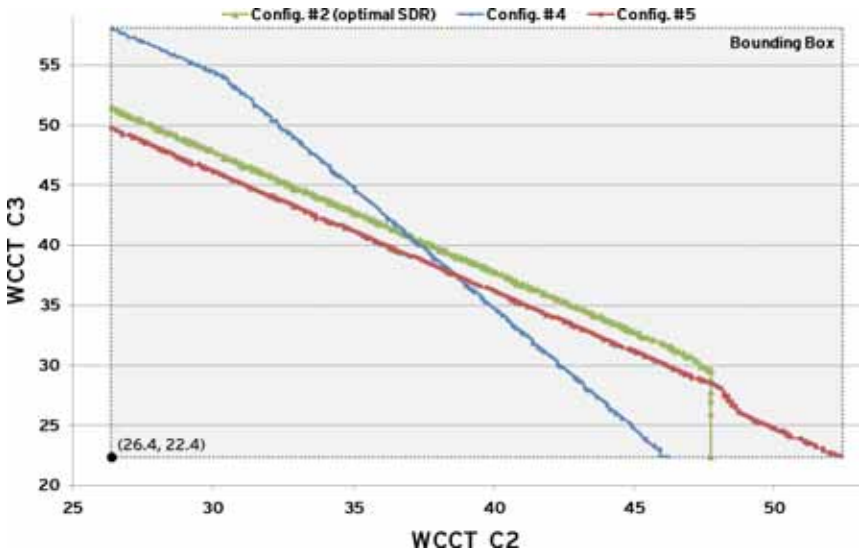
**Communication channels C4 and C5.** The original configuration possesses a robustness of

$$\mathrm{SDR}_{org}^{\mathrm{dep}}(\{C4, C5\}) = 0.459$$

with respect to WCCT variations of the communication channels C4 and C5. This rather low robustness can be increased through static robustness optimization by more than factor 7. The robustness of the optimal configuration # 6 is equal to




(a) Static design robustness - configuration #2 has optimal SDR properties




(b) Dynamic design robustness - reconfigurability on CAN

Fig. 4.27: Two-dimensional static and dynamic WCCT robustness of communication channels C2 and C3

$$\mathrm{SDR}^{\mathrm{dep}}_{\#6}(\{C4, C5\}) = 3.3164.$$

The spaces of feasible property value combinations for the original and the optimized parameter configurations are visualized in Figure 4.28.

Note that in contrast to the first two experiments the WCCT robustness of C4 and C5 cannot be improved through dynamic system behavior. In other words, the static and dynamic design robustness with respect to WCCT variations of the communication channels C4 and C5 are equivalent:

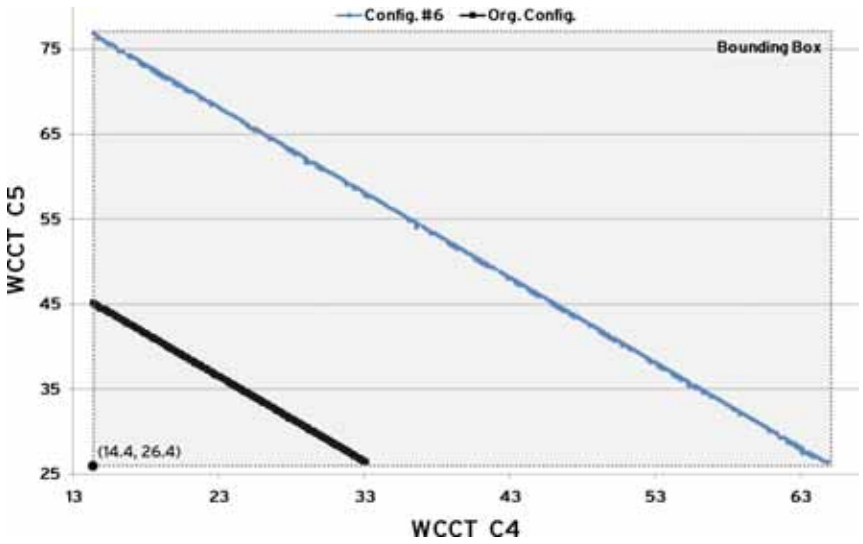



Fig. 4.28: Two-dimensional static and dynamic WCCT robustness of communication channels C4 and C5. Dynamic robustness is equal to static robustness.

$$\mathrm{SDR}^{\mathrm{dep}}_{\#6}(\{C4, C5\}) = \mathrm{DDR}^{\mathrm{dep}}_{\{CAN,ARM,DSP,uC,PPC\}}(\{C4, C5\}).$$

**Communication channel C3 and task T1.** In the last two-dimensional experiment the WCCT/WCET robustness of communication channel C3 and task T1 is optimized. The robustnesses of the original parameter configuration and the parameter configurations exhibiting optimal SDR properties (configuration #3) are visualized in Figure 4.29a. The robustness of the original configuration is very low:

$$\mathrm{SDR}^{\mathrm{dep}}_{org}(\{C3, T1\}) = 0.0503.$$

However, by means of static robustness optimization configuration #3 is found increasing robustness by more than factor 6:

$$\mathrm{SDR}^{\mathrm{dep}}_{\#3}(\{C3, T1\}) = 0.3226.$$

Robustness can be further increased if dynamic system behavior is assumed. For instance, reconfigurability of the communication channel priorities on the CAN bus leads to the discovery of three additional configurations (#4, #8, and #9) increasing robustness by 5%:

$$\mathrm{DDR}^{\mathrm{dep}}_{\{CAN\}}(\{C3, T1\}) = 0.339.$$

Unlike in the previous experiments additional robustness improvement can be achieved through reconfigurability on uC. Altogether, a robust-

ness increase of nearly 10 % is obtained through reconfigurability on the CAN bus and uC:

$$\mathrm{DDR}^{\mathrm{dep}}_{\{CAN,uC\}}(\{C3,T1\}) = 0.3531.$$

The space of feasible property value combinations for the dynamic case is visualized in Figure 4.29b. As can be seen, the low robustness is mainly due to T1. Therefore, reconfigurability on uC is particularly interesting since it increases the dynamic robustness in the dimension of T1.

### 4.5.2 Three-dimensional robustness optimization

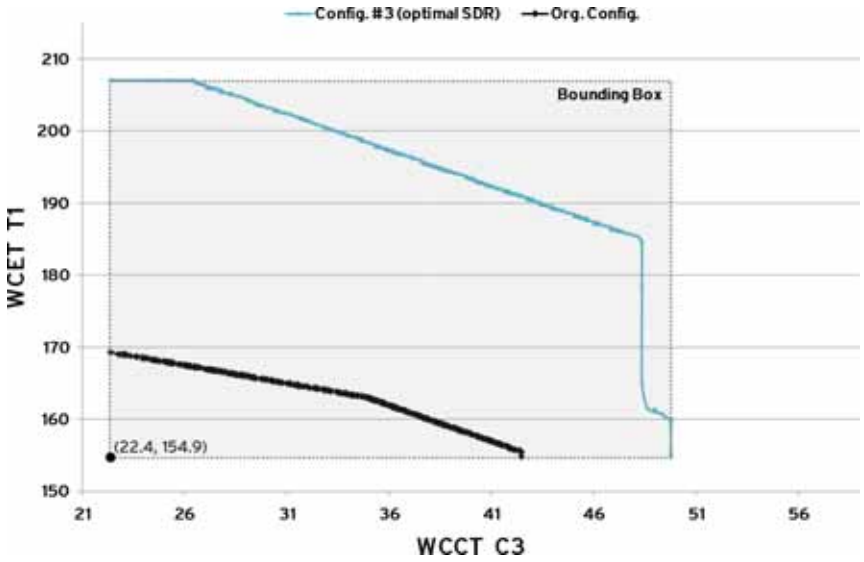
In this section two three-dimensional robustness optimizations scenarios are discussed. First, the robustness of task T1 together with incoming and outgoing communication channels C0 and C1 is investigated. Second, the robustness of the tasks belonging to the application $Cam \rightarrow V_{out}$, i.e. T3, T4, and T5, is optimized. For both experiments the involved tasks and communication channels are weighted 1.

Like in the two-dimensional case the experiments were performed on an Intel Core Duo T2400 (2*1.83 GHz). The outer loop varying scheduling parameters was left untouched (20 generation with 25 individuals each). However, the inner loop performing stochastic sensitivity analysis of feasible parameter configurations needs to run longer for accurate robustness assessment in the three-dimensional case. Therefore, a generation count of 10 each consisting of 100 individuals was chosen. Thereby, the average analysis time per individual was 150 ms. Hence, together with three one-dimensional sensitivity analyses that are necessary to determine the bounding cube (3*5 seconds), the overall evaluation time for one feasible parameter configuration added up to approximately 90 seconds (using both cores).

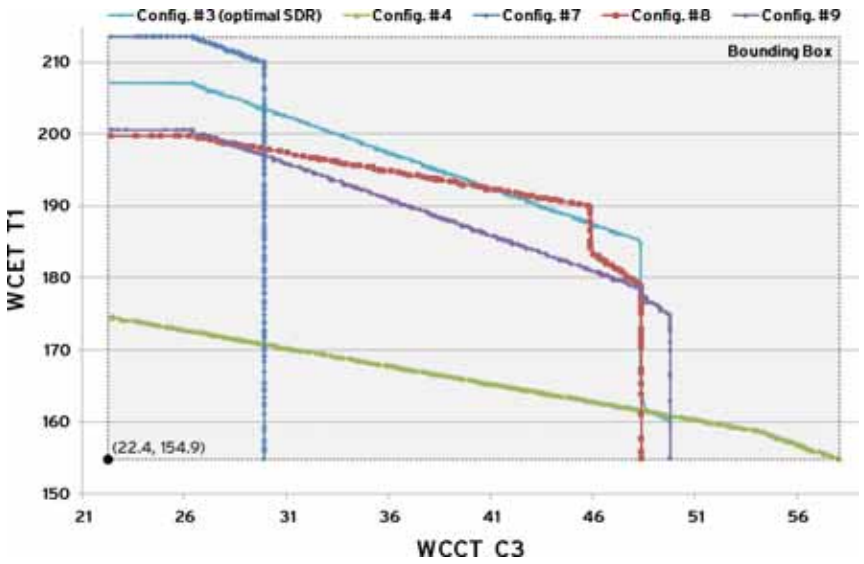
During exploration on average about 100 feasible parameter configurations were found and needed to be evaluated, translating into an overall robustness optimization time of approximately 2 hours and 30 minutes. Note that the run-time is independent of whether static or dynamic robustness properties are considered.

**Task T1 with incoming and outgoing communication channels C0 and C1.** Figure 4.30 visualizes the robustness of the original parameter configuration. Its robustness is rather low:

$$\mathrm{SDR}^{\mathrm{dep}}_{org}(\{C0,C1,T1\}) = 0.0336.$$




(a) Static design robustness - configuration #3 has optimal SDR properties




(b) Dynamic design robustness - reconfigurability on CAN and uC (configuration #7 is obtained through reconfigurability on uC)

Fig. 4.29: Two-dimensional static and dynamic WCCT / WCET robustness of communication channel C3 and task T1

However, it can be substantially increased by means of robustness optimization. The parameter configurations that were discovered during static and dynamic robustness optimization of task T1 with communication channels C0 and C1 are listed in Table 4.2.

Parameter configuration #2 possesses optimal static robustness properties:

$$\mathrm{SDR}^{\mathrm{dep}}_{\#2}(\{C0, C1, T1\}) = 1.3138.$$

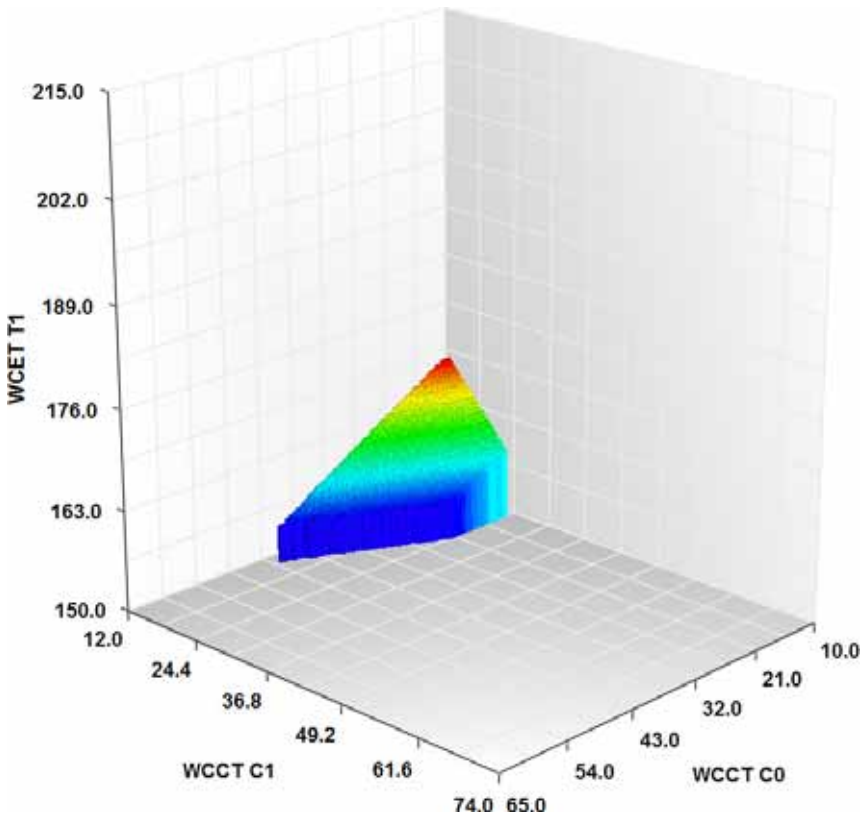



Fig. 4.30: Three-dimensional WCCT/WCET robustness of communication channels C0 and C1 as well as task T1 - original configuration

| # | *CAN* | *ARM* | *DSP* | *uC* | *PPC* |
|---|---|---|---|---|---|
| 1 | $C0 > C4 > C5 > C2 > C3 > C1$ | $T6 > T1 > T8$ | $T7 > T4$ | $T3 > T0$ | $T2 > T5$ |
| 2 | $C4 > C0 > C5 > C2 > C1 > C3$ | $T6 > T1 > T8$ | $T7 > T4$ | $T3 > T0$ | $T2 > T5$ |
| 3 | $C4 > C0 > C5 > C2 > C3 > C1$ | $T6 > T1 > T8$ | $T7 > T4$ | $T3 > T0$ | $T2 > T5$ |
| 4 | $C4 > C2 > C5 > C0 > C1 > C3$ | $T6 > T1 > T8$ | $T7 > T4$ | $T0 > T3$ | $T2 > T5$ |
| 5 | $C4 > C5 > C0 > C2 > C1 > C3$ | $T6 > T1 > T8$ | $T7 > T4$ | $T0 > T3$ | $T2 > T5$ |
| 6 | $C4 > C5 > C0 > C2 > C1 > C3$ | $T6 > T1 > T8$ | $T7 > T4$ | $T3 > T0$ | $T2 > T5$ |
| 7 | $C4 > C5 > C0 > C2 > C3 > C1$ | $T6 > T1 > T8$ | $T7 > T4$ | $T3 > T0$ | $T2 > T5$ |
| 8 | $C4 > C5 > C2 > C0 > C1 > C3$ | $T6 > T1 > T8$ | $T7 > T4$ | $T3 > T0$ | $T2 > T5$ |

Table 4.2: Considered system parameter configurations, i.e. priority assignments, during three-dimensional WCCT/WCET robustness optimization of communication channels C0 and C1 as well as task T1

This corresponds to a robustness increase by factor of 40 compared to the original parameter configuration. The space of valid property value combinations covered by configuration #2 is visualized in Figure 4.31a.

If reconfigurability is included into robustness considerations this value can be further increased. In the first step it is, therefore, assumed that the communication channel priorities on the CAN bus are reconfigurable. This measure leads to a dynamic robustness increase of nearly 10% compared to the static case:

$$\mathrm{DDR}^{\mathrm{dep}}_{\{CAN\}}(\{C0, C1, T1\}) = 1.4369.$$

Further experiments reveal that an additional dynamic robustness improvement is possible by adding uC to the reconfiguration space:

$$\mathrm{DDR}^{\mathrm{dep}}_{\{CAN,uC\}}(\{C0, C1, T1\}) = 1.4898.$$

This corresponds to a dynamic robustness increase of 13.4% compared to the static case. The overall dynamic robustness, i.e. assuming reconfigurability on the CAN bus and uC, is visualized in Figure 4.31.

**Tasks T3, T4, and T5.** Figure 4.32 visualizes the space of feasible property value combinations covered by the original parameter configuration. Its robustness is equal to

$$\mathrm{SDR}^{\mathrm{dep}}_{org}(\{T3, T4, T5\}) = 0.0193.$$

Through robustness optimization it is tried to find parameter configuration with better robustness properties. The parameter configurations that were discovered during static and dynamic robustness optimization of tasks T3, T4, and T5 are listed in Table 4.3.

| # | *CAN* | *ARM* | *DSP* | *uC* | *PPC* |
|---|---|---|---|---|---|
| 1 | $C0, C4, C2, C5, C3, C1$ | $T6, T1, T8$ | $T7, T4$ | $T3, T0$ | $T2, T5$ |
| 2 | $C2, C0, C5, C4, C3, C1$ | $T6, T1, T8$ | $T7, T4$ | $T3, T0$ | $T2, T5$ |
| 3 | $C2, C4, C5, C0, C3, C1$ | $T6, T1, T8$ | $T7, T4$ | $T0, T3$ | $T2, T5$ |

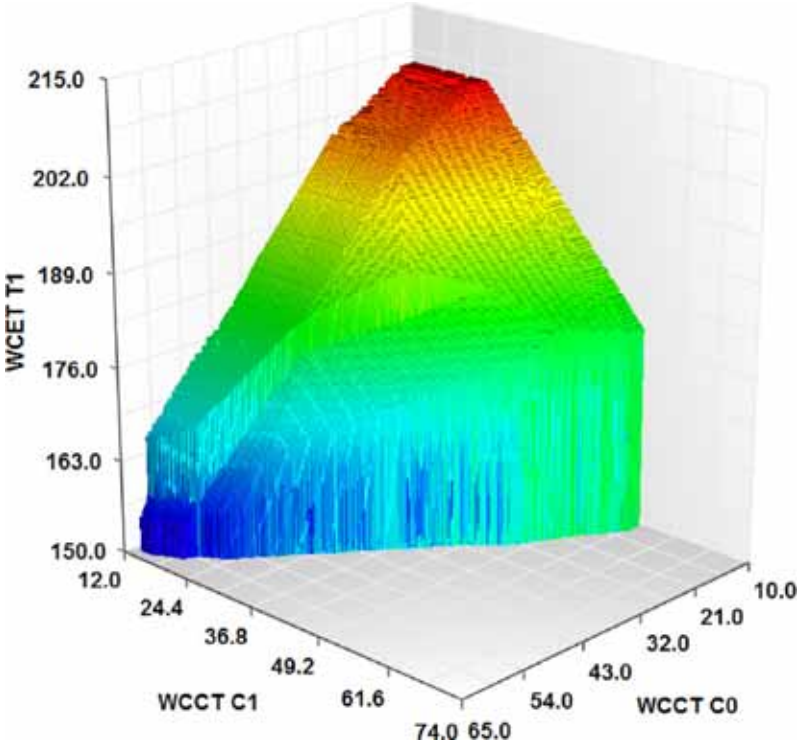
Table 4.3: Considered system parameter configurations, i.e. priority assignments, during three-dimensional WCET robustness optimization of tasks T3, T4, and T5

Parameter configuration #2 possesses optimal static robustness properties:

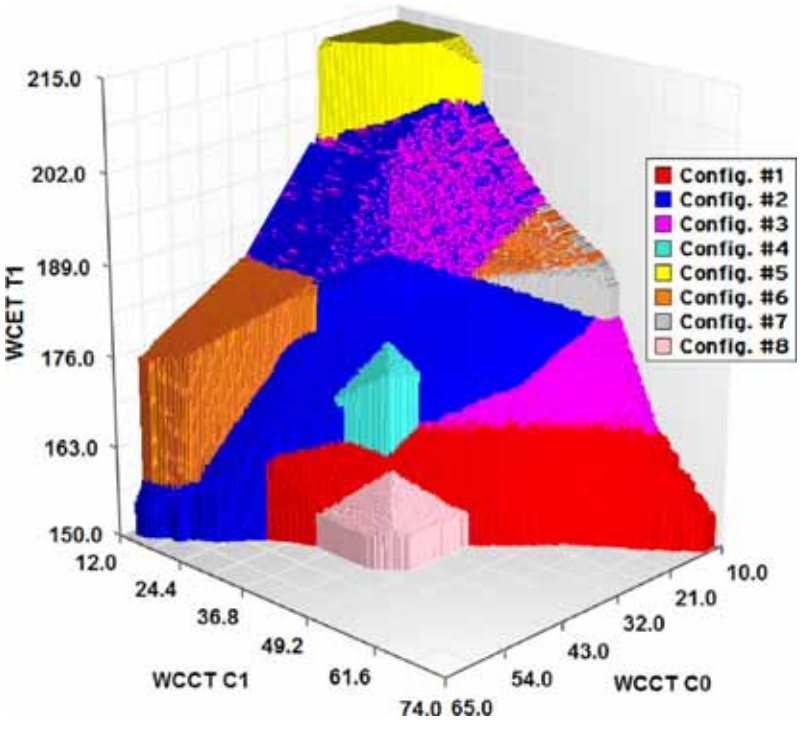
$$\mathrm{SDR}^{\mathrm{dep}}_{\#2}(\{T3, T4, T5\}) = 0.0452.$$

This corresponds to a robustness increase of 134% compared to the original parameter configuration. The space of valid property value combinations covered by configuration #2 is visualized in Figure 4.33a.

Again, it is tried to further increase robustness through reconfiguration mechanisms. Reconfigurability of the communication channel priorities on the CAN bus leads to a dynamic robustness increase of more than 10% compared to the static case:




(a) Static design robustness - configuration #2 has optimal SDR properties




(b) Dynamic design robustness - reconfigurability on CAN and uC (configurations #4 and #5 are obtained through reconfigurability on uC)

Fig. 4.31: Three-dimensional static and dynamic WCCT/WCET robustness of communication channels C0 and C1 as well as task T1

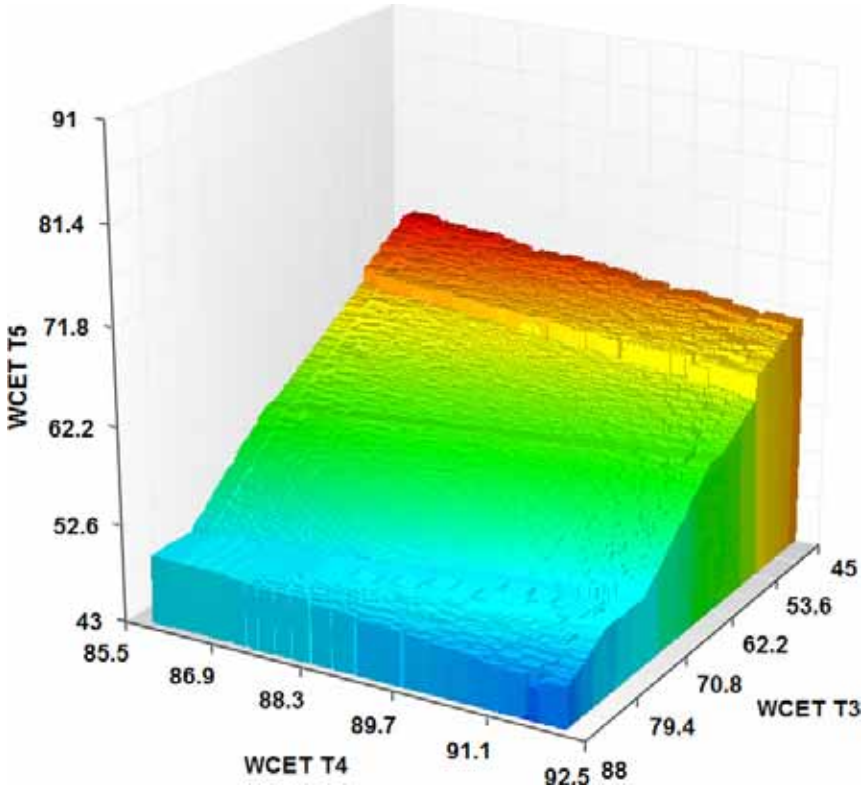



Fig. 4.32: Three-dimensional static and dynamic WCET robustness of tasks T3, T4, and T5 - original configuration

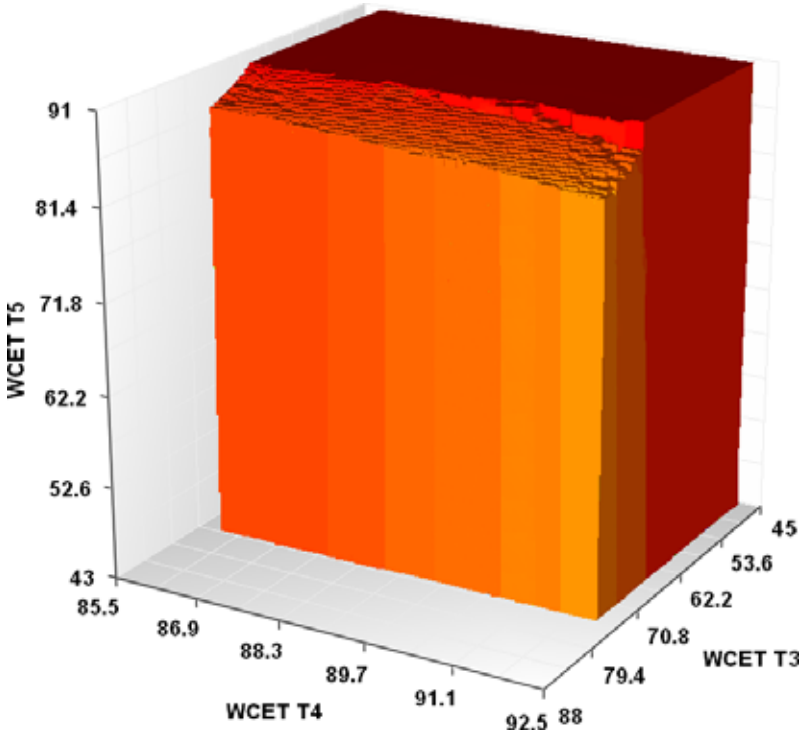
$$\mathrm{DDR}^{\mathrm{dep}}_{\{CAN\}}(\{T3, T4, T5\}) = 0.04996.$$

Like in the previous experiment an additional dynamic robustness improvement is possible by adding uC to the reconfiguration space:

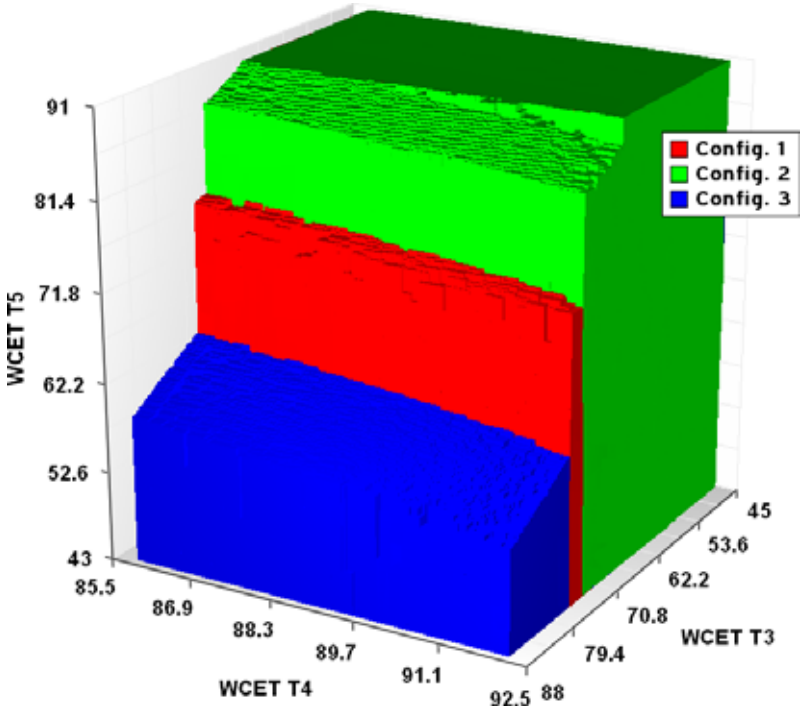
$$\mathrm{DDR}^{\mathrm{dep}}_{\{CAN,uC\}}(\{T3, T4, T5\}) = 0.0532.$$

This corresponds to a robustness increase of 17.7% compared to the static case. The overall dynamic robustness, i.e. assuming reconfigurability on the CAN bus and uC, is visualized in Figure 4.33b.

Compared to the previous experiments it can be concluded that only low robustness can be achieved for the application $Cam \rightarrow V_{out}$, even if dynamic system behavior is assumed. The main reason is the fragility of task T4 running on the DSP. Its original WCET of 86 time units cannot be increased further than 92.15 time units, which corresponds to a slack of little more than 7% (compare Figure 4.33b).




(a) Static design robustness - configuration #2 has optimal SDR properties




(b) Dynamic design robustness - reconfigurability on CAN and uC (configuration #3 is obtained through reconfigurability on uC)

Fig. 4.33: Three-dimensional static and dynamic WCET robustness of tasks T3, T4, and T5

Chapter 5

# COMBINED PERFORMANCE ANALYSIS OF EMBEDDED SYSTEMS

Over the last decade, many analytical methods for performance analysis of embedded systems were proposed, examples can be found in [79, 131, 77, 115, 84, 43, 113, 50]. Most of them are specialized and dedicated to specific domains, whereas others try to be general enough to serve for several application scenarios. However, the heterogeneity of nowadays embedded systems challenges state-of-the-art performance analysis methods with respect to analysis capabilities and accuracy. To solve this problem, modular design approaches have been proposed that allow to compose analyses of different resource sharing strategies. They share a similar overall analysis methodology but use different models of computation. They have demonstrated high modeling accuracy and simplicity for certain areas but are less appropriate for others.

This fact is pointed out in [83], where different state-of-the-art modular performance analysis approaches are compared, including SymTA/S (Symbolic Timing Analysis for Systems) [50], MPA (Modular Performance Analysis) [126] based on RTC (Real-Time Calculus) [17], UPPAAL [8, 46], and MAST (Modeling and Analysis Suite for Real-Time Applications) [43]. The authors show that even for small heterogeneous distributed systems the different approaches exhibit remarkable differences in terms of analysis accuracy. Thereby, none of the compared approaches performs best in all cases. It can, therefore, be concluded that the choice of an appropriate abstraction fitting the characteristics of the analyzed system is important to obtain tight and expressive analysis results.

In this chapter we want to contribute in solving the problem of providing accurate performance predictions for general distributed real-time systems. Rather than searching for a unified more complex model, we

propose to exploit a *compositional system level analysis methodology* to couple different modular performance analysis techniques.

As an example the coupling of SymTA/S and MPA is demonstrated. The strength of SymTA/S is its flexibility in modeling activation and execution dependencies allowing to precisely analyze complex real-world operating systems and bus protocols. Examples are ERCOSEK [29, 70] and CAN [1, 15] known from the automotive domain. MPA offers a very intuitive and efficient way to capture scheduling hierarchies [125], which requires substantial modeling effort using the SymTA/S approach. By coupling both approaches the advantages offered by both models of computation can be combined, allowing to model and analyze components in complex distributed system with the best fitting method. As will be seen in the experimental Section 5.5 this leads to tighter analysis results compared to monolithic analyses.

The remainder of this chapter is structured as follows. First, the compositional performance analysis methodology is briefly discussed (Section 5.1). Afterwards, the two modular performance analysis approaches that are couple in this chapter are introduced: SymTA/S (Section 5.2) and MPA (Section 5.3). Then, it is explained in detail how both approaches can be combined by exploiting their common formal basis (Section 5.4). Finally, the seamless analysis interaction and accuracy benefits over using the individual approaches alone are demonstrated by means of a complicated example with dependencies and feedback (Section 5.5).

Note that the method and the results presented in this chapter have been developed to equal parts together with Simon Künzli (ETH Zürich), and that the approach was originally published in [61].

## 5.1 Compositional performance analysis

In the past years *compositional performance analysis* approaches [50, 17, 114, 52] have received increasing attention in the real-time and embedded systems research communities. Compositional performance analyses exhibit great flexibility and scalability for timing and performance analysis of complex distributed embedded real-time systems. Their basic idea is to integrate local performance analysis techniques, e.g. scheduling analysis techniques known from real-time research, into system level analyses. This composition is achieved by connecting the component's inputs and outputs by stream representations of their communication behavior.

The possibility to compose existing local performance analysis techniques to system level analyses represents a major advantage over holistic analysis approaches [115, 43, 84]. Holistic approaches represent analysis techniques for fixed system architectures. They have, therefore, diffi-

culties to scale to larger systems with new components. More drastically, in case of system changes it is possible that holistic analyses are no longer applicable without major adaptations. However, holistic approaches may make it easier to take global performance effects into account. Even though, compositional approaches have been recently extended to consider global performance dependencies such as relative offsets and relative jitters [48], specialized holistic approaches might yield tighter analysis results in some cases.

The flexibility of compositional performance analysis even spans different approaches based on different models of computation. For instance, in this chapter it is demonstrated how two state-of-the-art compositional performance analysis methods, namely SymTA/S [50, 87] and Real-Time Calculus (RTC) [17, 126], can be coupled using a bi-directional conversion algorithm for their internal stream representations. The possibility to couple different methods represents another big advantage of the compositional performance analysis methodology. It allows to combine the strengths of several methods, that are often tailored for specific system architectures or application domains, without the need to search for a unified more complex model.

The remainder of this section is structured as follows. First, the concept of local component analysis in the context of compositional performance analysis is discussed (Section 5.1.1). Afterwards, the compositional system level analysis loop solving the system level performance analysis problem for distributed systems is explained (Section 5.1.2). In cases that two or more components are circularly dependent, system-level analysis cannot be performed without generating a so-called *starting point*. This issue is discussed in Section 5.1.3.

### 5.1.1 Local component analysis

The main idea of the compositional performance analysis methodology consists in integrating local component analyses into system level analysis. Note that the basic performance model that is used by many performance analysis approaches at component level has been introduced in Section 2.2.

Based on the underlying resource sharing strategy as well as stream representations of the incoming workload modeled through so-called *activating event models*, local component analyses systematically derive worst-case scenarios to calculate worst-case (sometimes also best-case) task response times (BCRT, WCRT), i.e. the time between task activation and task completion, for all tasks sharing the same resource. Thereby, local component analyses guarantee that all observable re-

sponse times fall into the calculated [best-case, worst-case] interval. These analyses are therefore called *conservative*.

Note that different approaches use different models of computation to perform local component analyses. SymTA/S, for instance, is based on the algebraic solution of so-called response time formulas using the *sliding window technique* proposed by Lehoczky [68], whereas the Real-Time Calculus utilizes arrival curves and service curves to characterize workload and processing capabilities of components, and determine their real-time behavior [112]. These concepts are based on the *network calculus*. For details please refer to [13].

Additionally, local component analyses determine the communication behavior at the outputs of the analyzed tasks by considering the effects of scheduling. Therefore, it is usually assumed that tasks produce output events at the end of each execution. Like the input timing behavior, also the output timing behavior is captured by event models. The way in which these *output event models* are derived depends on the underlying analysis technique.

### 5.1.2 Compositional system level analysis loop

The basic idea of the compositional system level analysis is visualized in Figure 5.1, see e.g. [50, 96, 17, 113].

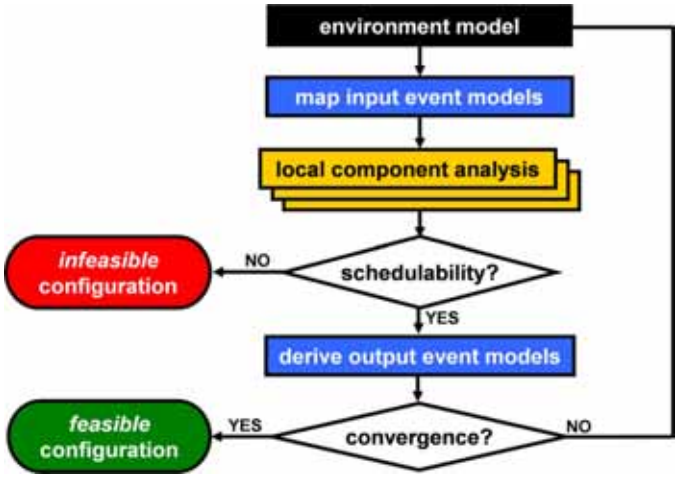



Fig. 5.1: Compositional system level analysis loop

Compositional system level analysis alternates local component analysis as explained in Section 5.1.1 and output event model propagation.

More precisely, in each global iteration of the compositional system level analysis, local analysis is performed for all component to derive response times and output event models. Afterwards, the calculated output event models are propagated to the connected components, where

they are used as activating event models for the subsequent global iteration.

Obviously, this iterative analysis represents a fix-point problem. If after an iteration all calculated output event models stay unmodified, convergence is reached and the last calculated task response times are valid. This holds for analysis techniques that do not contain states in their analytical model, otherwise not only unchanged output event models, but also steady internal state has to be considered as convergence criterion (e.g. for the models presented in [18]). In cases where no convergence is reached, no statement can be made about the analyzed system.

### 5.1.3 Starting point generation

In order to successfully apply compositional system level analysis, the input event models of all components need to be known or must be computable by local component analysis. Obviously, for systems containing feed-back between two or more components this is not the case, and thus system level analysis cannot be performed without additional measures. The concrete strategy to overcome this problem depends on the component types and their input event models. One possibility is the so-called *starting point generation* that was proposed in the context of the SymTA/S approach [50, 96], but that is also applicable to other compositional performance analysis approaches such as MPA.

The starting point generation consists in propagating external event models along all task chains in the system without considering the effects of scheduling, until initial activating event models are available for all tasks. This approach is safe since, on the one hand, scheduling cannot change the period of an event stream, and, on the other hand, scheduling can only *increase* the jitter contained in an event stream [115]. Since increased jitter leads in the worst-case (best-case) to potentially more (less) events in any given time interval $\Delta t$, the initial assumption of ignoring scheduling effects is safe.

## 5.2 Symbolic Timing Analysis for Systems SymTA/S

At component level SymTA/S uses formal analysis techniques based on the busy window technique proposed by Lehoczky [68]. Currently, SymTA/S offers local analysis techniques for fixed priority scheduling (preemptive and non-preemptive), TDMA, Round Robin [94], EDF [71], CAN [15], Flexray [33], and various OSEK variants [70] such as ERCOSEK [29]. In this section it is briefly explained how SymTA/S couples

local component analyses according to the compositional performance analysis methodology.

### 5.2.1 Composition using standard event models

The SymTA/S approach [96, 54, 50] utilizes so-called *standard event models* to describe the timing behavior at component inputs and outputs. These models represent the necessary abstraction to perform local component analyses as described in Section 5.1.1, and enable their easy composition according to the compositional system level analysis loop discussed in Section 5.1.2. *Standard event models* are characterized by three parameters:

- The *activation period* $P$ denotes the average interval between two consecutive task activations.
- The *activation jitter* $J$ denotes the maximum task activation delay relative to its activation period.
- The *minimum inter-arrival distance* $d$ denotes the minimum time between two consecutive activations of the same task.

Note that standard event models have already been discussed in Section 2.2.

### 5.2.2 Output event model calculation

The compositional performance analysis methodology defines very clear component interfaces. One of the key requirements is that components must be capable of determining their output communication behavior, i.e. output event models, considering the effects of scheduling.

The SymTA/S standard event models allow to specify very simple rules to obtain output event models during local component analysis. Note that in the simplest case, i.e. if tasks produce exactly one output event for each activating event, the output event model period equals the activation period. A discussion about how output event model periods are determined for more complex semantics, e.g. considering rate transitions, can be found in [54].

The output event model jitter $J_{out}$ is calculated by adding the difference between maximum and minimum response times, *the response time jitter*, to the activating event model jitter $J_{in}$ [96]:

$$\mathcal{J}_{out} = \mathcal{J}_{in} + (t_{resp,max} - t_{resp,min})$$

Note that if the calculated output event model jitter is larger than the period, this information alone would indicate that early output events

could occur before late previous output events. In reality, output events cannot follow closer than the minimum response time of the producer task. To reflect this, the calculated output event model can be refined using the *minimum distance* parameter $d$.

An important aspect of the output event model calculation of SymTA/S is that not only worst-case but also best-case response times are calculated. The accurate calculation of best-case response times leads to smaller output jitters limiting transient load peaks on connected components, which, in turn, leads to tighter response times [130, 40].

Note that recently a more exact output jitter calculation algorithm for local component analysis based on standard event models was proposed [51]. The approach exploits that the task activation arriving with worst-case jitter does not necessarily experience the worst-case response time.

## 5.3 Modular Performance Analysis MPA

In [112], Thiele et al. proposed Real-Time Calculus (RTC), a mathematical framework for system-level performance analysis of distributed embedded systems. Like SymTA/S (see Section 5.2) also the RTC approach is based on the principles of the compositional performance analysis methodology presented in Section 5.1. However, the underlying analysis method [17, 113] is based on different formal models. RTC utilizes arrival curves and service curves to characterize workload and processing capabilities of components. System level analysis is performed by connecting arrival and service curves according to the system topology and utilized scheduling policy.

The concepts of RTC are implemented in a toolbox, called MPA [126] (Modular Performance Analysis). Using MPA, it is possible to determine worst-case bounds for on-chip memory requirements, overall throughput, and delay. MPA supports complex task activation schemes, including OR and AND, and several scheduling policies, such as fixed priority preemptive, EDF, GPS, and TDMA. Since resource capabilities are first class citizens of the RTC analysis method (see Section 5.3.2), hierarchical scheduling methods can be easily modeled and analyzed by arbitrarily combining the above mentioned basic schemes [125]. In the following sections the basic principles of the RTC/MPA approach are explained.

### 5.3.1 Arrival curves

For a given event stream, let $R(s,t)$ denote the number of events that arrive in the time interval $[s,t[$. The upper arrival curve, denoted by $\alpha^u(\Delta)$, gives an upper bound on the number of events in any interval $\Delta$.

Similarly, a lower bound on the number of events is given by the lower arrival curve $\alpha^l(\Delta)$. $R$, $\alpha^u$ and $\alpha^l$ are related as follows:

$$\forall s \; : \quad \alpha^l(\Delta) \leq R(s, s+\Delta) \leq \alpha^u(\Delta)$$

Arrival curves describe timing properties of a whole class of event streams, for example the average rate, burstiness, long-term, and short-term behavior. Upper and lower arrival curves can also be given for the standard event models that are used by SymTA/S (Figure 5.2).

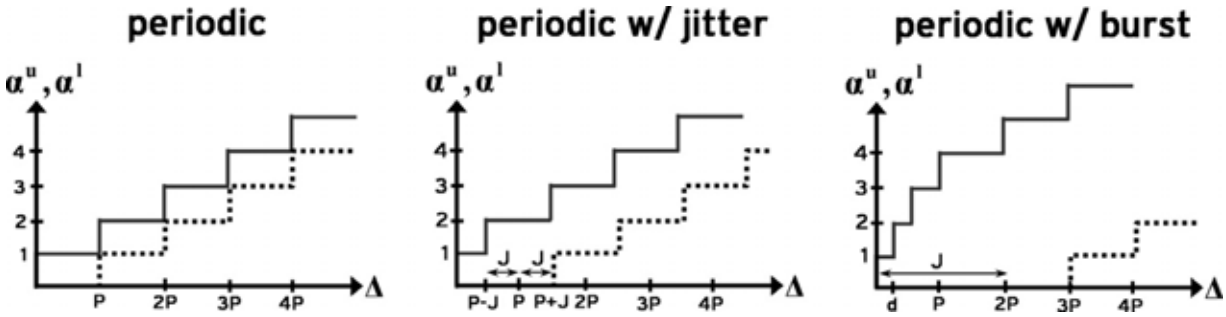



Fig. 5.2: Arrival curves for standard event models

### 5.3.2 Service curves

Similar to the description of workload with arrival curves, the RTC methodology uses so-called *service curves* to model processing capability of resources.

Let $C(s,t)$ denote the amount of processing units (e.g. in terms of time units) available on a resource in the time interval $[s,t[$. The upper service curve, denoted by $\beta^u(\Delta)$, gives an upper bound on the amount of available processing units in any time interval $\Delta$. A corresponding lower bound is given by the lower service curve $\beta^l(\Delta)$. $C$, $\beta^u$ and $\beta^l$ are related as follows:

$$\forall s \; : \quad \beta^l(\Delta) \leq C(s, s+\Delta) \leq \beta^u(\Delta)$$

### 5.3.3 System level performance analysis

Given the upper arrival curve $\alpha^u$ describing the worst-case workload as well as the lower service curve $\beta^l$ describing the minimum available processing capability of the executing resource, the RTC approach defines an elegant algebra to derive the worst-case response time as well as the worst-case buffering requirements (execution backlog) of a given task.

Additionally, the RTC methodology defines formulas to calculate the remaining processing capabilities of the executing resource, and the out-

put arrival curve describing the stream of processed events. The detailed formulas can be found in [17]. However, at this point a graphical interpretation shall be given of how worst-case response times and backlog can be calculated using the RTC approach (Figure 5.3).

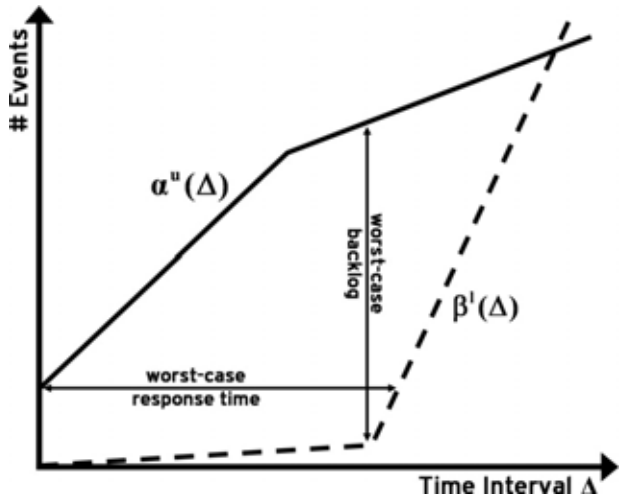



Fig. 5.3: Determination of worst-case response time and backlog using the RTC approach

In order to extend single task analysis to component analysis, scheduling needs to be considered. The RTC approach models scheduling by connecting involved processes by service curves according to the utilized resource sharing policy. For instance, static priority scheduling is modeled by connecting the highest priority task with the initial service curve of the executing resource. The output service curve, describing the remaining service capability after the highest priority task is served, is then granted to the task with the second highest priority. The remaining tasks are connected in the same way until the lowest priority task is served. Using different strategies to connect tasks competing for resource access, RTC can be applied to various additional scheduling policies including EDF, GPS, and TDMA.

This way of modeling scheduling dependencies is one of the strengths of the RTC approach. For instance, hierarchical resource sharing strategies can be elegantly modeled using *server components* [125]. However, this advantage also comes with restrictions. For example, some scheduling characteristics are not inherently supported by the RTC approach, and might require considerable modeling effort and additional operations. Examples are operating system overhead, cooperative task behavior, and event correlations.

According to the compositional performance analysis approach, local component analyses based on the RTC approach can be composed with high flexibility and scalability. This is done by building so-called *schedul-*

*ing networks*. Tasks are connected according to the global application structure by event streams described using arrival curves. System level performance analysis can then be performed according to the compositional system level analysis loop described in Section 5.1.2.

### 5.3.4 Computational efficiency

For arbitrarily complex arrival and service curves the determination of worst-case response times and worst-case backlogs according to the RTC formulas visualized in Figure 5.3 is computationally expensive. The reason is that finding the maximum distances (horizontally and vertically) between both curves is not a trivial task, and that, therefore, many alternatives (all jump discontinuities) need to be checked.

However, for computational efficiency, arrival and service curves can be represented by finite aperiodic and periodic parts, see [123, 126]. The former describes the short term bursty behavior, whereas the latter describes the long term periodic behavior. Figure 5.4 shows an upper arrival curve with an aperiodic part of length $\Delta_0$, and a periodic part with period $k$ that is repeated infinitely often. Note that this specific representation of arrival and service curves significantly decreases the computational complexity of the RTC formalisms for their implementation in the MPA toolbox [126].

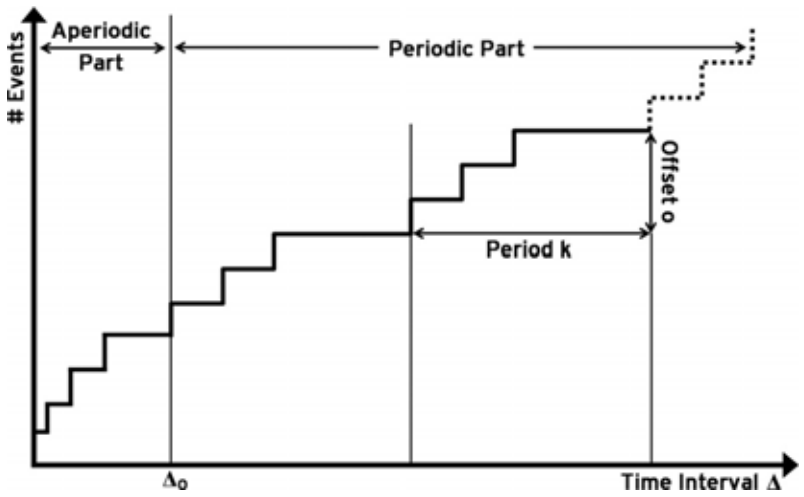



Fig. 5.4: Arrival curve with aperiodic and periodic part

## 5.4 Coupling SymTA/S and MPA

In this section, the necessary operations to integrate SymTA/S and MPA into a combined performance evaluation approach are described. Figure 5.5 visualizes the concept that is utilized to couple both methods. On system level *SymTA/S components* and *MPA components* are

distinguished. The former are analyzed by SymTA/S, whereas the latter are analyzed by MPA. Note that each component may consist of several functionally dependent tasks. However, each component is assumed to represent a single computational or communication resource.

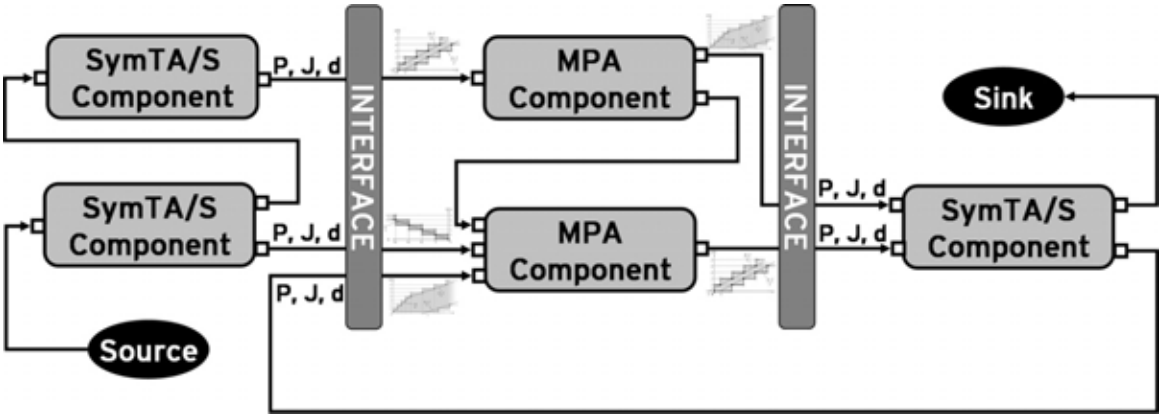



Fig. 5.5: Coupling of SymTA/S and MPA analyzed system components

As previously mentioned, SymTA/S uses standard event models to couple local component analyses according to the compositional performance analysis approach, whereas MPA uses arrival curves for this purpose. Obviously, for seamlessly analyzing SymTA/S and MPA components within the compositional analysis loop, conversion algorithm between both stream representations are needed. Additionally, starting point generation rules need to be defined for the combined approach to resolve cyclic component interdependencies. These topics are discussed in the following sections.

### 5.4.1 Event model conversion

In this section, it is described how MPA arrival curves can be obtained from SymTA/S standard event models and vice versa.

The translation from SymTA/S standard event models to MPA arrival curves is lossless and straightforward, since standard event models can directly be mapped to arrival curves. For example, a periodic event model with jitter can be represented by staircase functions with step width equal to the period $P$, and a gap between the upper and lower arrival curve that is equal to two times the jitter $J$. For a graphical representation and other examples see Figure 5.2.

The opposite direction is more involved. In the following a general method is described for deriving standard event models (i.e. its parameters $P$, $J$, and $d$) from arbitrary arrival curves $\alpha^u$ and $\alpha^l$ with $o > 0$, i.e. a non-zero average rate. For arrival curves with $o = 0$ the approximation with standard event models does not make sense, and purely

bursty approximations are more appropriate. Note that the proposed translation is not lossless but provides a safe approximation.

**Step 1: Period.** The period $P$ is calculated to fit the periodic part of the upper MPA arrival curve $\alpha^u$. This can simply be achieved by dividing the duration $k$ of the periodic part by the number of events that occur during one period (called offset in Figure 5.4).

$$P = \frac{k}{\alpha^u(\Delta_0 + k) - \alpha^u(\Delta_0)}$$

Figure 5.6 visualizes this first step of the event model conversion algorithm for example lower and upper arrival curves.

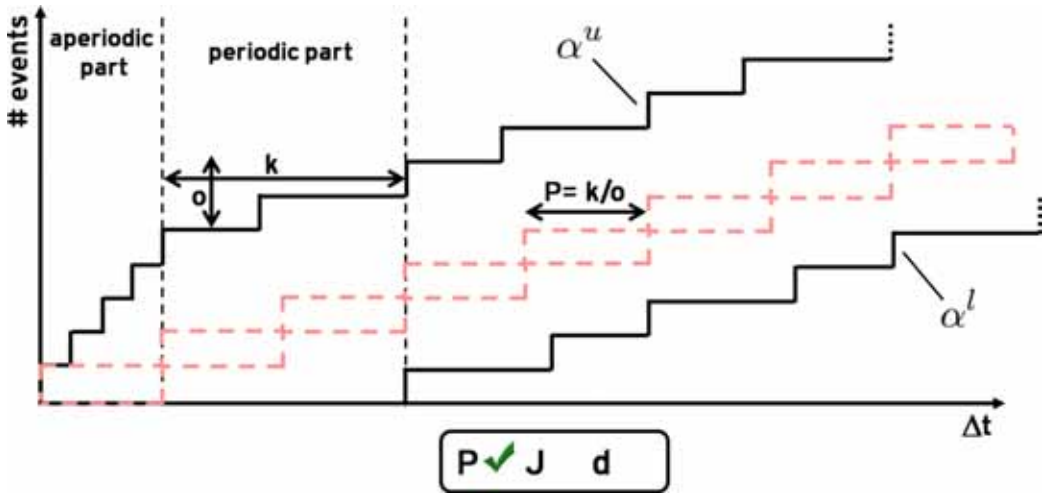



Fig. 5.6: First step of the event model conversion algorithm: determination of the period $P$

**Step 2: Jitter.** The determination of the jitter $J$ is the most complicated part of the approximation algorithm. The already calculated period $P$ defines a strictly periodic standard event model that matches the long term periodic behavior of the input arrival curves $\alpha^u$ and $\alpha^l$. However, jitter must be added to this periodic stream for it to completely cover the input arrival curves, and thus to represent a conservative approximation. Two different cases must be considered.

First, the periodic event stream with the period $P$ is shifted to the left, until the upper input arrival curve $\alpha^u$ is completely covered.

$$J_1 = \min\{J_0 : \lceil \frac{\Delta + J_0}{P} \rceil \geq \alpha^u(\Delta),\ \forall \Delta \in [0, \Delta_0 + k]\}$$

Secondly, the periodic event stream with the period $P$ is shifted to the right, until the lower input arrival curve $\alpha^l$ is completely covered.

$$J_2 = \min\{J_0 : \max\left(0, \lfloor \frac{\Delta - J_0}{P} \rfloor\right) \leq \alpha^l(\Delta),\ \forall \Delta \in [0, \Delta_0 + k]\}$$

Figure 5.7 visualizes the two candidate jitter $J_1$ and $J_2$ for the sought-after standard event model.

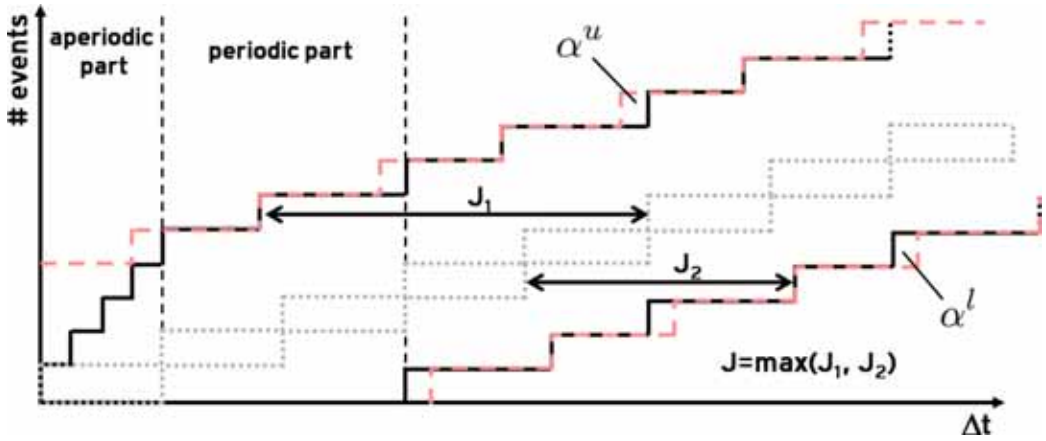



Fig. 5.7: Second step of the event model conversion algorithm: determination of the two candidates jitters $J_1$ and $J_2$

Note that for coverage tests only the jump discontinuities of the corresponding input arrival curve between 0 and $\Delta_0 + k$ need to be checked. In both cases the shifting of the periodic stream is achieved by successively adding jitter. The larger jitter can safely be used as parameter for the approximating standard event model.

$$J = \max\{J_1, J_2\}$$

Figure 5.8 visualizes the approximation of the given arrival curves after the second step of the conversion algorithm.

**Step 3: Minimum distance.** In case that the jitter $J$ is larger or equal than the period $P$, the minimum distance parameter $d$ can be determined to refine the so far calculated approximating standard event model. Note that this refinement is not strictly necessary, since the given arrival curves are already conservatively approximated after the first two steps of the conversion algorithm. However, it reduces the conservatism of the approximation by relaxing the initial burst of the calculated standard event model.

Since the minimum distance parameter $d$ only affects the maximum number of events, it is obvious that $d$ can be chosen corresponding to the steepest slope of the upper input arrival curve $\alpha^u$. Consequently, $d$

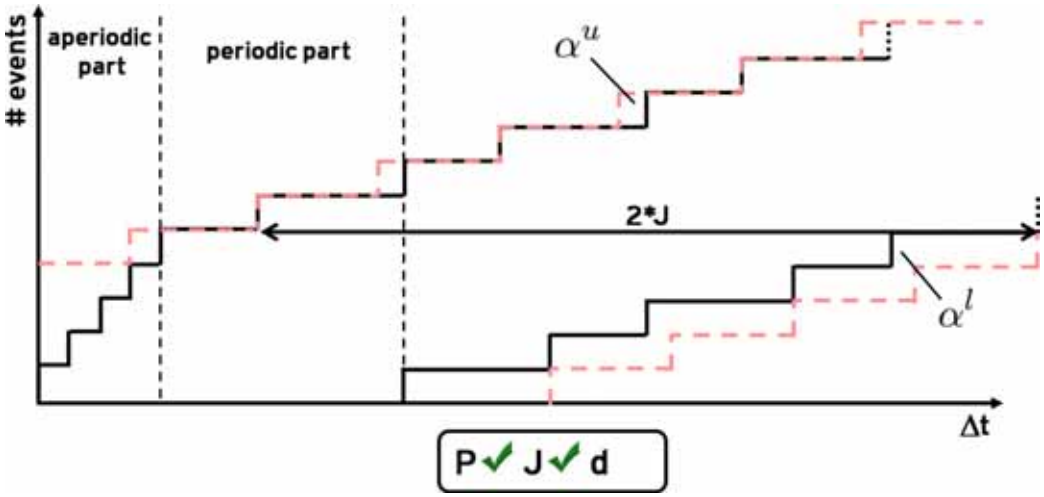



Fig. 5.8: Second step of the event model conversion algorithm: approximation of the arrival curves using period $P$ and jitter $J$

corresponds to the minimum distance that can be observed between two arbitrary events in $\alpha^u$.

$$d = \min\{d_0 : \alpha^u(d_0) = 2\}$$

Note that from an algorithmic point of view it is not necessary to check the whole arrival curve $\alpha^u$ to determine $d$. Since arrival curves are subadditive functions, the minimum distance between two events can be observed at the beginning of the aperiodic part of $\alpha^u$.

Figure 5.9 visualizes the final approximation of the given arrival curves by a standard event model.

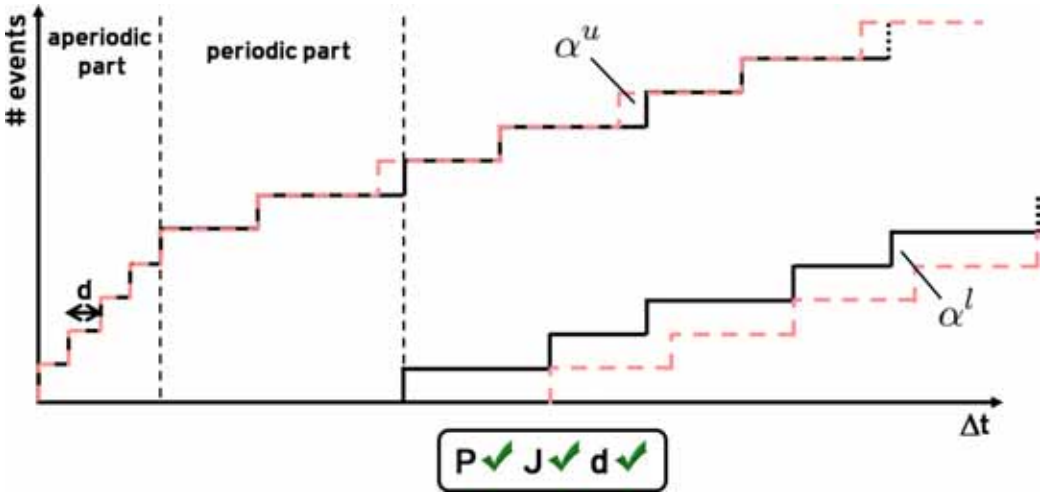



Fig. 5.9: Third step of the event model conversion algorithm: refinement of the approximation using the minimum distance parameter $d$

### 5.4.2 Starting point generation

As mentioned in Section 5.1.2, a so-called starting point needs to be generated under certain circumstances to successfully apply compositional system level performance analysis. An analysis starting point is needed, for instance, if several components of the analyzed system are cyclically interdependent, i.e. one component cannot be analyzed without the analysis results of another component, and vice versa. Concrete starting point generation strategies depend on the considered components.

The starting point generation explained in Section 5.1.3 is separately applicable to both presented compositional analysis approaches. However, since SymTA/S and MPA internally use different event stream representations, it has to be explained how the starting point generation works between both analysis domains.

Without loss of generality, it is assumed that components analyzed by MPA are integrated into the starting point generation in the SymTA/S domain. One possible simple starting point generation strategy, that allows to consider components analyzed by MPA as black-boxes, is visualized in Figure 5.10.

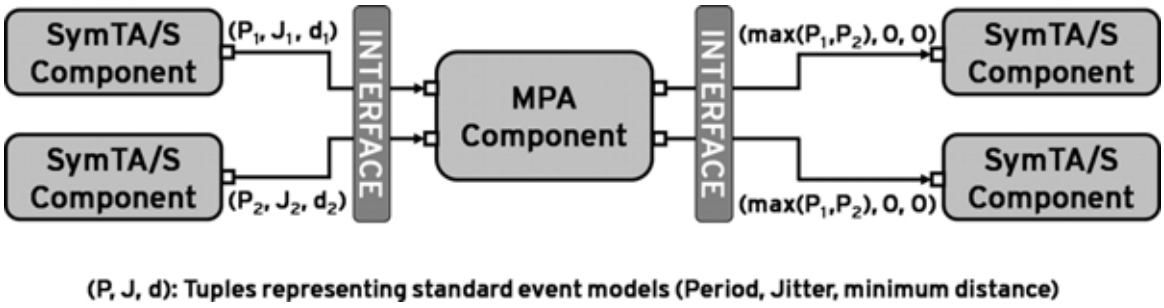



Fig. 5.10: Starting point generation for combined analysis approach. Standard event models used by SymTA/S are propagated over a component analyzed by MPA.

The proposed strategy consists in propagating the largest period $P$ occurring at the inputs of the MPA component to all its outputs, and to assume an initial jitter $J$ and an initial minimum distance $d$ of zero. This black-box approach leads to a safe starting point since the application model utilized by MPA cannot lead to period increases[1], and thus the load induced in any given time interval $\Delta t$ by the standard event model that is initially assumed at all outputs of the MPA component represents a conservative lower bound (compare Section 5.1.3). During system-level

[1]The only period changing application property used in MPA is the OR task activation. OR activation, however, leads to period decrease [56].

analysis, the load generated of the MPA component at all its outputs will only increase due to the effects of scheduling, or once the event streams connected to the inputs of the MPA component are propagated internally to the corresponding outputs.

Note that for components analyzed by performance analysis approaches supporting, for instance, rate transitions, such a black-box approach is not possible, and the responsibility for the starting point generation must be delegated to the components themselves.

## 5.5 Experiments

In this section the distributed example system shown in Figure 5.11 is considered to demonstrate the efficiency and benefits of the proposed combined performance analysis approach.

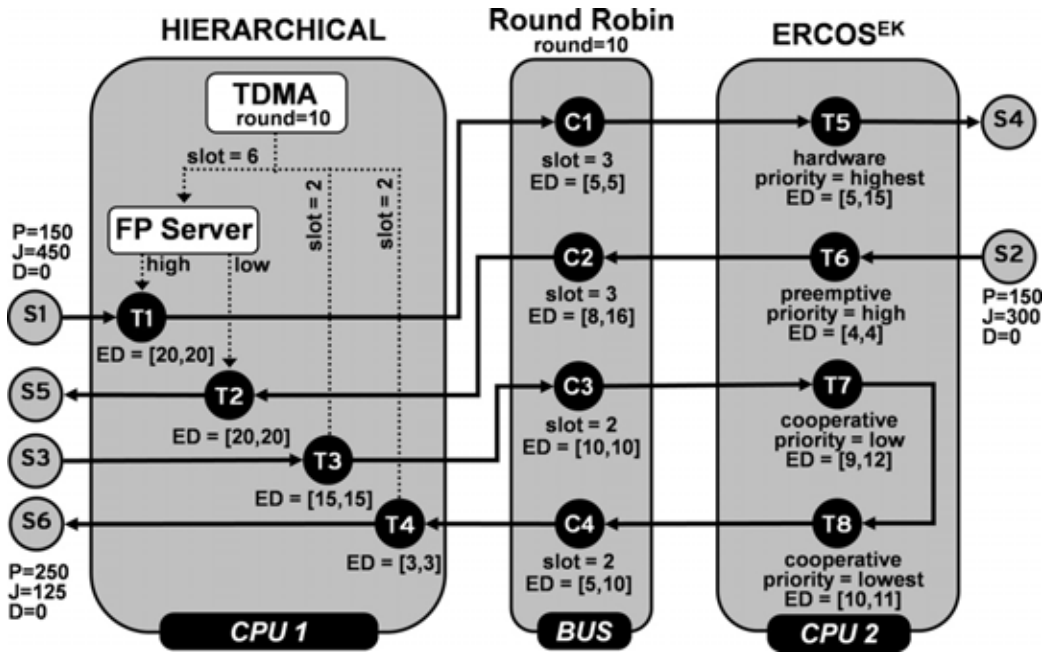



Fig. 5.11: Distributed example system

The system consists of two computational resources interconnected via a round robin arbitrated bus. $CPU1$ executes four tasks, that are scheduled according to a hierarchical scheduling policy with TDMA resource sharing at the top level. $T1$ and $T2$ are dispatched by a priority scheduled server and share 60% of the CPU's service capacity. $T3$ and $T4$ are each assigned 20% of the available CPU time. Since the resource sharing policy of $CPU1$ is hierarchical, MPA is best fitted to analyze this part of the system.

$CPU2$ executes four tasks, that are scheduled by the priority based ERCOSEK operating system. The specialty on $CPU2$ is the cooperative behavior of $T7$ and $T8$. Generally, $T7$ has a higher priority than

$T8$. However, due to the cooperative policy $T7$ can interrupt $T8$ only at specific points in time. In the given example system, $T8$ can be interrupted by $T7$ only after 2, 3, 4, 6, or 11 time units during its execution. More details about ERCOSEK can be found in [29]. The ERCOSEK scheduled $CPU2$ as well as the round robin arbitrated $BUS$ are best analyzed using the specialized analysis libraries of SymTA/S.

Note that for correct operation the system has to satisfy three end-to-end latency constraints: the maximum latency along the paths $S2 \rightarrow S5$ and $S3 \rightarrow S6$ must be less than 400 time units, whereas the maximum latency along the path $S1 \rightarrow S4$ must not exceed 200 time units.

### 5.5.1 Path latency analysis

In this section system level analysis of the given distributed example system is performed. First, the system is analyzed using SymTA/S and MPA alone. Afterwards, the obtained results are compared with the combined performance analysis approach.

**Analysis using SymTA/S alone.** Since SymTA/S does not directly support hierarchical scheduling policies, the timing behavior of the tasks running on $CPU1$ needs to be approximated to analyze the example system. Figure 5.12 shows one possible approximation using standard SymTA/S components.

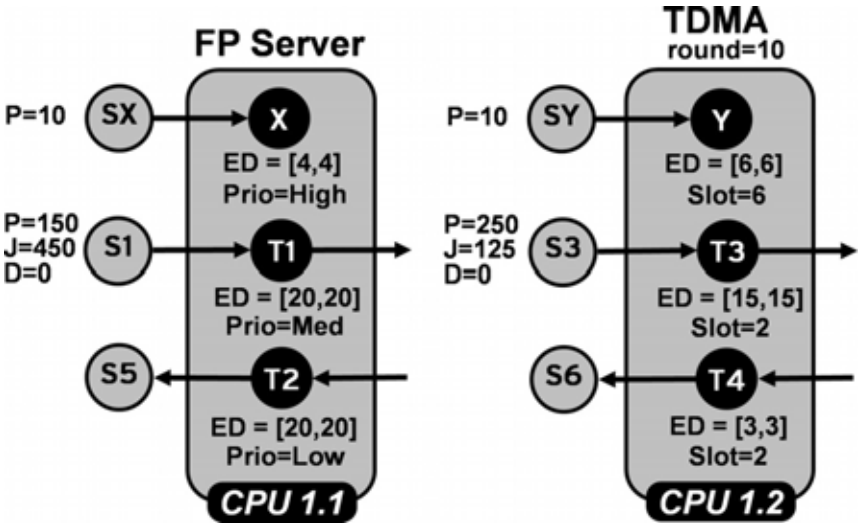



Fig. 5.12: Approximation of the hierarchical scheduling on $CPU1$ with SymTA/S modeling techniques

The FP server is modeled by the fixed priority scheduled $CPU1.1$. To account for the reduced service capacity of the FP server, that is due to the TDMA resource sharing at the top hierarchy level, the high priority task $X$ is introduced. $X$ is activated every 10 time units and executes 4

time units. Consequently, it reduces $CPU1.1$'s service capacity by the amount of resources claimed by $T3$ and $T4$. The timing behavior of the TDMA-scheduled tasks $T3$ and $T4$ is approximated by $CPU1.2$. The TDMA slot allocated to the FP server is modeled by the additional task $Y$.

Using this approximation, the following path latencies are obtained by the SymTA/S analysis:

| | $S1 \rightarrow S4$ | $S2 \rightarrow S5$ | $S3 \rightarrow S6$ |
|---|---|---|---|
| SymTA/S alone | 170 | 376 | 422 |
| Constraint | 200 | 400 | 400 |
| Status | $\surd$ | $\surd$ | $\times$ |

As can be seen the timing constraint for the path $S3 \rightarrow S6$ is violated, and hence SymTA/S evaluates the given example system as *infeasible*. Note that in the considered case the approximation of the hierarchical scheduling on $CPU1$ with SymTA/S is quite straightforward. However, in the general case, i.e. considering more complicated hierarchical resource sharing policies with several levels of hierarchy, similar approximations are far more complicated and result in large overestimations of the timing behavior.

**Analysis using MPA alone.** MPA is currently not able to analyze ERCOSEK scheduled resources. Consequently, the timing behavior of the ERCOSEK scheduled resource $CPU2$ needs to approximated to analyze the given example system with MPA alone.

It was chosen to approximate the timing behavior of $CPU2$ using a standard preemptive fixed priority scheduler. The tasks $T5$ and $T6$ are assigned the highest and second highest priority, respectively. In order to approximate the best-case and worst-case response times of the higher priority cooperative task $T7$, it is assigned a higher and a lower priority compared to $T8$, respectively. By this means, the possible partial blocking of $T7$ by the lower priority task $T8$, that is due to the cooperative policy, is replaced by full blocking. Obviously, this represents a conservative approximation.

Using this approximation, the following path latencies are obtained by the MPA analysis:

| | $S1 \rightarrow S4$ | $S2 \rightarrow S5$ | $S3 \rightarrow S6$ |
|---|---|---|---|
| MPA alone | 170 | 430 | 412 |
| Constraint | 200 | 400 | 400 |
| Status | $\surd$ | $\times$ | $\times$ |

As can be seen the timing constraints for the paths $S2 \rightarrow S5$ and $S3 \rightarrow S6$ are violated, and hence MPA evaluates the example system as

*infeasible*. Note that this simple and not overly pessimistic approximation with MPA was only possible because of the simple ERCOSEK task configuration chosen for the illustrative example. Only the timing effects of two cooperative tasks ($T7$ and $T8$) needed to be considered. However, in general the scheduling effects on ERCOSEK resources can be far more complicated disabling a reasonable approximation by MPA. Examples are so-called "Time Tables" specifying phase offsets between periodic tasks. Furthermore, "Alarms" use dynamic time-out mechanisms, and can be issued and disabled at any point in time. Finally, scheduling-related OS routines can request a significant amount of execution time at various priority levels.

**Analysis using the combined approach.** In the last experiment the proposed mixed performance analysis approach is applied. The hierarchical scheduled $CPU1$ is analyzed by MPA, whereas the round robin arbitrated $BUS$ as well as the ERCOSEK scheduled $CPU2$ are analyzed by SymTA/S.

The combined analysis approach yields the following path latencies:

| | $S1 \rightarrow S4$ | $S2 \rightarrow S5$ | $S3 \rightarrow S6$ |
|---|---|---|---|
| MPA alone | 170 | 376 | 389 |
| Constraint | 200 | 400 | 400 |
| Status | $\surd$ | $\surd$ | $\surd$ |

Using the combined analysis approach it is discovered that the example system perfectly satisfies all imposed path latency constraints, and can, thus, be considered as *feasible*. This is remarkable since both, SymTA/S and MPA, rejected the given design as discussed in the first two experiments. Altogether, the worst-case latencies predicted by the combined approach are more than 10% tighter than the results achieved by the single methods for some end-to-end paths. Considering the small size and the simplicity of the given example system this represents a remarkable analysis accuracy improvement.

# Chapter 6

# CONCLUSIONS

Embedded systems can be found everywhere and are going to be even more pervasive in the near future. In order to meet growing productivity demands and cost pressure, embedded systems need to be designed very efficiently. Obviously, due to the increasing complexity and distributed nature of modern systems this is by far no easy task. One key factor to increase the development process efficiency of complex distributed embedded systems is reuse. Concepts like the platform-based design style and standardization efforts on the software level allow to conceive whole product families and variants based on the same set of reusable components and sub-systems. However, while reuse helps to increase design efficiency at the functional level, it does not solve another key embedded systems challenge that arises with system complexity and sub-system integration: the control of non-functional properties, such as timing, power consumption, or dependability.

This thesis introduced several techniques allowing to control and optimize non-functional system properties during the design flow and the lifetime of an embedded system.

First of all, a flexible exploration framework was introduced. The proposed framework allows performing partial explorations and provides the possibility to dynamically extend and restrict the search space without loosing previously obtained results. The main motivation to realize the high degree of flexibility was that nowadays distributed embedded systems are usually not designed by a sole manufacturer who controls all design parameters. In the automotive industry, for instance, it is common practice that the OEM delegates the development of complex system features to different independently working suppliers. In this context neither OEM nor single suppliers control all necessary model pa-

rameters, and are reluctant to share design details due to IP-protection issues. As a consequence, system optimization and exploration can be tackled best in an iterative process, where only few uncritical performance information need to be exchanged between the involved design teams that independently optimize the system parts they are responsible for. Such an approach is enabled by the proposed compositional exploration framework.

In the second part of this thesis system robustness was discussed. In the field of embedded system design, robustness is usually associated with reliability and resilience. However, in this thesis a different kind of robustness was considered: robustness to variations of system properties. Informally, a system is called robust if it can sustain system property modifications without severe consequences for system performance and integrity. It was shown that accounting for property variations early during design is key, since even small modifications in systems with complex performance dependencies can have drastic non-intuitive impact on the overall system behavior, and might lead to severe performance degradation effects. Since performance evaluation and exploration do not cover these effects, it is clear that they are insufficient to systematically control performance along the design flow and during system lifetime. Therefore, explicit robustness evaluation and optimization techniques that build on top of performance evaluation and exploration were introduced in this thesis. They enable the designer to introduce robustness at critical positions in the design, and thus help to avoid critical performance pitfalls.

In order to systematically assess and optimize system robustness, metrics for different assumptions and design scenarios were proposed. The proposed metrics are based on sensitivity analysis. Thereby, robustness metrics for system properties with and without performance dependencies were distinguished. In the case of independent properties, the value of one property does not have any influence on the admissible values of other properties. In the case of dependent properties, the modification of one property value leads to restrictions for dependent properties, i.e. their flexibility with respect to modifications decreases. Especially the robustness metrics for the dependent case are computationally expensive, since multi-dimensional sensitivity analysis is necessary to capture the interdependencies. To circumvent this complexity problem, a scalable stochastic sensitivity analysis method was developed that is capable of efficiently approximating system robustness characteristics by formulating sensitivity analysis as multi-criterion optimization problem. Using this technique, it was shown how the proposed robustness metrics

can efficiently be approximated with upper and lower robustness bounds leading to a significant exploration speed-up.

It is known that already for small systems different state-of-the-art performance analysis approaches exhibit remarkable differences in terms of accuracy. As a consequence, it can be stated that the expressiveness of the methods that were proposed in this thesis highly depends on the choice of the right abstraction, i.e. analysis engine, for the considered system. For this reason, the problem of providing accurate and expressive performance predictions for general embedded systems was tackled in the last part of this thesis. Rather than searching for a unified more complex analysis model, it was proposed to exploit the so-called *compositional performance analysis methodology* to couple modular performance analysis techniques. By this means, it was demonstrated how the advantages offered by the individual models of computation can easily be integrated into a cross-domain analysis that allows modeling and analyzing each system component with the best fitting method.

# List of Figures

# List of Tables

# Bibliography


[1] ISO 11898-1. Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication. *ISO Standard-11898, International Standards Organisation (ISO)*, November 1993.

[2] S.G. Abraham, B.R. Rau, and R. Schreiber. Fast Design Space Exploration Through Validity and Quality Filtering of Subsystem Designs. Technical Report HPL-2000-98, Hewlett-Packard Laboratories, 2000.

[3] B. Andersson, S.K. Baruah, and J. Jonsson. Static-Priority Scheduling on Multiprocessors. In *Proc. of the IEEE International Real-Time Systems Symposium (RTSS)*, London, England, December 2001.

[4] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard Real-Time Scheduling: The Deadline Monotonic Approach. In *Proc. of the 8th IEEE Workshop on Real-Time Operating Systems*, pages 133–137, 1991.

[5] AUTOSAR Automotive Open System Architecture, *http://www.autosar.org*.

[6] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

[7] I. Bate and P. Emberson. Incorporating Scenarios And Heuristics To Improve Flexibility In Real-Time Embedded Systems. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, San Jose, USA, April 2006.

[8] G. Behrmann, A. David, K. G. Larsen, J. Hakansson, P. Pettersson, W. Yi, and M. Hendriks. UPPAAL 4.0. In *Proc. of the International Conference on the Quantitative Evaluation of Systems (QEST)*, Riverside, CA, USA, September 2006.

[9] G. Berry and G. Gonthier. The Esterel Synchronous Programming Language: Design, Semantics, Implementation. *Science of Computer Programming*, 2002.

[10] C. Bierwirth, D.C. Mattfeld, and H. Kopfer. On permutation representations for scheduling problems. In *Proc. Parallel Problem Solving from Nature (PPSN)*, Berlin, Germany, September 1996.

[11] E. Bini, M. Di Natale, and G. C. Buttazzo. Sensitivity Analysis for Fixed-Priority Real-Time Systems. In *Proc. of the Euromicro Conference on Real-Time Systems (ECRTS)*, Dresden, Germany, July 2006.

[12] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. Pisa - a platform and programming language independent interface for search algorithms. In *Proc. of the Conference on Evolutionary Multi-Criterion Optimization (EMO)*, Faro, Protugal, April 2003.

[13] J.Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001.

[14] Bruno Bouyssounouse and Joseph Sifakis. *Embedded Systems Design: The ARTIST Roadmap for Research and Development*, volume 3436 of *Lecture Notes in Computer Science*. Springer, 2005.

[15] R. Bosch GmbH. CAN Specification V2.0B, *http://www.semiconductors.bosch.de/en/20/can/3-literature.asp*.

[16] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg. Volcano - A Revolution in On-Board Communications. *Volvo Technology Report*, 1:9–19, 1998.

[17] S. Chakraborty, S. Künzli, and L. Thiele. A General Framework for Analysing System Properties in Platform-Based Embedded System Designs. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Munich, Germany, 2003.

[18] S. Chakraborty, L. X. Phan, and P. S. Thiagarajan. Event Count Automata: A State-Based Model for Stream Processing Systems. In *Proc. of the IEEE International Real-Time Systems Symposium (RTSS)*, pages 87–98, Washington, DC, USA, 2005.

[19] J.M. Chateau. Flexible Platform-Based Design. *CommsDesign*, February 2001.

[20] Microsoft Corporation. The Component Object Model, *http://www.microsoft.com/com/default.mspx*.

[21] CORBA/IIOP specification V3.0.3 ,*http://www.omg.org/technology/documents/corba_spec_catalog.htm*.

[22] R.I. Davis, A. Burns, R.J. Bril, and J.J. Lukkien. Controller Area Network (CAN) Schedulability Analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[23] K. Deb. *Multi-objective optimization using evolutionary algorithms.* John Wiley, Chichester, 2001.

[24] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[25] R. P. Dick and N. K. Jha. MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Co-synthesis of Hierarchical Heterogeneous Distributed

Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):920–935, October 1998.

[26] T. Ehlers, M. Harms, J.U. Varchmin, M. Mutz, and M. Horstmann. STEP-X: Strukturierter Entwicklungsprozess für eingebettete Systeme im Automobilbereich. In *23. Tagung Elektronik im Kfz*, Stuttgart, June 2003.

[27] P. Emberson and I. Bate. Minimising Task Migration And Priority Changes In Mode Transitions. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Seatlle, WA, USA, April 2007.

[28] M. Emmerich, N. Beume, and B. Naujoks. An EMO Algorithm Using the Hypervolume Measure as Selection Criterion. In *Proc. of the Conference on Evolutionary Multi-Criterion Optimization (EMO)*, Guanajuato, Mexico, March 2005.

[29] ETAS GmbH. ERCOSEK V4.1 user's guide, *http://en.etasgroup.com/*.

[30] R. Ernst. Codesign of Embedded Systems: Status and Trends. *IEEE Design & Test of Computers*, April 1998.

[31] J. Filipiak. *Real Time Network Management*. North-Holland, 1991.

[32] M. Fleischer. The Measure of Pareto Optima: Applications to Multi-objective Metaheuristics. *Lecture Notes in Computer Science*, 2632:519–533, 2003.

[33] FlexRay consortium. FlexRay specification V2.1, *http://www.flexray.com*.

[34] B. Florentz and P. M. Hofmann. Ein Model Driven Architecture Ansatz für die Softwareentwicklung im Automotive-Bereich. In *24. Tagung Elektronik im Kfz*, Haus der Technik, Essen, June 2004.

[35] C. M. Fonseca, L. Paquete, and M. Lopez-Ibanez. An Improved Dimension-Sweep Algorithm for the Hypervolume Indicator. In *Proc. of the IEEE Congress on Evolutionary Computation (CEC)*, Vancouver, BC Canada, July 2006.

[36] D. Gajski, F. Vahid, S. Narayan, and J. Gong. System-level Exploration with SpecSyn. In *Proc. of the ACM/IEEE Design Automation Conference (DAC)*, San Francico, CA, USA, June 1998.

[37] T. Givargis, F. Vahid, and J. Henkel. System-level Exploration for Pareto-optimal Configurations in Parameterized System-on-a-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(4):416–422, 2002.

[38] O. Gonzalez, H. Shrikumar, J.A. Stankovic, and K. Ramamritham. Adaptive Fault Tolerance and Graceful Degradation under Dynamic Hard Real-Time Scheduling. In *Proc. of the IEEE International Real-Time Systems Symposium (RTSS)*, San Francisco, CA, USA, December 1997.

[39] D. Gu, F. Drews, and L. Welch. Robust Task Allocation for Dynamic Distributed Real-Time Systems subject to Multiple Environmental Parameters. In *Proc. of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, Columbus, Ohio, USA, June 2005.

[40] J. C. Palencia Gutiérrez, J. J. Gutiérrez García, and M. González Harbour. Best-Case Analysis for Improving the Worst-Case Schedulability Test for Distributed Hard Real-Time Systems. In *Proc. of 10th Euromicro Workshop on Real-Time Systems*, June 1998.

[41] W. Haid and L. Thiele. Complex Task Activation Schemes in System Level Performance Analysis. In *Proc. of the IEEE/ACM International Conference on HW/SW Codesign and System Synthesis (CODES-ISSS)*, Salzburg, Austria, September 2007.

[42] A. Hamann, R. Racu, and R. Ernst. A Formal Approach to Robustness Maximization of Complex Heterogeneous Embedded Systems. In *Proc. of the IEEE/ACM International Conference on HW/SW Codesign and System Synthesis (CODES-ISSS)*, Seoul, South Korea, October 2006.

[43] M. González Harbour, J. J. Gutiérrez García, J. C. Palencia Gutiérrez, and J. M. Drake Moyano. MAST: Modeling and Analysis Suite for Real-Time Applications. In *Proc. Euromicro Conference on Real-Time Systems (ECRTS)*, page 125, Washington, DC, USA, 2001.

[44] David Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[45] G.J. Hekstra, G.D. La Hei, P. Bingley, and F.W. Sijstermans. TriMedia CPU64 Design Space Exploration. In *Proc. of the International Conference on Computer Design (ICCD)*, Austin, Texas, USA, October 1999.

[46] M. Hendriks and M. Verhoef. Timed Automata Based Analysis of Embedded System Architectures. In *Proc. of the Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, Island of Rhodes, Greece, April 2006.

[47] R. Henia and R. Ernst. Context-Aware Scheduling Analysis of Distributed Systems with Tree-shaped Task-Dependencies. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Paris, France, March 2005.

[48] R. Henia and R. Ernst. Improved Offset-Analysis Using Multiple Timing-Reference. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Munich, Germany, March 2006.

[49] R. Henia and R. Ernst. Scenario Aware Analysis For Complex Event Models And Distributed Systems. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, Tucson, Arizona, USA, December 2007.

[50] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System Level Performance Analysis - The SymTA/S Approach. *IEE Proceedings Computers and Digital Techniques*, 152(2):148–166, March 2005.

[51] R. Henia, R. Racu, and R. Ernst. Improved Output Jitter Calculation for Compositional Performance Analysis of Distributed Systems. In *Proc. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, Long Beach, CA, USA, March 2007.

[52] T. Henzinger and S. Matic. An Interface Algebra for Real-Time Components. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2006.

[53] V. Izosimov, P. Pop, P. Eles, and Z. Peng. Design Optimization of Time- and Cost-Constrained Fault-Tolerant Distributed Embedded Systems. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Munich, Germany, March 2005.

[54] M. Jersak. *Compositional Performance Analysis for Complex Embedded Applications*. PhD thesis, Technical University of Braunschweig, 2004.

[55] M. Jersak, R. Henia, and R. Ernst. Context-Aware Performance Analysis for Efficient Embedded System Design. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Paris, France, March 2004.

[56] M. Jersak, K. Richter, and R. Ernst. Performance analysis for complex embedded applications. *International Journal of Embedded Systems, Special Issue on Codesign for SoC*, 1(1/2):33–49, 2005.

[57] D. Juedes, F. Drews, L. Welch, and D. Fleeman. Heuristic resource allocation algorithms for maximizing allowable workload in dynamic, distributed real-time systems. In *Proc. of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Santa Fe, New Mexico, USA, April 2004.

[58] H. Kargupta, K. Deb, and D. Goldberg. Ordering Genetic Algorithms and Deception. In *Proc. of Parallel Problems Solving from Nature (PPSN)*, Brussels, Belgium, September 1992.

[59] D.I. Katcher, H. Arakawa, and J.K. Strosnider. Engineering and Analysis of Fixed Priority Schedulers. *Software Engineering*, 19(9):920–934, 1993.

[60] H. Kopetz. *Real-time Systems-Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[61] S. Künzli, A. Hamann, R. Ernst, and L. Thiele. Combined Approach to System Level Performance Analysis of Embedded Systems. In *Proc. of the IEEE/ACM International Conference on HW/SW Codesign and System Synthesis (CODES-ISSS)*, September 2007.

[62] S. Künzli, F. Poletti, L. Benini, and L. Thiele. Combining Simulation and Formal Methods for System-Level Performance Analysis. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Munich, Germany, 2006.

[63] T.-W. Kuo and A.K. Mok. Load Adjustment in Adaptive Real-Time Systems. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 160–171, 1991.

[64] M. Laumanns, L. Thiele, E. Zitzler, E. Welzl, and K. Deb. Running time analysis of multi-objective evolutionary algorithms on a simple discrete optimization problem. In *Proc. Parallel Problem Solving From Nature (PPSN)*, Granada, Spain, September 2002.

[65] S. Edwards L. Lavagno, E.A. Lee, and A. Sangiovanni-Vincentelli. Design of Embedded Systems: Formal Models, Validation, and Synthesis. *Proceedings of the IEEE*, 85(3):366–390, 1997.

[66] E. A. Lee and Th. M. Parks. Dataflow Process Networks. *Proceedings of the IEEE*, May 1995.

[67] P.A. Lee, T. Anderson, J.C. Laprie, A. Avizienis, and H. Kopetz. *Fault Tolerance: Principles and Practice*. Springer Verlag, Secaucus, NJ, USA, 1990.

[68] J. Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, 1990.

[69] J. Lehoczky, L. Sha, and Y. Ding. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 166–171, IEEE Computer Society Press, 1989.

[70] J. Lemieux. *Programming in the OSEK/VDX Environment*. CMP Books, 2001.

[71] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[72] C. Lu, J.A. Stankovic, S.H. Son, and G. Tao. Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms. *Real-Time Systems Journal*, 23(1-2):85–126, 2002.

[73] L. Lundberg. Analyzing Fixed-Priority Global Multiprocessor Scheduling. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, San Jose, CA, USA, September 2002.

[74] A. Maxiaguine, S. Künzli, S. Chakraborty, and L. Thiele. Rate Analysis for Streaming Applications with On-chip Buffer Constraints. In *Proc. of the IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC)*, Yokohama, Japan, January 2004.

[75] Industrieanlagen-Betriebsgesellschaft mbH (IABG). *V-Modell '97 Spezifikation: Entwicklungsstandard für IT-Systeme des Bundes - Vorgehensmodell*. www.v-modell.iabg.de, 2002.

[76] Mentor Graphics. Vehicle Network Design. `http://www.mentor.com/products/vnd`.

[77] A. Nandi and R. Marculescu. System-level Power/Performance Analysis for Embedded Systems Design. In *Proc. 38th conference on Design automation (DAC)*, pages 599–604, New York, NY, USA, 2001.

[78] NXP Nexperia platform, *http://www.nxp.com*.

[79] C. Norström, A. Wall, and W. Yi. Timed Automata as Task Models for Event-Driven Systems. In *Proc. 6th International Conference on Real-Time*

*Computing Systems and Applications (RTCSA)*, page 182, Washington, DC, USA, 1999.

[80] D.-I. Oh and T. P. Baker. Utilization Bounds for N-Processor Rate Monotone Scheduling with Static Processor Assignment. *Real-Time Systems*, 15(2):183–192, 1998.

[81] J. C. Palencia and M. G. Harbour. Schedulablilty analysis for tasks with static and dynamic offsets. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, Madrid, Spain, December 1998.

[82] M. Palesi and T. Givargis. Multi-objective Design Space Exploration Using Genetic Algorithms. In *Proc. of the International Symposium on Hardware/Software Codesign (CODES)*, Estes Park, Colorado, May 2002.

[83] S. Perathoner, E. Wandeler, L. Thiele, A. Hamann, S. Schliecker, R. Henia, R. Racu, R. Ernst, and M. González Harbour. Influence of Different System Abstractions on the Performance Analysis of Distributed Real-Time Systems. In *Proc. ACM Conference on Embedded Systems Software (EMSOFT)*, Salzburg, Austria, October 2007.

[84] T. Pop, P. Eles, and Z. Peng. Holistic Scheduling and Analysis of Mixed Time/Event-Triggered Distributed Embedded Systems. In *Proc. of the International Symposium on Hardware/Software Codesign (CODES)*, pages 187–192, New York, NY, USA, 2002.

[85] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing Analysis of the FlexRay Communication Protocol. In *Proc. Euromicro Conference on Real-Time Systems (ECRTS)*, Dresden, Germany, July 2006.

[86] K. Poulsen, P. Pop, V. Izosimov, and P. Eles. Scheduling and Voltage Scaling for Energy/Reliability Trade-offs in Fault-Tolerant Time-Triggered Embedded Systems. In *Proc. of the IEEE/ACM International Conference on HW/SW Codesign and System Synthesis (CODES-ISSS)*, Salzburg, Austria, October 2007.

[87] SymTA/S Project. http://www.symta.org. Institute of Computer and Communication Network Engineering, Technical University of Braunschweig, Germany.

[88] S. Punnekkat, R. Davis, and A. Burns. Sensitivity Analysis of Real-Time Task Sets. *ASIAN*, pages 72–82, 1997.

[89] R. Racu. *Performance Characterization and Sensitivity Analysis of Real-Time Embedded Systems.* PhD thesis, Technical University of Braunschweig, 2008.

[90] R. Racu and R. Ernst. Scheduling Anomaly Detection and Optimization for Distributed Systems with Preemptive Task-Sets. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, San Jose, USA, April 2006.

[91] R. Racu, A. Hamann, and R. Ernst. A Formal Approach to Multi-Dimensional Sensitivity Analysis of Embedded Real-Time Systems. In *Proc. of the Euromicro Conference on Real-Time Systems (ECRTS)*, Dresden, Germany, July 2006.

[92] R. Racu, A. Hamann, and R. Ernst. Automotive System Optimization using Sensitivity Analysis. In *Proc. of the International Embedded Systems Symposium (IESS)*, Irvine, CA, USA, May 2007.

[93] R. Racu, M. Jersak, and R. Ernst. Applying Sensitivity Analysis in Real-Time Distributed Systems. In *Proc. of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, San Francisco, California, March 2005.

[94] R. Racu, L. Li, R. Henia, A. Hamann, and R. Ernst. Exact Response Time Analysis of Tasks Scheduled under Preemptive Round Robin. In *Proc. of the IEEE/ACM International Conference on HW/SW Codesign and System Synthesis (CODES-ISSS)*, Salzburg, Austria, 2007.

[95] O. Redell. Analysis of Tree-Shaped Transactions in Distributed Real Time Systems. In *Proc. of the Euromicro Conference on Real-Time Systems (ECRTS)*, Catania, Italy, June 2004.

[96] K. Richter. *Compositional Performance Analysis using Standard Event Models*. PhD thesis, Technical University of Braunschweig, 2004.

[97] K. Richter and R. Ernst. Real-Time Analysis as a Quality Feature: Automotive Use-Cases and Applications. In *Proc. of the Embedded World Conference*, Nuremberg, Germany, Feb 2006.

[98] K. Richter, M. Jersak, and R. Ernst. How OEMs and Suppliers can face the Network Integration Challenges. In *Automotive Designers Forum. IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Munich, Germany, Mar 2006.

[99] K. Richter, R. Racu, and R. Ernst. Scheduling Analysis Integration for Heterogeneous Multiprocessor SoC. In *Proc. of the 24th IEEE Real-Time Systems Symposium (RTSS)*, Cancun, Mexico, December 2003.

[100] J. Rox and R. Ernst. Modeling Event Stream Hierarchies with Hierarchical Event Models. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Munich, Germany, March 2006.

[101] A. Sangiovanni-Vincentelli. Defining Platform-based Design. *EEDesign*, March 2002.

[102] S. Schliecker, M. Ivers, and R. Ernst. Integrated Analysis of Communicating Tasks in MPSoCs. In *Proc. of the IEEE/ACM International Conference on HW/SW Codesign and System Synthesis (CODES-ISSS)*, Seoul, Korea, October 2006.

[103] S. Schliecker, S. Stein, and R. Ernst. Performance Analysis of Complex Systems by Integration of Dataflow Graphs and Compositional Performance Analysis.

In *Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Nice, France, 2007.

[104] J. Schlosser. Requirements for Automotive System Engineering Tools. In *Proc. of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Los Alamitos, CA, USA, 2002.

[105] L. Sha, R. Rajkumar, and J. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[106] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode Change Protocols for Priority-Driven Preemptive Scheduling. *Real-Time Systems*, 1(3):243–264, 1989.

[107] W.T. Shiue and C. Chakrabarti. Memory exploration for Low Power, Embedded Systems. In *Proc. of the ACM/IEEE Design Automation Conference (DAC)*, New Orleans, LA, USA, June 1999.

[108] G. Snider. Automated Design Space Exploration for Embedded Computer Systems. Technical Report HPL-2001-220, Hewlett-Packard Laboratories, 2001.

[109] G. Syswerda. Schedule Optimization Using Genetic Algorithms. In *Handbook of Genetic Algorithms*, New York, 1990. Van Nostrand Reinhold.

[110] Infineon Technologies AG. TC1796 Users Manual, V1.0, 32 Bit Single-Chip Microcontroller: System Units, June 2005.

[111] L. Thiele, S. Chakraborty, M. Gries, A. Maxiaguine, and J. Greutert. Embedded Software in Network Processors - Models and Algorithms. In *Proc. of the ACM Workshop on Embedded Software (EMSOFT)*, Lake Tahoe (CA), USA, October 2001.

[112] L. Thiele, S. Chakraborty, and M. Naedele. Real-Time Calculus for Scheduling Hard Real-Time Systems. In *Proc. International Symposium on Circuits and Systems*, pages 101–104, Geneva, Switzerland, March 2000.

[113] L. Thiele, E. Wandeler, and S. Chakraborty. A Stream-Oriented Component Model for Performance Analysis of Multiprocessor DSPs. *IEEE Signal Processing Magazine*, 22(3):38–46, May 2005.

[114] L. Thiele, E. Wandeler, and N. Stoimenov. Real-Time Interfaces for Composing Real-Time Systems. In *Proc. ACM International Conference On Embedded Software (EMSOFT)*, Seoul, Korea, 2006.

[115] K. Tindell and J. Clark. Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. *Microprocessing & Microprogramming*, 50(2-3):117–134, April 1994.

[116] K. Tindell, H. Kopetz, F. Wolf, and R. Ernst. Safe automotive software development. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Munich, Germany, March 2003.

[117] K. W. Tindell. Adding Time-Offsets to Schedulability Analysis. Technical Report YCS 221, University of York, 1994.

[118] K.W. Tindell, A. Burns, and A. J. Wellings. Calculating controller area network (can) message response times. *Control Engineering Practice*, 3(8):1163–1169, Aug 1995.

[119] K.W. Tindell, H. Hansson, and A.J. Wellings. Analysing real-time communications: Controller Area Network (CAN). In *Proc. of the IEEE International Real-Time Systems Symposium (RTSS)*, San Juan, Puerto Rico, Dec 1994.

[120] TTA group. TTP specification V1.1, *http://www.ttagroup.org/technology/specification.htm*.

[121] M. Verhoef, E. Wandeler, L. Thiele, and P. Lieverse. System Architecture Evaluation Using Modular Performance Analysis - A Case Study. In *Proc. of the 1st IEEE/ACM International Symposium on Leveraging Applications of Formal Methods (ISOLA)*, Pafos, Cyprus, Oct 2004.

[122] S. Vestal. Fixed-Priority Sensitivity Analysis for Linear Compute Time Models. *IEEE Transactions on Software Engineering*, 20(4), april 1994.

[123] E. Wandeler. *Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems*. PhD thesis, Swiss Federal Institute of Technology, September 2006.

[124] E. Wandeler, A. Maxiaguine, and L. Thiele. Performance Analysis of Greedy Shapers in Real-Time Systems. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Munich, Germany, 2006.

[125] E. Wandeler and L. Thiele. Interface-Based Design of Real-Time Systems with Hierarchical Scheduling. In *Proc. of 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 243–252, San Jose, USA, April 2006.

[126] E. Wandeler and L. Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.

[127] E. Wandeler and L. Thiele. Workload Correlations in Multi Processor Hard Real-Time Systems. *Journal of Computer and System Sciences (JCSS)*, 73(2):207–224, March 2007.

[128] L. While, P. Hingston, L. Barone, and S. Huband. A Faster Algorithm for Calculating Hypervolume. *IEEE Transactions on Evolutionary Computation*, 10(1):29–38, February 2006.

[129] D. L. Whitley and N. Yoo. Modeling Simple Genetic Algorithms for Permutation Problems. In *Foundations of Genetic Algorithms III*, pages 163–184, San Francisco, CA, 1995. Morgan Kaufmann.

[130] T. Yen and W. Wolf. Performance Estimation for Real-Time Distributed Embedded Systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(11), November 1998.

[131] T.-Y. Yen and W. Wolf. Performance Estimation for Real-Time Distributed Embedded Systems. In *Proc. International Conference on Computer Design (ICCD)*, pages 64–71, Washington, DC, USA, 1995.

[132] Q. Zhuge, Z. Shao, B. Xiao, and H.M.S. Edwin. Design Space Minimization with Timing and Code Size Optimization for Embedded DSP. In *Proc. of the IEEE/ACM International Conference on HW/SW Codesign and System Synthesis (CODES-ISSS)*, Newport Beach, CA, USA, October 2003.

[133] D. Ziegenbein. *A Compositional Approach to Embedded System Design*. PhD thesis, Technical University of Braunschweig, 2003.

[134] E. Zitzler. Hypervolume metric calculation:. *ftp://ftp.tik.ee.ethz.ch/pub/people/zitzler/hypervol.c*, 2001.

[135] E. Zitzler and S. Künzli. Indicator-Based Selection in Multiobjective Search. In *Proc. 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*, volume 3242 of *Lecture Notes in Computer Science*, Heidelberg, Germany, September 2004. Springer.

[136] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Proc. Evolutionary Methods for Design, Optimisation, and Control*, pages 95–100, Barcelona, Spain, 2002.

[137] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[138] E. Zitzler, L. Thiele, M. Laumanns, C. M. Foneseca, and V. Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.