Optimisation of the Allocation of Functions in Vehicle Networks

Der Technischen Fakultät der
Universität Erlangen-Nürnberg

zur Erlangung des Grades

D O K T O R – I N G E N I E U R

vorgelegt von

Dipl.-Ing. (FH) Bernd Hardung

Erlangen – 2006

Als Dissertation genehmigt von
der Technischen Fakultät der
Universität Erlangen-Nürnberg

Tag der Einreichung:	30. Juni 2006
Tag der Promotion:	03. November 2006
Dekan:	Prof. Dr. Alfred Leipertz
Berichterstatter:	PD Dr. Gabriella Kókai
	Prof. Dr. Wolfgang Schröder-Preikschat

Optimisation of the Allocation of Functions in Vehicle Networks

Abstract

Modern vehicles have a complex distributed system of electronic control units (ECUs), networked by several bus systems. There are up to 80 network nodes in a single car. ECUs are becoming more complex, integrating an increasing number of functions and software logic, and are connected to several hardware components, like sensors and actuators.

In the early development phase, there is quite a big degree of freedom in the allocation of these components to ECUs. Multiple objectives, like costs, quiescent current or busload, have to be considered for optimising the allocation, not to mention the many constraints that have to be fulfilled, like memory, pin or bandwidth availability.

Human experts usually have problems to effectively optimise multiple objectives simultaneously. In order to recognise shortcomings in earlier phases of the development, it is necessary to support the decision process.

In this work, the most important criteria are examined. Furthermore, a framework is introduced supporting a multi-objective optimisation of the allocation. A database model is introduced, allowing to store all relevant information. Several heuristic optimisation algorithms are compared and improved for this specific problem. Finally, an application example is shown.

Optimierung der Zuordnung von Funktionen in Fahrzeugnetzwerken

Kurzfassung

Moderne Fahrzeuge haben ein komplexes, verteiltes System von Steuergeräten, die mittels verschiedener Bussysteme miteinander vernetzt sind. Bis zu 80 Netzknoten befinden sich in einem einzelnen Fahrzeug. Die Steuergeräte werden zunehmend komplexer und integrieren eine steigende Anzahl von Funktionen und Software-Logik. Die Steuergeräte sind zudem verbunden mit verschiedenen Hardware-Komponenten, wie Sensoren und Aktuatoren.

In frühen Entwicklungsphasen hat die Zuordnung dieser Komponenten zu Steuergeräten noch viele Freiheitsgrade. Allerdings müssen viele Kriterien, wie Kosten, Ruhestrom oder Buslast bei der Festlegung berücksichtigt werden. Zudem müssen Nebenbedingungen wie Speicherplatz, Pins oder Bandbreite in Betracht gezogen werden.

Menschen haben normalerweise Probleme, mehrere Kriterien gleichzeitig zu optimieren. Um Unzulänglichkeiten in möglichst frühen Entwicklungsphasen zu entdecken, ist es notwendig, den Entscheidungsprozess zu unterstützen.

In dieser Arbeit werden dazu die wichtigsten Kriterien untersucht. Weiterhin wird ein Rahmenprogramm vorgestellt, welches eine Mehrkriterienoptimierung der Zuordnung ermöglicht. Ein Datenmodell wird eingeführt, mit dessen Hilfe alle relevanten Informationen gespeichert werden können. Verschiedene heuristische Optimierungsverfahren werden verglichen und für dieses spezielle Optimierungsproblem angepasst. Abschließend werden die Optimierungsverfahren an einem realitätsnahen Anwendungsbeispiel getestet.

Danksagung

An erster Stelle möchte ich mich bei PD Dr. Gabriella Kókai bedanken, die es mir ermöglicht hat, diese Dissertation anzufertigen. Sie hat mich mit ihren kritischen Reviews immer wieder motiviert, die Ergebnisse zu überdenken und mir unermüdlich mit Rat und Tat zur Seite gestanden. Weiterhin hat sie das Erstgutachten erstellt. An dieser Stelle möchte ich mich auch bei Prof. Dr. Wolfgang Schröder-Preikschat für die Erstellung des Zweitgutachtens bedanken.

Zu Dank verpflichtet bin ich der AUDI AG, wo mir die Möglichkeit geboten wurde, diese Dissertation neben meiner Industrietätigkeit anzufertigen. Mein besonderer Dank gilt den Betreuern Dr. Andreas Krüger und Jens Kötz für ihre Unterstützung sowohl in fachlicher als auch administrativer Hinsicht. Bedanken möchte ich mich an dieser Stelle auch bei Prof. Dr. Herbert Forster von der Georg-Simon-Ohm Fachhochschule Nürnberg für die Vermittlung zu Audi.

Ich möchte mich bei den Studenten Christoph Neumann, Csanad Viragos, Thomas Kollert, Stefan Federzoni, Bettina Bickel, Aron Csendes, Szilvia Papp und Manuel Förster bedanken. Sie haben mich nicht nur bei der Implementierung unterstützt, sondern mir auch viele neue Ideen und Anregungen gegeben.

Mein Dank gilt meinen Kollegen an der Universität und bei Audi für die anregenden Diskussionen und die Reviews meiner Arbeit. Namentlich erwähnen möchte ich hier Wolfgang Frieß, der wiederholt und ausdauernd meine Seiten gelesen hat sowie immer für Fachdiskussionen zur Verfügung stand.

Meine Eltern haben in großem Maß zum Gelingen dieser Arbeit beigetragen. Sie haben mir ihre Gene vererbt, mich erzogen und mich auch sonst in jeder erdenklichen Weise unterstützt.

Allen Freunden gilt mein Dank. Durch sie fand ich den notwendigen Ausgleich, so dass ich immer wieder mit neuer Kraft an die Arbeit gehen konnte.

Zum Schluß möchte ich mich bei meiner Lebensgefährtin Melanie für ihre Geduld sowie ihre Unterstützung vor allem in der letzten Phase der Arbeit bedanken.

Erlangen, 03. November 2006

Contents

Nomenclature

Abbreviations

ACO Ant Colony Optimisation, page 19

B&B Branch & Bound, page 123

BPMF Body Power Module Front, page 7

BPMR Body Power Module Rear, page 7

CAN Controller Area Network, page 5

CPU Central Processing Unit, page 9

EA Evolutionary Algorithm, page 16

ECU Electronic Control Unit, page 4

ER Entity Relationship, page 26

FK Foreign Key constraint, page 26

FLU Front Light Unit, page 76

HAL Hardware Abstraction Layer, page 5

HEUROPT HEURistic OPTimisation, page 61

HP Horse Power, page 33

HW HardWare, page 68

I/O Input/Output, page 5

JVM Java Virtual Machine, page 62

LCM Lowest Common Multiple, page 48

LIN Local Interconnect Network, page 5

MOEA Multi-Objective Evolutionary Algorithm, page 16

MOOP Multi-Objective Optimisation Problem, page 13

MOOVE Multi-Objective Optimisation of Vehicle Electronics, page 61

MOST Media Oriented System Transport, page 5

MSC Message Sequence Chart, page 64

NPGA Niched Pareto Genetic Algorithm, page 16

NSGA-II Non-dominated Sorting Genetic Algorithm II, page 16

OA Orthogonal Array, page 102

OE Objective Estimation, page 66

OEM Original Equipment Manufacturer, here: car manufacturer, page 57

PK Primary Key constraint, page 26

POSA Pattern Oriented Software Architecture, page 62

RAM Random Access Memory, page 4

RLU Rear Light Unit, page 76

ROM Read Only Memory, page 4

SA Simulated Annealing, page 124

SCP Set-Covering Problem, page 58

SI Swarm Intelligence, page 19

SMSC Switch Module Steering Column, page 7

SPEA2 Strength Pareto Evolutionary Algorithm 2, page 16

SPF Shortest Path First algorithm, page 49

SPLP Simple Plant Location Problem, page 120

SQL Simple Query Language, page 26

SW SoftWare, page 68

TSP Travelling Salesman Problem, page 19

U Unique Constraint, page 26

VCP Variant Combination Problem, page 112

VEGA Vector Evaluated Genetic Algorithm, page 14

VO Variant Optimisation, page 112

WLP Warehouse Location Problem, page 120

XML eXtensible Markup Language, page 64

Symbols

A Ampere, page 56

a allocation of a component to an ECU, page 10

$[a, b]$ closed interval from a to b, page 98

$]a, b[$ open interval from a to b, page 52

$\binom{a}{b}$ binomial coefficient: $\binom{a}{b} = \frac{a!}{b!(a-b)!}$, page 120

B Byte, page 35

$\mathfrak{C}$ virtual cost unit, page 80

c component, page 5

e electronic control unit (ECU), page 5

kW Kilo Watt, page 33

$\mathcal{P}(\mathbf{x})$ power set or set of all subsets. Example: $\mathbf{x} = \{1, 2, 3\}$, then $\mathcal{P}(\mathbf{x}) = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{3, 1\}, \{1, 2, 3\}, \{\}\}$, page 122

s solution, page 10

Symbols – Criteria and Variant Optimisation

b_{f} frame length without data, page 45

b_{oh} worst-case protocol overhead, page 45

b_{s} number of stuff bits, page 45

b_{sig} length of the signal, page 45

C_{bat} net capacity of a battery, page 57

$\mathbf{c}_e$ set of ECU-allocated components, page 113

$\mathbf{c}_{\mathrm{e},\mathfrak{v}}^{\min}$ minimal component combination, page 114

$\mathfrak{c}_c^{\mathrm{ext}}$ external component costs, page 40

$\mathfrak{c}_c^{\mathrm{int}}$ internal component costs, page 40

$\mathfrak{c}_c^{\mathrm{int,nVO}}$ not-VO-reducible component costs, page 113

$\mathfrak{c}_c^{\mathrm{int,VO}}$ VO-reducible component costs, page 113

$\mathfrak{c}_e$ average ECU costs, page 40

$\mathfrak{c}_e^{\mathrm{bas,VO}}$ basic VO ECU costs, page 115

$\mathfrak{c}_e^{\mathrm{bas}}$ basic ECU costs, page 40

$\mathfrak{c}_e^{\mathrm{hand}}$ handling costs per variant, page 116

$\mathfrak{c}_{e,\mathbf{V}}^{\mathrm{avg}}$ average ECU costs of a variant combination, page 117

$\mathfrak{c}_v$ costs of an ECU variant, page 115

d_{b} breadth of the wiring harness, page 37

d_{h} distance from the bottom in the wiring harness, page 37

d_{r} distance from rear in the wiring harness, page 37

F feature, page 112

$\mathbf{F}_{\mathrm{e},\mathfrak{v}}$ feature combination, page 113

$\mathbf{F}_{\mathrm{e}}$ set of ECU features, page 112

f_{s} sample frequency, page 46

l_N net busload on a network, page 50

N network, page 44

$n_{\mathfrak{v}}$ number of cars per vehicle partition, page 113

$n_{v^{\min}}$ predicted number of ordered minimal variants, page 115

$p_{\mathrm{i},e}$ installation rate of an ECU, page 40

$p_{\mathrm{i},v}$ variant installation rate, page 117

$p_{\mathrm{o},c}$ order rate of a component, page 40

p_{oh} protocol overhead factor, page 50

$p_{\mathfrak{U}}$ power consumption in a certain use case $\mathfrak{U}$, page 56

r maximum transfer rate, page 44

$\mathbf{sig}_{N,\mathrm{tr}}$ set of signals that have to be transported on the specified network for a given solution, page 48

t_{d} maximum delivery time, page 44

$t_{\mathrm{D},N}$ maximum transmission delay, page 44

$T_{\mathfrak{U}}$ time requirement for a use case $\mathfrak{U}$, page 57

t_{u} minimum update time, page 43

$\mathfrak{U}$ power use case, page 55

$\mathfrak{v}$ vehicle partition, page 113

$\mathbf{V}$ variant combination, page 116

$\mathbf{v}$ set of possible variants, page 116

v ECU variant, page 115

$\mathbf{V}^*$ optimal variant combination, page 118

$\mathbf{V}_m^*$ local optimal variant combination, page 118

$\mathbf{v}^{\min}$ set of all minimal variants, page 115

$v^{\min}$ minimal variant, page 115

Symbols – Optimisation

A_t archive in generation t, page 16

c_{c_1,c_2} correlation value, page 98

D density, page 17

d_{i} Euclidian distance, page 25

$\mathfrak{d}_{\mathrm{constr}}(s_1, s_2)$ The dominance operator in the constraint space, returns if s_1 dominates s_2 in the constraint space, page 24

$\mathfrak{d}_{\text{Deb}}(s_1, s_2)$ The Deb-dominance operator, returns if s_1 dominates s_2 according to the modified definition of Deb, page 23

$\mathfrak{d}_{\text{Oyama}}(s_1, s_2)$ The Oyama-dominance operator, returns if s_1 dominates s_2 according to the modified definition of Oyama, page 24

$\mathfrak{d}_{\text{new}}(s_1, s_2)$ The new-dominance operator, returns if s_1 dominates s_2 according to the new definition, page 94

$\mathfrak{d}(s_1, s_2)$ The original dominance operator, returns if s_1 dominates s_2, page 14

ε small tolerance, page 11

F SPEA2-fitness, page 17

I_{G} generational distance indicator, page 25

I_{N} number of feasible solution indicator, page 25

I_{H} hypervolume indicator, page 25

$\mathfrak{p}_{\text{CER}}$ correlation evaporation rate, page 100

$\mathfrak{p}_{\text{CP}}$ crossover probability, page 91

$\mathfrak{p}_{\text{ERP}}$ exploitation random parameter, page 98

$\mathfrak{p}_{\text{MMR}}$ move mutation rate, page 91

$\mathfrak{p}_{\text{MR}}$ normalised mutation rate, page 91

$\mathfrak{p}_{\text{PER}}$ pheromone evaporation rate, page 100

$\mathfrak{p}_{\text{UGS}}$ update factor global solution set, page 100

$\mathfrak{p}_{\text{ULF}}$ update factor local feasible solutions, page 100

$\mathfrak{p}_{\text{ULN}}$ update factor local non-dominated solutions, page 100

$\mathfrak{p}_{\text{FUP}}$ factor for update by step, page 101

$\mathfrak{p}_{\text{NCPO}}$ number of colonies per objective, page 99

$\mathfrak{p}_{\text{SPF}}$ sub population size factor, page 99

p_{M} mutation rate, page 91

$\mathfrak{p}_{\text{PSF}}$ population size factor, page 90

$P(s)$ penalty on solution s, page 23

P_t population in generation t, page 16

$p_{\mathfrak{U}}^{\max}$ maximum power consumption in a certain use case $\mathfrak{U}$, page 57

R raw fitness, page 17

S strength value, page 17

$\mathfrak{T}_{\text{bound}}$ modified SPEA2 truncation operator not removing boundary solutions, page 93

$\mathfrak{T}_{\text{new}}$ new SPEA2 truncation operator for constraints modelled as objectives, page 95

$\mathfrak{T}_{\text{orig}}$ original SPEA2 truncation operator, page 18

$\tau_{c,e}$ pheromone value, page 97

Part I.

Introduction

1. Description of the Task

In the past years, a rapid evolution of electronics in vehicles has taken place. The public discusses this development controversially. It is a correct conclusion, that a non-existing function cannot be defect. However, electronic features are motivated by more than convenience.

The share of electronic failures in the break down statistics has been rising over the past years. Although the share is increasing, the total number of electronic defects has been quite constant over the last years. It has to be considered that the percentage of electronics in cars has been rising at the same time. It can be concluded, that the dependability of vehicle electronics is much better than its image.

The number of deadly accidents is declining world-wide, since active and passive safety systems like *Antilock Brake System* (ABS), *Electronic Stability Program* (ESP), and airbag are assembled into cars in significant numbers. A further improvement is expected, once driver assistance systems like *lane assist* and *adaptive cruise control* get more common.

It is interesting, that the fuel consumption of modern cars is declining, while weight and power are rising. This is due to electronic injection and control systems. Not to forget, that in the premium as well as in lower classes, nowadays comfort functions are available that were reserved for the luxury class or even not available at all a few years ago.

All these arguments explain, why the share of electronics in vehicles will further increase. This is also supported by a study of Mercer Management Consulting [MH01], which forecasts a share of 40 % of the production value of a car in 2010 for electronics.

1.1. Motivation

Due to this manifold of electronic functions, modern vehicles have a complex distributed system of *Electronic Control Unit*s (ECUs), networked by several different bus systems. There are up to 80 network nodes in a single car. ECUs are becoming more complex, integrating an increasing number of functions and software logic. They are connected to a huge amount of hardware components, like sensors and actuators. All electronic components that are not inside the

ECU are called sensors and actuators. Sensors (for example switches) normally provide information to the ECU—actuators (for example motors or bulbs) normally execute commands from the ECU, see also [HKK04].

In early development phases, there is quite a big degree of freedom in the allocation of these components to ECUs. Multiple objectives, like costs, energy consumption, or busload have to be considered for optimising the allocation—not to mention the many constraints that have to be fulfilled, like memory, pin, or bandwidth availability.

Human experts usually have problems to effectively optimise multiple objectives simultaneously, especially if the complexity of this allocation problem is considered. In order to recognise shortcomings in earlier phases of the development it is necessary to support the decision process.

Electronic architectures are often historically grown. Major changes in the architecture cause a high risk for a new vehicle type series. The goal is a tradeoff between changing the whole system for directly saving money on the one side, and changing as few as possible in comparison to an existing type series on the other side.

A point aiming in the same direction is therefore the traceability of architectural decisions. An engineer might ask two years after a decision has been passed, why the decision has been made in this way. It is often very difficult to trace the reason.

The main reason for starting this work is the increasing functionality and especially their interconnectivity in modern cars. How can an optimum for an architecture be found if a huge number of new functions have to be brought in series in shortening time periods. At the same time an optimum has to be found in order to resist the increasing cost pressure.

1.2. Control Networks in Vehicles

In order to describe the task handled in this work, some basics of electronic architectures in vehicles are explained in the following two sections.

Every modern vehicle has a complex network of interacting functions, composed of so-called software and hardware components. The components are assigned to ECUs. On an ECU, *hardware components* like network interfaces, power electronics, and pins are located that allow the attachment of sensors and actuators. Thus, hardware components can be located on ECUs (like power electronics or pins) or outside ECUs like sensors and actuators, which are connected to the pins with cables. Micro-controllers provide for example resources like ROM/RAM and timers that allow the assignment of *software components*.

Components c are referred to as an aggregation of software and hardware components. This aggregation even allows to model hardware and software dependent on each other as single components. It is a common case that actuators need power electronics directly connected to an *ECU e*, and that device driver software must be on the same ECU as well due to timing and/or safety requirements.

Nowadays, ECUs are developed using standardised software components. A detailed overview about the Volkswagen standard software can be found in [KWEP03]. Among other tasks, these standard software modules support an abstraction of hardware, like network drivers [HIS03, LIN03] or I/O-drivers [HIS04], subsumed called *Hardware Abstraction Layer* (HAL).

Modern architectures often have a central gateway and a number of different networks connected to it. Notable network types are the *Controller Area Network* (CAN, [CAN05]), *Local Interconnect Network* (LIN, [LIN03]), *Media Oriented System Transport* (MOST, [MOS05]), and *FlexRay* [Fle05]. Additionally, the architecture contains sub networks. The kind of network is depending on the communication needs of the connected ECUs. Architectures and whole network nodes are often reused in several type series' [Köt05].

1.3. Middleware Concepts

Besides the abstraction of hardware described above, several research projects have been carried out developing middleware concepts for the automotive domain. Examples are *Architecture Electronique Embarquée* (AEE, [AEE05]), *Titus* [FRHW00], *Dependable Embedded Components and Systems* (DECOS, [DEC06]), and *Embedded Architecture and Software Technology—Embedded Electronic Architecture* (EAST-EEA, [ITE05, TEF$^+$03]). This research resulted in the *AUTOmotive Systems ARchitecture* (AUTOSAR, [AUT05]) project, which is still ongoing. Currently, more than 40 companies are members in AUTOSAR.

The common idea of the research projects is defining a middleware layer for achieving an abstraction of the location of software components in a network. Software components are not directly communicating with the network drivers or the ECU-local interface drivers. Software components only know the interface of the middleware. All signals needed for communication by the software components are provided in the required form by configuration of the middleware.

Figure 1.1 shows the flow of signals in an example. Two ECUs *e* are connected to a network. Two signals are exchanged between application 1 and application 2 and application 3, respectively. One signal is submitted via network from

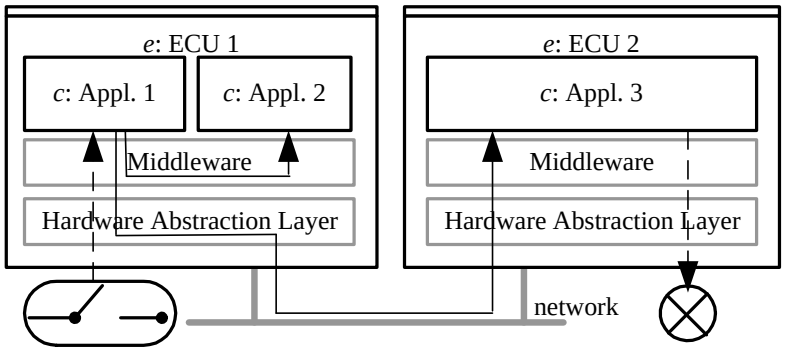

Figure 1.1.: *Middleware between hardware abstraction and application*

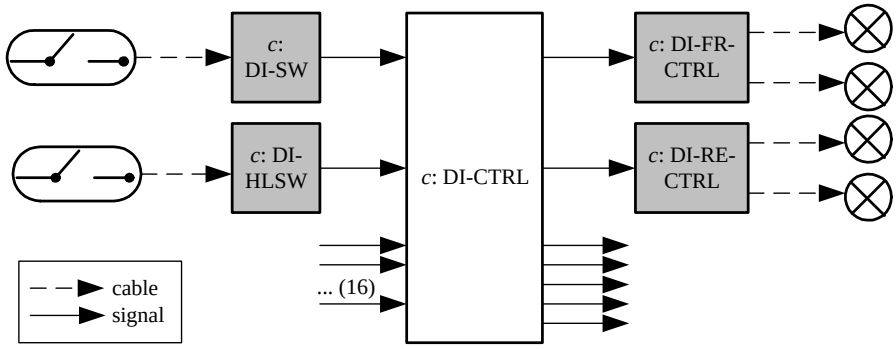

Figure 1.2.: *The function direction indication as an example for hardware independent modelling*

application 1 to application 3. A second signal is exchanged locally, without invocation of the HAL, directly by the middleware between application 1 and 2.

The described concept of middleware adds a lot of flexibility to the location of application software. Software components can be allocated more freely in the network. In addition, applying this approach, hardware accessors can be moved with little effort. The middleware cares for the transportation of the signals to the respective software components via network.

The impact of re-allocating functions on the hardware is shown in an example: In Fig. 1.2, the function *direction indication* can be seen without hardware. The dotted lines are discrete cable connections and the solid lines symbolise signal exchange between components. The grey boxes are hardware/hardware-near software components (for convenience, in the following simply called hardware components) and the bright box is a pure software component.

The *Direction Indication Master Control* (DI-CTRL) is the central component of the function. It is responsible for the prioritisation and execution of the different

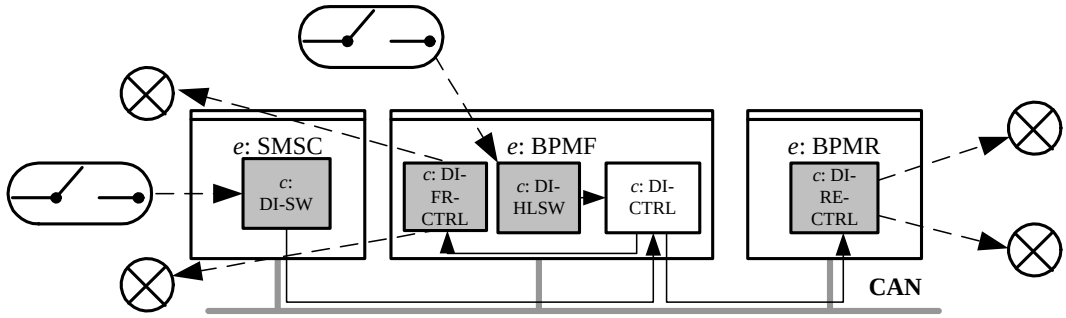

Figure 1.3.: *The function direction indication allocated to a network topology*

functions. Only the component DI-CTRL has about ten sub functions like direction indication, hazard-warning signal flasher, crash signal flasher, panic signal flasher, central door locking confirmation, trailer direction indication, theft alarm light, and so on. All together, 18 different input signals are necessary including *direction indication switch* (DI-SW), *hazard lights switch* (DI-HLSW), *trailer detected*, and *ignition on* among others. On the output side, two additional control components are necessary for the front bulbs (DI-FR-CTRL) and the rear bulbs (DI-FR-CTRL).

Hardware components include the path from the ECU-pin over the circuit board, over the micro-controller-pin to the debouncing[1] software. These hardware components have to be re-allocated completely if necessary. Hardware components, which are sensors, contain the way from the power electronics to the ECU pin.

In Fig. 1.3, the same function is shown allocated to hardware. Three ECUs *e* are connected to one CAN network. It can be seen that the direction indication switch is read by the *Switch Module Steering Column* (SMSC). The switch signal is transported via the network to the *Body Power Module Front* (BPMF) where the direction indication control software DI-CTRL component is located. This software component in turn is spreading the activation signals to the four bulbs both locally and again via CAN network to the *Body Power Module Rear* (BPMR).

If, contrary to Fig. 1.3, all components are allocated to a single ECU, no signals are exchanged at all. However, this results in higher costs for cables.

1.4. Objectives for the Allocation

Several objectives have to be considered during the allocation of functions:

[1] Debouncing is the removal of contact bounce that occurs due to the springy metal in switches and relays.

- The *busload* depends on the allocation of functions. In order to reduce the overall busload, functions with a lot of need for communication can be located on the same networks or even on the same ECUs. A simplified example of a network architecture with a central *gateway* and two different networks of the types *Controller Area Network* (CAN) can be seen in Fig. 1.4.

- In addition, different energy states have to be considered for reducing the *electrical energy consumption*. In the state *ignition off*, several networked functions are offered to the user, like for example radio or hazard lights. Putting the components belonging to these functions onto different control units requires the networks to be awake and consuming energy.[2] The energy consumption can be decreased by reducing the number of active ECUs in some use cases. In the use case *ignition off and no function active*, the energy consumption can be reduced by locating functions that have to be always active, on a number of ECUs as small as possible. Examples for components, which have to be active always, are receivers for the radio key or theft protection.

- A very important factor are *costs*. Costs can be decreased by reducing the length of cables in the wiring harness. Additionally, the behaviour of customers has to be taken into consideration. There might be functions that are not always ordered by the customer. The whole ECU might not be installed or these functions can be deactivated. Another possibility is to create variants of the ECU. The work done on the determination of ECU variants is described in [HKKK05, HK05].

- Another objective to be considered is called *supplier complexity*. For each component, a set of suppliers, which is able to develop and produce it, is defined. For a given allocation and a given ECU, the supplier complexity is defined by the minimum number of suppliers involved in the development of the ECU. The problem of calculating this minimum number can be transformed into a so-called *set-covering problem*. Algorithms for solving this problem type are summarised in [CFT98]. As global value for the objective supplier complexity, the sum of the supplier complexities of all ECUs is applied.

Besides the mentioned objectives, many *pure constraints* are involved in the search, like

- memory consumption,

[2] The front and rear hazard lights at least have to be synchronised from time to time.

- pin consumption,
- timer consumption, and
- space consumption on the circuit board in the ECU.

In addition, constraints that are objectives at the same time have to be considered. An example is busload, which is limited to 100 % or less. Often, the boundary is much lower, since free capacity has to be reserved for additional functions that are introduced in a later phase of the product life cycle of the new car. Since a lower busload gives more degrees of freedom for the development of functions in the future, busload is still an objective, even if it is below the boundary.

No general criteria can be determined, according to which the objectives and constraints can be grouped or ordered. In Chap. 3, it is shown in detail, how the different objectives can be handled, how quality ratings can be calculated, and which data is necessary for them.

1.5. Allocation of Components

With the knowledge of how to map functions on different locations within the electronic architecture, another question arises, which is not addressed by the projects in Sect. 1.3: What is a good location to move the components onto while considering the different objectives and constraints? During the development of a new type series of a car, the system architects have to define which control units communicate with each other and which functionality to put on which control unit.

This problem type is subject to research in form of static task allocation for several years. There exist one-pass greedy algorithms [BS98] that could be applied to the vehicle domain as long as the aspects were limited to CPU and memory consumption. Some of these algorithms even solve global resource problems such as the busload [THW95], but without consideration of whole networks of bus systems with their routing problems.

It is important to notice that the allocation in this work is done in very early phases of development, when no implementations are available. This is due to the fact that most of the functions are developed together with the suppliers of the according ECU after the allocation. There is recent work from Blickle, Teich, and others [BTT98, SHT05] in the area of hardware and software co-design addressing similar problems. The main focus of their work lies on real time properties, communication issues, and task and bus schedules. They address

the problem mainly in a later phase of the development process where implementation and/or detailed specifications of the functions are available. In this work, the early development phase is addressed from the view of an OEM. In the early phase, the available information is not sufficient for an exact determination of the system properties.

Nowadays, the allocation of the components is usually generated by senior engineers and their expert knowledge, without any tool-support. In future, after the allocation decision, tools like DaVinci [Veco5], INTECRIO [ETA05], or TRESOS [3SO06] can be used for configuring the middleware. Code can be generated, that performs the actual communication of a selected component, adapting to the specific network(s) that are available on the chosen ECU.

During the allocation of components to ECUs, several solutions are compared and assessed. Multiple objectives and constraints influence the allocation decision, as already explained in Sect. 1.4. Human experts often encounter problems in effectively optimising multiple objectives simultaneously. Thus, shortcomings are often recognised in a late construction phase, when the vehicles are already being built. Therefore, it is necessary to support or even automate this decision process.

Not every new vehicle series demands a total re-development of the whole system architecture. Already the addition of some new functions is a very demanding task. Considering ten new functions for a new vehicle type series and five possible locations for each, already $5^{10} = 9,765,625$ possibilities for the allocation exist, although constraints reduce this number significantly.

Sometimes, components have to be modelled in a fine granularity depending on the application. The reason could for example be the ability to model costs and weight of single integrated circuits. So-called *component units* are introduced in order to ensure both the possibility of modelling fine granular and the atomic allocation. A component unit is always combining a number of components that have to be allocated together to ECUs. In this work, it is always spoken of components, if no differentiation is necessary.

More formal, every solution s can be defined as a set of q allocations $a = \{c \overset{\text{asg}}{\rightarrow} e\}$, since any of the q components c has to be allocated to an ECU e. Therefore, every solution is defined by $s = \{a_1, a_2, ..., a_q\}^{\mathrm{T}}$. This is illustrated in Fig. 1.4.

The problem of function allocation has the specific property, that often a high percentage of the possible solutions are infeasible due to tight constraints. A formal problem description can be given as follows: The optimisation deals with m objective functions and n constraints. A solution s is defined by a set of q decision variables a. Without loss of generality, the problem can be written as

$$\text{minimise } \mathbf{f} = \{f_1(s), f_2(s), ..., f_m(s)\}^{\mathrm{T}} \tag{1.1}$$

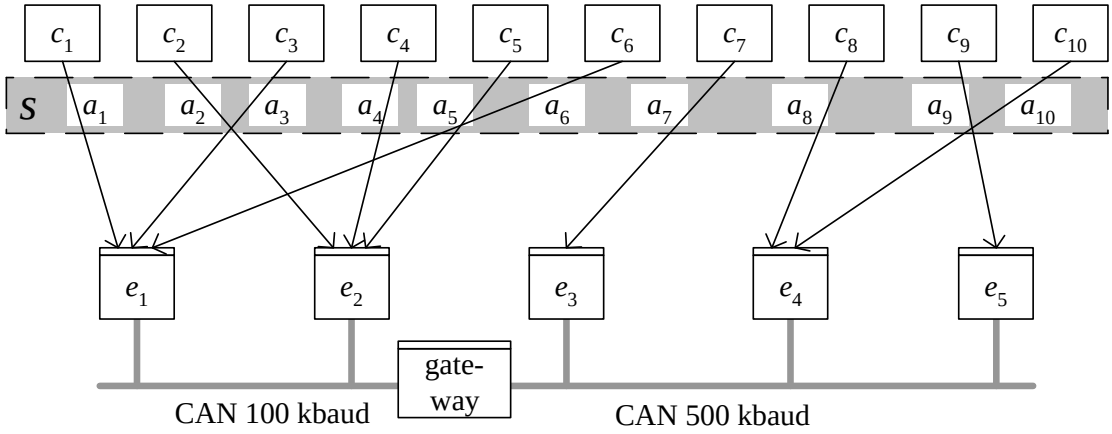

Figure 1.4.: *Example for a possible allocation set (solution s) of 10 components c to five possible ECUs e in two networks*

$$\text{subjected to } \mathbf{g} = \{g_1(s), g_2(s), ..., g_n(s)\}^{\mathrm{T}} \leq \{0, 0, ..., 0\}^{\mathrm{T}} \tag{1.2}$$

where $s = \{a_1, a_2, ..., a_q\}^{\mathrm{T}}$ and each a describes the assignment of one component to an ECU. All constraints $\mathbf{g}$ are formulated as functions with a value less or equal to zero if they are not violated. All solutions are *feasible* unless at least one constraint in $\mathbf{g}$ is violated. All equality constraints $h(s) = 0$ can be transformed into inequality constraints [Coe02] in order to fit in (1.2) via

$$|h(s)| - \varepsilon \leq 0, \tag{1.3}$$

where ε is a small tolerance.

1.6. Summary of the Task

Several tasks are handled within this work. They can be summarised as follows:

- Characterise objectives for the allocation and develop ways to calculate them.
- Develop and compare strategies for a multi-objective optimisation of the allocation of functions to control units. Constraint for all optimisation algorithms is that they have to work with optimisation problems of realistic size.
- Develop a database model including all data for evaluating the objectives and applying the optimisation algorithm.

- Implement a framework for verification of the developed strategies.
- Show an application example including results.

1.7. Outline

In this work, a review of related work is performed in Chap. 2. The allocation criteria are examined in Chap. 3. Chapter 4 gives an overview on the architecture of the system used for performing all tests. Additionally, a realistic application example in is provided Chap. 5. It is presented, how the introduced type of problems can be solved using multi-objective optimisation algorithms in Chap. 6. They are improved specifically for the problem of function allocation. All optimisation algorithms have certain parameters to be adjusted. Approaches for adjusting and finding good parameters are shown in Chap. 6 as well. Chapter 7 explains, how the special problem of ECU variants can be handled. It is a sub problem of the global optimisation of the function allocation. The work is concluded in Chap. 8.

2. Review of Related Work

In this chapter, related work is presented necessary for understanding the other sections. First, several population based optimisation algorithms are presented in Sect. 2.1. They are the basis for the optimisation in Chap. 6. Additionally, an overview on modelling data structures is given in a depth as used in the further context (see Sect. 2.2).

2.1. Population Based Optimisation Algorithms

Classical optimisation techniques focus on finding values for variables in order to optimise according to a single objective. It is already outlined in Sect. 1.4 that there are several objectives to consider during the optimisation of function allocation in vehicle electronics. The classical approaches can be adopted for *Multi-Objective Optimisation Problems* (MOOP) as well. Parameters can be introduced giving an importance or weight to the different objectives. By this transformation of multi-objective problems into single-objective problems, the complexity of multi-objective optimisation can be avoided.

From a practical point of view, only one solution is needed for realisation. However, how to set the weight parameters? Non-linear relationships between the objectives are even more complicated to handle. Furthermore, it might be an interesting information to choose between several solutions having advantages in different objectives each. Modern optimisation strategies as described in this section are able to handle multiple objectives at the same time and provide a set of solutions as optimised result. Thus, no weight parameters are necessary and the user can choose between several optimal solutions.

Heuristic optimisation is usually applied to problems with a range of possible solutions so big that an exhaustive search is not possible. Furthermore, they are not bound to a special problem, but applicable to different problem classes. Special forms of heuristic optimisation algorithms called *population based optimisation algorithms* manage a population of solutions. Many of these algorithms are suitable for multi-objective problems as well. Classical optimisation algorithms, modified for MOOPs, usually find only one solution per optimisation run, whereas population based algorithms can find a set of solutions within a

single run. A more detailed motivation of the use of population based algorithms for MOOPs, has been published by Deb in [Deb01].

Population based optimisation algorithms are characterised by a set of solutions. This set of solutions is called *population*. Typical examples are *evolutionary algorithms* and *ant colony optimisation* as introduced in the course of this section.

This section gives a brief introduction into basic terms and techniques that are required to understand population based multi-objective optimisation strategies. At first, the term Pareto front is explained. Additionally, an overview over population-based optimisation algorithms is given. Several frequently studied population based optimisation algorithms are reviewed. Current methods for handling constraints and constraint violations are explained. Finally, a method for benchmarking the quality of an optimisation algorithm is presented.

2.1.1. Pareto Optimality

If there is only one objective involved in the optimisation, normally, only one solution exists that has an optimal quality rating. This solution is called an optimal solution for the given problem. *Vector Evaluated Genetic Algorithm* (VEGA) has been described in [Sch85] for handling multiple objectives at the same time. In VEGA, the population is sorted in sub-populations, one for each objective. Subsequently, each sub-population is evaluated according to a single objective. It is difficult to explore the entire *Pareto front* when applying this method.

Later methods apply the concept of Pareto optimality. Multi-objective optimisation results in multiple solutions being superior in different objectives each. This results in a set of optimal solutions. This set of best solutions is called the *Pareto front*[1] according to Vilfredo Pareto. The Pareto front can be determined in accordance to the quality ratings of the objectives and the concept of *dominance* according to Deb [Deb01]:

Definition 1 *A solution s_1 is said to dominate $\mathfrak{d}(s_1, s_2)$ the other solution s_2, if both conditions are true:*

- *The solution s_1 is no worse than s_2 in all objectives f*

$$\forall f(s_1) \leq f(s_2) \tag{2.1}$$

- *The solution s_1 is strictly better than s_2 in at least one objective*

$$\exists f(s_1) < f(s_2) \tag{2.2}$$

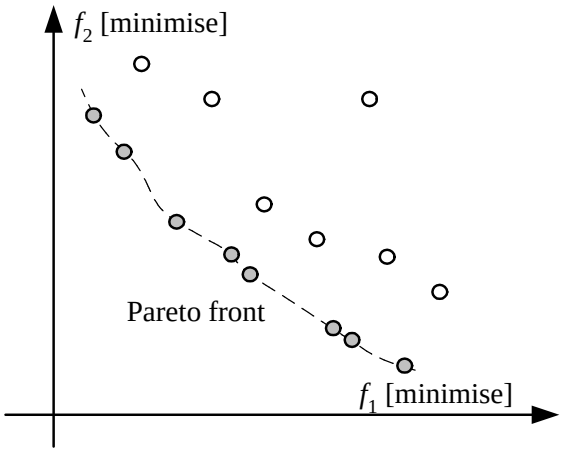

Figure 2.1.: *A population of several solutions and its Pareto front for two objectives and their objective estimation functions f_1 and f_2*

The Pareto front is the set of solutions that are *non-dominated* by any others at all, as is shown in Fig. 2.1. More information can be found in [Deb01].

The human expert running the optimisation system is primarily provided with the Pareto front. The task of selecting the one single best solution that is implemented finally can only partially be supported automatically. Some aggregations of the quality ratings could be applied. Plain aggregation models are for example linear combinations like weighted sum method, goal programming, or goal attainment [Coe00]. This way, the user can be supported in selecting one single best solution.

Nevertheless, considering that most objectives are hardly correlated and that the formalisation or quantification of importance is usually not possible—aggregation models do not give proper results. Anyway, there is little necessity for aggregation, because human experts, provided with a Pareto front, are superior in selecting the single best solution for their problem in short time.

2.1.2. Evolutionary Algorithms

Evolutionary Algorithms (EA, [Hol92]) are typical representatives of population based algorithms. Potential solutions are represented as *individuals*. The individuals are encoded as *genomes*. A whole set of solutions/individuals is called *population*. Imitating nature's breeding, EAs apply operations like *mutation* and *crossover* of genomes or pairs of genomes in order to evolve the population.

[1]The Pareto front is sometimes called *Pareto trade-off frontier*, *Pareto set*, or *non-dominated set*.

Since the cardinality of populations usually should remain constant or within some boundaries, a selection of the genomes is applied based on the numeric *fitness* of a genome. For multiple objectives, this fitness is a vector.

Seeding of the population refers to the creation of solutions from scratch. Seeding can be done *initially* or *iterative*. Initial seeding means that after the initialisation phase, new solutions are created exclusively by applying evolutionary algorithms on existing individuals or genomes. Whereas iterative seeding means that even after the initialisation phase the algorithm may decide to create new individuals from scratch.

Standard operations for EAs are mutation and crossover. The *mutation* operation prevents the population from becoming too similar and helps to avoid local minima. The *crossover* is a recombination of usually two individuals. Several examples are the one-point, two-point, uniform, and half-uniform crossover.

Three basic classes of evolutionary algorithms are named *Genetic Algorithms* (GA, [Gol89]), *Genetic Programming* (GP, [Koz92]) and *Evolutionary Strategy* (ES, [Rec94]). They differ for example in their used genotype and in their applied genetic operation.

Finally, an *interrupt condition* (or often called stop criterion) has to be defined. The easiest one could be to stop the optimisation after a given number of generations. A better one could be to check the improvement of the solution in the last generation and stop the optimisation if the improvement was less than a certain percentage. This could fail if local optima are found. Better interrupt conditions for example combine both approaches.

Sorted Pareto Evolutionary Algorithm

It has been decided to use and improve (see Sect. 6.1) the *Strength Pareto Evolutionary Algorithm* 2 (SPEA2, [ZLT01]), a state of the art *Multi-Objective Evolutionary Algorithm* (MOEA). According to Zitzler et al. [ZLT01], SPEA2 has advantages especially for optimisation problems with a high number of objectives in comparison to other MOEAs like the *Non-dominated Sorting Genetic Algorithm* (NSGA-II, [DAPM00]) or the *Niched Pareto Genetic Algorithm* (NPGA, [HNG94]). It is also applied by Schlichter et al. [SHT05] in a similar optimisation problem. In SPEA2, only a fixed number of individuals in the population survive in each generation due to the applied selection operations. The complete SPEA2 algorithm is shown in Alg. 1. One attribute of SPEA2 is that it manages an archive A_t of fixed size N for the storage of parent generation individuals. As first step, an initial population P_0 is generated (seeding). For example, it can be filled with random solutions.

A sophisticated fitness calculation algorithm is applied for reducing the size of the archive together with the new generation after the application of evolu-

Algorithm 1: Complete SPEA2 Algorithm

```
1  t := 0;
2  Generate the initial population, P_0, and archive, A_0;
3  while t < T do
4  |  Calculate fitness F = R + D of all individuals in P_t ∪ A_t;
5  |  A_{t+1} := non-dominated individuals in P_t ∪ A_t;
6  |  if |A_{t+1}| ≤ N then
7  |  |  Fill A_{t+1} from dominated solutions in P_t ∪ A_t ordered by the fitness;
8  |  else
9  |  |  Remove solution returned from truncation operator T_orig (see Alg. 2)
   |  |  from A_{t+1} until |A_{t+1}| = N;
10 |  end
11 |  Fill the mating pool P_t by binary tournament selection with replacement
   |  on A_{t+1};
12 |  Apply evolutionary operators to mating pool and place results in P_{t+1};
13 |  t := t + 1;
14 end
15 return A_t;
```

tionary operations (see Alg. 1, line 4). The calculation algorithm for the *SPEA2-fitness F* exploits information about domination and density:

- First, the *strength value S* is assigned to each solution. The strength is the number of other solutions that are dominated.
- Second, the *raw fitness value R* is assigned to each solution. The raw fitness value for each solution is sum of the strength values of all dominating solutions. The domination relation $\mathfrak{d}(s_1, s_2)$ is according to Def. 1.
- A so-called *density D* is applied as a second criterion. The density value is only crucial, if two solutions are equal in means of the raw fitness value. In order to determine D of all solutions in $P_t \cup A_t$, the distance D' to the k-nearest neighbour is determined for each solution. The value of k is determined with $k = \sqrt{|P_t \cup A_t|}$. Thus, the density can be determined with $D = \frac{1}{D'^k + 2}$.
- The assigned SPEA2-fitness $F = R + D$ is only applicable, if the archive size is bigger than the number of non-dominated individuals in the archive.

Figure 2.2 shows values for S,R, and D on the left hand side and F on the right hand side.

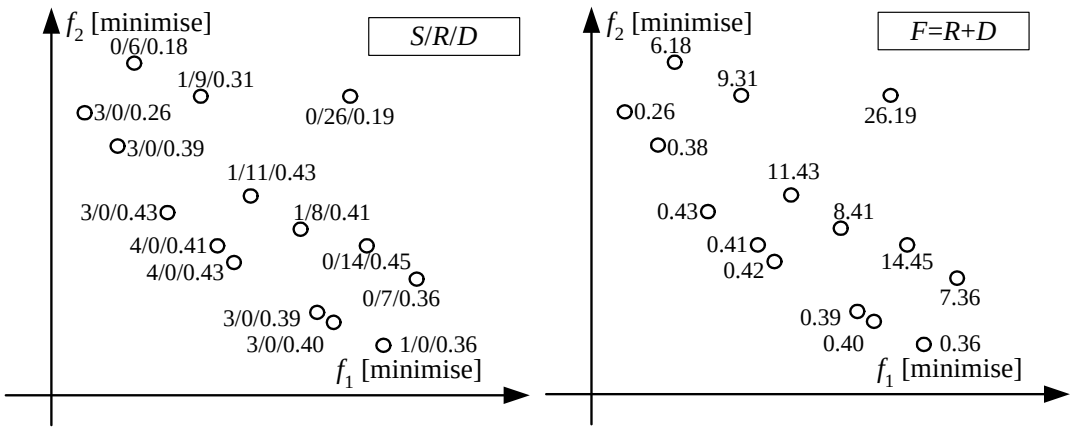

Figure 2.2.: *Example for the determination of the SPEA2-fitness F (right hand figure) based on the strength value S, the raw fitness R, and the density D*

If the archive size N is bigger than the non-dominated set, the archive is filled from $P_t \cup A_t$, ordered by the fitness value (see Alg. 1, line 7).

If the archive size N is smaller than the non-dominated set, a truncation operator $\mathfrak{T}_{\text{orig}}$ returns individuals to be removed from this non-dominated set. It is defined as follows: Remove the individual that has the minimum distance to its neighbour. If several individuals exist with the same minimum distance, the tie is broken by considering the second smallest distances and so forth. This operator as shown in Alg. 2 helps to prevent boundary solutions to be removed and to keep diversity. The method getSolsWithNearestNeighbour($\mathbf{s}_{\mathfrak{T}}, k, A$) returns solutions with the k-nearest neighbour to the member of A from a solution set $\mathbf{s}_{\mathfrak{T}}$. For example, if $k = 2$, the solutions with the closest distance to the second-nearest neighbour in A from $\mathbf{s}_{\mathfrak{T}}$ are returned. Figure 2.3 shows the order of solutions removed from a Pareto front using Alg. 2.

Algorithm 2: $\mathfrak{T}_{\text{orig}}$: Original SPEA2 Truncation Operation

Input: An archive A with non-dominated solutions s

```
1 s_T := A;                                   // initialize truncation candidates
2 k := 1;                                     // Distance to the k-nearest neighbour
3 while |s_T| ≠ 1 do
4 |   s_T := getSolsWithNearestNeighbour(s_T, k, A);
5 |   k := k + 1;
6 end
7 return s_T;                                 // exactly one solution to be truncated
```

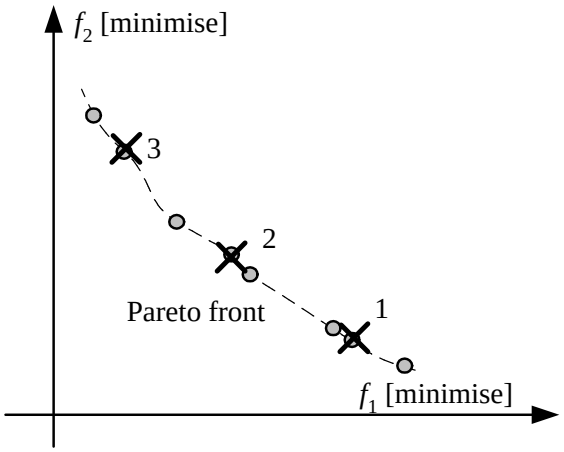

Figure 2.3.: *Order of solution removal in a Pareto front using the truncation operator* $\mathfrak{T}_{\text{orig}}$

A mating pool P_t (see Alg. 1, line 11) is filled by binary tournament selection from A_t, before evolutionary operators are applied to it. Finally, the results are placed in P_{t+1}. Binary tournament selection means that random pairs are chosen from the archive. These are compared according to the SPEA2-fitness. The better one is placed in the mating pool until its target size is reached.

2.1.3. Ant Colony Optimisation

Another representative of population based optimisation algorithms is *Swarm Intelligence* (SI, [BDT99, KE01]). One type of SI is called *Ant Colony Optimisation* (ACO, [CDM91]). The algorithm is based on the behaviour of real world ant: Real world ants are marking the way they are walking with *pheromones*. These pheromones evaporate after time. The following ants decide which way to go with a probability depending on the strength of the pheromone trail. If an ant has found food and walks back, it amplifies the pheromone trail on its way to the nest. Following ants prefer this path more and more. After a short while, all ants walk the shortest way to the food with a high probability.

ACO algorithms use the analogy of ants finding food. They have originally been developed for the *Travelling Salesman Problem* (TSP, [SD99]). The optimum solution for the TSP is an order of cities in which a travelling salesman should visit each city once, covering a minimum distance. The decision, which city to visit next is influenced by randomising based on pheromones and heuristic

information. The probability p_{ij} that a city j in the neighbourhood of city i is chosen as next city is given with

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum_{j \in N} \tau_{ij}^{\alpha} \eta_{ij}^{\beta}}. \tag{2.3}$$

The variable τ_{ij} is the so-called pheromone value, η_{ij} the heuristic information and N the set of all neighbour cities of i. The pheromone concentration is higher on shorter (or better) paths, because ants travel on them more frequently. In the case of the TSP, the heuristic information η_{ij} is the inverse distance between the cities. Thus, a small distance increases the probability for choosing the according city. For the sake of diversification, the probability for less-pheromone paths should be above zero. If the probability is above zero, the algorithm is able to find new solutions and not only to affirm known ones. The parameters α and β weight the influence of pheromones and heuristic information.

An important point of the algorithm is the update of the pheromones. In one method, pheromone values are changed after each single movement of the ants. In this case, the value of τ_{ij} is reduced first. This part is called *evaporation* and simulates the natural evaporation of pheromone trails. In the second step, the pheromone value is increased. The new pheromone value at the time $t+1$ is determined with

$$\tau_{ij}(t+1) = \rho \tau_{ij}(t) + \Delta \tau_{ij}(t, t+1), \tag{2.4}$$

where the value of ρ is a measure for the evaporation or persistence of the pheromones. The increment $\Delta \tau_{ij}(t, t+1)$ is either a constant or a value depending on the distance between the cities i and j.

In a further method, pheromones are updated after completion of a whole tour. In this case, all edges within each tour are increased indirect proportional to the length of the whole tour.

The standard algorithms explained before are not able to handle multiple objectives and constraints. In the remainder of this section, these two problems are addressed. The previous ant colony optimisation algorithms refer to single objective optimisation problems. In several studies, ACO is applied to multi-objective optimisation problems as well. An overview can be found in [GMCH04]. A common approach is to establish one ant colony for each objective. The colonies are sorted according to the importance of the corresponding objectives. This approach has already been applied to the vehicle routing problem [GTA99] and the TSP [MM99].

In many use cases, it is not possible to sort the objectives. In [IMM01], an approach is introduced avoiding this problem. Like in the previous approaches

[GTA99, MM99], there is one colony for each objective. Each colony has its own pheromone information and the ants in a colony only use this information for the update. In order to provide indirect communication between the colonies, all solutions are collected in a global pool. The non-dominated solutions from this pool update the pheromone values. There are two different ways to do this. Either, ants only update the pheromone values of their own colony (*update-by-origin*) or the solutions are sorted according to an arbitrary objective (*update-by-region*). The second method allows guiding the colonies explicitly in different regions of the search space.

Several ACO algorithms are able to consider constraints, like in [MH04, BP96]. The applied constraint handling techniques are similar to the method of ensuring feasibility as shown in Sect. 2.1.4. In this approach, the feasibility is guaranteed by the update operators. The main drawback is that for any additional objective, an adaptation of the ACO operators is mandatory for ensuring the feasibility. In Chap. 6, a method will be shown avoiding this disadvantage. A rating of the different methods and a selection of the applied ACO algorithms will be performed in Sect. 6.2.

2.1.4. Constraint Handling Techniques

Most population-based algorithms have in common, that they do not directly support constraints. As already outlined in Sect. 1.4, several constraints have to be considered during the optimisation like memory or space consumption. Several constraint handling techniques that can be used for population based optimisation algorithms are compared in [Coe02] and [KAW02]. The relevant types are:

- Discard infeasible solutions
- Ensure feasibility
- Repair infeasible solutions
- Penalty approach
- Modifying the dominance relation

The following requirements, that constraint handling techniques have to fulfil, can be identified:

- *Extendibility*. It must be easy to extend the optimisation with additional objectives. Therefore, it is not wished that the constraint handling technique requires support in the form of additional algorithms.

- *Parametrisation*. Each parameter eventually has to be adapted to the specific optimisation problem. Therefore, only constraint handling techniques are considered that do not involve additional parameters, like for example weight parameters.

- *Performance*. A good performance for all kinds of optimisation problems, especially including problems with tight constraints, is desired.

Discard Infeasible Solutions

A very easy constraint handling technique is discarding infeasible solutions. For example, every operation where an infeasible solution could occur can be repeated until a feasible solution is found [HS96]. This technique is only applicable, if few infeasible solutions are in the problem space. Otherwise, even after a number of generations not a single valid solution might be found at all. This approach does not consider infeasible regions at all. Thus, a movement of solutions through infeasible regions is not possible. Concluding, the extendibility and parametrisation of this approach is very good, but the performance is poor.

Ensure Feasibility of Solutions

A further constraint handling technique is ensuring feasibility. This can be done by representation [PK94] or by operators [Kow97]. In order to ensure feasibility by representation, the solution has to be represented in a way so that infeasible solutions are not possible. The main drawback of this method is the poor extendibility. For each type of problem and even for each new objective, that representation and/or the operators have to be adopted.

Repair Infeasible Solutions

Besides ensuring the feasibility of solutions, infeasible solutions could be repaired before they are further considered. For optimisation problems with tight constraints, this might be as difficult as the whole optimisation. Thus, performance is quite poor. Furthermore, these approaches are problem dependent. Nevertheless, successful applications have been shown for example in [TS95, XMT97, ZT99].

Penalty Approach

A very common method is the penalty approach. It has been presented by Goldberg in [Gol89]. The basic approach is to redefine the fitness values of a solution s as defined in (1.1):

$$\text{minimise } \mathbf{f}' = \mathbf{f} + P(s), \tag{2.5}$$

where $P(s)$ is a *penalty* dependent on the current solution s. It is assumed that $P(s) = 0$ if s is feasible, since these solutions are not penalised.

One easy method is penalising solutions for just being infeasible. Since no further information about the infeasibility is used, the performance is quite poor.

It is furthermore possible to penalise the amount of infeasibility of solutions. One approach is to penalise the number of violated constraints [MQ98]. This approach yields in a mid range performance and has the advantage that it needs no parameters.

There exist approaches for determining the severity of the constraint violation. For single objective optimisation problems, $P(s)$ could be determined with

$$P(s) = \sum_{\forall g \in \mathbf{g}} p_g g(s), \tag{2.6}$$

where p_g is a penalty weight parameter necessary for each constraint g. For multi-objective optimisation problem, several more complex strategies for adding a penalty [Coe02, OY05] exist, for example static [BK94], dynamic [JH94], annealing [MA94] or adaptive [HAB97] penalties. The performance of all this approaches seems to be quite good, if the needed parameters for this approach are set correctly. Due to the necessity for parameters, the requirement parametrisation is not met.

Modifying the Dominance Relation

Another approach is modifying the dominance relation. The basic idea is to redefine the optimisation of $\mathbf{f}$ so that the constraints $\mathbf{g}$ are represented as additional objectives. Thus, all population based optimisation algorithms can be applied without modification to the resulting new optimisation problem.

In [Deb00], Deb proposed a method that can be formulated as a modified domination operator. The *domination operator of Deb* $\mathfrak{d}_{\text{Deb}}(s_1, s_2)$ is re-defined according to Def. 2.

Definition 2 *A solution s_1 is said to Deb-dominate $\mathfrak{d}_{\text{Deb}}(s_1, s_2)$ the other solution s_2, if any of the following conditions is true:*

- *Solutions s_1 and s_2 are feasible and s_1 dominates ($\mathfrak{d}(s_1, s_2)$ according to Def. 1) s_2.*
- *s_1 is feasible and s_2 not.*
- *s_1 and s_2 are both infeasible, but s_1 has a smaller amount of constraint violation.*

The problem with this method is again, that the amount of constraint violation has to be compared. This requires the introduction of weight parameters if many constraints are applied. Otherwise, the comparison prefers solutions violating constraints with certain constraint severity value ranges.

A more recent method has been proposed by Oyama et al. in [OSF05]. Again, the dominance operator is redefined:

Definition 3 *A solution s_1 is said to Oyama-dominate $\mathfrak{d}_{\text{Oyama}}(s_1, s_2)$ the other solution s_2, if any of the following conditions is true:*

- *Solutions s_1 and s_2 are both feasible and s_1 dominates ($\mathfrak{d}(s_1, s_2)$ according to Def. 1) s_2.*
- *s_1 is feasible and s_2 not.*
- *s_1 and s_2 are both infeasible, but s_1 dominates ($\mathfrak{d}_{\text{constr}}(s_1, s_2)$ according to Def. 4) s_2 in the constraint space.*

Definition 4 *A solution s_1 is said to dominate $\mathfrak{d}_{\text{constr}}(s_1, s_2)$ the other solution s_2 in the constraint space, if both conditions are true:*

- *The solution s_1 is no worse than s_2 in all constraints*

$$\forall G(s_1) \leq G(s_2) \tag{2.7}$$

- *The solution s_1 is strictly better than s_2 in at least one constraint.*

$$\exists G(s_1) < G(s_2) \tag{2.8}$$

where $G(s) = \max(0, g(s))$.

This method does not need any parameters since the concept of dominance is directly transferred into the constraint space. Unfortunately, this new promising approach has only been tested with one example until now [OSF05].

Some of the described constraint handling techniques are compared and a new one is developed in Sect. 6.

2.1.5. Benchmarking Multi-Objective Optimisation Algorithms

Several multi-objective optimisation algorithms are compared in the course of this work. The following criteria have been identified for benchmarking the quality of a multi-objective population based optimisation algorithm:

- The *quality* of the gathered solutions regarding all relevant objectives
- The *number* of feasible solutions provided to the user
- The *diversity* of the solutions provided to the user
- The number of necessary *objective evaluations* and the execution time

There are several approaches in order to benchmark algorithms according to these criteria. A first quality indicator is just counting the *number of feasible solutions* I_N. No other criterion from the list can be assessed with this metric. Therefore, it is only recommended to use this metric together with others.

A further indicator is the *generational distance* I_G according to [VL00]:

$$I_G = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n}, \tag{2.9}$$

where n is the number of solutions and d_i is the *Euclidian distance* (in objective space) between each solution s and the nearest member of the overall non-dominated set. The complexity of the problems is so high, that exhaustive search (see Sect. 5.3.2) cannot always determine the overall non-dominated set. Instead, the non-dominated set of all solutions tested during all runs can be aggregated. This generational distance indicator cannot assess an algorithm according to number of feasible solutions and diversity. Like I_N, the usage is only recommended together with other indicators.

Quite a big number of quality indicators have been compared in [ZTL+03]. It has been proved that the *hypervolume indicator* I_H according to [ZT99] is among the best ones known currently. In order to determine a value for the hypervolume indicator, maximum values f_{max} must be known for each objective. From each solution, a multi-dimensional volume can be determined against these maximum values. The hypervolume indicator is the inverse of this volume. An example for the calculation of the hypervolume indicator in the 2-dimensional case can be seen in Fig. 2.4. Three solutions build rectangles. The area (in the 2-dimensional case, the hypervolume is an area) of the union of those three rectangles is called I'_H. The hypervolume indicator can be determined with $I_H = f_{1,max} \cdot f_{2,max} - I'_H$.

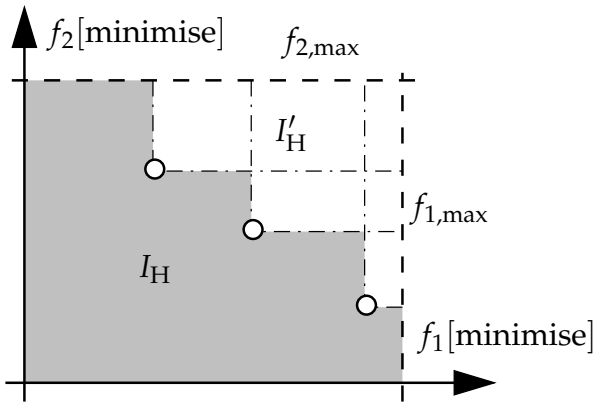

Figure 2.4.: *Example for the calculation of the hypervolume indicator*

The hypervolume indicator covers all criteria claimed at the beginning of this section. Therefore, it is used in the course of this work, especially in Chap. 6. It is important to notice that only feasible solutions are considered.

2.2. Database Models

For the description of the optimisation input—a description of the electronic system in a vehicle—database models are used. Semantic database models represent requirements in the studied problem field from the view of data. They present the structure and the relation between the data in a formal way (for example in a diagram).

The most commonly used specialisation of a database model is the *Entity-Relationship* (ER) model, originally proposed by Chen in [Che76]. A part of the model is the ER diagram, representing the data objects. Entity relationship diagrams can be easily understood even by non-experts in database design.

The advantage of ER models is that SQL[2] code can be generated for building the database structure. Entities are translated into tables—relations into foreign key constraints. SQL is a language for querying and manipulating data. The current standard—SQL:1999—is described more in detail in [GP99].

The notation used in all database models shown in this work expresses entities with *rectangles*. The name of the entity is at the top of each rectangle in a grey box and its attributes below. Each attribute has tags for *Primary Keys* (PK), *Foreign Keys* (FK) and *Unique* (U) constraints, if applicable.

The application of a unique constraint prevents duplicate values in a column. If a unique constraint is applied on more than one column, no two equal com-

[2] *Simple Query Language* (SQL)

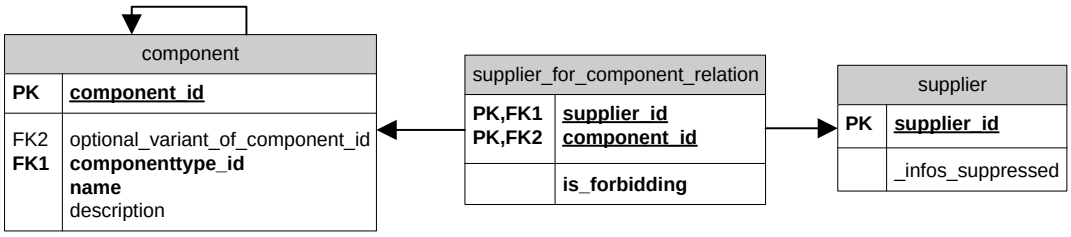

Figure 2.5.: *Entities component and supplier and their relations*

binations of these columns are allowed anymore. The primary key is the identifier of values within a relation. A primary key is unique and not allowed being empty. Only primary keys can be referenced by foreign key constraints. A foreign key constraint on a column ensures that the value in that column is found in the primary key of another table. Foreign keys ensure consistency between the data in different tables and simplify the implementation of the application using the database.

Figure 2.5 shows the entities component and supplier.[3] The entity supplier_for_component_relation is actually a relation. It expresses an n-to-m relation between supplier and component. Therefore, it is tagged with the suffix _relation. An *n-to-m relation* is a relation where each entry of entity *a* is possibly related to several entries of entity *b* and vice versa. In SQL, this is expressed in an additional table. This is the reason for symbolising it with a rectangle here.

In the remainder of this work, database models are used to show the data that is necessary as input for the optimisation. The database model is designed as generic as possible in order to be sustainable for future requirements. Therefore, there are quite complex semantic relationships between the entities. Thus, a human expert is required to input data. For the application in productive environments, it is recommended to provide a graphical user interface, supporting the users.

[3]The relation between component and supplier is described more in detail in Sect. 3.8

Part II.

Application

3. Criteria for the Allocation of Components

There are multiple criteria to be considered at the same time during the allocation of components. This chapter discusses these criteria. Only the ones are considered that can be influenced by the allocation of functions. Criteria that imply a change of the network topology are neglected. For example, reduction of busload is an objective. Nevertheless, the cost effect of replacing CAN networks by LIN networks—if the busload is sufficient low—is not considered. Due to the fixed network topology and hardware, additional constraints have to be considered like memory consumption or maximum busload in a network.

In the course of this chapter, all optimisation objectives/constraints are derived (Sect. 3.1). Common attributes of all optimisation objectives are explained in Sect. 3.2. After that, the different objectives/constraints resources (Sect. 3.3), weight (Sect. 3.4), costs (Sect. 3.5), busload (Sect. 3.6), electrical energy consumption (Sect. 3.7), and supplier complexity (Sect. 3.8) are presented. Database models are provided explaining the information necessary for the calculation of quality ratings for the objectives and related constraints.

3.1. Derivation of the Optimisation Criteria and Objectives

Most car manufacturers are commercial companies. As all commercial companies, their main goal is to maximise the *profit*. This can be done by reducing the *expenses* and/or by increasing the *revenue*. The revenue can be increased by improving the *customer benefit*. This leads to a bigger number of sold cars and a higher prize that can be realised.

There are several possibilities to increase the profit. In the following, only *criteria* are considered that can be influenced by the allocation of components: In order to reduce the expenses, the *production costs* and the *warranty charges* can be reduced. In order to increase the customer benefit, *dependability* and *fuel consumption* have been identified as relevant. The upper part of Fig. 3.1 summarises a number of direct criteria.

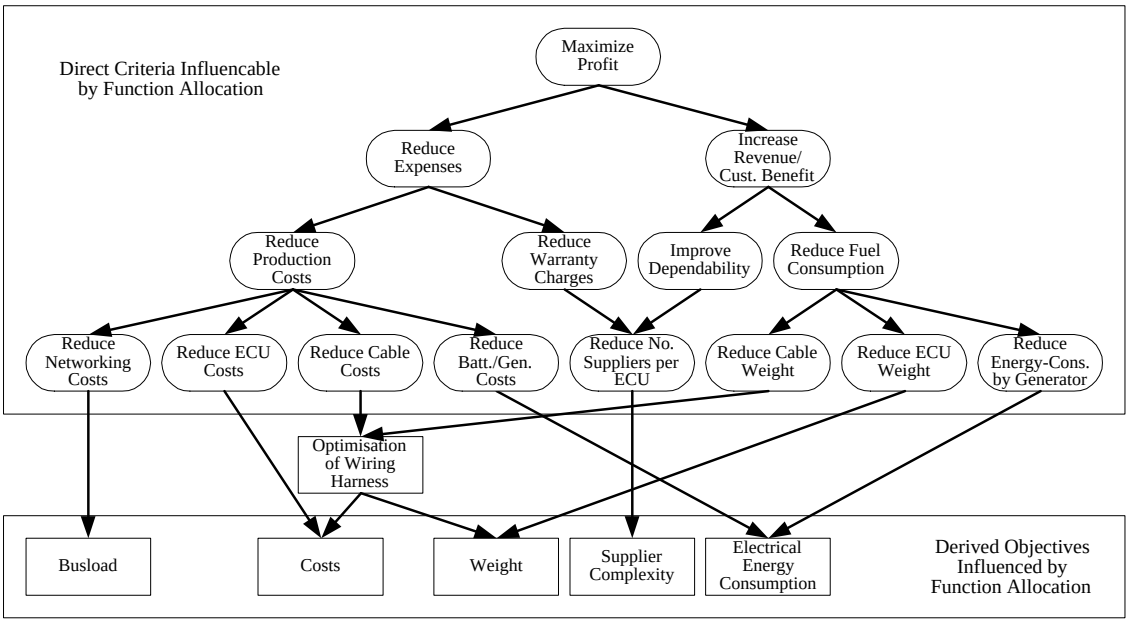

Figure 3.1.: *Derivation of the Objectives*

In the lower part of the figure, several *objectives* are derived. The difference between criteria and objectives is that objectives can be evaluated. It is explained in the course of this chapter, how this can be done for each objective.

It is important to note that it is not claimed to provide a complete list of all objectives. Important objectives from the view of one car manufacturer are identified and examined. It is possible to extend the method with further objectives, since population based optimisation algorithms allow the easy integration of additional objectives. Constraints are developed and explained within this section as well.

It is additionally important to note that an ideal optimisation would only have the criterion *profit*. Since network topology and hardware are assumed fixed, additional objectives like busload have to be taken into consideration. The mentioned ideal optimisation would also require for example an estimation of the revenue effect of weight savings.

3.2. Requirements for Optimisation Objectives/Constraints

This section explains requirements that have to be fulfilled by all implementations of objectives and constraints.

It is recommended to combine objectives and their according constraints within single implementations of objective estimations giving back only one value and a flag indicating if the solution is valid according to the objective

A common feature of all objective/constraint implementations the avoidance of plateaus during optimisation. Sometimes, the benefit of two solutions equals according to one objective. Still, it can be determined that one of the two solutions is nearer to an even bigger benefit. Plateaus are areas where the solution changes but the quality rating is constant although this difference can be determined. All constraints are examples for that fact. If a constraint is violated, the solution has no benefit at all—it is infeasible and therefore not realisable. Anyway, in many cases a severity of constraint violation can be determined. This difference should manifest in the quality ratings. If plateaus occur, the optimisation algorithm does not know in which direction allocations have to be changed in order to improve the solution. Thus, the optimisation process gets more difficult.

A requirement derived from the above one is the following: All quality ratings of an objective with constraint violation must be worse than all quality ratings without constraint violation. This is because the optimisation relies on values indicating how strong the violation is. If this is not possible, separated objective estimation instances have to be provided. It is shown in Sect. 4.2.5 how objective estimation objects can describe objectives and/or constraints at the same time. It is shown in Chap. 6, how these requirements comply with all optimisation and constraint handling techniques.

Furthermore, for each objective, it must be possible to calculate estimations for the extreme values, their quality ratings can return. This requirement is derived from Sect. 2.1.5 where the calculation of the hypervolume indicator requires these values.

As units for the calculation of quality ratings for the objectives/constraints, often, input values like for example energy consumption have to be provided. As far as possible, it is recommended to use the *Le Système international d' unités* [BIP98]. For example, instead of the unit *Horse Power* (HP), *Kilo Watt* (kW) should be used.

3.3. Resources

The goal of this work is to optimise the component allocation. The underlying hardware is not changed by the optimisation and is seen as given. This leads to constraints. Micro-controllers have a limited amount of memory, the size of the circuit board is limited, and the number of available I/O-pins might be restricted [BTT98]. All these constraints are very similar.

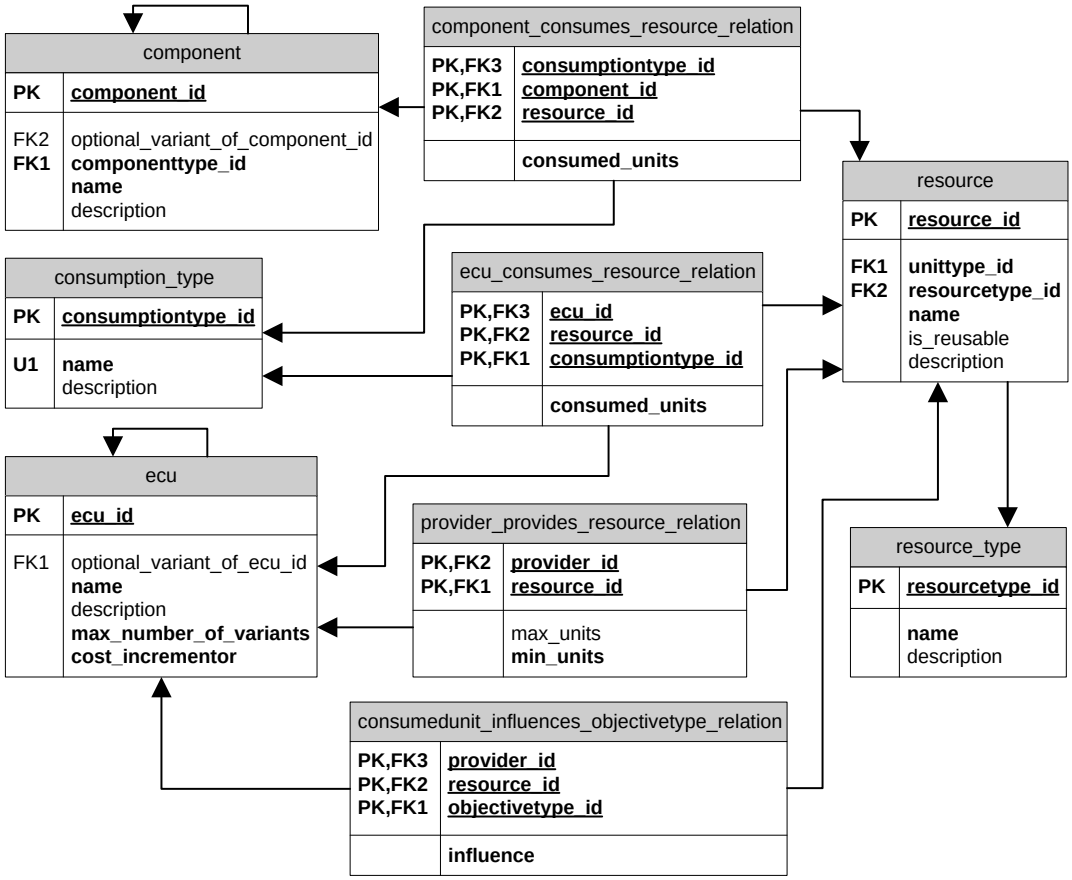

Figure 3.2.: *Database model of ECUs, components, resources, and their relations*

In order to be able to describe these constraints in a unique way and without to change the database model frequently, the concept of *resources* is applied. The basic concept of resources is as follows: resources are provided by ECUs and consumed by the ECUs themselves or components (see Fig. 3.2).

ECUs or components (see p. 5) consume resources with a number of consumed_units. The attribute min_units in provider_provides_resource_relation indicates the minimum number of units that are considered as consumed—although the according ecu and component are consuming less. The attribute max_units constraints the resource consumption.

Resources are not only used as stand-alone constraints. They are also necessary as input for the calculation of the quality ratings for many of the other objectives. In the remainder of this chapter, several other objectives are applying this concept as well. The relation consumption_type is important for distinguishing between different types of consumption by these other objectives. The resource_type is necessary for grouping types of resources.

There are two kinds of resources: not-reusable resources and reusable resources. They are defined by the attribute is_reusable in the relation resource. Not-reusable resources are consumed by summing consumption of each resource. An example is the usage of RAM/ROM. Reusable resources are not consumed in addition. The maximum single consume is assumed instead. For example, the stack on a micro-controller can be a reusable resource, if no preemption is allowed between the tasks.

Beside the application of resources within objective implementations, they are applied as stand-alone constraints. Examples for resources as stand-alone constraints are:

- RAM/ROM consumption
- CPU-load consumption
- Consumption of timers
- I/O-pin consumption
- Circuit board consumption (Components consume space on the ECU's circuit board)
- Cable consumption
- Temperature ranges in the vehicle and temperature critical components

Until now, only non-redundant components are considered. Another interesting use case of resources is redundant functionality. In the future, it might make sense implementing two components that are executing the same function for redundancy and safety purposes. It makes no sense to locate these components on the same ECU due to redundancy. This can be achieved by creating an abstract resource that mirrors functionality. Both components consume one unit of this resource and all relevant ECUs provide it limiting the consume by the components ($\mathsf{min\_units} = 0$; $\mathsf{max\_units} = 1$).

As quality rating, a sum of the severity of constraint violations is determined. For example, a ratio or difference between consumed resource and provided amount is applied. It makes often sense to use several concrete objective implementations for different resource_type. Different resource types mostly also have different units and are therefore not comparable. For example, the units of the resource types *circuit board consumption* and *memory consumption* cannot be compared (m^2 and B). Thus, different objective instances returning separate quality ratings are instantiated.

3.4. Weight

The *weight* of cars is growing in the last years. The weight of the vehicle electronics is a significant portion of the vehicle weight. According to [MH01], electronics cause 5 % or 67 kg of the weight of an average mid range vehicle. Reducing the weight has several positive effects. It directly reduces the fuel consumption of the car. Additionally, the power impression and dynamics of the vehicle is improved.

The weight of the electronics can be reduced by using new materials with the same attributes but lower weight. This is a process independent from and not influenceable by the allocation of functions to ECUs. How can the allocation of functions influence the objective weight?

Two issues can be identified: The weight of the wiring harness and the number of necessary ECUs. The quality rating can be calculated by returning the weight of the resulting wiring harness and the weight of the necessary ECUs.

3.4.1. Weight of the Electronic Control Units

Resources allow modelling the necessary data for the estimation of the weight of ECUs including their allocated components. Each ECU consumes a certain amount of the resource *weight* itself for micro-controller, other electronic components, boxing, and so on. The weight of the boxing is depending on the installation location, for example, since mostly boxings have more weight in wet areas.

The weight can be reduced by finding a subset of ECUs executing all components with a low weight. The weight of a specific ECU has not to be added to the quality rating if

- no component is allocated to the ECU
- and no signal[1] has to be routed by the ECU.

The second criterion ensures that gateway ECUs, which are necessary for routing signals, are considered.

3.4.2. Weight of the Wiring Harness

In addition to the weight of ECUs, the overall weight is influenced by cable lengths resulting from the function allocation. It is influenced by the distance between the ECUs and the according sensors/actuators. This is modelled in the database using resources according to Sect. 3.3. For each cable

[1] For the definition of signal, see Sect. 3.6.

resource, an influence of the objective weight is specified (see Fig. 3.2, consumedunit_influences_objectivetype_relation). In this case, this influence means the weight per meter of the specified cable. For example, in the allocated function direction indication in Fig. 1.3, cables from the front direction indication bulbs to the BPMF and from the rear direction indication bulbs to the BPMR have to be laid.

Routing of cables through the available cable ducts is an optimisation in itself. Methods for automated wiring harness optimisation have been published already several years ago [PCCL94, KS94]. This optimisation is so complex in itself that nowadays, human experts often use virtual reality in order to determine an optimum [SRHR02]. At any time, one of these automated routing algorithms and the according data can be included in order to determine the cable lengths between the ECU and the according sensors/actuators in a more precise way.

The disadvantage of all of these algorithms is that the performance is very poor due to the amount of data and the complexity of the sub-optimisation problem.

3.4.3. Determination of the Cable Lengths

In order to improve the performance of the calculation of the resulting cable lengths and the quality rating depending on the function allocation, a simplification of the real wiring harness is made in this work. Cars at the Volkswagen group normally have a wiring harness in the form of an H (see Fig. 3.4, right). This means, that the main cable ducts go on the left and right side from the rear to the front and that there is exactly one crossing for instance under the dashboard or the rear seats.

The H-form is due to production requirements. A second crossing would not allow assembling the wiring harness into the car in one piece. With this information, a simplified (and fast computable) model of the wiring harness is developed that can be used for the calculation of the quality rating. Figure 3.3 shows a database model for modelling the location of components (component_instance) and ECUs (network_node) and defining the according wiring_harness.

The parameters distance_from_rear d_r, distance between the left and the right part (breadth) d_b and distance_from_bottom d_h define the size and form of the simplified wiring harness. The wiring harness is shown in Fig. 3.4.

A location has to be defined for all components that are sensors and/or actuators and for all network nodes. The location is defined by the three dimensional location denoted in $\{x, y, z\}$. Another parameter (harness_fraction) with the values l, r, or c for left, right, or crossing has to be provided, defining to which of the three cable ducts it is connected. The presented database model allows only one sensors/actuator location per component instance. If more than one

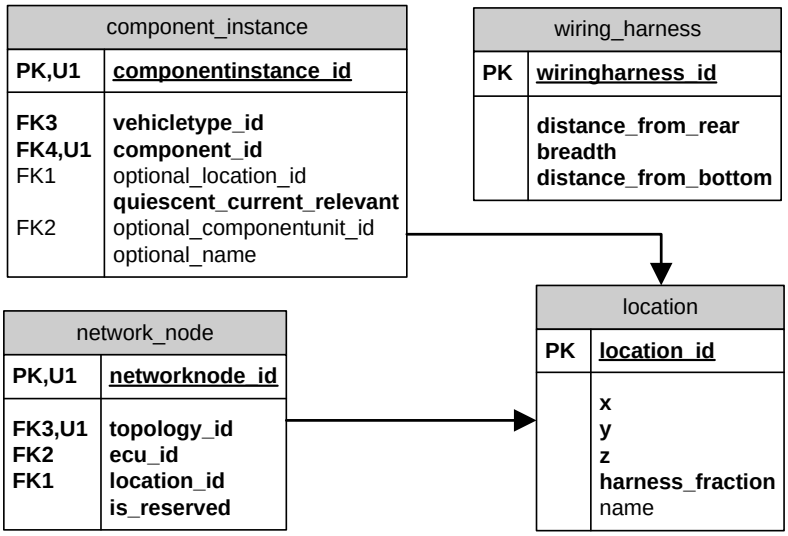

Figure 3.3.: *Database model of locations and the wiring harness*

sensors/actuators are connected to a technical unit, it has to be modelled as separate components. In order to calculate the distance between the location of an actuator and its according allocated network node connecting to it, the following lengths are added:

1. The direct (shortest) way from the 3-dimensional location of the network node to the wiring harness (H)
2. A way from this point to the same point for the actuator location (which is specified with the according component)
3. The direct way from there to the component

If both actuator and network node are on the same connected cable duct and the same harness fraction, there is no need to consider the whole way back to the cable duct. Figure 3.5 shows (2-dimensional) the distance between the component *direction-indication switch* and the ECU BPMR for all distances in steps 1-3 ($l_1, \ldots, l_3$, see example in Fig. 1.3).

3.5. Costs

A very important objective during the development of vehicles is the minimisation of costs. All other objectives influence the profit indirectly as well. In this work, only hardware costs for electronics are considered. Two different costs can directly be influenced by the allocation of components. First, an algorithm

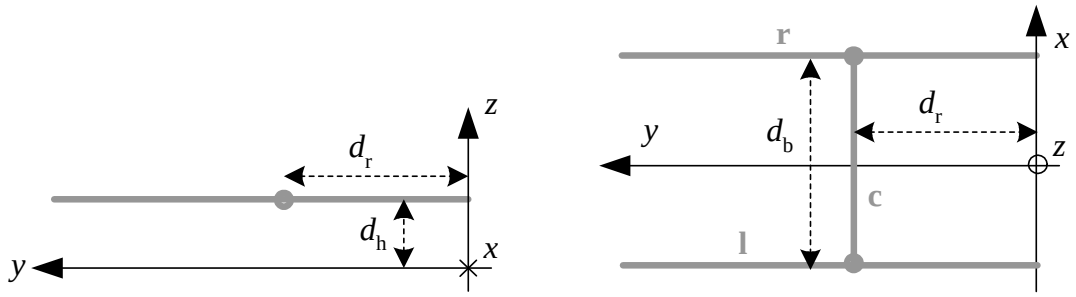

Figure 3.4.: *Wiring harness (grey) in the co-ordinate system from the side (left) and top (right)*

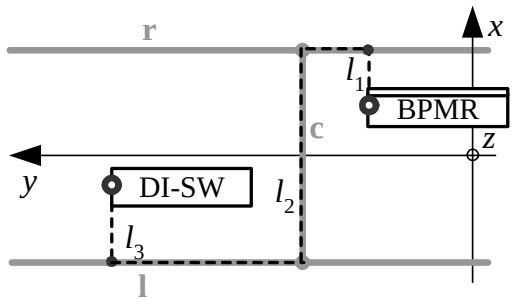

Figure 3.5.: *Distance between component direction indication switch (DI-SW) and BPMR*

for the estimation of the average ECU costs and second, for the determination of the costs of the cables between ECUs and sensors/actuators is explained. In this work, the symbol $\mathfrak{c}$ is used for costs.

3.5.1. Average ECU Costs

The allocation of functions influences the *average ECU costs* $\mathfrak{c}_e$. Some functions are representing extra equipment. If these functions are put on an ECU together with standard equipment, the according ECU has to be built in every vehicle. This is the case although the extra equipment is not ordered. Thus, the extra equipment is still causing costs in this car. In the described case, only the ECU-internal fractions of the components cause *internal component costs* $\mathfrak{c}_c^{\text{int}}$. Sensors/actuators (and their according *external component costs* $\mathfrak{c}_c^{\text{ext}}$) are only as-

sembled in cars with the *order rate* $p_{o,c}$ of the according feature(s). The external fractions are consumed with a different consumption_type.

The average ECU costs $\mathfrak{c}_e$ per car are determined by predicting the *installation rate* $p_{i,e}$ of each ECU. The installation rate equals the ratio of vehicles in which a specific ECU has actually to be installed depending on the allocated components. The resulting average costs caused by an ECU e can be determined with

$$\mathfrak{c}_e = \mathfrak{c}_e^{\text{bas}} p_{i,e} + \sum_{\forall c | c \xrightarrow{\text{asg}} e} \left(\mathfrak{c}_c^{\text{int}} p_{i,e} + \mathfrak{c}_c^{\text{ext}} p_{o,c} \right), \tag{3.1}$$

where $\mathfrak{c}_e^{\text{bas}}$ are the *basic ECU costs* caused by resource consumption (ecu_consumes_resource, see Fig. 3.2) of the ECU itself. The total average ECU costs of a vehicle are determined by summing all average ECU costs $\mathfrak{c}_e$ in the vehicle. ECUs are not considered at all, if no components are assigned and/or no signals have to be routed (see Sect. 3.6).

The prediction of the installation rate of ECUs $p_{i,e}$, depending on the allocated components, is the crucial point in this calculation. It is not sufficient to determine the installation rates of the allocated components. Additionally, the dependencies between the components must be to known. For example, two features with an order rate of 40 % each—both allocated on the same ECU—could imply an installation rate in a range between 40 % (if they are always ordered together) and 80 % (if they cannot be ordered together at all) of the according ECU. In order to express these dependencies, the introduction of a *conditional probability* value could help. This value would have to be provided for each possible combination of components. The number of possible combinations is very high already for a low number of components.

A better way is providing the necessary interdependencies indirectly in form of a list of produced cars. More precisely, it is proposed to use an aggregated prediction of each single manufactured car in the future. Such a prediction can be determined from currently produced cars. Some newly available features are considered in the prediction as well. It is additionally considered that the order rate of some features might differ for future cars. The predictions are aggregated to so-called vehicle_partitions $\mathfrak{v}$, since only features are regarded that have impact on the currently optimised set of components. Figure 3.6 shows the according fraction of the database model.

The relation feature allows defining features. Each feature can be used in several vehicle_types (see relation feature_to_vehicletype_relation). The relation componentinstance_to_feature_relation allows assigning a set of features to each component_instance. If one of these features is *ordered*, the component instance has to be assembled in the car and the allocated ECU is installed. If necessary, the order rate $p_{o,c}$ of the components can be determined from this data as well.

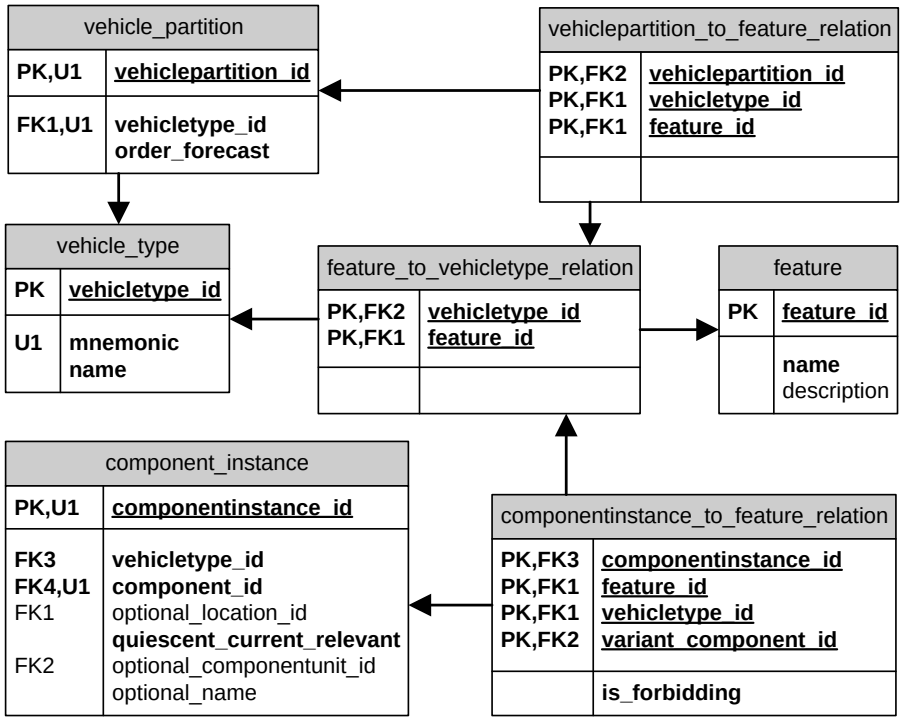

Figure 3.6.: *Database model of predictions of customer orders*

This is shown in an example: All components of the feature *keyless entry* c_{KES} and the components of the feature *rain/light sensor* c_{RLS} are allocated on the same ECU. Table 3.1 shows four different vehicle_partitions $\mathfrak{v}$ with their order_forecasts (number of ordered cars). For instance, 450,000 cars are ordered with the features rain light sensor and without keyless entry. These orders result in an order rate $p_{\mathrm{o},c_{\mathrm{KES}}} = 45\,\%$ and $p_{\mathrm{o},c_{\mathrm{RLS}}} = 70\,\%$ for the considered 1,000,000 cars. The installation rate $p_{\mathrm{i},e}$ of the according ECU can be determined with 90 %, since only 10 % of the cars need none of the two features.

3.5.2. Costs for Cables between ECUs and Sensors/Actuators

The lengths of the cables are determined according to Sect. 3.4.3. Similar to the objective *weight*, for each cable resource, an influence on the objective costs is specified (see Fig. 3.2, consumedunit_influences_objectivetype_relation). In this case, the influence means the *costs* per meter of the specified cable.

This section shows how the average costs of the electronic system can be determined focussed on re-allocable components. Costs of the ECUs themselves and the wiring harness arise depending on the allocation of components. The resulting average costs can be used as quality rating without any transformation.

Table 3.1.: *Example for the determination of order rates $p_{o,c}$ and installation rates $p_{i,e}$*

vehicle partition	order forecast	component	
		c_{KES}	c_{RLS}
$\mathfrak{v}_1$	100,000	☐	☐
$\mathfrak{v}_2$	200,000	☒	☐
$\mathfrak{v}_3$	450,000	☐	☒
$\mathfrak{v}_4$	250,000	☒	☒
	$p_{o,c}$	45 %	70 %
	$p_{i,e} \vert \{c_{KES}, c_{RLS}\} \overset{asg}{\rightarrow} e$	90 %	

In a further step, additional cost saving potential can be realised by creating variants of ECUs. These differ in their hardware. The question, how to configure the variants, is an optimisation on its own and is explained in detail in Chap. 7.

3.6. Busload

A number of years ago, almost no bus systems were installed in vehicles. Due to the rising number of networked functions communicating with each other, a large amount of signals has to be exchanged. For each signal, a discrete signal cable was used in the past. Many signals can be transmitted over single cables using bus systems.

This section explains the objective *busload*. A modern electronic architecture consists of several different networks. This is because different functions from domains like infotainment, body electronics, or power train have different requirements in relation to communication safety and amount of data. Figure 3.7 shows a database model allowing modelling this so-called network topology. Each network_node can be connected to several networks and different network_types. The networks can be configured by setting the flag is_time_triggered, baud_rate and protocol_overhead factor. How to configure these parameters for different networks is explained in the course of this section.

In order to exchange the function relevant information, signal transmission takes place between components. Components are reusable in different vehicle type series. Therefore, signals are not statically bound to components. In fact, signals are bound to interfaces[2]. Figure 3.8 shows a partition of the database

[2]It is distinguished between producer_interfaces (Output) and consumer_interfaces (Input)

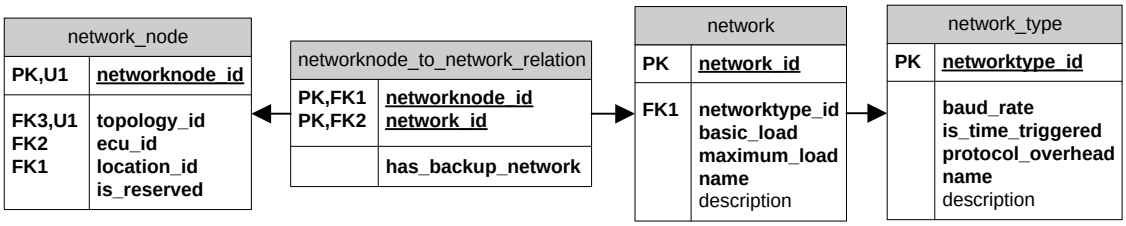

Figure 3.7.: *Database model of the network topology*

model used for modelling the functional network. It allows to model component_instances. Therefore, components only have signal interfaces for input and output communication. Each interface is defined by the interface_type. The communication between components can only be modelled between concrete component instances. The interface types of the two communication partners have to be identically. Each communication between component instances—called *signal*—always has exactly one sender and an arbitrary number of receivers.

The estimation of the busload of a network considers signal communication between component instances that are not located on the same ECUs. In the presented model, signals are modelled independently from the used bus system. The common about CAN, LIN, FlexRay and MOST is the fact that signals are specified by a bit-size. To calculate a representation of the busload, additional parameters are necessary.

Functions have different requirements on signal transmission. For each producedsignal_to_consumer_relation, values for maximum_delivery_time t_d and minimum_update_time t_u are specified. It is shown how to determine if the maximum delivery times are kept in Sect. 3.6.1. Section 3.6.2 shows how the best networks for routing signals to the receivers is determined. The influence of signals on the busload is explained in Sect. 3.6.3. The maximum delivery times are also important for the determination of the resulting busload, especially for time-triggered networks. Finally, in Sect. 3.6.4 the determination of a quality rating is presented.

3.6.1. Maximum Delivery Time of a Signal

The worst-case delivery time of a signal must be lower than the specified maximum_delivery_time t_d in order to ensure the specified function. In relation to the function allocation, only the network relevant delivery times have to be considered. The time from sensor to micro-controller to the network controller cannot be influenced by the allocation. In this section, only times are incorporated that are maximum reachable if the according bus systems are configured ideally.

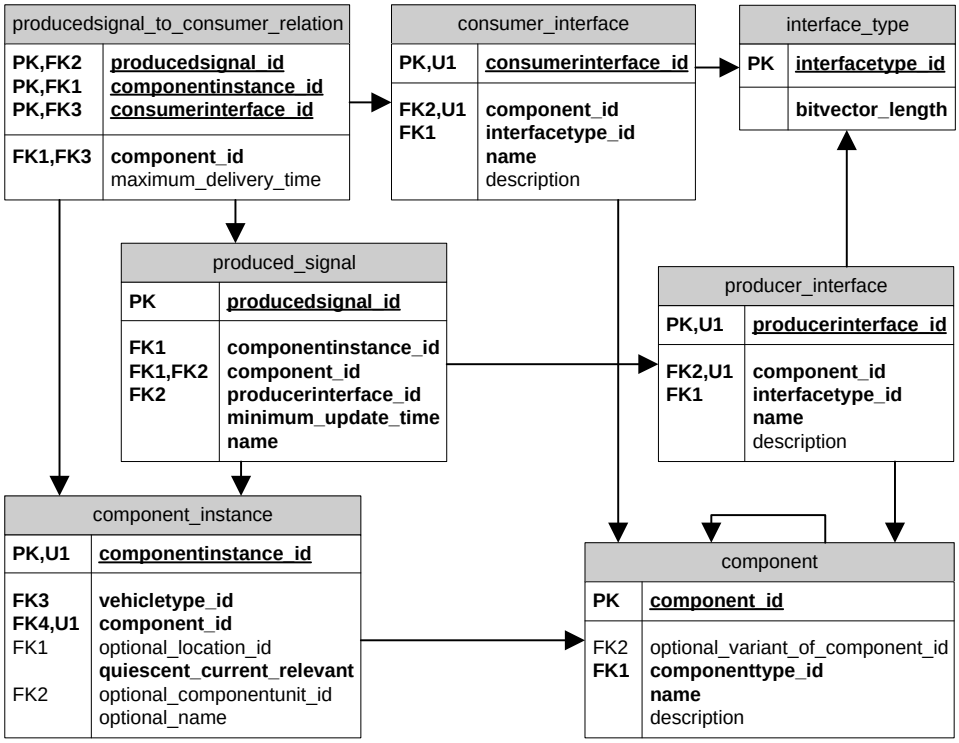

Figure 3.8.: *Database model of components, component instances, interfaces, and signals*

With the *maximum transfer rate* (baud_rate) r, the *maximum transmission delay* $t_{\mathrm{D},N}$ can be calculated for one *network* N with

$$t_{\mathrm{D},N} = t_{\mathrm{prop}} + t_{\mathrm{trans}} + t_{\mathrm{prio}}. \tag{3.2}$$

The variables are defined as follows:

- The *propagation time* t_{prop} is the time needed for the propagation of signals from the application on sender side to the application on receiver side. Times for network controller t_{contr}, transceiver t_{tc} and network route itself t_{net} have to be summed up. The wave transmission speed in cable is about $2 \cdot 10^8 \frac{\mathrm{m}}{\mathrm{s}}$. For 100 m cable this results in $t_{\mathrm{net}} \approx 0.5\,\mathrm{ns}$ and can be neglected here. On high speed CAN, common values are $t_{\mathrm{contr}} \approx 250\,\mathrm{ns}$ and $t_{\mathrm{tc}} \approx 50\,\mathrm{ns}$. Similarly, for low speed CAN, t_{prop} is only in the range of ¯s. All networks have to be defined so that t_{prop} is somehow smaller than the time amount needed for the transmission of one bit. Thus, for all networks by definition, t_{prop} does not affect $t_{\mathrm{D},N}$ significantly, since it is much smaller than t_{trans}.

- The *transmission time* t_{trans} is the time needed for the transmission of the whole signal from the sender to the receiver. It is depending on the protocol overhead and can depend on the *length of the signal* b_{sig} (see Fig. 3.8, bitvector_length).

- The *prioritisation time* t_{prio} is the time until the transmission of the message can be started.

For the calculation of the maximum transmission delays, some values have to be rounded up. The symbol $\overset{rb}{\hat{a}}$ is defined as rounding up its argument a to the next multiple of the *rounding base* rb. For example $\overset{8}{\hat{12}} = 16$.

The delay times $t_{D,N}$ for CAN, FlexRay, MOST and LIN can be calculated as follows:[3]

- For CAN, t_{trans} can be calculated with frame length divided by the maximum transfer rate r. According to [Lawoo], there exist two different versions of CAN with 11- and 29-bit identifiers. The *frame length without data* b_{f} is 47 bit and 67 bit, respectively. Additionally, stuff bits have to be considered. According to [NHN03], the worst-case *number of stuff bits* b_{s} for 11- and 29-bit identifier is

$$b_{\text{s}} = \frac{\{34, 54\}\,\text{bit} + \overset{8}{\hat{b}}_{\text{sig}} - 1\,\text{bit}}{4}, \tag{3.3}$$

  where b_{sig} is the length of the signal. The worst-case *protocol overhead* $b_{\text{oh}} = b_{\text{f}} + b_{\text{s}}$ can be calculated with $47\,\text{bit} + 25\,\text{bit} = 72\,\text{bit}$ and $67\,\text{bit} + 30\,\text{bit} = 97\,\text{bit}$ for 11- and 29-bit identifier. The resulting t_{trans} can be calculated with

$$t_{\text{trans}} = \frac{b_{\text{oh}} + \overset{8}{\hat{b}}_{\text{sig}}}{r}. \tag{3.4}$$

  Important CAN messages can be sent with a high priority. Even for the CAN message with the highest priority, a message with a lower priority will finish sending. The maximum frame length is 64 bit. Thus, $t_{\text{prio}} \geq \frac{b_{\text{oh}} + 64\,\text{bit}}{r}$. Concluding, $t_{\text{D,CAN}}$ can be calculated with

$$t_{\text{D,CAN}-11} \approx t_{\text{trans}} + t_{\text{prio}} < \frac{72\,\text{bit} + \overset{8}{\hat{b}}_{\text{sig}}}{r} + \frac{136\,\text{bit}}{r} = \frac{208\,\text{bit} + \overset{8}{\hat{b}}_{\text{sig}}}{r} \tag{3.5}$$

[3] t_{prop} can be neglected and is not discussed anymore.

and

$$t_{\mathrm{D,CAN-29}} \approx t_{\mathrm{trans}} + t_{\mathrm{prio}} < \frac{97\,\mathrm{bit} + \overset{8}{\hat{b}}_{\mathrm{sig}}}{r} + \frac{161\,\mathrm{bit}}{r} = \frac{258\,\mathrm{bit} + \overset{8}{\hat{b}}_{\mathrm{sig}}}{r}. \quad (3.6)$$

These values can only be guaranteed for the message with the highest priority. Thus, CAN is not real time capable.

- According to [LIN03], LIN (2.0) has a protocol header of 34 bit. In the response space, additional 10 bit have to be considered for the checksum. The length of the carried data can only be specified Byte per Byte with a maximum of 8 Byte. At the beginning and end of each Byte, a start and stop bit has to be transmitted. The frequency of the oscillator has low requirements on accuracy. In comparison to the nominal time, this results in 40 % additional time for the transmission of the whole signal. The value t_{trans} can be calculated with

$$t_{\mathrm{trans}} = 1.4\frac{44\,\mathrm{bit} + 2(\overset{8}{\hat{b}}_{\mathrm{sig}}\ /8) + \overset{8}{\hat{b}}_{\mathrm{sig}}}{r}. \quad (3.7)$$

For time-triggered networks, the prioritisation time t_{prio} is smaller or equal to the cycle time t_{cyc}[4] of the frame where they are included. With $44\,\mathrm{bit} + 2(\overset{8}{\hat{b}}_{\mathrm{sig}}\ /8) < 60\,\mathrm{bit}$, $t_{\mathrm{D,LIN}}$ can be calculated with

$$t_{\mathrm{D,LIN}} \approx t_{\mathrm{trans}} + t_{\mathrm{prio}} < 1.4\frac{60\,\mathrm{bit} + \overset{8}{\hat{b}}_{\mathrm{sig}}}{r} + t_{\mathrm{cyc}}. \quad (3.8)$$

For t_{cyc}, it is not necessary to consider the factor 1.4, since the master, which is triggering all signal transmissions, has tighter oscillator accuracy requirements.

- On MOST, the control channel is used for data exchange comparable to the other networks. According to [MOS05], for a common value of the *sample frequency* f_{s} of $f_{\mathrm{s}} = 44.1\,\mathrm{kHz}$, 2670 control messages per second can be transmitted. One frame has 512 bit and includes 2 Byte of one control messages. A complete control message has a length of 32 Byte. So, $t_{\mathrm{trans}} = \frac{16 \cdot 512\,\mathrm{bit}}{r}$ with $r = f_{\mathrm{s}} \cdot 512\,\mathrm{bit}$[5]. Only every third control message can be received by a MOST controller. Therefore, even for the message

[4] It is explained in Sect. 3.6.3 how to estimate t_{cyc}.

[5] This value $r = 44.1\,\mathrm{kHz} \cdot 512\,\mathrm{bit} = 22580\,\mathrm{kbit/s}$ is neglecting inter-message delays and stuff bits, since they are not defined in [MOS05]. The physical value is about $r = 24.8\,\mathrm{Mbit/s}$.

Table 3.2.: *Comparison of the transmission time relevant parameters of different in-car networks for an 8 bit signal and optimal design*

network	r $\frac{\text{kbit}}{\text{s}}$	t_{trans} ms	t_{prio} ms	cap. of real time	$t_{\text{D},N}$ ms
Low-Speed CAN, 11-bit-id	100	0.8	1.36	□	2.16
High-Speed CAN, 29-bit-id	500	0.21	0.322	□	0.532
LIN	19.2	4.96	10	⊠	14.96
MOST	22580	0.363	1.088	□	1.451
FlexRay	10000	0.0326	10	⊠	10.0326

with the highest priority, in the worst case, another three messages have to be awaited. Concluding, $t_{\text{D,MOST}}$ can be calculated with

$$t_{\text{D,MOST}} \approx t_{\text{trans}} + t_{\text{prio}} = t_{\text{trans}} + 3\,t_{\text{trans}} = 4\frac{16 \cdot 512\,\text{bit}}{r}. \tag{3.9}$$

- According to [Fle05], the overhead of the FlexRay protocol is $b_{\text{oh}} = 86\,\text{bit}$ at best. Normally, the length of slots in the static segment is fixed. A commonly used value is 24 Byte. The dynamic segment is not considered since signals that have a bounded delivery time should always be transmitted via the static segment. Each transmitted Byte is started with a Byte-start-sequence of 2 Byte. This results in a minimum of $t_{\text{trans}} \leq \frac{86\,\text{bit}+24\cdot 10\,\text{bit}}{r}$. For FlexRay, the netto transfer rate r is used since the value is highly dependent on the bus configuration of the bus. In reality, a value of $r = 10\,\text{Mbit/s}$ can be reached when installing two channels. Similarly to LIN, t_{prio} is depending on the cycle time of the according frame. Concluding, $t_{\text{D,FR}}$ can be calculated with

$$t_{\text{D,FR}} \approx t_{\text{trans}} + t_{\text{prio}} \leq \frac{86\,\text{bit} + 24 \cdot 10\,\text{bit}}{r} + t_{\text{cyc}}. \tag{3.10}$$

Table 3.2 shows the maximum delay times for an 8-bit signal if the networks are designed for fast transmission of critical messages. For time-triggered networks, a cycle time of $t_{\text{cyc}} = 10\,\text{ms}$ is used.

In order to check whether all delay times comply with the constraints (all maximum delivery times t_{d}), all maximum transmission delays t_{D} on the whole way through the network have to be summed up and compared with the constraints. At each gateway, an additional *processing time* t_{proc} is needed for transmitting the

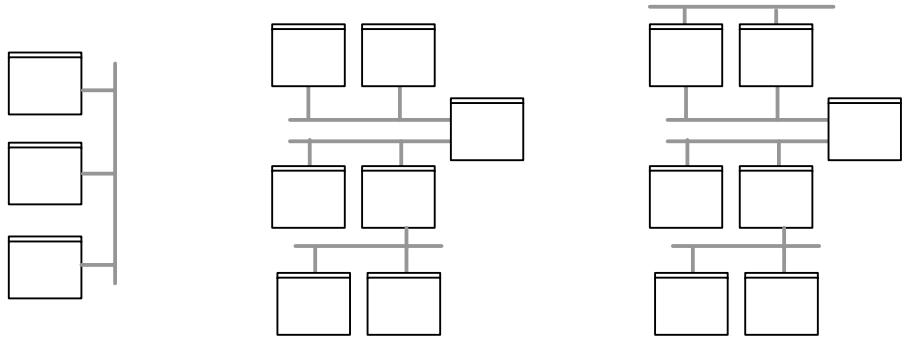

Figure 3.9.: *Examples for network topologies with one network (left), topologies in tree form (mid), and graph form (right)*

signal to the other network. This time is consumed for looking into the routing table, finding the correct target(s), and copying the needed data into the send buffer. A connection between two FlexRay networks is a special case, since it is possible to synchronise two different FlexRay networks. The resulting delay $t_{\mathrm{D,FR,n}}$ for n succeeding FlexRay networks is

$$t_{\mathrm{D,FR},n} \approx t_{\mathrm{trans}} + t_{\mathrm{prio}} \leq \frac{86\,\mathrm{bit} + 24 \cdot 10\,\mathrm{bit}}{r} + t_{\mathrm{cyc,LCM}} + (n-1)t_{\mathrm{proc}}, \qquad (3.11)$$

where $t_{\mathrm{cyc,LCM}}$ is the *Lowest Common Multiple* (LCM) of the cycle times of the submitting frames on the transmission path.

3.6.2. Allocation of Signals to Networks

As a next step, for each network, the set of signals $\mathbf{sig}_{N,\mathrm{tr}}$ has to be determined that have to be transported. If only one network is considered, this can be done as follows: If any of the receivers of a signal is not allocated on the same ECU as the sender, the signal has to be transmitted over the network.

Another step of difficulty is a network in the form of a tree. This is the most common case for current vehicle architectures in the Volkswagen group. A tree means that there is only one possible way for the signals from the senders to the receivers. For each sender-receiver-relation the networks can be determined in this way.

There exist network topologies, which are not in a tree but in a graph form. Figure 3.9 shows examples of the three types of networks. Different ways to route signals from a sender to a receiver can be thought of. In this work, the *Shortest Path First* (SPF, [Dij59]) algorithm is applied in order to determine the

necessary networks for all signals. This algorithm solves the single-source shortest path problem for a directed graph with non-negative *edge* weights. Therefore, each *vertex* (here equivalent to network nodes) in the graph keeps a list of the shortest known routes to all other vertices. This list is initialised with the direct neighbours and updated iterative until the shortest path from the starting point to the ending point is known. Since a tree is a special case of a graph, this algorithm is also applicable for networks in the form of a tree.

3.6.3. Influence of Signals on the Busload

There are different influences of signals on the busload. As explained at the beginning of this section, two parameters are known for signals. For each produced_signal_to_consumer_relation, maximum_delivery_time t_d and minimum_update_time t_u can be specified.

If the update time t_u is quite small in comparison to the maximum delivery time t_d, the signal has to be transmitted always after a time of t_u, independent from the network ($t_\mathrm{cyc} = t_\mathrm{u}$). If the maximum delivery time t_D is very small, it has to be differentiated between time-triggered and non-time-triggered networks, since the influence of a signal on the busload is different for these two cases:

- For *non-time-triggered networks* (n-tt-N), the maximum transmission time cannot be influenced by adjusting t_cyc (see (3.6)). A signal with a large t_u is potentially transmitted very rarely, almost not influencing the busload although the maximum delivery time is quite small.

- This is different for *time-triggered networks* (tt-N). The value t_cyc directly influences t_d (see (3.10)). It is necessary that

$$\sum_{\substack{\forall N \\ \in \text{route}}} t_\mathrm{cyc} < t_\mathrm{d} - \sum_{\substack{\forall \text{gateways} \\ \in \text{route}}} t_\mathrm{proc} - \sum_{\substack{\forall \text{n-tt-N} \\ \in \text{route}}} t_{\mathrm{D},N} - \sum_{\substack{\forall \text{tt-N} \\ \in \text{route}}} t_\mathrm{trans} \tag{3.12}$$

  in order to fulfil the timing constraints. The sum of the cycle times t_cyc of all networks on the route has to be smaller than the maximum delivery time t_d decreased by all processing times t_proc of the gateways on the route, by the maximum transmission delays $t_{\mathrm{D},N}$ on the route, and by all transmission times t_trans. It is proposed to split up the cycle times on the different time-triggered networks on the route proportionally to the transmission rate r. If one signal has several receivers with different t_d, this can sometimes be used to relax the timing requirements for some networks.

In difference to the calculation of the worst-case delay times, here the signals are considered to be mapped to message frames in a way optimal for low busload consumption. This means, for each signal it is assumed that other signals can be found, forming a message with a maximum possible data length in order to save proportional overhead. Overhead is only considered proportionally for each transmitted signal. Thus, for each network N, the *net busload* l_N can be estimated with

$$l_N = \frac{1}{r} \sum_{\forall \text{sig} \in \mathbf{sig}_{N,\text{tr}}} \frac{p_{\text{oh}} b_{\text{sig}}}{t_{\text{cyc}}}. \tag{3.13}$$

The *protocol overhead factor* p_{oh} is the ratio of a gross message in comparison to the length of the user data. It is calculated in the case of optimal design. That means, that for example for CAN it is assumed that each message has 64 bit of user data in this optimal case.

For the different networks, the *protocol overhead factor* can be calculated with:

- CAN has a worst-case protocol overhead of $b_{\text{oh}} = 72/97$ bit for networks with 11/29-bit identifiers. The according p_{oh}'s can be calculated with $p_{\text{oh,CAN}-11} = \frac{72+64}{64} = 2.125$ and $p_{\text{oh,CAN}-29} = \frac{97+64}{64} \approx 2.52$.
- LIN has an overhead of $b_{\text{oh}} = 44$ bit. Additionally, a factor of 1.4 (see Sect. 3.6.1) has to be considered. Totally, the protocol overhead factor can be calculated with $p_{\text{oh,LIN}} = 1.4\frac{44+64}{64} \approx 2.35$.
- MOST uses the control channel for signal transmission. A number of 16 bit of the 512 bit of a message are used for transmission of control messages. According to [MOS05], only 62 out of 64 control messages can be used for user data. Each control message has an overhead of 14 Byte in comparison to 18 Byte user data. Concluding, $p_{\text{oh,MOST}} = \frac{512}{16}\frac{64}{62}\frac{32}{18} \approx 58.72$.
- FlexRay has an overhead of $b_{\text{oh}} = 86$ bit. The maximum frame length is 255 Byte. Often, the maximum frame length is not used. A common value is 24 Byte. With that value, the protocol overhead factor can be calculated with $p_{\text{oh,FR}} = \frac{86+24\cdot10}{24\cdot10} \approx 1.36$.

Table 3.3 compares the net busload influence of an 8 bit signal transmitted over exactly one network without regarding gateways.

3.6.4. Estimation of a Quality Rating for Busload

This section explains how to calculate a quality rating for busload q_{busload}. Architectures claim to ensure extensibility for future features not known at design

Table 3.3.: *Comparison of the net busload influence of an 8 bit signal with* $t_\mathrm{u} = 25\,\mathrm{ms}$ *and* $t_\mathrm{d} = 20\,\mathrm{ms}$ *transmitted only over one network*

network	r $\frac{\mathrm{kbit}}{\mathrm{s}}$	t_trans ms	t_cyc ms	l_N %
Low-Speed CAN, 11-bit-id	100	0.8	25	0.68
High-Speed CAN, 29-bit-id	500	0.21	25	0.16128
LIN	19.2	4.96	20.04	4.88
MOST	22580	0.363	25	0.0832
FlexRay	10000	0.0326	24.9674	0.00436

time. Therefore, the real maximum value for the busload l_N^max (maximum_load) is much lower than 100 %. Additionally, for non-time-triggered networks some bandwidth has to be reserved for safety purposes and in order to allow time critical messages getting access to the bus relatively fast. A basic_load is caused by signals that are not modelled. It can be set with the parameter l_N^bas.

One way to handle the busload on different networks is having separate objective instances for each. The returned quality rating equals the busload on each network. This could add quite a number of objectives to the optimisation problem depending on the number of networks. In order to improve the performance, an aggregation model is developed. An aggregated quality rating has the following requirements:

- If the busload l_N is far away from the maximum busload $l_N^\mathrm{max} - l_N^\mathrm{bas}$, additional busload should be weighted weaker.
- Near to the maximum busload, the weight should be higher, since it can be considered more important.
- For the busload $l_N = l_N^\mathrm{max} - l_N^\mathrm{bas}$, the value $q_{N,\mathrm{busload}}$ shall exactly equal 1. If this is known, the number of constraint violations can be added to the quality rating. This guarantees that a quality rating for a constraint violating solution is always worse than for a valid one.
- No weighting parameter for each network N is necessary.

In order to fulfil these requirements, it is proposed to calculate a quality rating for all aggregated networks N with

$$q_\mathrm{busload} = \sum_{\forall N} q_{N,\mathrm{busload}} + \text{number of constraint violations} \tag{3.14}$$

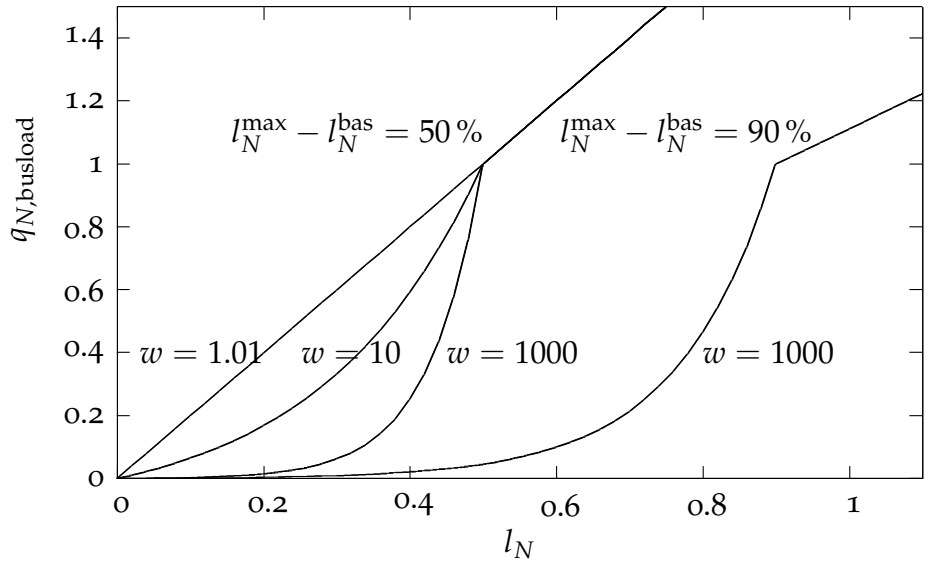

Figure 3.10.: $q_{N,\text{busload}}$ *for different values of* w *and* $l_N^{\max} - l_N^{\text{bas}}$

and

$$q_{N,\text{busload}} = \begin{cases} \dfrac{w^{\left(\frac{l_n}{l_N^{\max} - l_N^{\text{bas}}}\right)} - 1}{w - 1} & \text{if } l_N < l_N^{\max} - l_N^{\text{bas}} \\ \dfrac{l_N}{l_N^{\max} - l_N^{\text{bas}}} & \text{else} \end{cases}, \tag{3.15}$$

with $w =]1, \infty[$ for globally adjusting the relaxation of the busload quality rating for low busloads. With $w \gtrsim 1$, $q_{N,\text{busload}}$ represents the busload in comparison to the constraint value ($l_N < l_N^{\max} - l_N^{\text{bas}}$). For bigger values of w, low busload values are considered less important. Figure 3.10 shows $q_{N,\text{busload}}$ for different values of w and $l_N^{\max} - l_N^{\text{bas}}$.

The compliance with the maximum delivery times should be considered in a different objective. All violations of the timing constraint could for example just be summed up.

3.7. Electrical Energy Consumption

This section deals with the allocation objective *electrical energy consumption*. An explanation is given on how to estimate the allocation quality according to energy consumption in different states of the vehicle.

The wish to reduce the energy consumption of a car has several reasons. On the one hand, the size of the generator can be reduced if less power is consumed.

On the other hand, the size of the battery can be reduced. In both cases, costs as well as weight can be reduced. Another point is that operating a generator increases the fuel consumption of the car, since it consumes power from the engine.

Since all networks and ECUs are normally active if the engine is running, the power consumption of the functions in this use case cannot be influenced so much by the allocation of components on ECUs. In order to reduce this power consumption, the function itself has to be reduced in its power consumption.

However, if the engine is off and especially if the ignition is off as well, the energy consumption is highly dependent on the allocation. This energy consumption is very critical, since the capacity of the battery and the energy consumption of the whole vehicle limits the time of availability of the functions. For comfort functions like radio, a limited time of availability might be acceptable. For safety relevant functions like *parking light* or *hazard lights*, it is not.

In the state *ignition off and no backlashes active*, the power consumption is highly dependent on the allocation. Several functions that have to be active all the time can be grouped together on single ECUs in order to reduce power consumption. For example, features like *radio remote controlled central locking* or *anti-theft protection* require sensor components to be active all the time. Some functions are activated by the user, like for example *hazard lights*. They often require some of the networks to be active.[6] This causes a high amount of energy consumption, since the network transceivers of all other ECUs have to be active at the same time. An active network transceiver often means that the whole ECU is active and consuming power. This can be avoided by adding so-called backup networks to ECUs enabling an active network transceiver without keeping the whole ECU awake.

3.7.1. Consideration of Number of Active ECUs

In the early development phase, the specification of the electronic system is quite vague. In this phase, a quite easy calculable possibility for estimating the allocation quality is to count the number of ECUs with at least one function that has to be active all the time. The disadvantage is that different use cases like warning light cannot be considered at all with this approach. In addition, the quality of an allocation can only be estimated very inaccurate.

[6] The front and rear hazard lights at least have to be synchronised from time to time.

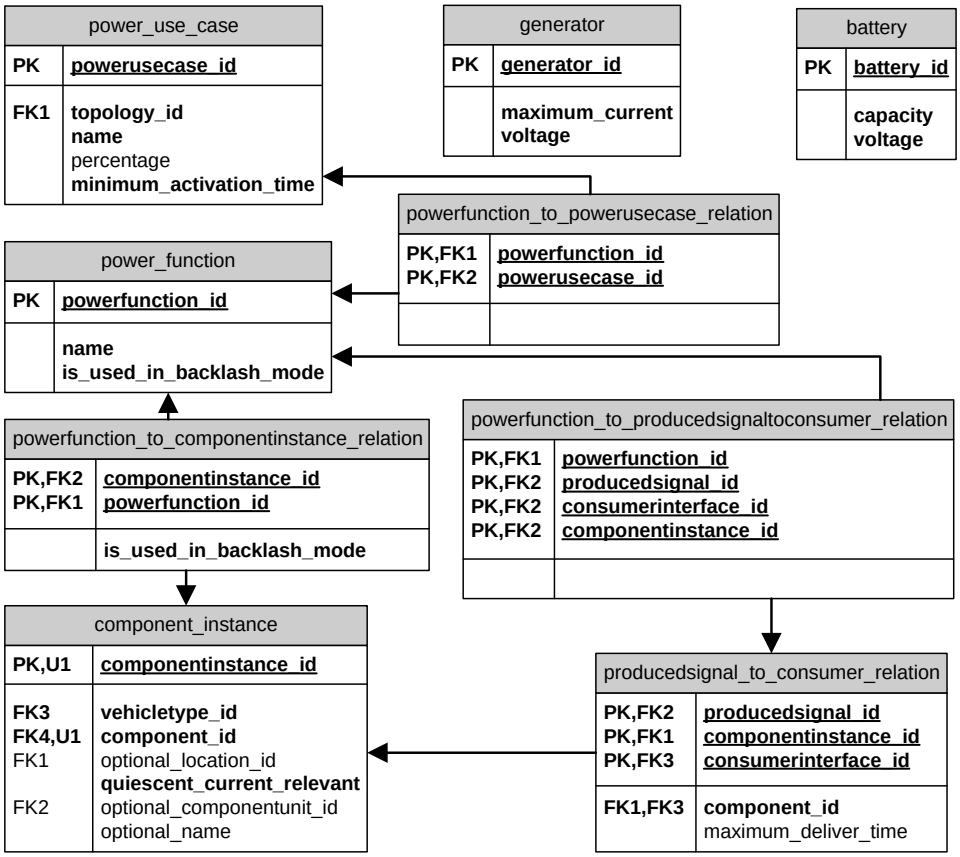

Figure 3.11.: *Database model of power functions*

3.7.2. Consideration of Different Use Cases

A more precise way for the prediction of power consumption is to estimate different use_cases. It is proposed to define a set of so-called *power relevant functions* (power_function). Again, a database model is provided in Fig. 3.11. Each power relevant function is defined by a set of components and a set of signals that have to be transmitted during execution of the function. To go back to the example *direction indication* (see Fig. 1.2 and Fig. 1.3), besides the components, also the signal used for synchronisation belongs to the power relevant function hazard lights.

Besides different intrinsic functions, additionally, *ignition on*, *engine on*, and *backlash on* must be defined as power relevant functions. For these, all according components and signals have to be modelled as well. Backlash is the mode, where most functions are already deactivated, but some ECUs and functions are still awake until they are finally shut down.

It is important to note that some functions keep whole networks awake if they are activated. For example, for the hazard lights a network might be awake in

order to synchronise the frequency and offset of the bulbs of front and rear. More exact, the route from the sender of a signal to the specific receiver has to be defined as belonging to a function. If such a signal is required by a power relevant function, all networks on its route have to be active if the according function is active. If gateways do not know the use case, the vehicle is currently in, all routes of the signal to the other receivers' networks might be activated as well.

A further layer of abstraction is the definition of power_use_cases. Each *power use case* $\mathfrak{U}$ is defined by a user-defined set of power relevant functions. There are an innumerable number of different use cases for a vehicle. Some of the most important ones are:

- Ignition off – backlash off
- Ignition off – backlash on
- Ignition off – warning lights on
- Ignition off – radio on
- Ignition off – radio on – warning lights on
- Ignition on – motor off
- Ignition on – motor on

If the information about power relevant functions is available, it is very easy to define new interesting combinations of them as new use cases.

The concrete consumption of power can be covered by the concept of resources as explained in Sect. 3.3. It is distinguished between three types of power consumption (consumption_type, see Fig. 3.2):

- *Operation power consumption*, if at least one function or the ECU itself is active
- *Special power consumption*, consumed for communication by the ECU if no *backup network*[7] implemented or in the *backlash mode* by the components
- *Idle power consumption*, always consumed, particularly important in the use case *ignition off – backlash off*

[7] A backup network allows enabling the transceiver without activating the micro-controller.

Table 3.4.: *Power consumption $p_{\mathfrak{U}}$ of several use cases $\mathfrak{U}$ for an ECU e_1 and one component c_1*

$\mathfrak{U}$	$p_{\mathfrak{U}}$ power equivalent in mA				
all off	0.1	+	0.05	=	0.15
only network active (e_1 has backup network)	10	+	0.05	=	10.05
only network active (e_1 has no backup network)	100	+	0.05	=	100.05
c_1 in backlash mode	100	+	5	=	15
c_1 active	100	+	50	=	150

All three kinds of power consumption can be consumed by ECUs as well as by components. *Idle power consumption* is the amount of power that is always consumed, even if no function is activated at all. Nowadays, most ECUs are connected directly to the battery and not switched by ignition. Thus, they are consuming a little amount of idle power as well.

The special power consumption is different for ECUs and components. ECUs consume the special power if the network has to be active and they have a backup network. Components consume special power if they are in the backlash mode. In this mode, the function can already be deactivated, but there are still some tasks to complete. Therefore, only the reduced special power is consumed.

If at least one component on an ECU is in backlash mode or active, the whole ECU has to be activated as well. Additionally, the ECU has to be activated if the according network is necessary for transmission of signals and if no backup network is available. Components only consume operation power if they are activated for the specific use case.

For each use case, an exact value for *power consumption* $p_{\mathfrak{U}}$ can be determined in the described way. Table 3.4 shows an example with an ECU e_1 and one component c_1 with an operation power consumption of 100 mA and 50 mA, a special power consumption of 10 mA and 5 mA, and an idle power consumption of 0.1 mA and 0.05 mA. For several use cases $\mathfrak{U}$, the according power consumption $p_{\mathfrak{U}}$ is determined in the right column.

For each use case, a single objective instance can be used, similarly to the objective busload. This would lead to a high number of objectives. A further possibility is weighting the objectives with a certain percentage

Additionally, constraints are involved: For each use case, a *maximum power consumption* $p_{\mathfrak{U}}^{\max}$ can be defined. The maximum power consumption can be

determined with

$$p_{\mathfrak{U}}^{\max} = \frac{C^{\text{bat}} U^{\text{bat}}}{T_{\mathfrak{U}}}, \tag{3.16}$$

where C^{bat} is the net capacity of the battery, U^{bat} is the voltage and $T_{\mathfrak{U}}$ (minimum_activation_time) is the required time of availability for a specific use case $\mathfrak{U}$. The maximum power consumption of all use cases is defined by the maximum_current $I^{\max}$ and the voltage U^{gen} of the generator

$$p_{\mathfrak{U}} < p_{\mathfrak{U}}^{\max} = U^{\text{gen}} I^{\max}. \tag{3.17}$$

If constraints are involved, for practical application, a method similar to Sect. 3.6.4 can be used:

$$q_{\text{energy}} = \sum_{\forall \mathfrak{U}} q_{\mathfrak{U},\text{energy}} + \text{number of constraint violations} \tag{3.18}$$

and

$$q_{\mathfrak{U},\text{energy}} = \begin{cases} \frac{w^{\frac{p_{\mathfrak{U}}}{p_{\mathfrak{U}}^{\max}}} - 1}{w - 1} & \text{if } p_{\mathfrak{U}} < p_{\mathfrak{U}}^{\max} \\ \frac{p_{\mathfrak{U}}}{p_{\mathfrak{U}}^{\max}} & \text{else} \end{cases} \tag{3.19}$$

Thus, the optimisation tends to allocate functions that need network communication with low busload requirements although ignition is off onto same networks. Additionally, it tends to move functions that are in communication with each other on the same network nodes in order to safe power consumption in the idle state.

3.8. Supplier Complexity

This section shows how a quality rating q_{supplier} for the so-called *supplier complexity* can be determined. Supplier complexity is a measure for the number of suppliers participating at the development of single ECUs.

The goal of *Original Equipment Manufacturers* (OEM) is to reduce the number of suppliers developing and supplying an ECU because

- the coordination work between OEMs and suppliers is higher,
- the coordination work between the suppliers is higher,
- beside the interfaces of the ECU, additional interfaces within the ECU between the components have to be specified in detail, and

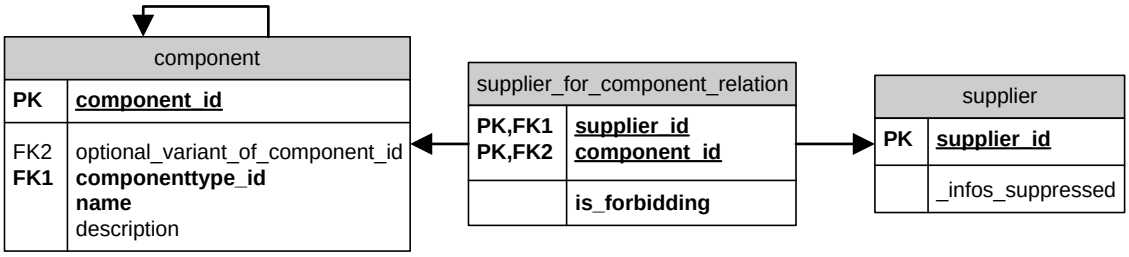

Figure 3.12.: *Database model of suppliers for a component*

- business models have to be developed ensuring the responsibility for errors during the development.

Why not contract a single supplier for each ECU? Not all innovations are made by the OEMs themselves. Suppliers often develop new functions on their own or together with OEMs. This is the reason that often there is only a limited number of suppliers being able to develop and manufacture a specific component.

If several components are allocated on one ECU, it can happen that there exists no single supplier being able to develop and produce an ECU with all these components. The quality rating for this objective has to return a worse quality rating in these cases.

In order to estimate a quality rating, for each component, a set of suppliers has to be defined that have the ability to supply an ECU including this component. The according database model is shown in Fig. 3.12.

From a mathematical point of view, the problem is to find a minimal set of suppliers that are able to develop and produce the ECU or at least to find the number of suppliers. For ECUs with a large number of components, this might be quite difficult.

This problem is similar to the so-called *Set-Covering Problem* (SCP). As input for a set-covering problem, several sets are given. The elements in these sets can occur multiple times in the different sets. The optimum solution for the set-covering problem is a minimum number of these sets so that each element is covered. In this section, a number of suppliers is given for each component and a minimum number of suppliers has to be found so that the according ECU can be developed and produced. Each supplier can supply a subset of all components on an ECU and the task is to find the minimum number of the suppliers so that all components are covered. Table 3.5 shows an example with four components and five suppliers. The minimum number of suppliers in this example is two—possible solutions among others are for example $\{B, E\}$ or $\{C, D\}$.

Table 3.5.: *Example for a set of four components (c_1,c_2,c_3,c_4) and the according suppliers (A,B,C,D,E)*

supplier	c_1	c_2	c_3	c_4
A	☒	☐	☒	☐
B	☒	☒	☐	☐
C	☒	☒	☒	☐
D	☐	☒	☐	☒
E	☐	☐	☒	☒

The first step in order to solve the problem efficiently is reducing the problem size. Reduction procedures that allow removing redundant rows and columns can be found in [CFT98]. One example for reduction rules is the removal of rows and columns that are subsets of each other. Columns with only one element can be removed while adding the according supplier to the solution.

If these reduction procedures do not solve the problem yet, *branch & bound* algorithms can be used. Branch & bound algorithms put one supplier to the used supplier set as one *branch*. The other branch is that the supplier is never in the supplier set anymore. As soon as a valid solution has been found, this is the new *upper bound* (the currently smallest number of suppliers). The *lower bound* is the sum of the number of suppliers necessary already and an estimation of a minimum number of suppliers necessary for the remaining components. No lower bounds that have a bigger or equal number of suppliers than the current upper bound are considered anymore. Different algorithms are mainly differing in heuristics, which try to find the best choice for the next supplier to take or not to take into the supplier set. This is tight related to the determination of the lower bound, since branches with small lower bounds are normally good choices. An overview of algorithms to solve the set-covering problem can be found in [CFT98, CFSC01].

One possible intuitive quality rating for this objective would be the average number of suppliers over all ECUs. The disadvantage is that the removal of an ECU (no component allocated to that ECU) with a low supplier complexity would lead to a worse quality rating although this can be considered as an improvement. Another approach is using the highest number of necessary suppliers as quality rating. The drawback of this idea is that changes in mid-range quality ECUs can not be considered anymore. It is proposed to sum up all numbers of necessary suppliers for all ECUs in order to overcome the mentioned disadvantages.

Table 3.6.: *Database fractions necessary for the estimation of the objectives (O) and constraints (C)*

Fraction	Fig.	Resources	Weight	Costs	Busload	Energy	Supplier Complexity
Resources	3.2	C	O	O		O/C	
Wiring Harness	3.3		O	O			
Customer Orders	3.6			O			
Communication/Network	3.7, 3.8				O/C	O/C	
Power Functions	3.11					O/C	
Comp. Def. Suppliers	3.12						O

3.9. Summary

In this chapter, parallel to the description of the different objectives, a database model is developed. There are several sections in the database. Table 3.6 summarises, for which objective/constraint, which of these fractions of the data is necessary.

Several objectives are introduced and several different ways are proposed how to calculate the quality ratings for each objective. It is explained how *resources*, *weight*, *costs*, *busload*, *electrical energy consumption*, and *supplier complexity* can be assessed. It is also shown, which constraints are involved related to the objectives.

4. System Architecture Overview

This chapter gives an overview on the system architecture of MOOVE (Multi-Objective Optimisation of Vehicle Electronics) and HeurOpt (HEURistic OPTimisation), which have been developed in the course of this work. Section 4.1 explains requirements on the system architecture. Section 4.2 explains the architecture of HeurOpt, Sect. 4.3 the implementation of the domain-specific extension MOOVE. The results of this chapter are summarised in Sect. 4.4. The design and implementation of the HeurOpt framework is based on the work of Neumann [Neu05].

4.1. Motivation and Requirements

First, the motivation for the architecture of the system is explained. This section shows which requirements to the system architecture exist and how they can be fulfilled.

The main interest is to have a framework allowing the application of different optimisation strategies to the allocation problem described in this work. In Chap. 6, some optimisation strategies are presented and compared based on this framework. The population-based strategies that the framework aims at are for example evolutionary algorithms or ant colony optimisation.

Existing implementations for heuristic optimisation with a single objective [MFM01] as well as multiple ones [BLTZ03, SU05], usually are purpose-built for specific problems and are often programmed in C or C++. In contrast, the framework architecture presented in this chapter is required to be as independent as possible from the function allocation problem. This is ensuring its applicability to different problems in the future.

The most important criteria for the architecture of the presented framework have been:

- Separation of concerns
- Flexibility
- Portability and platform independence
- Small overhead

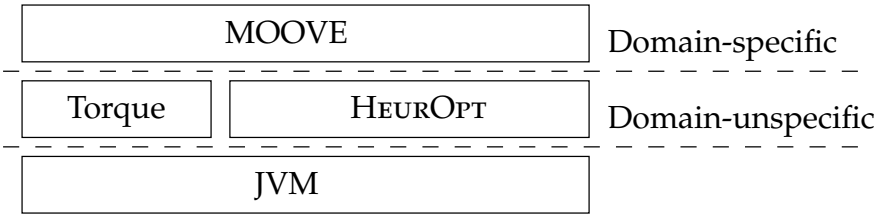

Figure 4.1.: HeurOpt *and* MOOVE *software layers*

These criteria are explained in the following:

Separation of Concerns. It is required to strictly distinguish between domain-specific and domain-unspecific layers, because the framework intends its application to arbitrary problems beside the function allocation. The separation of the concrete problem domain of *vehicle electronics* from the unspecific optimisation logic is done by defining the two layers MOOVE and HeurOpt (see Fig. 4.1). Object-oriented programming is used for structuring the spheres of responsibility by inheritance and aggregation: The inheritance hierarchy and structure of the framework adheres to the hierarchy of the problem domain layers.

Portability and platform independence. The framework is implemented in Java [GJSB05]. Despite the widely known statement that C++ performs better than Java, modern *Java Virtual Machine*s (JVM) do not lack general performance any more [BSPF01]. The Java support of database persistence layer frameworks like Apache Torque [Apa05] and platform independent development environments like Eclipse [Ecl05] in addition to the built-in platform independence of Java were significant for the decision to use Java as programming language for the framework. Therefore, the JVM can be found at the bottom of the overview in Fig. 4.1. HeurOpt is designed to deal with solution populations, solutions, objectives, and constraints in a generic way by defining interfaces and implementing abstract classes, component repositories, and configuration management. In spite of the complex requirements, a kind of simplicity is achieved by adherence to software development patterns from the well-known *Group of Four* [GHJV94] and *Pattern Oriented Software Architecture* (POSA, [BMR+96, SSRB00]) books. To understand this chapter in detail, knowledge about design patterns is required. The interrelationships between solutions and objectives/constraints are implemented based on the *model-view* design pattern, as shown in Sect. 4.2.4.

Flexibility. A further primary goal of the framework is the application of different optimisation strategies to the allocation problem of vehicle electronics. It is required to provide interfaces for solutions, populations, objectives, and constraints in a flexible way. Since the problem of allocating one set (of components or abstract targeting items) completely to another set (of network nodes or abstract targets) does not only relate to vehicles, implementations for this abstract

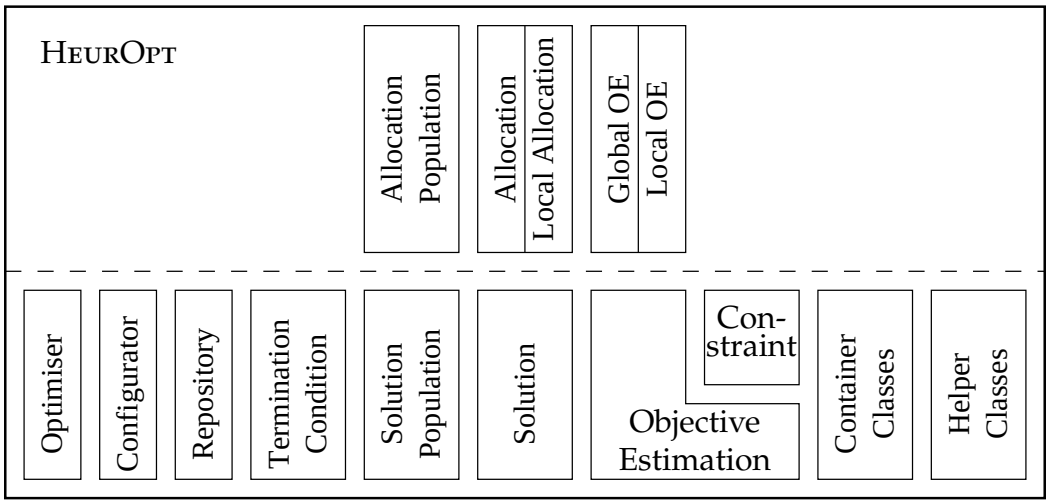

Figure 4.2.: *The* HeurOpt *layers*

problem domain are part of HeurOpt. They can be reused for similar allocation problems. They are the basis for the MOOVE project discussed in Sect. 4.3.

Small Overhead. To ensure a minimal overhead, efficient object cloning is required, rebuilding the relationship between populations, solutions, and objective estimations. Another possibility to reduce overhead is to inform objective estimations implementations of solution (or concrete: allocation) changes in an incremental way. For example, mutation and crossover operations from evolutionary algorithms change only parts of a solution. The objective or constraint implementation that has to evaluate and rate the new solution can decide either to use the update information or to access the whole solution/allocation information if necessary.

4.2. HeurOpt

In this section, the domain independent framework called HeurOpt is introduced. The most difficult part of HeurOpt is the initialisation process as shown in Sect. 4.2.1. The principal optimisation sequence is described in Sect. 4.2.2. The relationships between population and solutions as well as between solution and objective estimations are explained in Sect. 4.2.3 and Sect. 4.2.5. Figure 4.2 gives an overview of the HeurOpt classes.

The lower part of Fig. 4.2 shows the allocation independent part of HeurOpt. The upper part of Fig. 4.2 consists of classes that support the problem of allocating one set (of targeting items) completely to another set (of targets), bearing no

relation to vehicles. Vehicles are not considered until Sect. 4.3, with its descriptions of the MOOVE extensions.

4.2.1. Initialisation

Component[1], component repository, and component configurator are design patterns from [SSRB00, pp. 75–107]. A HeurOpt Optimiser is the aggregation of a TerminationCondition and a SolutionPopulation. The Configurator initialises the TerminationCondition with a reference to the SolutionPopulation.

The basic component interface just provides standards for initialisation, finalisation, and information retrieval. The optimiser implements the ResumableComponent interface and not the basic Component interface. The resumable component supports run(..), resume(..), and suspend(..).

A Configurator is the first object visible to the client application. The Client provides a ConfigurationDirective to the configurator's processDirective(..) method. A ConfigurationDirective can for example be an XML file[2] including the settings for the current optimisation run. The configuration directive in turn returns a reference to the created optimiser.

4.2.2. Optimisation Sequence

Before the population with its solutions and objective estimations is discussed, a *Message Sequence Chart* (MSC) of the interaction between client, component, and termination condition is given, shown in Fig. 4.3.

The Optimiser in its role as a ResumableComponent allows to suspend(..) and resume(..) the optimisation iteration. The suspension can be triggered asynchronously and suspends the component before the next generation. After the optimisation start with run(..), the optimiser uses the TerminationCondition as Iterator, calling hasNext(..) to find out whether the termination condition is still not met, and calling next(..) to initiate the next iteration. The TerminationCondition in turn calls the regeneratePopulation(..) of the SolutionPopulation.

A simple implementation of the TerminationCondition could be to specify a number of iterations. More sophisticated implementations can easily be integrated—for example they can be based on thresholds for objectives and the evaluation of the Pareto front. The Pareto front can be accessed by getParetoFront(..) and used for the termination condition internal checkCondition(..) method.

[1] The design pattern component is different from the component defined in Chap. 1.2.
[2] XML is the abbreviation for *eXtensible Markup Language* defined in [W3C04].

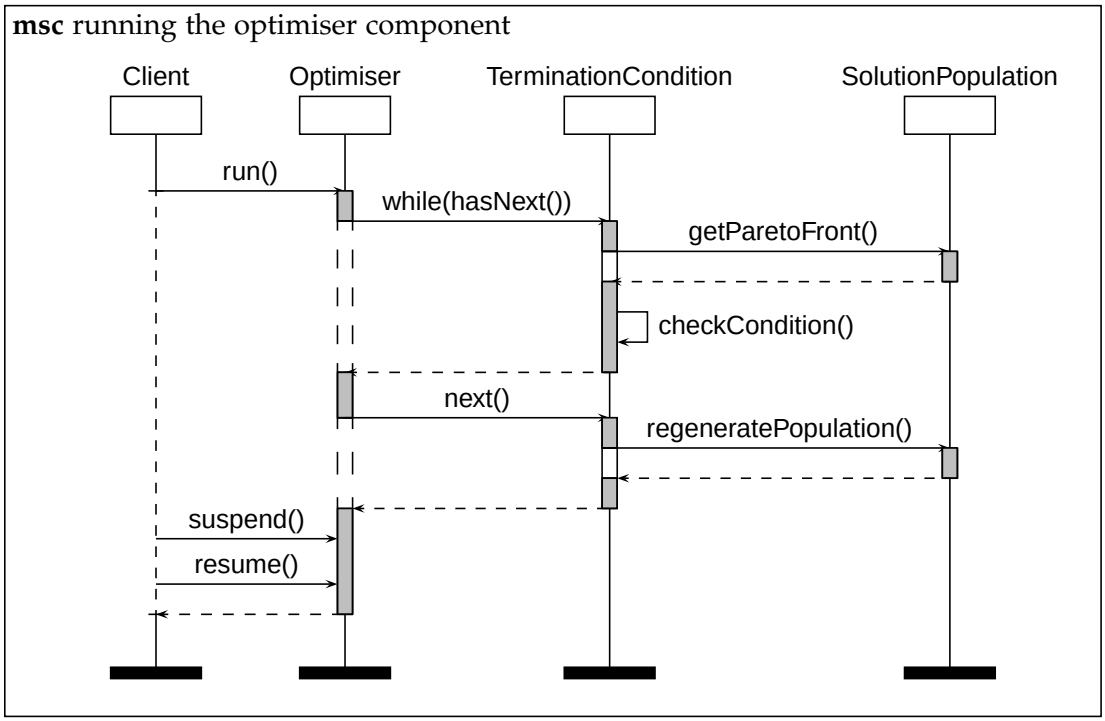

Figure 4.3.: *Running the optimiser component*

4.2.3. Populations

The population is defined by the SolutionPopulation interface. The population stores the solutions, using a collection based on the SortedSolutionSet interface (see Sect. 4.2.6).

The Solution interface (see Sect. 4.2.4) allows generic storage and access to the quality ratings of its objectives. The operations that the population applies to its solutions in order to modify them are dependent on the optimisation strategy. For example it is depending on, whether it is a colony of ants or an evolutionary population of individuals. Because of that, a population must explicitly define a set of supported solution classes. The class names of the applied population and solution can be configured in the ConfigurationDirective.

An important functionality of a SolutionPopulation is the method generateInitialPopulation(..). During initialisation, a first single solution including the relationships to the objective estimations is set up. The population has to implement the generateInitialSolution(..) that leads to this *initial solution*. The initial solution is allowed to contain some actual solution information that is constant to all solutions of one optimisation cycle. For example, when some components are fix allocated (see Sect. 4.2.7), these are part of the initial solution. Additionally, heuristics can be used to deduce additional parts of the solution that are

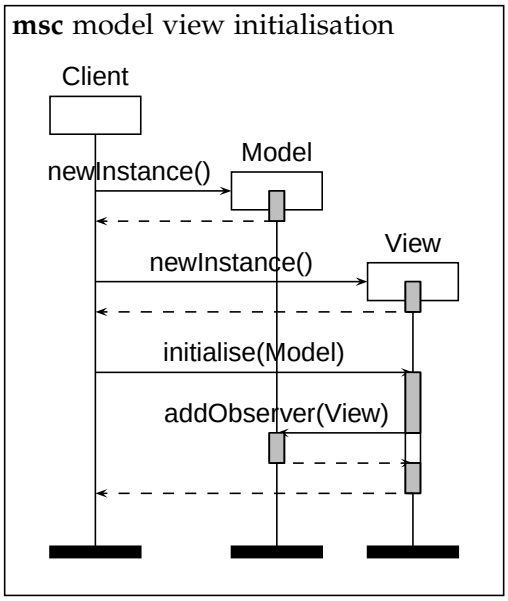

Figure 4.4.: *Instantiation and initialisation of model and view*

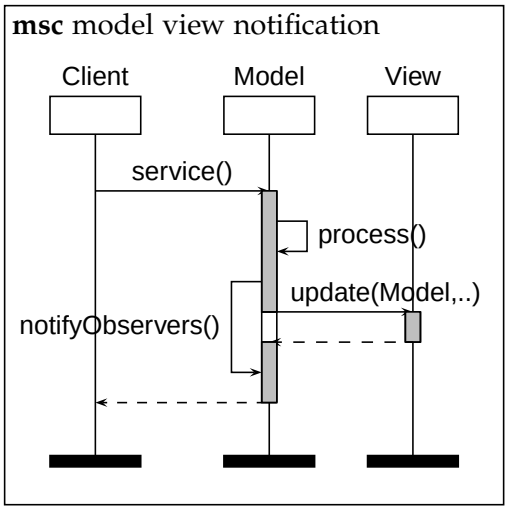

Figure 4.5.: *Notification and update between model and observers*

changeable. The information stored in the initial solution can help to reduce the solution space significantly.

4.2.4. Solutions

The most important role of a solution is the Model, according to the model-view paradigm. It is primarily defined in Java [GJSB05] by the java.util.Observable interface. An alternative is the *model-view-controller* (MVC) pattern in [BMR+96, pp. 125–143]. There is some semantic gap between the Java definition of model and view and the definition from the MVC pattern. The MVC pattern differentiates between observer (or controller) and view. The model-view paradigm treats them synonymously.

HeurOpt only makes use of the Model, ModelData and Observer interfaces, although the design pattern contains more. The class View is used with the semantics of an Observer, in accordance to the Java definition and in contrast to the MVC pattern definition. The Model interface is implemented by solutions. The View interface is implemented by the ObjectiveEstimations (see Sect. 4.2.5). The two standard use cases of the model-view pattern are shown in Fig. 4.4 and Fig. 4.5: the initialisation as well as the notification and update.

The ObjectiveEstimation objects evaluate a solution according to an objective. The result is provided as a numeric value, represented as QualityRating object.

Chapter 3 explained how to calculate these quality ratings. In Sect. 4.2.5, it is explained how constraints can be defined. The most important role of an ObjectiveEstimation in relation to a solution is the one as a View.

The Model has two tasks. First, it has to store the data of the model. SolutionModelData is extending the MVC-class ModelData and is explained more in detail in Sect. 4.2.7. The model data is further extended by the optimisation domains.

The second task of the Model is to behave like an Observable. Therefore, the observers that are registered at the observable have to be managed. In the context of optimisation iteration, the ObjectiveEstimation classes are the only classes that implement the Observer interface. Since the Solution implements the role of a model, and since the objective estimations implement the role of an observer, there are two equivalencies to remember: the Solution/Model/Observable and the ObjectiveEstimation/View/Observer. The significant methods of the Model as Observer are addObserver(..), deleteObserver(..) and notifyObservers(..). Especially, the addObserver(..) is used during initialisation of the framework to register the OEs at a solution.

The notifyObservers(..) has to be invoked when the model is changed. The model changes during the initialisation or during a call of regeneratePopulation(..). In Fig. 4.5, these methods are summarised as service(..). After a call of service(..), the model will process(..) the changed data. Finally, the model iterates over its observers invoking their update(..) method. According to the design pattern, the parameters of the update methods contain a reference to the model and an object that should represent the model change. Although each view stores a reference to the model, this reference is still provided as parameter of the update method because it is possible for an observer to register at different models. For solutions and OEs, this is not necessary, since separate OE instances are used for each solution. Support for cloning is provided instead of reusing OE instances for multiple solutions, which would introduce management overhead to the OEs for the de-multiplexing, especially for incremental updates.

4.2.5. Objective Estimations

ObjectiveEstimation objects have to evaluate a solution according to an objective. The result is provided as a numeric value by the method calculateQualityRating(..). Each Solution has several ObjectiveEstimations.

ObjectiveEstimations handle constraints as well. Constraints have to implement the ConstrainedObjectiveEstimation interface, extending the ObjectiveEstimation. Constraints are declared like objective estimations and provide their degrees of violation in form of their QualityRating. If a constraint is violated, the isViolated(..) method has to return true.

In comparison to the Chap. 3, additional objective estimation can be defined by implementing the ObjectiveEstimation interface and adding the implemented class to the ConfigurationDirective.

4.2.6. Container Classes

Many populations need a container class in order to store solutions sorted according to different ratings. The SortedSolutionSet interface defines the getParetoFront(..) method. The concrete implementation can be defined in the ConfigurationDirective.

One possible implementation that is used by an evolutionary algorithm is the SPEA2 archive. According to the description in Sect. 2.1.2, a SPEA2SortedSolutionSet can be implemented providing an additional truncate(..) method.

4.2.7. Solution Model Data for Allocations

This section introduces some concepts dealing with the problem of allocating a set of *targeting items* completely to another set of *targets*. These classes/interfaces are used by the MOOVE extension in Sect. 4.3: TargetingItems are the SW/HW-components that have to be allocated to the ECUs (specialisations of Targets). The classes in this section are not directly related to the optimisation and the strategy, but are aggregated by Solutions to represent their ModelData as explained in Sect. 4.2.4. Additionally, this section provides understanding of the allocation problem, and is necessary to understand the concept of *local models* in the following Sect. 4.2.8.

For expressing the allocation of a set of targeting items to a set of targets, the concept of edges is applied: EdgeID aggregates one Target and one TargetingItem object, and therefore represents an identifier for an edge of the allocation.

An edge has a status type, expressing whether it is actually allocated or not. For this purpose, the enumeration class StatusType exists. It realises the main types ALLOCATED and NOTALLOCATED. Additionally, the types INIT for initialisation purposes with the semantics of *unknown*, and the types ALWAYSALLOCATED and NEVERALLOCATED for fixed allocation information are available. The aggregation of an edge identifier with a status type is standardised as EdgeStatus class.

The information for the allocation itself is implemented as AllocationModelData. Its initialisation requires a TargetingItemSet and a TargetSet. It constructs a matrix with all edge identifiers in the status type INIT as coefficients. The status types are changed during the initial phase. First, the fixed allocation information are filled. During each optimisation iteration, the allocation is changed by the

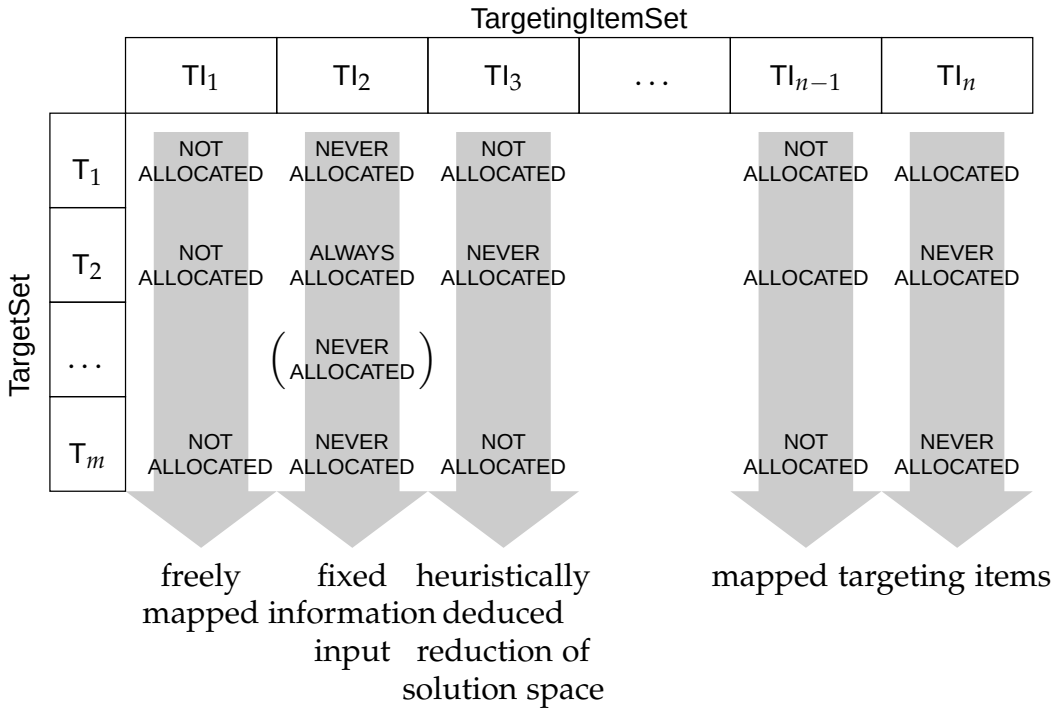

Figure 4.6.: *The allocation information structure (example)*

dynamic operations. Figure 4.6 shows an example for an initialised allocation structure. It is not allowed for a targeting item to be allocated on two different targets at the same time.

The GlobalAllocationSolution uses the AllocationModelData as model data. The GlobalAllocationSolution is further extended by the MOOVE extension in Sect. 4.3, introducing vehicle semantics. The implementations in the HeurOpt packages deal with targets and targeting items in a generic way. The Global-prefix is explained in the next section, when the *local models* are introduced.

An EdgeOperation or a set of edge operations is used as change-object by the notifyObservers(..) and update(..) methods between a GlobalAllocationSolution and its GlobalObjectiveEstimation observers. The GlobalAllocationSolution provides an apply(..) method that transforms EdgeOperation objects into model change and update notifications.

4.2.8. Local Models

This section introduces a concept of the integration of inner optimisation loops into outer optimisation iterations. The motivation lies within the concrete problem domain of control networks for vehicles.

For many objective estimations, the determination of a quality rating working only with the data of one ECU is much easier. A global objective estimation can aggregate the quality ratings of the local ones. For example, the implementation of a cost estimation including the variant optimisation according to Chap. 7 could cover only single ECUs. The global cost estimation has to add the cost values in order to determine the total costs.

The framework supports an integration of target-local solutions into a global allocation solution. This means that for example the update and notification between global and local solutions as well as the according global and local OEs are supported. Additionally, the cloning mechanism must support the rebuilding of these interrelationships. The framework does not support any decision process of *when to run the local optimisation*. The local optimisation might introduce significant overhead—it is often not applicable to run it for each newly generated global allocation solution and all dependent local models. The global solution and the global objective estimations must be aware of the local models and make this decision on their own.

The target-local solution is represented by the LocalAllocationSolution interface. It extends the Solution interface, but also extends the View interface, being a view of the GlobalAllocationSolution objects.

In contrast to the global allocation solution, which represents an allocation, the LocalAllocationSolution just represents a single Target with a set of targeting items that are (currently) allocated to the target. The TargetingitemsPerTargetSet is used as ModelData—it is a partition of the AllocationModelData used by the global solution.

The global optimisation changes the set of targeting items for each target, because the allocation-edges are changed. During a local optimisation, its set of targeting items must not change. The set is represented by a TargetingItemSet. A local solution is not conceptually related to allocations.

Figure 4.7 provides an exemplary overview of the structure and interrelationship that are the results of the initialisation for one solution. The upper half of Fig. 4.7 shows a solution s with its registered OEs. As examples for the OEs, just use a subset of the ones explained in Chap. 3 (C: costs, BL: busload, W: weight, E: Energy, HWA: Hardware Availability (instance of resources)) is used. The HWA in Fig. 4.7 is a constraint and in the same role as the other objective estimations. The lower half of the figure represents the local models for the two ECUs e_1 and e_2 and their relation to the global objects. The arrows indicate access availability.

A GlobalObjectiveEstimation can decide to use no LocalObjectiveEstimations (local OE). It must indicate this by returning true for the isPureGlobal(..) method. In Fig. 4.7, the example for this is the busload (BL), because the busload only needs

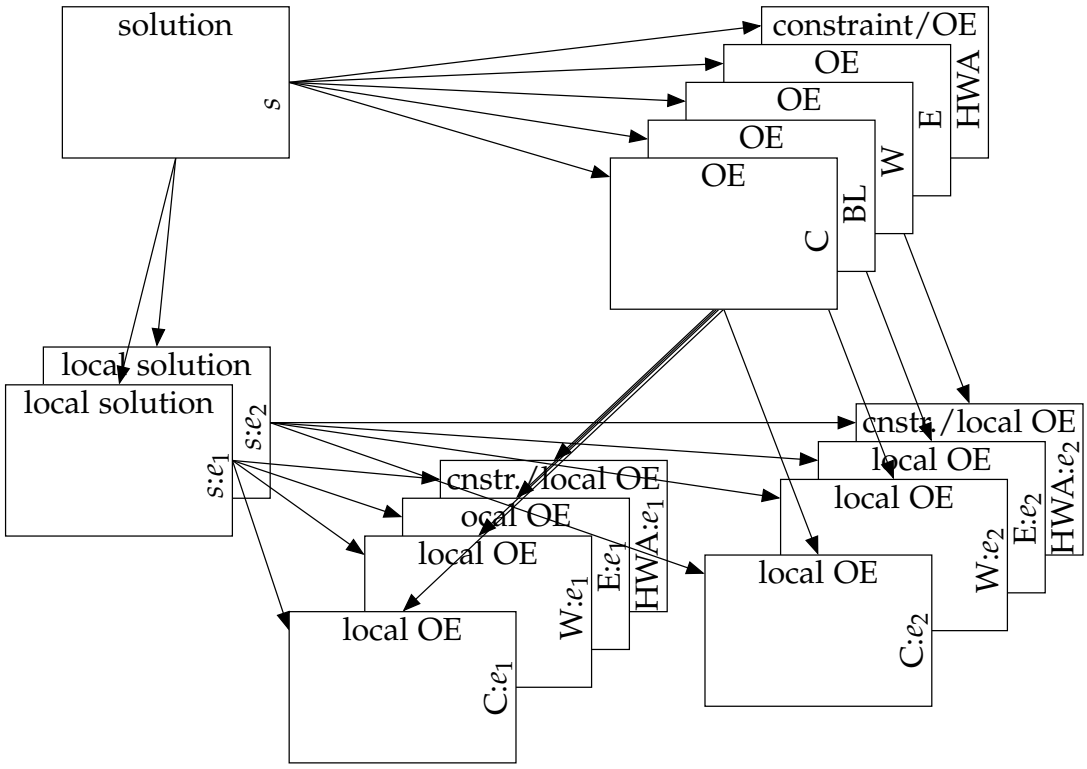

Figure 4.7.: *Connection structure between solutions and objective estimations*

the global view under any circumstances. The isPureGlobal(..) method is mainly used during the creation process for the local OEs, skipping such global OEs. After the creation and initialisation of the local OEs, the local solution is now also initialised according to its first role.

4.3. Multi Objective Optimisation of Vehicle Electronics

Based on the HeurOpt framework, the MOOVE project consists of concrete implementations for the vehicle domain, and aims to solve the problem of allocating software and hardware components to ECUs. An overview can be found in Fig. 4.8. It can be seen that only a small portion of the architecture is optimisation algorithm specific. This makes it easy to implement and test different optimisation strategies. A database aware configurator is available as well as several implementations of optimisation algorithms. The database model introduced in Chap. 3 is part of the MOOVE project as well as objective estimations including their constraint implementations.

MOOVE implements a special configurator and directive extending the HeurOpt Configurator that is aware of the database. The MOOVE project pro-

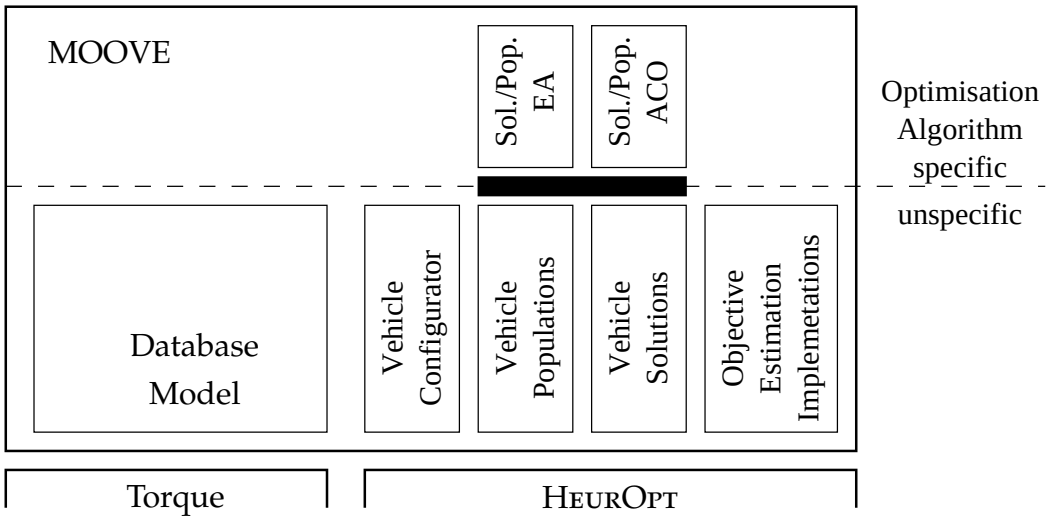

Figure 4.8.: *The* MOOVE *layers*

vides the VehicleConfigurator. It uses the database as input for the TargetSet and the TargetingItemSet. The implementation of the Torque-generated class[3] ComponentInstance for accessing the tuples of the component_instance relation is changed so that the TargetingItem interface is supported. The class for the network_node relation is equally extended to support the Target interface.

The VehiclePopulation class implements the generateInitialSolution(..) method. The manually provided information about fixed allocations from the database is accessed and applied to the initial solution, preparing the AllocationModelData like it was shown in Sect. 4.2.7.

Furthermore, the VehiclePopulation class implements the method generateInitialPopulation(..) using the randomFill(..) method to create solutions for the initial population. The creation of the initial population is highly dependent on the optimisation algorithm. Further extensions are explained in Chap. 6.

According to the discussion of local models in Sect. 4.2.8, the VehicleLocalSolution is available as ECU-local solution and uses the generic TargetingitemsPerTargetSet as model data. It delegates update notifications to the local OEs calling the notifyObservers(..) method inherited from the HeurOpt implementations.

Several objective estimations are available as part of the MOOVE project. They have been implemented based on the objectives and database models in Sect. 3.

[3]Torque generates a Java class for each relation in the database.

4.4. Summary

In this chapter, the HeurOpt framework and the MOOVE project are explained. It is shown that the requirements from Sect. 4.1 are fulfilled. It is explained how an optimisation algorithm is composed from different solutions, populations, iterators, and objective estimations. It is also shown, how the framework can be extended with further objective estimations. Torque can be connected to a big number of different database implementations. All these database implementations care for the consistency of the stored data by checking the primary key, unique, and foreign key constraints. The implementation has been performed based on Java to reach platform independence. The architecture of the framework ensures reusability of large parts of the code for different problem domains.

With HeurOpt and MOOVE, a framework is available enabling all implementations and benchmarks necessary in the course of this work.

5. Description of the Application Example

In this chapter, an application example is shown. As already explained previously, there is an endless number of electronic functions in modern vehicles providing both comfort and safety to the driver. In this chapter, a small number of these functions have been chosen. Still, this example is representing the typical tasks, architects of electronic systems in vehicles are confronted with. The application example is an allocation task for the functions *keyless entry*, *central door locking*, *direction indication*, and *exterior light*. Since some of the functions might be new to the reader, all functions are explained first (see Sect. 5.1). A detailed function specification is given in Sect. 5.2. Section 5.3 shows, which results can be gathered with existing methods.

5.1. Functions Used in the Application Example

In this section, all functions that are part of the application example are explained and the necessary components are introduced. A total of 22 components are defined.

5.1.1. Exterior Light

The basic sub-functions of the function *exterior light* are *driving light*, *parking light*, *high beam*, *braking light*, *fog light*, *fog rear light*, and *reverse gear light*.

Drivers can control the functions *driving light*, *parking light*, *fog light*, and *fog rear light* by a *light turn switch* $c_{\mathrm{EL-LTS}}$. The functions *parking light* (only left or right) and *high beam* are controlled by the *direction indicator switch* $c_{\mathrm{DI-SW}}$ (see Sect. 5.1.2). The functions *braking light* and *fog light* are controlled by other functions, which are not modelled in this example.

In order to perform the above functions, an *exterior light control component* $c_{\mathrm{EL-CTRL}}$ is necessary. It is computing the prioritisation, and it is proofing other conditions like ignition status and so on. For example, it is not only depending on the status of the light turn switch, but also on the ignition status, the

rain/light sensor, and the coming/leaving home function, if the driving light is on.

Two different actuators have been modelled: The *Front Light Unit* (FLU) and the *Rear Light Unit* (RLU). The according four instances of components—a left and a right one in each case—are named $c_{\mathrm{EL-FLU-L}}$, $c_{\mathrm{EL-FLU-R}}$, $c_{\mathrm{EL-RLU-L}}$, and $c_{\mathrm{EL-RLU-R}}$. They already include the direction indication lights.

If a *rain/light sensor component* $c_{\mathrm{EL-RLS}}$ is assembled, the additional function *coming/leaving home* is available. The coming/leaving home function is supposed to switch on the light during the driver's way to or from the car at night. The function *leaving home* switches on the exterior light, as soon as the car is unlocked by the radio remote key (see Sect. 5.1.3) and until a door is opened. The function *coming home* keeps the exterior light on, once the driver leaves the car for a certain time. The rain/light sensor component is necessary for these functions, since they are only active, if it is dark outside.

5.1.2. Direction Indication

The function *direction indication* consists of the sub-functions *direction indication*, *hazard-warning signal flasher*, *crash signal flasher*, *panic signal flasher*, *central door locking confirmation*, *trailer direction indication*, and *theft alarm light* among others. Some of the sub-functions are implemented since decades in cars, like for example direction indication and hazard-warning signal flasher. The central door locking confirmation has the task to confirm, once the door locks are secured or unsecured after operation of the remote key. After the notification of a crash by another car system, the crash signal flasher is automatically operated similar to the hazard-warning signal flasher. If somebody without authorisation wants to open/start the car, the theft alarm light is started beside the horn.

In this application example, the components *direction indication switch* $c_{\mathrm{DI-SW}}$ and *hazard light switch* $c_{\mathrm{DI-HLSW}}$ provide input signals. There are three control components, the *direction indication master control* $c_{\mathrm{DI-CTRL}}$, the *direction indication front control* $c_{\mathrm{DI-FR-CTRL}}$, and the *direction indication rear control* $c_{\mathrm{DI-RE-CTRL}}$. Different control components for front and rear a useful in order to allow more degrees of freedom during the allocation.

5.1.3. Central Door Locking and Keyless Entry

The function *central door locking* secures all door locks, once the driver communicates this wish by locking one door lock with the key or by pressing the according button on the radio remote control. For executing this function, the *central door locking control component* $c_{\mathrm{CDL-CTRL}}$ needs a *radio receiver* $c_{\mathrm{CDL-REC}}$

and the four *door lock components* $c_{\text{CDL-DL-FL}}$, $c_{\text{CDL-DL-FR}}$, $c_{\text{CDL-DL-RL}}$, and $c_{\text{CDL-DL-RR}}$.

Keyless entry is an extension of the normal central door locking function. The benefit is that the driver can keep the key in his pocket. Once the driver approaches the car, several antennas within the car recognise the location of the key. A signal is sent to the central door locking component to open the locks, once the driver approaches the according door handle. A button on each door lock allows the driver to secure the car without touching the key when leaving the car. A detailed description of a similar system can be found in [SKPR98]. For the execution of this function, an additional *keyless entry control component* $c_{\text{KES-CTRL}}$ and three *antenna components* $c_{\text{KES-AN1}}$, $c_{\text{KES-AN2}}$, $c_{\text{KES-AN3}}$ are necessary.

5.2. Detailed Specification of the Example

This section specifies the functions and components introduced in the last section more in detail. Thereby, the database model explained previously in Chap. 3 is used. An overview of all components and their specification can be found in Tab. 5.1. All values in this table are explained in the course of this section.

Components can be composed from software, hardware, and sensors/actuators. If a component contains a sensor/actuator, a number of cables have to be connected from the allocated ECU to the location of the sensor/actuator. The locations (columns x, y, and fraction) specified in Tab. 5.1 are the locations of the sensor/actuators. Only components, that have values for their location, contain a sensor/actuator.

5.2.1. Network Topology

The network topology consists of five ECUs. They are called *body power module front* (BPMF, e_{BPMF}), *body power module rear* (BPMR, e_{BPMR}), *switch module steering column* (SMSC, e_{SMSC}), *door control unit driver* (DCUD, e_{DCUD}), and *door control unit co-driver* (DCUC, e_{DCUC}). They are connected to a CAN network with 100 kBaud. The other network nodes existing in a real car on this network are not modelled in the example. The network is additionally connected to a *gateway* connecting the ECUs to all other necessary functions, for example the gear box providing the reverse gear signal in order to enable the reverse gear light (see Sect. 5.1.1). An overview of the network topology is presented in Fig. 5.1. An overview of all parameters of the network topology can be seen in Tab. 5.2. All these values are explained in the course of this section.

Table 5.1.: *Overview of the definition of the components*

component	space	ROM kB	thin cable	thick cable	idle active	supplier	x m	y m	fraction
c_{EL-LTS}	1	0	1	0	☒	A,B,C	−0.4	2.9	l
$c_{EL-CTRL}$	1	64	–	–	□	A,B,C	–	–	–
$c_{EL-FLU-L}$	1	0	0	10	□	A,B,C	−0.5	4.2	l
$c_{EL-FLU-R}$	1	0	0	10	□	A,B,C	0.5	4.2	l
$c_{EL-RLU-L}$	1	0	0	15	□	A,B,C	−0.6	0.0	r
$c_{EL-RLU-R}$	1	0	0	15	□	A,B,C	0.6	0.0	r
c_{EL-RLS}	1	0	1	0	□	A	0.1	2.95	l
c_{DI-SW}	1	16	1	0	□	A,B,C	−0.2	2.9	l
$c_{DI-HLSW}$	1	16	1	0	☒	A,B,C	−0.35	2.9	l
$c_{DI-CTRL}$	1	64	–	–	□	A,B,C	–	–	–
$c_{DI-FR-CTRL}$	1	32	–	–	□	A,B,C	–	–	–
$c_{DI-RE-CTRL}$	1	32	–	–	□	A,B,C	–	–	–
$c_{CDL-CTRL}$	1	32	–	–	☒	A,B,C	–	–	–
$c_{CDL-REC}$	1	0	0	2	☒	A,B,C	0.1	2.75	l
$c_{CDL-DL-FL}$	1	0	0	2	□	B	−1.0	2.2	l
$c_{CDL-DL-FR}$	1	0	0	2	□	B	1.0	2.2	r
$c_{CDL-DL-RL}$	1	0	1	0	□	B	−1.0	1.4	l
$c_{CDL-DL-RR}$	1	0	1	0	□	B	1.0	1.4	r
$c_{KES-CTRL}$	1	16	–	–	☒	C	–	–	–
$c_{KES-AN1}$	1	16	0	2	☒	C	0.3	2.7	l
$c_{KES-AN2}$	1	16	0	2	☒	C	0.75	1.9	r
$c_{KES-AN3}$	1	16	0	2	☒	C	−0.2	0.0	r

5.2.2. Resources

In this example, the number of components per ECU is limited. Therefore, a resource called *space* for components is defined. Each component consumes

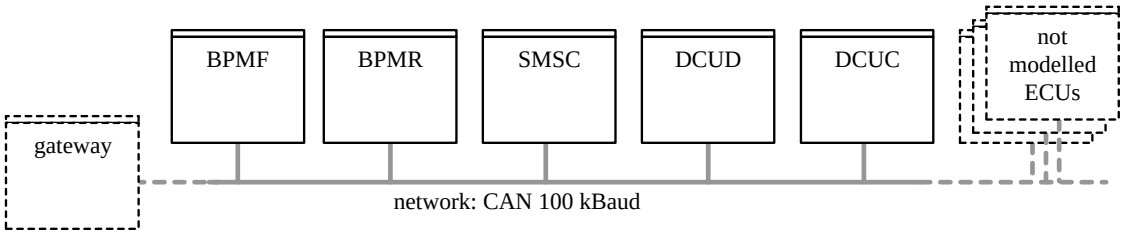

Figure 5.1.: *Network topology of the application example*

Table 5.2.: *Overview of the definition of the hardware topology*

ECU	number	space	ROM kB	x m	y m	fraction
e_{BPMF}	1	max_units = 5	max_units = 128	−0.6	3	l
e_{BPMR}	2	max_units = 5	max_units = 128	0.6	0.9	r
e_{SMSC}	3	max_units = 5	max_units = 128	−0.3	2.9	l
e_{DCUD}	4	max_units = 5	max_units = 64	−0.75	2.8	l
e_{DCUC}	5	max_units = 5	max_units = 64	0.75	2.8	r

one unit of this resource. The ECUs in the network topology limit the number of components to be allocated by providing space with max_units = 5. In this example, the space resource is representative for all other kinds of resources, like for example circuit board space or timer/CPU load constraints. Once, more than a certain number of components are allocated to an ECU the constraint is violated. The according quality rating q_{space} equals to the sum of the number of supernumerous components on all ECUs.

A further resource is called *ROM*. The two door control units provide a smaller amount of ROM than the others (see Tab. 5.2). The amount of ROM consumed by the components is shown in Tab. 5.1. Quality ratings q_{ROM} for the resource ROM are determined analogue to q_{space}. If the constraints are violated each Byte that does not fit on the micro-controller is summed up.

There are two different types of the resource *cable*. They are called *thin cable* and *thick cable*. Cables are consumed with a number of cables. The values in the two cable columns in Tab. 5.1 represent this number of cables. The number of consumed cable resources indicates how many cables are necessary in order to fulfil the specified function. For example, a front light unit needs 10 thick cables, because there are 10 bulbs to control. Thick cables are mainly used, if power has to be transmitted like for lights. Thin cables are primarily used for signals. For example the rain/light sensor pre-processes the sensor data and sends the signals about the current weather or light condition via a single thin cable to the receiver ECU. No quality rating is necessary for the cable resource, since there are no constraints on this resource. It is not constrained in this example, although in reality it could make sense to limit the number of cables leaving an ECU depending on the installation location.

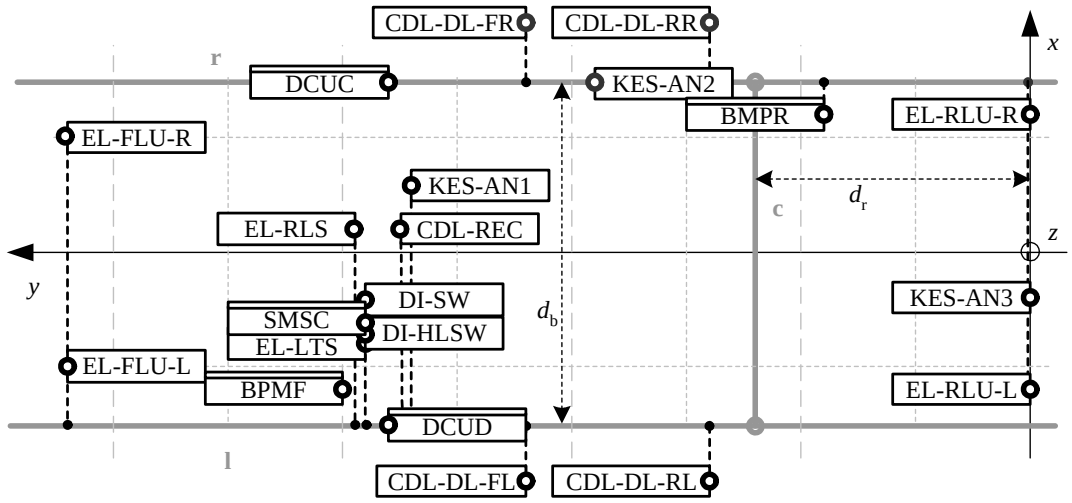

Figure 5.2.: *Overview of the location of ECUs and sensor/actuators contained in components*

5.2.3. Costs

Cable resources are defined above. In this example, cables are the main influence factor for costs that can be influenced by the allocation of components in this example. Costs that can be saved for example by optimisation of variants are not taken into consideration. An additional example for variant optimisation is presented within Chap. 7. A cost influence of 0.05 €/m for the thin cable and of 0.1 €/m for the thick cable is specified, where € is a *virtual cost unit*. Basic ECU costs are not defined at all. Thus, the costs determined by the quality rating q_{costs} represent only the wiring harness.

For the sake of transparency, the wiring harness is specified only 2-dimensional. The parameters distance from rear, breadth, and distance from bottom (see Sect. 3.4.3) are set to $d_{\text{r}} = 1.2\,\text{m}$, $d_{\text{b}} = 1.5\,\text{m}$, and $d_{\text{h}} = 0\,\text{m}$. The locations in x- and y-direction as well as the connected *fraction* of the wiring harness can be seen in Tab. 5.2 for the network topology and in Tab. 5.1 for the components. Only the left (l) and right (r) wiring harness fraction are used in this example. An overview of the locations is given in Fig. 5.2. Components are symbolised by simple rectangles, ECUs show a second horizontal line at the top.

5.2.4. Weight

For the sake of easiness, the objective weight is not used in this example. The weight of a cable is often almost direct proportional to its costs. Thus, an opti-

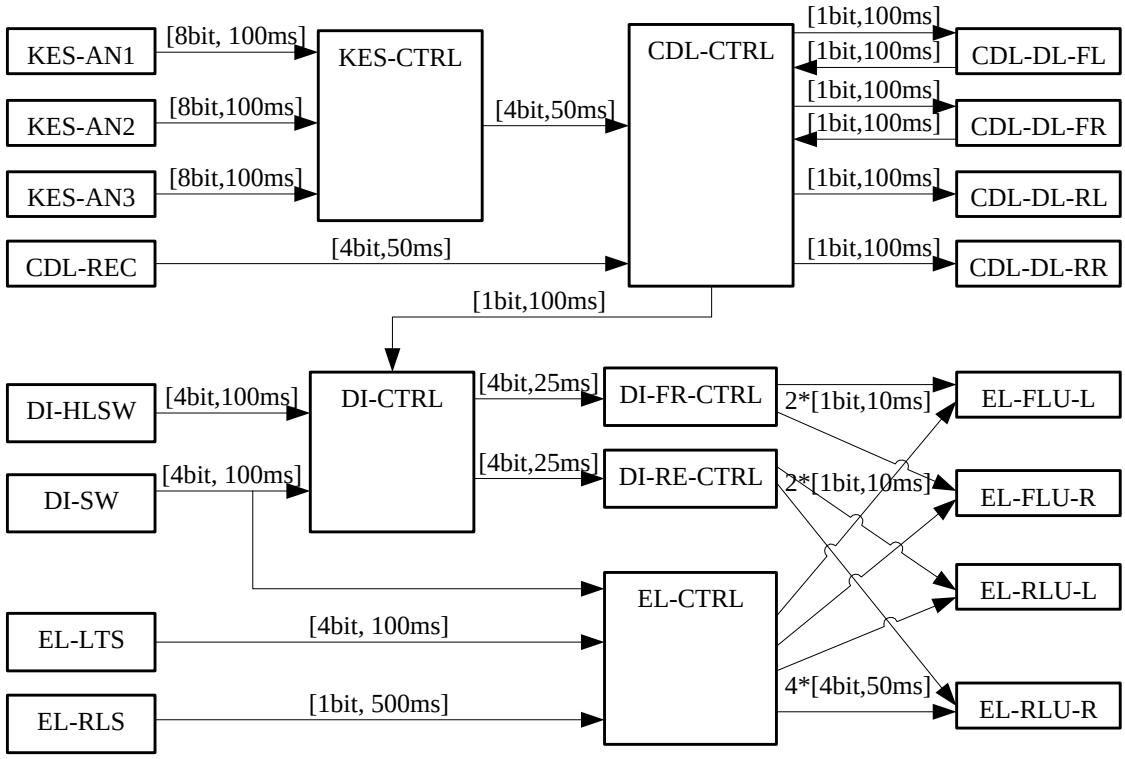

Figure 5.3.: *Functional network including all components and signals specified in the form $[b_{\text{sig}}, t_{\text{u}}]$ of the application example*

misation of costs is optimising the weight already.

5.2.5. Busload

As already shown in Sect. 5.1, the functions of this example are highly networked. All network interconnections are collected and summarised in Fig. 5.3. All signals are tagged in the form $[b_{\text{sig}}, t_{\text{u}}]$, denoting the length of the signal b_{sig} and the minimum update time t_{u} according to Sect. 3.6.

In order to calculate the busload, first of all, the network has to be specified. The network is a CAN network with a transmission rate of $r = 100\,\frac{\text{kbit}}{\text{s}}$. The protocol overhead factor is $p_{\text{oh}} = 2.125$, according to Sect. 3.6.3. Since CAN is not time-triggered, delay times are not relevant in this example. Thus, t_{cyc} in (3.13) equals the update time t_{u} of the signals. The variable $\mathbf{sig}_{N,\text{tr}}$ denotes the set of signals that have to be transported via the network. In this example, it contains all signals between two components that are not located on the same

ECU. The net busload l_N caused by all signals using the network is determined with

$$l_N = \frac{1}{r} \sum_{\forall \text{sig} \in \mathbf{sig}_{N,\text{tr}}} \frac{p_{\text{oh}} b_{\text{sig}}}{t_{\text{u}}}. \tag{5.1}$$

In Sect. 3.6.4, an aggregation model is proposed. Since only one network is involved in this example, the parameter w of this aggregation model can be set near 1. Thus, the quality rating q_{busload} is determined according to (3.14) (see p. 51) with

$$q_{\text{busload}} = \frac{l_N}{l_N^{\max} - l_N^{\text{bas}}}. \tag{5.2}$$

The value for $l_N^{\max} - l_N^{\text{bas}}$ is set to $l_N^{\max} - l_N^{\text{bas}} = 0.017$. It is assumed that the basic load of the network without these new functions $l_N^{\max}$ has a certain value and the maximum busload is 1.7 % higher due to extendibility. For example, if only one signal with $b_{\text{sig}} = 4\,\text{bit}$ and $t_{\text{u}} = 100\,\text{ms}$ has to be transferred, with (5.1) follows $l_N = 0.00085$ and thus, $q_{\text{busload}} = \frac{0.00085}{0.017} = 0.05$. That means, this signal occupies 5 % of the available busload.

5.2.6. Electrical Energy Consumption

For the application example, the *number of active ECUs* according to Sect. 3.7.1 is applied instead of using the more complex consideration of different *use cases* (see Sect. 3.7.2). The quality rating q_{energy} represents the number of ECUs that have to be active, when the car is in the idle mode. It gives a good estimation, if a high quiescent current is caused by the allocation of functions. For example, if all keyless entry antennas are placed on different ECUs, all of them have to be awake in the idle mode. The column *idle active* in Tab. 5.1 contains all necessary data for this objective. Each component that is active in the idle mode is ticked in Tab. 5.1.

5.2.7. Supplier Complexity

Supplier complexity represents the sum of necessary suppliers for each ECU as described in Sect. 3.8. For this example, three imaginary suppliers A, B, and C have been assigned to the components (see column *supplier* in Tab. 5.1). If there are no restriction in relation to the supplier, all suppliers are possible. The quality rating q_{supplier} is the sum of all numbers of suppliers for all of the five ECUs. Thus, if all five ECUs are used, the best thinkable value is $q_{\text{supplier}} = 5$.

5.3. Optimisation Results With Existing Methods

This section shows existing methods for determining a solution. These methods are used and it is tried to determine the best solution for the problem given above. Solutions s are normally described according to the definition in Sect. 1.5 and $s = \{a_1, a_2, ..., a_q\}^{\mathrm{T}}$, with $a = \{c \overset{\text{asg}}{\rightarrow} e\}$. In this section, a short form is used in order to have a more compact representation. A solution is described with a sequence of ciphers without separator in curly brackets. Each cipher indicates the ECU, on which a specific component is allocated. Each cipher is the allocation of a component according to the order in Tab. 5.1. The numbers of the ECUs are sorted according to the order in Tab. 5.2 beginning with 1. For example: $s = \{\mathbf{2}22222\mathbf{3}33335555554444\} = \{\underbrace{c_{\mathrm{EL-LTS}} \overset{\text{asg}}{\rightarrow} e_{\mathrm{BPMR}}}_{2}, \ldots, \underbrace{c_{\mathrm{DI-SW}} \overset{\text{asg}}{\rightarrow} e_{\mathrm{SMSC}}}_{3}, \ldots, \underbrace{c_{\mathrm{KES-AN3}} \overset{\text{asg}}{\rightarrow} e_{\mathrm{DCUD}}}_{4}\}^{\mathrm{T}}$.

5.3.1. Manual Optimisation

In reality, often an approach is applied for optimising the function allocation to ECUs similar to the following:

1. *Heritage*. In order to reduce risk, the architecture of older type series' is cloned.

2. *Domain grouping*. New functions are allocated *near*[1] to other functions of the same domain.

3. *Improvement proposals*. In the third step, the solution is made feasible. Then, further improvements proposals are made. Once a solution is compliant with the constraints, the most important objectives are safety and costs.

Since there is no history in the presented example, it is directly started with grouping the functions by domain. Grouping by domain means that first, the four domains exterior light, direction indication, central door locking, and keyless entry are treated as units and they are allocated only together. With this approach, only $5^4 = 625$ solutions have to be tested. If domination (see Def. 1) is only applied to the severity of the constraint violation, there is only one solution dominating all others: The exterior light is allocated to BPMR, direction indication to SMSC, central door locking to DCUC, and keyless entry to DCUD. Using only four ECUs seems to save a complete ECU, but the quality ratings of the solution {22222223333355555554444}

[1] *Near* in this sense can mean allocated onto the same ECU or connected to the same network.

are $\mathbf{q} = \{q_{\text{space}}; q_{\text{ROM}}; q_{\text{costs}}; q_{\text{busload}}; q_{\text{energy}}; q_{\text{supplier}}\} = \{3.0; 32000; 21.965; 0.613; 4.0; 5.0\}$. Thus, this solution is infeasible.

This solution has three critical ECUs, where the constraint violations take place: DCUC carries six components, but is only allowed to have five allocated on. BPMR carries seven components instead of five. The amount of used ROM on SMSC is 160 kB instead of 128 kB. No components are allocated to the BPMF. After re-allocating the components EL-RLS, EL-LTS, DI-HLSW, DI-SW, and CDL-DL-FL to the BMPF, the solution $\{122222111333515554444\}$ is feasible. The according quality ratings are $\mathbf{q} = \{0.0; 0.0; 20.97; 0.590; 3.0; 7.0\}$.

Finally, an emulation of the process of improvement proposals is performed. With the current solution, all movements of single components to different ECUs are tested. Due to the tight constraints q_{space} and q_{ROM}, all exchanges of all combinations of two components located on different ECUs are tested as well. It can be argued that costs are very important. On the other hand, how much cost causes a high number of ECUs active during the idle mode of the car? Therefore, in two different runs, two different criteria have been defined for accepting a movement of one or an exchange of two components as improvement:

- The new solution is improving the old one in at least on objective and is not worsening it in the other ones.
- The quality rating for the objective costs has improved and the solution is still feasible (or the new solution is dominating the old one).

In the first case, only three improvement proposals can be found and the resulting solution $s_{\text{dom}} = \{1233221113335545554444\}$ has quality ratings of $\mathbf{q} = \{0.0; 0.0; 14.465; 0.540; 3.0; 7.0\}$. In the second case, 22 found improvement proposals led to a solution $s_{\text{costs}} = \{3231221443331145425452\}$ with the following quality ratings: $\mathbf{q} = \{0.0; 0.0; 11.8125; 0.915; 5.0; 9.0\}$. The costs are much lower, but in the other objectives, significant worsenings has to be accepted. Here, the main drawback of this kind of optimisation can be seen clearly. No trade-off solutions are available in order get a better result according to the other objectives while investing a small amount of costs.

5.3.2. Exhaustive Search

Another possibility for determining solutions is to perform an exhaustive search. Each possibility can be tried in order to get a good result. The computer used for this test manages to evaluate about 200 possible solutions per second. The test problem described above has 22 components allocated to a maximum of five ECUs. In order to test the complete possible solution space, $5^{22} \approx 2 \cdot 10^{15}$ evaluations have to be performed. With the test computer, it would

Table 5.3.: *Pareto set gathered with a random run of one day*

solution	q_{space}	q_{ROM}	q_{costs}	q_{busload}	q_{energy}	q_{supplier}
{5333221532325452511144}	0.0	0.0	15.4575	0.9025	4.0	8.0
{1131145152322225534444}	0.0	0.0	25.3975	0.8275	4.0	7.0
{5414223231112245335553}	0.0	0.0	14.3525	0.9150	3.0	9.0
{4332321452221111534445}	0.0	0.0	23.8100	0.7650	3.0	8.0
{5321335531114454152223}	0.0	0.0	30.6750	0.7650	4.0	7.0
{1152132542223331354414}	0.0	0.0	34.0225	0.8275	3.0	7.0
{4111453323135532524244}	0.0	0.0	22.3600	0.8775	3.0	8.0
{2533551123334515214142}	0.0	0.0	19.6350	0.8400	4.0	9.0
{4241224251123351315445}	0.0	0.0	15.2425	0.9775	3.0	8.0
{1244225311415533225531}	0.0	0.0	16.3275	0.9400	3.0	8.0
{5221225111123344354534}	0.0	0.0	18.9075	0.8775	4.0	10.0
{5441424221153142253333}	0.0	0.0	18.8325	0.9625	4.0	7.0
{5534255223331214421411}	0.0	0.0	18.4725	0.8875	4.0	8.0
{1213252341115525434434}	0.0	0.0	17.6650	0.8875	4.0	9.0
{4243225543324152351411}	0.0	0.0	15.0875	0.9900	2.0	7.0
{1141212542224453135455}	0.0	0.0	20.0200	0.8900	3.0	7.0
{4311354551114435252232}	0.0	0.0	21.7525	0.7900	4.0	8.0
{1244224113434152155255}	0.0	0.0	13.9300	0.9900	4.0	8.0
{5144311522425515213333}	0.0	0.0	25.9800	0.8625	3.0	7.0
{5211223413134445413235}	0.0	0.0	13.4775	0.9525	5.0	8.0
{4535535412224412113334}	0.0	0.0	27.0525	0.9375	3.0	6.0

take about 400,000 years to evaluate all possible solutions. Even if the number of evaluations could be increased to 200,000 solutions per seconds, which is currently only thinkable by massive parallelisation or a signification improvement of the evaluation implementation, the result would be available only after 400 years. After a one day run or about $17 \cdot 10^6$ random evaluations, a Pareto set had been found as shown in Tab. 5.3. The solution s_{dom} dominates over 85 % of the randomly found solutions and is not dominated at all. Although, s_{costs} is not dominating a single one, the value for the objective costs is much better. Another interesting aspect is that only about 0.001897 % of the solutions were valid. It can be concluded that this random run of one day did not find good results for this application example.

5.4. Summary

In this chapter, an application example has been presented. Four different functions have been explained and specified. It has been shown that existing methods are not sufficient for getting a bunch of good trade-off solutions. In the course of this work, it is shown that it is possible to find much better solutions for this application example. In reality, the most allocation problems are more constrained. For example, there are rarely components that have a degree of freedom as high as in this example. Mostly, there are only two or three possible target ECUs. Furthermore, this approach is only thought for local optimisation problems and not intended for optimising the function allocation of the whole vehicle in one step. Thus, the application example can be seen as a kind of worst-case scenario as benchmark for the optimisation algorithms.

Part III.

Optimisation

6. Multi-Objective Optimisation of the Function Allocation

This chapter presents optimisation algorithms that can be applied in order to find an optimal function allocation. First, the usage of evolutionary algorithms is examined (see Sect. 6.1). In Sect. 6.2, ant colony optimisation is applied to the same problem. Many *parameters* $\mathfrak{p}$ are involved in the described optimisation algorithms like for example *population size* or *mutation rate*. The appendix contains a summary of the parameters of the evolutionary algorithm (see Tab. A.1) and the ant colony optimisation algorithm (see Tab. A.2). Section 6.3 explains, which parameters to tune in which way in order to improve the optimisation results. Finally, in Sect. 6.4, evolutionary algorithms and ant colony algorithms are compared. It is discussed, which algorithm should be preferred.

6.1. Optimisation of the Function Allocation with Evolutionary Algorithms

As shown in Sect. 2.1.2, SPEA2 gives a good basis for optimisation problems with a high number of objectives. Nevertheless, in SPEA2 several points are not defined and have to be adopted for this specific optimisation problem. Open points according to Alg. 2 (see p. 18) are the representation of a solution, the generation of the initial population, and the application of evolutionary operators. Furthermore, there is no idea given how to handle constraints.

6.1.1. Generation of the Initial Population

The first task according to Alg. 2 is the generation of an initial population. There are several possibilities to do this. In principle, any optimisation method can be used in order to generate the initial individuals.

However, a widely known method is just filling the initial population with random solutions. A random solution in this problem domain can be generated by allocating each component to a randomly chosen ECU.

Another approach is generating local search solutions. Local search algorithms make a local optimum choice at each iteration with the hope of finding

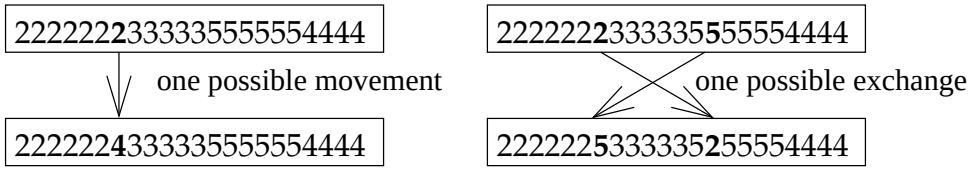

Figure 6.1.: *Examples for movement and exchange of components*

the global optimum. Unfortunately, due to this local optimum choice, there is a risk that the global optimum is not found. For the solution representation in this work, a local search optimisation algorithm according to a single objective is defined in Alg. 3 applying movement and exchange of single components. Figure 6.1 shows examples for a movement and an exchange for the solution $s = \{22222233333555554444\}$.[1]

Algorithm 3: Local Search Optimisation of a Solution According to a Single Objective

Input: A not optimised solution, an objective to optimise according to
1 **while** *improvement can be found* **do**
2 Try all movements of components to other ECUS;
3 Try all exchanges of the allocation of all pairs of components;
4 Apply movement or exchange with the most improvement in the given objective;
5 **end**
6 **return** *The local search optimised solution;*

In the evolutionary algorithm applied in this work, the population size $|P|$ has a fixed size. A mixture of both methods is applied in order to generate the initial population. First, according to each objective, one local search solution is generated. The rest of the population is filled with random solutions. The desired size of the population can be configured by the user. A parameter *population size factor* $\mathfrak{p}_{\text{PSF}}$ defines a factor that is multiplied with the number of objectives in order to determine the desired $|P|$. For example, with the initially used value $\mathfrak{p}_{\text{PSF}} = 4$, 25 % of the initial solutions are generated by local search and 75 % randomly: With the application example from Chap. 5 (6 objectives) and a $\mathfrak{p}_{\text{PSF}}$ of $\mathfrak{p}_{\text{PSF}} = 4$, the population size is 24. Thus, 6 solutions are generated by local search and 6 by local randomly.

[1] The notation of s is according to the definition in Sect. 5.3.

6.1.2. Evolutionary Operators

Since the realisation of evolutionary operators is mainly dependent on the representation of solutions, these operators have to be specifically defined for each representation. Standard types of evolutionary operators are for example mutation and recombination.

Zitzler et al. demonstrate in [LZT01] a dependency between the size of the solution representation and the necessary mutation rate. Therefore here, the resulting *mutation rate* p_M is defined by the size of the solution representation, indicated by the number of components $|s|$, and the parameter *normalised mutation rate* $\mathfrak{p}_{MR}$ with $p_M = \mathfrak{p}_{MR}/|s|$. Initially, a value between the values proposed by Zitzler et al. is chosen with $\mathfrak{p}_{MR} = 2$.[2] In the application example, this is resulting in a mutation rate of $p_M = 2/22 \approx 0.091$.

The mutation algorithm itself is depending on the representation. In this work, two different mutation operators are defined. With a probability of p_M, the assignment of a component is mutated. With a probability according to the parameter *move mutation rate* $\mathfrak{p}_{MMR}$, the so-called move mutation is performed instead of other mutation operations. The move mutation changes the assignment of the specific component to another randomly chosen ECU similarly to the local search described above.

Due to the possibly quite tight resource constraints, it makes sense to define an additional type of mutation operator that is called *exchange mutation*. This mutation operator exchanges the allocation of two randomly chosen components. With a probability of $1 - \mathfrak{p}_{MMR}$, the exchange mutation is performed instead of the move mutation. The exchange mutation randomly chooses another component and exchanges the allocated ECUs of both. Initially, the move mutation rate is set to $\mathfrak{p}_{MMR} = 0.5$. Both mutation operators are similar to the move and exchange during the generation of the initial solution population as shown in Fig. 6.1.

Another issue beside mutation is *recombination*. Similar to the exchange mutation, the *crossover operator* exchanges the assigned ECUs of the same component in two different randomly chosen solutions. With a probability of the *crossover rate* $\mathfrak{p}_{CR}$, the assigned ECU of each component is switched. In a first approach, the value $\mathfrak{p}_{CR}$ is set to $\mathfrak{p}_{CR} = 0.5$ in order to get offspring solutions that are composed of 50 % of each parent solution in average.

6.1.3. Modified Truncation Operator Preserving Boundary Solutions

Until now, the solution representation and operators are independent from SPEA2. In this section, a shortcoming of the original truncation operator in

[2] In [LZT01], values for $\mathfrak{p}_{MR}$ between 1 and 10 are proposed.

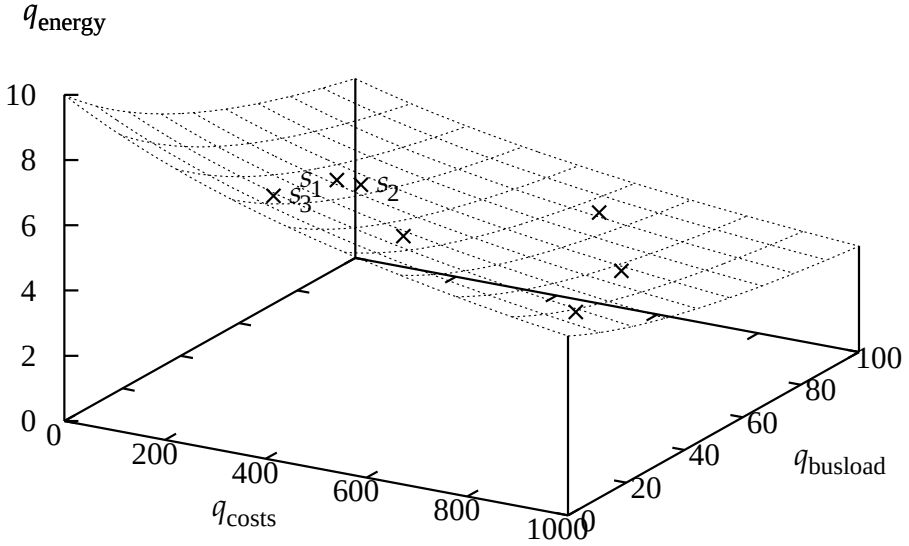

Figure 6.2.: *Example for solutions in a non-dominated set*

SPEA2 is explained and a proposal is presented how to overcome it.

The original SPEA2 truncation operator $\mathfrak{T}_{\text{orig}}$ (see Alg. 2, p. 18) is designed for keeping diversity and preserving boundary solutions from being removed. This works fine for optimisation problems with only two objectives. For three or more objectives, the counter example in Fig. 6.2 shows that boundary solutions can still be removed. This is not acceptable, since for example a cost minimal solution could be eliminated.

The effect of removing is demonstrated on the following example. Consider the quality ratings for the three optimisation criteria busload q_{busload}, energy consumption q_{energy}, and costs q_{costs}, which are all subject to minimisation. The current archive contains seven non-dominated solutions, which are all feasible, and the archive size N is six. Figure 6.2 shows the values of q_{busload}, q_{energy}, and q_{costs}, for these seven solutions. Now, one individual has to be chosen for truncation. The three interesting solutions are marked with s_1, s_2, and s_3. The quality rating for energy consumption is slightly falling from the origin while moving to higher values in the objectives costs and busload. The solution s_1 is not dominating s_2 because it is worse in the objective energy consumption.

The original truncation operator $\mathfrak{T}_{\text{orig}}$ chooses solutions with the shortest distance to the first neighbour. This results in solutions s_1 and s_2. The solution s_1 has the shortest distance to the second-nearest neighbour and the truncation operator chooses it for removal. Thus, the solution with the minimum costs is removed from the archive. This behaviour is different to the statement from Zitzler et al. [ZLT01] saying that the truncation operation prevents removal of boundary solutions.

The operator $\mathfrak{T}_{\text{orig}}$ is modified in order to overcome the mentioned problem

with $\mathfrak{T}_{\text{bound}}$ (see Alg. 4). The procedure getNotOptimalSolutions(A) returns all solutions from the archive that do not have the globally unique best value for at least one objective. That way, these solutions are not candidates for removal any more.

Algorithm 4: $\mathfrak{T}_{\text{bound}}$: Modified SPEA2 Truncation Operation not removing boundary solutions

Input: An archive A with non-dominated solutions s

```
1 s_T := getNotOptimalSolutions(A);
2 k := 1;                                   // Distance to the k-nearest neighbour
3 while |s_T| ≠ 1 do
4 |   s_T := getSolsWithNearestNeighbour(s_T, k, A) ;
5 |   k := k + 1;
6 end
7 return s_T;                               // exactly one solution to be truncated
```

If $\mathfrak{T}_{\text{bound}}$ is applied, it has to be ensured that the size of the archive $|A|$ is larger than the number of objectives. Otherwise, it can happen that each solution is the unique global optimum for a different objective and no solutions remain for removal. Thus, if a smaller archive size is used, $\mathfrak{T}_{\text{orig}}$ has to be applied. In the example in Fig. 6.2, $\mathfrak{T}_{\text{bound}}$ removes s_2 instead of s_1 first. Thus, the cost optimal solution is not truncated any more.

6.1.4. Modifications for Constrained Optimisation Problems

The problem of function allocation has the specific feature that often a high percentage of possible solutions are infeasible due to tight constraints. SPEA2 is not directly capable to deal with constrained problems. Several constraint handling techniques are introduced in Sect. 2.1.4.

It is shown that from existing constraint handling techniques, the modification of the domination relation is most promising. Especially, the domination operators developed by Deb $\mathfrak{d}_{\text{Deb}}(s_1, s_2)$ and Oyama $\mathfrak{d}_{\text{Oyama}}(s_1, s_2)$ are interesting (see Sect. 2.1.4). Deb does not describe how to measure the amount of constraint violation. In order to test the amount of constraint violation, just the number of constraint violations is used. A more detailed measure of the amount of constraint violation is very difficult, since different constraints would have to be compared in that case.

Additionally to the modifications of the domination relation by Deb and Oyama, in Sect. 2.1.4, another modification for solving constrained problems with SPEA2 is proposed by Hubley et al. [HZR03]. With this modified domina-

tion relation $\mathfrak{d}_{\text{Hubley}}(s_1, s_2)$, each feasible solution dominates all infeasible solutions. If the solutions s_1 and s_2 are both either feasible or infeasible, the original domination relation $\mathfrak{d}_{\text{orig}}$ (see Def. 1) is applicable.

For optimisation problems with multiple constraints, another improvement to $\mathfrak{d}_{\text{Hubley}}(s_1, s_2)$ is proposed that is called new domination relation $\mathfrak{d}_{\text{new}}(s_1, s_2)$.

Definition 5 *A solution s_1 is said to new-dominate $\mathfrak{d}_{\text{new}}(s_1, s_2)$ the other solution s_2, if any of the following conditions is true:*

- *s_1 has a smaller number of violated constraints than s_2.*
- *Solutions s_1 and s_2 have the same number of constraint violations and s_1 dominates ($\mathfrak{d}(s_1, s_2)$ according to Def. 1) s_2.*

Similar to $\mathfrak{d}_{\text{Deb}}$, it regards only the number of constraint violation. It additionally considers the severity of the constraint violation by applying the original domination operation, if two solutions have the same number of constraint violations.

Another algorithm contrary to the adaptation of the domination relation is leaving the domination relation untouched by applying $\mathfrak{d}_{\text{orig}}$. Instead, the constraints are modelled as objectives and the truncation operator is modified.

It has been shown by Vieira et al. [VAKV02], that transforming the n constraints into n more objectives is a promising approach. Now, the problem from (1.1) and (1.2) (see p. 11) can be reformulated as

$$\text{minimise } \mathbf{f} = \{f_1(s), f_2(s), ..., f_m(s), g_1(s), g_2(s), ..., g_n(s)\}^{\mathrm{T}} \tag{6.1}$$

It is important to note that if pure constraints are transformed into objectives, they should return a constant minimal value, if the constraint is not violated.

As already explained in Chap. 3, a large number of objectives and constraints exist for the presented application. A modified modelling approach is the basis for the implementations of the different objectives and optimisation algorithms. It is able to combine objectives and constraints. Some objectives are constraints at the same time. An example is busload, which is object of minimisation, but constrained at the same time.

All objectives with a constrained boundary value can be modelled as combined constraints/objectives as already described in Sect. 4.2.5. The number of constraints modelled as objectives can be reduced this way. More formalised, the set of constraints $\mathbf{g}_f$, that are objectives at the same time can be determined with

$$\mathbf{g}_f = \{g_j \in (f_i, g_j) | \exists c_i, f_i - c_i = g_j, f_i \in \mathbf{f}, g_j \in \mathbf{g}\}, \tag{6.2}$$

with all c_i constant values as boundaries and $\mathbf{f}$ and $\mathbf{g}$ according to (1.1) and (1.2). Now, the optimisation problem can be re-formulated with fewer objectives as

$$\text{minimise } \mathbf{f} = \{f_1(s), f_2(s), ..., f_m(s), \mathbf{g} \setminus \mathbf{g}_f\}^{\mathrm{T}}. \tag{6.3}$$

For application of this new objective modelling approach, a modified elitism (or especially the truncation operator for SPEA2) is necessary in order to get satisfying results. Horn et al. published the *Niched Pareto Genetic Algorithm* (NPGA, [HNG94]). The elitism of NPGA has been modified by Vieira et al. [VAKV02]. For SPEA2, these modifications are presented here.

In general, at the beginning of an evolutionary algorithm, only few solutions in the archive are non-dominated. If all constraints are modelled as objectives, constraint-violating solutions are not removed automatically. After a number of generations, an increasing number of non-dominated solutions are found.

At this time, the truncation operator $\mathfrak{T}_{\text{bound}}$ removes feasible solutions as well as infeasible. If only small portions of the search space produce feasible solutions, many feasible solutions are removed while keeping infeasible solutions.

In order to overcome this problem, a new truncation operator $\mathfrak{T}_{\text{new}}$ is introduced in Alg. 5. The specific of the operator is the procedure getSolutionsWithMostConstraintViolations(A) (see line 2). This procedure finds solutions with the highest number of constraint violations. Only solutions found by this procedure are candidates for removal.

Algorithm 5: $\mathfrak{T}_{\text{new}}$: New SPEA2 Truncation Operation for Constraints Modelled as Objectives

Input: An archive A with non-dominated solutions s

```
1  if constraint violating solutions found then
2  |  s_T := getSolutionsWithMostConstraintViolations(A);
3  |  return selectionOperatorForInfeasibleSolutions(s_T);
4  else
5  |  s_T := getNotOptimalSolutions(A);
6  |  k := 1;                          // Distance to the k-nearest neighbour
7  |  while |s_T| ≠ 1 do
8  |  |  s_T := getSolsWithNearestNeighbour(s_T, k, A);
9  |  |  k := k + 1;
10 |  end
11 end
12 return s_T;                         // exactly one solution to be truncated
```

In the next step (see line 3), exactly one solution is picked from these candidates. It is the solution with the biggest distance to the nearest neighbour from

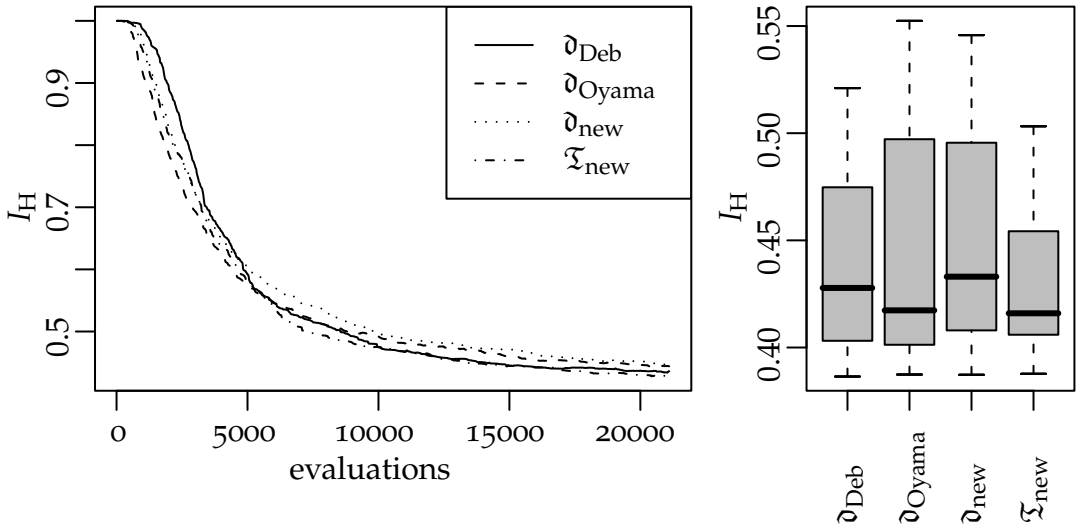

Figure 6.3.: *Influence of the constraint handling technique on the optimisation result with 20 runs each*

the set of feasible solutions. With other words, the infeasible solution with the biggest distance to the feasible region is picked for removal first. If no feasible solutions are found yet, $\mathfrak{T}_{\text{bound}}$ is applied. For each objective, the range between smallest and largest quality rating value of all feasible solutions in the current Pareto set is normalised to 1 in order to measure the distance.

Similar to $\mathfrak{d}_{\text{new}}$, $\mathfrak{T}_{\text{new}}$ first removes solutions that violate the most constraints. In contrast to $\mathfrak{d}_{\text{new}}$, $\mathfrak{T}_{\text{new}}$ first eliminates solutions from the set of most constraint violating solutions that are far from the feasible region. SPEA2 with $\mathfrak{d}_{\text{new}}$ tends to remove solutions from this set in regions, which are crowded, but potentially very near to the feasible region.

In order to decide, which constraint handling technique to apply, all four constraint handling techniques are compared. The optimisation problem presented in Chap. 5 is taken as application example. The parameters are set to the values as described before.

In Fig. 6.3, the results of the comparison are shown. In the comparison, the hypervolume indicator I_{H} as described in Sect. 2.1.5 is used. On the left-hand side, the mean of all 20 test runs with each constraint handling technique over the number of evaluations is displayed.

On the right-hand side, *box plots* of all Pareto sets of the last generation of each of the 20 test runs are compared. The grey filled box contains the middle fifty percent of all values. The horizontal bold line marks the *median value*. The whiskers at the top and the bottom mark the worst/best-case values. Values are called *outliers* and marked with a circle, if the distance of the value to the box is

more than one and a half the height of the box (see for example Fig. 6.9, p. 108). A more detailed explanation of box plots can be found in [MSVCS05].

It can be seen in Fig. 6.3 that—applied to the application from Chap. 5—all constraint handling techniques seem to have similar performance in finding good solution sets. Applying $\mathfrak{T}_{\text{new}}$ yields in the best results although the difference is not significant. The box plots show that both the median hypervolume and the worst-case hypervolume are minimal in comparison to the other constraint handling techniques. For the further investigations, $\mathfrak{T}_{\text{new}}$ is used.

6.2. Optimisation of the Function Allocation with Ant Colony Optimisation

State of the art ant colony optimisation algorithms have already been shown in Sect. 2.1.3. This section presents adaptations of the known ACO algorithms enabling them to solve the problem of function allocation in vehicle networks. This section is based on the work of Bickel [Bic06].

6.2.1. Problem Specific Pheromone Information

The first step during the adaptation of ACO to the function allocation problem is finding a suitable definition of the pheromone information. Especially, the interpretation of the pheromone values is important. Applied to the travelling salesman problem, the pheromone information represents the probability for choosing the next city. A first, naiv approach for the function allocation domain is the usage of a 2-dimensional pheromone matrix. The columns of the matrix represent the components and the rows the ECUs. The pheromone values $\tau_{c,e}$ in the matrix represent the probability that a component c is allocated to the according the ECU e.

This approach has a major drawback. It is normal for multi-objective problems, that there are several optimal solutions in the Pareto front. During the construction of solutions by the ants, only the information within the pheromone matrix is used. Therefore, the pheromone matrix must represent the past ant generations. The proposed naiv approach can only ensure this for a single solution in a proper way. If there are several solutions, the values kept in the naiv pheromone matrix just represent the number of solutions with a specific component allocated to a specific ECU. It does not store which combinations of component allocations lead to good solutions.

In order to improve this naiv approach, beside the pheromone matrix, a correlation matrix is introduced. The correlation matrix is also 2-dimensional, but

	c_1	c_2	c_3
e_1	τ_{c_1,e_1}	τ_{c_2,e_1}	τ_{c_3,e_1}
e_2	τ_{c_1,e_2}	τ_{c_2,e_2}	τ_{c_3,e_2}
e_3	τ_{c_1,e_3}	τ_{c_2,e_3}	τ_{c_3,e_3}
e_4	τ_{c_1,e_4}	τ_{c_2,e_4}	τ_{c_3,e_4}

	c_1	c_2	c_3
c_1	γ_{c_1,c_1}	γ_{c_2,c_1}	γ_{c_3,c_1}
c_2	γ_{c_1,c_2}	γ_{c_2,c_2}	γ_{c_3,c_2}
c_3	γ_{c_1,c_2}	γ_{c_2,c_3}	γ_{c_3,c_3}

Figure 6.4.: *Example for pheromone matrix (left) and correlation matrix (right) for three components and four ECUs*

contains all components over all components. The correlation values γ_{c_1,c_2} represent the probability that two specific components are allocated together onto the same ECU. Figure 6.4 shows an example of the pheromone and the correlation matrices for three components and four ECUs. Redundant values are printed in grey.

The probability $p_{c,e}$ for choosing a certain ECU e as allocation target for a component c, can be determined with

$$p_{c,e} = \frac{\tau_{c,e} \prod\limits_{\forall \gamma \in \gamma'} \gamma}{\sum\limits_{\forall e} \tau_{c,e} \prod\limits_{\forall \gamma \in \gamma'} \gamma} \tag{6.4}$$

where γ' contains all γ_{c_1,c_2} where $c_1 = c$ and c_2 is already allocated on e. In contrast to γ, which is containing all γ, γ' only contains the γ according to the components that are already allocated on the ECU.

Another proposal from Dorigo and Gambardella [DG97] improves the exploitation at the expense of exploration for the travelling salesman problem. It is applied as well in order to further improve the ACO algorithm. With a parametrisable probability q, the next city is chosen according to the highest product $\tau_{ij}^{\alpha}\eta_{ij}^{\beta}$ of all neighbour cities. In this application, the ECU e, chosen for allocation of component c, is determined with

$$e = \begin{cases} \max\limits_{\forall e} \tau_{c,e} \prod\limits_{\forall \gamma \in \gamma'} \gamma & \text{, if } q \leq \mathfrak{p}_{\text{ERP}} \\ \varepsilon & \text{, otherwise} \end{cases}, \tag{6.5}$$

where $\mathfrak{p}_{\text{ERP}}$ is an *exploitation random parameter* in the range $[0,1]$, q a random number in $[0,1]$, and ε a random ECU selected according to the distribution given in (6.4). With a probability of $\mathfrak{p}_{\text{ERP}}$, this algorithm chooses the currently best known ECU without considering the other paths.

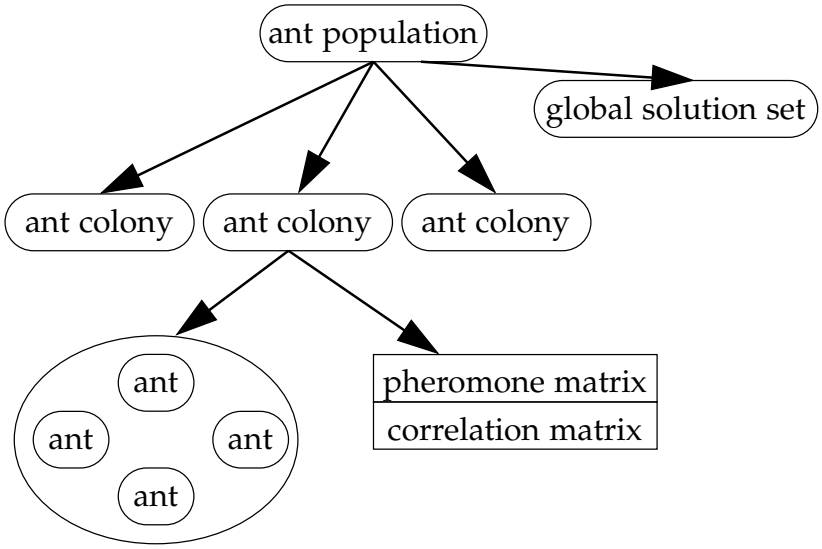

Figure 6.5.: *Hierarchy of the different groups of ants*

6.2.2. Usage of Multiple Colonies

The function allocation problem is a multi-objective optimisation problem. In Sect. 2.1.3, methods for handling multiple objectives in ACO are presented. The algorithm shown here is based on the work of IREDI et al. [IMM01]. Several colonies are created, but not necessarily one for each objective. The number of objectives is parameterised with the parameter *number of colonies per objective* $\mathfrak{p}_{\text{NCPO}}$. Additionally, a parameter called *sub population size factor* $\mathfrak{p}_{\text{SPF}}$ is involved for configuring the number of solutions in each sub population as multiple of the number of objectives.

Figure 6.5 shows the hierarchy of different groups of ants. The *ant* itself has the lowest rank. Its task is to generate a complete solution. Using the pheromone and correlation matrices, it allocates each component to a specific ECU. The *ant colony* is the next group of ants. It controls the individual ants and administrates the pheromone and correlation values. At the highest level, the *ant population* can be found. It collects the generated solutions, evaluates and compares them, and initiates the update of the matrices. The ant population stores the best solutions in a *global solution set* and provides them to the colonies for the matrix update. The definition of what the *best* solutions are is given below.

6.2.3. Update Strategies and Constraint Handling

This section shows strategies for updating the pheromone and correlation matrices. The first performed step is called evaporation. The evaporation reduces the pheromone and correlation values by a certain percentage. The according parameters are called *pheromone evaporation rate* $\mathfrak{p}_{\mathrm{PER}}$ and *correlation evaporation rate* $\mathfrak{p}_{\mathrm{CER}}$. Applying these parameters, (2.4) can be rewritten as

$$\tau_{c,e}(t+1) = (1-\mathfrak{p}_{\mathrm{PER}})\tau_{c,e}(t) + \Delta\tau_{c,e}(t,t+1) \tag{6.6}$$

and

$$\gamma_{c_1,c_2}(t+1) = (1-\mathfrak{p}_{\mathrm{CER}})\gamma_{c_1,c_2}(t) + \Delta\gamma_{c_1,c_2}(t,t+1). \tag{6.7}$$

In the matrices, the information of the past generations are stored and passed to the following generations. Only the *best* solutions update the matrices and are used for the determination of the values $\Delta\tau_{c,e}(t,t+1)$ and $\Delta\gamma_{c_1,c_2}(t,t+1)$ as described later. Within the travelling salesman problem, the best solution is the one with the shortest way. The function allocation problem includes several objectives and can involve several *best* solutions. As already described in Sect. 2.1.1, the non-dominated set or Pareto front is an analogue to the best solution for multi-objective optimisation problems.

The management of the *global solution set* (see Fig. 6.5) is a task similar to the management of the archive within SPEA2 as already shown in Sect. 2.1.2. In this work, the method from SPEA2 is transferred for ACO. Similarly, a parameter *population size factor* $\mathfrak{p}_{\mathrm{PSF}}$ denotes the number of solutions in the global solution set as factor to be multiplied with the number of objectives.

For the update of the pheromone and correlation matrices, this global solution set $\mathbf{s}_{\mathrm{g}}$ as well as the best ants (feasible $\mathbf{s}_{\mathrm{f}}$ and non-dominated $\mathbf{s}_{\mathrm{n}}$ solutions) from the own ant colony are included. As shown in the overview in Fig. 6.6, the solutions from the three sets global solution set, local feasible solutions, and local non-dominated solutions update the matrices with the update parameters *update factor global solution set* $\mathfrak{p}_{\mathrm{UGS}}$, *update factor local feasible solutions* $\mathfrak{p}_{\mathrm{ULF}}$, and *update factor local non-dominated solutions* $\mathfrak{p}_{\mathrm{ULN}}$. The values $\Delta\tau_{c,e}(t,t+1)$ and $\Delta\gamma_{c_1,c_2}(t,t+1)$ in (6.6) and (6.7) are determined at any time with

$$\Delta\tau_{c,e} = \mathfrak{p}_{\mathrm{UGS}}\mathrm{p}(c \overset{\mathrm{asg}}{\rightarrow} e, \mathbf{s}_{\mathrm{g}}) + \mathfrak{p}_{\mathrm{ULF}}\mathrm{p}(c \overset{\mathrm{asg}}{\rightarrow} e, \mathbf{s}_{\mathrm{f}}) + \mathfrak{p}_{\mathrm{ULN}}\mathrm{p}(c \overset{\mathrm{asg}}{\rightarrow} e, \mathbf{s}_{\mathrm{n}}) \tag{6.8}$$

and

$$\Delta\gamma_{c_1,c_2} = \mathfrak{p}_{\mathrm{UGS}}\mathrm{p}(c_1,c_2,\mathbf{s}_{\mathrm{g}}) + \mathfrak{p}_{\mathrm{ULF}}\mathrm{p}(c_1,c_2,\mathbf{s}_{\mathrm{f}}) + \mathfrak{p}_{\mathrm{ULN}}\mathrm{p}(c_1,c_2,\mathbf{s}_{\mathrm{n}}), \tag{6.9}$$

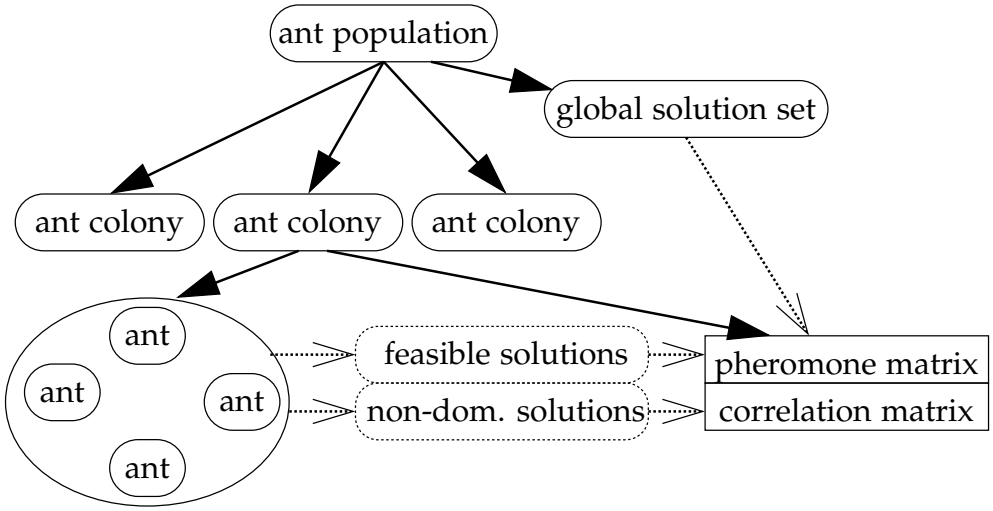

Figure 6.6.: *Ant groups updating the pheromone and correlation matrices*

where $\mathrm{p}(c \overset{\text{asg}}{\rightarrow} e, \mathbf{s})$ denotes the percentage of solutions in the solution set $\mathbf{s}$, where c is allocated to e and $\mathrm{p}(c_1, c_2, \mathbf{s})$ the percentage of solutions, where c_1 and c_2 are allocated to the same ECU. A colony only considers those solutions from the global set that are generated by their own previously. This is called *update-by-origin* method.

The last important decision is the usage of the *update-by-step* algorithm. The update-by-step algorithm decreases the value in the matrices after each allocated component. This way, the following ants will less likely follow the same path. The according values in the pheromone and correlation matrices are multiplied with a parameter called *factor for update by step* $\mathfrak{p}_{\mathrm{FUP}}$.

6.3. Adjustment of the Optimisation Parameters

Evolutionary algorithms and ant colony optimisation have been examined for solving the function allocation problem in the previous sections. Several parameters are involved in these algorithms. There are two different possibilities for setting the parameters: Static values can be determined or the parameters can be adjusted dynamically during the optimisation run. A survey of methods controlling parameters dynamically can be found in [EHM99]. Static methods allow a better insight in the algorithms. Thus, in order to compare evolutionary algorithms and ant colony optimisation, fixed values are used in this work. However, dynamic parameter control can additionally be applied at any time.

So far, in this work the parameters are fixed to values obtained from the literature or experiments. However, the parameters have potentially big influence

on each other and the optimisation result. That means, changing one parameter might only yield in a good optimisation result, if another parameter is adjusted as well. In this section, a method using orthogonal arrays is introduced, which is applied for finding the optimal combination of parameter settings.

6.3.1. Orthogonal Arrays

Design of experiments is a general term for methods reducing the number of necessary experiments for the determination of a good parameter set in any kind of process or problem. One main property of these methods is the fact that not every parameter is optimised singularly. Rather, all parameters are tested simultaneously in experiments. One possible method for the design of these experiments is the application of *Orthogonal Array*s (OA). TAGUCHI applied OAs in his method called *robust design* [Tag86, UD91], where OAs describe the experiments to be performed.

An orthogonal array $L(n,l)$ is a matrix with one out of a limited number of so-called levels l in each field. In this work, only orthogonal arrays with a *strength* of 2 are used, since they hold a smaller number of rows. For orthogonal arrays with a strength of 2, all combinations of 2 columns have the same property: All possible combinations of levels occur in these columns—and they occur with the same frequency. The variable n denotes the number of columns of the orthogonal array.

There are methods for constructing orthogonal arrays for example by transforming so-called *latin square*s. Additionally, there exist methods constructing near-orthogonal arrays [Ngu96]. For the application in this work, a library is used that can be found in [Sloo6]. Table 6.1 shows an example for an orthogonal array $L(4,3)$ with 4 columns, 3 levels (0, 1, and 2), and a strength of 2. For the application to the determination of optimisation parameters, each column is used for another parameter and each level represents a parameter setting for the specific parameter.

6.3.2. Determination of the Parameters for Evolutionary Algorithms

The described orthogonal array method is applied to the determination of the parameters of the evolutionary algorithm. Several parameters have been introduced in Sect. 6.1. For optimising them with an orthogonal array, several levels are assigned to all of them as shown in Tab. 6.2. For the parameters p_{PSF}, p_{MR}, p_{MMR}, and p_{CR}, values that make sense are guessed in a first step.

In the next step, a suitable orthogonal array is determined. A suitable orthogonal array $L(6,5)$ from [Sloo6] includes 25 different experiments. It can be found in the appendix in Tab. B.1. The last two columns are not used for this

Table 6.1.: *Orthogonal Array $L(4,3)$ [Sloo6]*

0	0	0	0
0	1	1	2
0	2	2	1
1	0	1	1
1	1	2	0
1	2	0	2
2	0	2	2
2	1	0	1
2	2	1	0

Table 6.2.: *Assignment of levels to concrete parameter values for EA*

level	$\mathfrak{p}_{\text{PSF}}$	$\mathfrak{p}_{\text{MR}}$	$\mathfrak{p}_{\text{MMR}}$	$\mathfrak{p}_{\text{CR}}$
0	2	0.2	0.00	0.01
1	3	0.5	0.25	0.05
2	4	1.0	0.50	0.10
3	5	1.5	0.75	0.25
4	6	2.0	1.00	0.50

parameter determination and removed resulting in the $L(4,5)$ OA applied in Tab. 6.3. For each experiment, 25 test runs are performed in order to be able to draw statistically reliable conclusions. The population is reduced to a size of 12 using the truncation operator before calculating the hypervolume indicator. The population size is exactly 12 if the parameter $\mathfrak{p}_{\text{PSF}}$ is set to the smallest value ($\mathfrak{p}_{\text{PSF}} = 2$ according to Tab. 6.2). Thus, the positive effect of a Pareto front containing more solutions is compensated.

Table 6.3 shows a summary of the results. On the top part of the table, the mean values (avg. I_{H}) of all 20 test runs are denoted beside the according experiments. The optimisation is stopped after 5000 evaluations in each test run. On the lower part of the table, the mean values of all avg. I_{H} are shown. For each level, only the avg. I_{H}'s are considered that are determined in an experiment with the according level setting of the parameter. The lowest value for each parameter marks the level to choose for an optimal performance. For example, the value 0.5696 for level 0 and $\mathfrak{p}_{\text{PSF}}$ is determined by averaging the avg. I_{H}'s of experiment 0...5. A graphical representation for easier orientation is shown in

Table 6.3.: *Determination of the parameters of evolutionary algorithms using an orthogonal array $L(4,5)$ with 20 runs per experiment*

experiment #	$\mathfrak{p}_{PSF}$	$\mathfrak{p}_{MR}$	$\mathfrak{p}_{MMR}$	$\mathfrak{p}_{CR}$	avg. I_H
0	0	0	0	0	0.5412
1	0	1	2	4	0.5458
2	0	2	3	1	0.5747
⋮	⋮	⋮	⋮	⋮	⋮
22	4	2	4	0	0.6130
23	4	3	2	1	0.6125
24	4	4	0	4	0.6255
level	$\mathfrak{p}_{PSF}$	$\mathfrak{p}_{MR}$	$\mathfrak{p}_{MMR}$	$\mathfrak{p}_{CR}$	
0	0.5696	**0.5437**	0.5669	0.5766	
1	0.5709	0.5675	**0.5427**	0.5731	
2	0.5650	0.5616	0.5624	**0.5623**	
3	**0.5554**	0.5844	0.5731	0.5672	
4	0.5948	0.5985	0.6106	0.5765	

Fig. 6.7.

With the orthogonal array method, the following parameter settings can be determined: $\mathfrak{p}_{PSF} = 5$, $\mathfrak{p}_{MR} = 0.2$, $\mathfrak{p}_{MMR} = 0.25$, $\mathfrak{p}_{CR} = 0.1$. In order to verify the results, another experiment with the optimal parameter set is executed. The mean value of the hypervolume indicator of 100 test runs is 0.5132.[3] As expected, this value is noticeable better than all mean values shown in Fig. 6.7.

[3] A number of 100 test runs is performed, since every value in Fig. 6.7 is the mean value of 100 single runs as well.

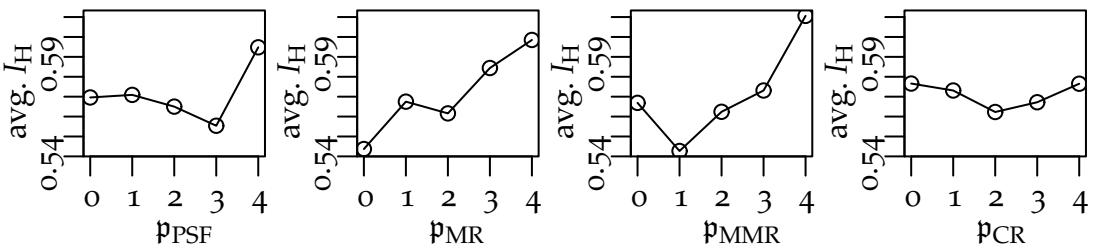

Figure 6.7.: *Influence of the different parameters on the performance of the EA*

Table 6.4.: *Assignment of levels to concrete parameter values for ACO (first run)*

level	$\mathfrak{p}_{\text{ERP}}$	$\mathfrak{p}_{\text{NCPO}}$	$\mathfrak{p}_{\text{SPF}}$	$\mathfrak{p}_{\text{PER}}$	$\mathfrak{p}_{\text{CER}}$	$\mathfrak{p}_{\text{PSF}}$	$\mathfrak{p}_{\text{UGS}}$	$\mathfrak{p}_{\text{ULN}}$	$\mathfrak{p}_{\text{ULF}}$	$\mathfrak{p}_{\text{FUP}}$
0	0.1	0.33	2	0.01	0.01	2	0.001	0.001	0.001	0.90
1	0.3	0.66	4	0.05	0.05	4	0.005	0.005	0.005	0.95
2	0.5	1.00	6	0.10	0.10	6	0.010	0.010	0.010	1.00

Table 6.5.: *Mean values of all experiments for each level for ACO with twelve test runs per experiments (first run)*

level	$\mathfrak{p}_{\text{ERP}}$	$\mathfrak{p}_{\text{NCPO}}$	$\mathfrak{p}_{\text{SPF}}$	$\mathfrak{p}_{\text{PER}}$	$\mathfrak{p}_{\text{CER}}$	$\mathfrak{p}_{\text{PSF}}$	$\mathfrak{p}_{\text{UGS}}$	$\mathfrak{p}_{\text{ULN}}$	$\mathfrak{p}_{\text{ULF}}$	$\mathfrak{p}_{\text{FUP}}$
0	**0.492**	0.542	0.549	0.566	0.632	**0.494**	0.562	**0.515**	0.511	**0.496**
1	0.534	**0.505**	**0.521**	**0.514**	0.505	0.527	**0.516**	0.518	0.597	0.604
2	0.585	0.564	0.541	0.531	**0.475**	0.590	0.534	0.578	**0.503**	0.511

This demonstrates the improvement of the EA due to the optimised parameters.

Another interesting point is the fact that the mutation rate is low in comparison to the results obtained by Laumanns et al. in [LZT01]. The explanation lies in the different measurement method. In this work, the number of objective evaluations instead of the number of generations is consulted for measurement. Applying low mutation rates, there are a number of solutions not changed at all inducing the mentioned difference.

6.3.3. Determination of the Parameters for Ant Colony Optimisation

Similarly to the EA, this section explains the determination of the parameters for the ant colony optimisation algorithm. The first step is the selection of a suitable orthogonal array. Totally, there are ten parameters to be configured. In order to keep the number of necessary experiments low, for ACO only three levels are chosen. An according orthogonal array $L(13, 3)$ with 27 experiments is shown in the appendix in Tab. B.2. The last three columns are ignored, since only ten parameters are optimised. The parameters and proposals for the levels for the first run are shown in Tab. 6.4. Table 6.5 shows the results of twelve runs for each experiment. Thus, each value in Tab. 6.5 is the mean value of $27/3 \times 12 = 108$ single runs.

The parameter $\mathfrak{p}_{\text{CER}}$ has the main influence on the optimisation result since it shows both the lowest and highest average hypervolume indicator value. In

Table 6.6.: *Assignment of levels to concrete parameter values for ACO (second run)*

level	$\mathfrak{p}_{ERP}$	$\mathfrak{p}_{NCPO}$	$\mathfrak{p}_{SPF}$	$\mathfrak{p}_{PER}$	$\mathfrak{p}_{CER}$	$\mathfrak{p}_{PSF}$	$\mathfrak{p}_{UGS}$	$\mathfrak{p}_{ULN}$	$\mathfrak{p}_{ULF}$	$\mathfrak{p}_{FUP}$
0	0.05	0.50	3	0.03	0.08	1	0.003	0.0005	0.001	0.90
1	0.10	0.66	4	0.05	0.12	2	0.005	0.0010	0.005	0.95
2	0.15	0.83	5	0.07	0.16	4	0.007	0.0020	0.010	1.00

Table 6.7.: *Mean values of all experiments for each level for ACO with twelve test runs per experiments (second run)*

level	$\mathfrak{p}_{ERP}$	$\mathfrak{p}_{NCPO}$	$\mathfrak{p}_{SPF}$	$\mathfrak{p}_{PER}$	$\mathfrak{p}_{CER}$	$\mathfrak{p}_{PSF}$	$\mathfrak{p}_{UGS}$	$\mathfrak{p}_{ULN}$	$\mathfrak{p}_{ULF}$	$\mathfrak{p}_{FUP}$
0	**0.477**	**0.465**	0.478	0.486	0.486	**0.479**	0.486	0.486	0.480	**0.475**
1	0.484	0.477	**0.477**	0.481	**0.475**	0.480	0.483	0.484	0.484	0.484
2	0.479	0.498	0.485	**0.472**	0.478	0.481	**0.471**	**0.469**	**0.475**	0.480

the second iteration, this parameter is adjusted as shown in Tab. 6.6. The ranges of the most of the other parameters are modified as well in order to approach the optimal settings. The values gathered with the second run can be found in Tab. 6.7. According to these results, the optimal parameters are marked bold in the table. The mean value of 100 runs with these parameter settings is 0.4575. This value is lower than all values in Tab. 6.7. The parameter $\mathfrak{p}_{PSF}$ does not seem to have noticeable influence since the distance between the values is very small. It is set to $\mathfrak{p}_{PSF} = 4$ contrary to Tab. 6.7, which is indicating a value of $\mathfrak{p}_{PSF} = 1$. The other parameters are set to $\mathfrak{p}_{ERP} = 0.05$, $\mathfrak{p}_{NCPO} = 0.5$, $\mathfrak{p}_{SPF} = 4$, $\mathfrak{p}_{PER} = 0.07$, $\mathfrak{p}_{CER} = 0.12$, $\mathfrak{p}_{UGS} = 0.007$, $\mathfrak{p}_{ULN} = 0.002$, $\mathfrak{p}_{ULF} = 0.01$, and $\mathfrak{p}_{FUP} = 0.90$.

In this context, another idea is tested. Some colonies might drift in regions where no or only few good solutions are found at all. An extension to the classical *update-by-origin* approach (see Sect. 6.2.3) is introduced here in order to overcome this drawback. In the classical approach, each solution in the Pareto front stores a reference to the colony from which it origins. In the new approach, this reference is re-linked. In the first step, the solutions are sorted in stacks according to their origin colonies. As long as the sizes of the largest and the smallest stack differ more the two, one solution from the largest stack is moved to the smallest stack by re-linking the reference to the origin colony. Figure 6.8 shows a comparison of the classical and the equalised approach. Beside two outliers, the equalised approach performs better. From this point, the equalised approach is applied.

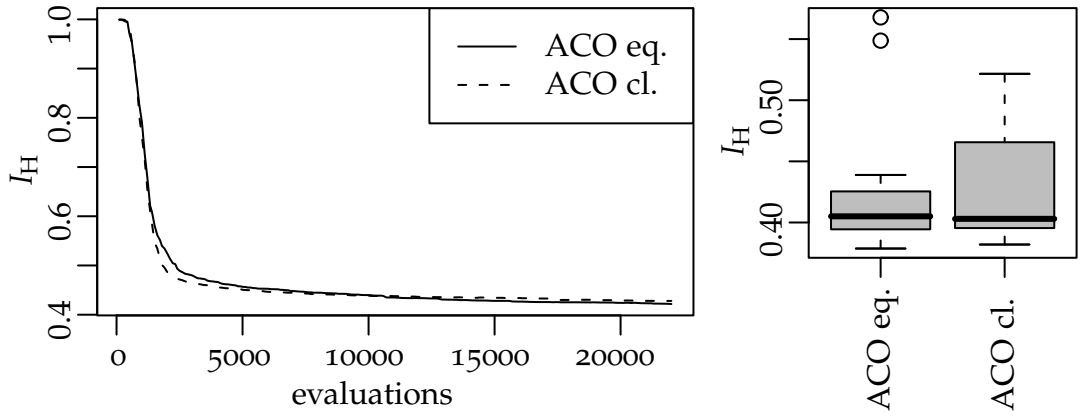

Figure 6.8.: *Comparison of the classical update-by-origin approach (ACO cl.) with equalised update-by-origin (ACO eq.) approach (twenty test runs)*

6.4. Summary

In this section, the results of this chapter are summarised. First, a comparison of evolutionary algorithm and ant colony optimisation is performed. Furthermore, an analysis of the obtained solutions is performed. The solutions are compared to the ones obtained in Sect. 5.3.

6.4.1. Comparison of Evolutionary Algorithm and Ant Colony Optimisation

The evolutionary algorithm and ant colony optimisation are compared in Fig. 6.9. Twenty test runs are performed for each optimisation algorithm. The parameters are set according to optima found in Sect. 6.3.

It can be seen that the ACO algorithm is faster in finding the good solution sets at the beginning of the runs. After a while, almost no better solutions are found by the ACO algorithm anymore. While the EA is slow in comparison to the ACO at the beginning, it is continuously improving in later phases. The solutions obtained at the end have very similar hypervolume values for both approaches. There is no noticeable difference in the quality of the obtained solution sets.

Concluding, at the moment the ACO is preferred instead of the EA, due to its better performance at the beginning of the optimisation runs. There seems to be further potential in the development of the ACO and the ideas provided in this work might help future researchers for developing ACOs with an even better performance.

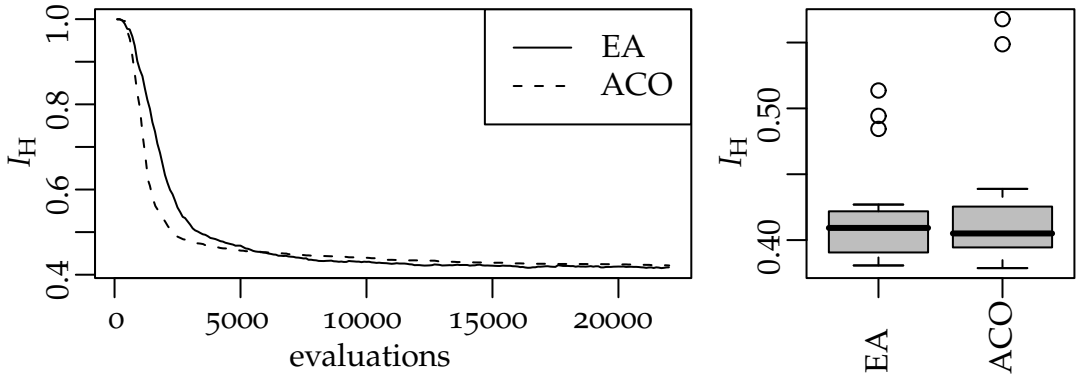

Figure 6.9.: *Comparison of evolutionary algorithm and ant colony optimisation with optimised parameters (twenty test runs)*

6.4.2. Analysis of Obtained Solutions

In order to illustrate the benefit of the optimisation for an OEM, the results are compared against the solutions obtained by existing methods in Sect. 5.3. Only two of the four dimensions—busload and costs—are displayed in Fig. 6.10. Three Pareto sets are shown: The results of the 24 hour random search (see Tab. 5.3), the manual optimisation, and an EA run with a hypervolume of $I_H \approx 0.39$. It is interesting that an EA run takes only approximately 10 minutes on the same computer. The solutions determined by the optimisation dominate all solutions determined in Chap. 5. The solutions determined by the random search are much worse than the optimised ones.

6.4.3. Conclusion

In this chapter, two different optimisation algorithms have been developed for solving the problem of function allocation in the automotive domain. Several modifications to existing EAs are proposed. Furthermore, a new ACO algorithm is presented outperforming the EA. For a quite complex optimisation problem and for both algorithms, the optimal parameters are determined applying orthogonal arrays. Thus, for optimisation similar or easier than the application example, it can be concluded that the method with the proposed parameters is sufficient to produce a set of architecture proposals.

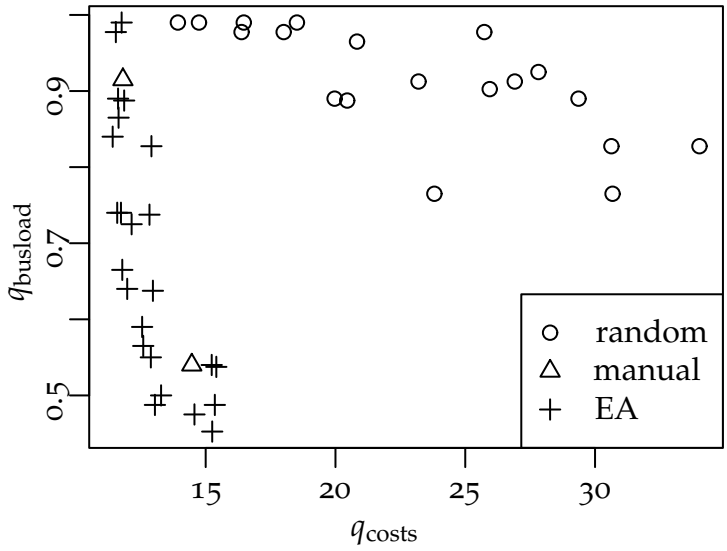

Figure 6.10.: *Comparison of existing methods and results of optimisation*

7. Optimisation of Variants of Control Units

In the last chapter, it has been shown how a set of optima for the allocation of components to ECUs—the Pareto front—can be found. This is called the *outer optimisation* (see also Sect. 4.2.8). In this chapter, an *inner optimisation* is introduced. An inner optimisation is an optimisation that has to be performed in order to calculate the quality rating for a solution proposed by the outer optimisation. Variant optimisation belongs to the objective *costs*, which is already described in Sect. 3.5. It can often not be performed together with the outer optimisation, due to the complexity of this inner optimisation problem. Thus, it is done after the outer optimisation or together with the last generations of the outer optimisation. Variant optimisation is explained in this own chapter because the results that can be gathered are promising on their own. The work presented in this chapter is based on the work of Kollert [HKKK05, HK05, Kol05].

As already explained in Sect. 1, the number of functions in modern cars is rising. There is a trend to provide increasingly personalised cars to the customer. This manifests not only in a growing number of new derivatives of cars. Additionally, a rising amount of extra equipment is offered to the customer in arbitrary combinations.

The number of new ECUs is not growing with the same speed as the number of new functions. There is even a trend to reduce the number of ECUs due to costs and complexity. The rising performance of micro-controllers allows the integration of an increasing number of functions together into single ECUs. Therefore, an increasing number of functions on singles ECUs can be ordered as optional equipment by the customer. In order to reduce costs, variants of the same ECUs are differing in hardware and software. Variants are created for instance by not populating sections of the circuit board. A variant can only save a share of the costs of the whole component. Component costs can optionally include proportionate values for circuit board and micro-controller resource. These cannot be saved by not populating sections of the circuit board.

Haubelt et al. [HTRE02] and Richter et al. [RZE+99] describe a representation of function variants. Their representation enables to describe variants differing in the selection of components and in the function networks for supporting different products. Thereby, a focus is on the verification of functional

correctness. In this work, an approach is shown considering customer orders and hardware mounting options of the ECU.

Each additional ECU variant causes expenses for logistics and development. Therefore, a cost optimal and preferably low number has to be found. Today, the process for the determination of variants is not automated. Non-technical dependencies like packages of equipment or commonly ordered combinations of equipment can only partially taken into account.

This chapter describes how the cost-optimal variants of an ECU can be determined. In Sect. 7.1, the problem is described more in detail. A mathematical problem description is presented in Sect. 7.2, before the transformation of the variant combination problem into a warehouse location problem is shown (see Sect. 7.3). Section 7.4 introduces an approach for reducing the problem complexity. Algorithms for solving the variant combination problem are presented in Sect. 7.5. Finally, the cost reduction potential of the variant optimisation is summarised in Sect. 7.6.

7.1. Description of the Variant Combination Problem

This section gives an overview on the *Variant Combination Problem* (VCP). The goal is to determine the average ECU costs $\mathfrak{c}_e$ originally defined in (3.1), see p. 40, under consideration of ECU variants. First, behaviour of customers is described by features. An extension of the definition of components allows describing the technical requirements of ECU variants. Sets of these variants are solutions for the VCP. During this section, it is explained how the average costs for the OEM can be calculated. The terms used in this section are based on the database model provided in Sect. 3.5.

7.1.1. Consideration of Customer Behaviour

The electronic system in a vehicle has to implement several *features F* based on the customer behaviour. It can be determined, which *set of ECU features* $\mathbf{F}_e$ is relevant for an ECU *e* using the database model shown in Fig. 3.6. It is the subset of all features that influence the currently considered ECU. Thereby, the set of ECU features for a specific ECU is dependent on the allocated components. The relation between features and components will be described more in detail below.

The customer decides which combination of features his car should cover—the configuration of his vehicle. For each ECU, configurations can be aggregated to ECU-specific *vehicle partitions* $\mathfrak{v}$. They influence the components that each ECU has to implement. The according subset of features $\mathbf{F}_{e,\mathfrak{v}} \subseteq \mathbf{F}_e$ for each

Table 7.1.: *Example for vehicle partitions, number of cars per vehicle partition, and feature combinations*

vehicle partition $\mathfrak{v}$	$n_{\mathfrak{v}}$	feature combination	F_1	F_2	F_3	F_4	F_5
$\mathfrak{v}_1$	20,000	$\mathbf{F}_{e_1,\mathfrak{v}_1}$	☐	☐	☒	☐	☒
$\mathfrak{v}_2$	5,000	$\mathbf{F}_{e_1,\mathfrak{v}_2}$	☒	☒	☒	☐	☒
$\mathfrak{v}_3$	15,000	$\mathbf{F}_{e_1,\mathfrak{v}_3}$	☐	☐	☐	☐	☒
$\mathfrak{v}_4$	10,000	$\mathbf{F}_{e_1,\mathfrak{v}_4}$	☐	☐	☐	☒	☐
$\mathfrak{v}_5$	10,000	$\mathbf{F}_{e_1,\mathfrak{v}_5}$	☐	☒	☒	☐	☒

vehicle partition and per ECU is called *feature combination* $\mathbf{F}_{e,\mathfrak{v}}$. Cars of a vehicle partition with a certain subset of features are ordered in arbitrary numbers—the *number of cars per vehicle partition* $n_{\mathfrak{v}}$.

In order to illustrate the various variables in this chapter, an example is presented. In the example, the ECU e_1 is optimised. The set of ECU features contains five features $F_1, \ldots, F_5$. Table 7.1 shows five different vehicle partitions $\mathfrak{v}_1, \ldots, \mathfrak{v}_5$. For each vehicle partition, a different feature combination defined. For example, 20,000 customers order a car with the options F_3 and F_5.

The determination of order forecasts for feature combinations is a non-trivial process. The proposed concept of vehicle partitions allows using past customer orders as input for creating these order forecasts. Additionally, marketing information is necessary for considering new features and the according future customer behaviour.

7.1.2. Involving Components

Until now, only the view of customers is taken into consideration. Contrary, in a technical view, it is spoken of components. All components allocated to an ECU are in the *set of ECU-allocated components* $\mathbf{c}_e$. The definition of the component costs is extended for an optimisation of variants. In Sect. 3.5.1, internal and external component costs ($\mathfrak{c}_c^{\text{int}}$ and $\mathfrak{c}_c^{\text{ext}}$) are defined to do this. In this chapter, only internal component costs $\mathfrak{c}_c^{\text{int}}$ are considered. Furthermore, only a share of these costs is cost saving potential, since it is not necessary to consider proportionate component costs like for example circuit board or micro-controller resources. Thus, only the *VO-reducible component costs* $\mathfrak{c}_c^{\text{int,VO}}$, which can be removed in variants of an ECU, are considered. Concluding, the *not-VO-reducible component costs* $\mathfrak{c}_c^{\text{int,nVO}}$ are defined with $\mathfrak{c}_c^{\text{int,nVO}} = \mathfrak{c}_c^{\text{int}} - \mathfrak{c}_c^{\text{int,VO}}$. Components that are not optional but mandatory are completely accounted in $\mathfrak{c}_c^{\text{int,nVO}}$. Since they have to be in all variants anyway, they are not affected by the variant optimisation.

Table 7.2.: *Example for components and their costs*

component c	$\mathfrak{c}_c^{\text{int,VO}}$		$\mathfrak{c}_c^{\text{int,nVO}}$		$\mathfrak{c}_c^{\text{int}}$	$\mathfrak{c}_c^{\text{ext}}$
	$\mathfrak{C}$		$\mathfrak{C}$		$\mathfrak{C}$	$\mathfrak{C}$
c_1	2.80	+	0.40	=	3.20	0.00
c_2	2.70	+	0.20	=	2.90	0.00
c_3	3.50	+	0.30	=	3.80	0.00

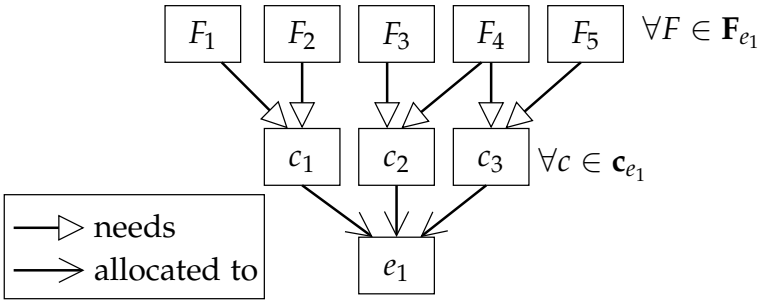

Figure 7.1.: *Example for the relation between features and components*

The example introduced above is extended with three components c_1, c_2, and c_3. Their according internal and external component costs are shown in Tab. 7.2 using the virtual cost unit $\mathfrak{C}$. External component costs are not relevant for variant optimisation and not modelled in this example.

In the example, all components are allocated to the ECU e_1. However, in which variants of this ECU to put them? This question is related to features ordered by customers. Each feature in $\mathbf{F}_\text{e}$ *needs* at least one component c. An example for the possible relations of features and components is shown in Fig. 7.1. The features F_1 and F_2 both need c_1. The feature F_4 needs c_2 and c_3. Similar to the features, in the following, only *optional* components are considered, since the optimisation result is not influenced by components that are necessary on each variant of an ECU. For convenience, the word *optional* is left out in the future, and it is just spoken of components and features.

Based on the relation between features and components of an ECU, it is possible to determine all components that are required to support the ordered feature combination $\mathbf{F}_{e,\mathfrak{v}}$ of an ECU. This combination of components is called *minimal component combination* $\mathbf{c}_{e,\mathfrak{v}}^{\min}$. All components in $\mathbf{c}_{e,\mathfrak{v}}^{\min}$ are required in order to support all ordered features in the vehicle partition $\mathfrak{v}$ and the according feature combination $\mathbf{F}_{\text{e},\mathfrak{v}}$.

Variants of ECUs are defined by a combination of components as well. They are built into cars in order to satisfy the demand for features. Equivalent minimal component combinations can be necessary for different vehicle partitions $\mathfrak{v}$

Table 7.3.: *Example of minimal component combinations their transformation to minimal variants*

$\mathbf{c}_{e,\mathfrak{v}}^{\min}$	$n_{\mathfrak{v}}$	c_1	c_2	c_3
$\mathbf{c}_{e,\mathfrak{v}_1}^{\min}$	20,000	□	⊠	⊠
$\mathbf{c}_{e,\mathfrak{v}_2}^{\min}$	5,000	⊠	⊠	⊠
$\mathbf{c}_{e,\mathfrak{v}_3}^{\min}$	15,000	□	□	⊠
$\mathbf{c}_{e,\mathfrak{v}_4}^{\min}$	10,000	□	⊠	⊠
$\mathbf{c}_{e,\mathfrak{v}_5}^{\min}$	10,000	⊠	⊠	⊠

$\Rightarrow$

$v^{\min}$	$n_{v^{\min}}$	c_1	c_2	c_3
$v_1^{\min}$	30,000	□	⊠	⊠
$v_2^{\min}$	15,000	⊠	⊠	⊠
$v_3^{\min}$	15,000	□	□	⊠

with a different feature combination. All equivalent minimal component combinations are aggregated to so-called *minimal variants* $v^{\min}$ in a next step. Minimal variants describe the minimum requirement for the ECU variant to be installed in the car. The *set of all minimal variants* $\mathbf{v}^{\min}$ contains all minimal variants derived from all feature combinations. The *predicted number of ordered minimal variants* $n_{v^{\min}}$ represents the according aggregation of the number of cars per vehicle partition $n_{\mathfrak{v}}$.

On the left hand, Tab. 7.3 shows five minimal component combinations derived from the feature combinations in Tab. 7.1 and the relation between features and components in Fig. 7.1. These five minimal component combinations are aggregated to three minimal variants on the right hand of Tab. 7.3. For example, $\mathbf{c}_{e,\mathfrak{v}_2}^{\min}$ and $\mathbf{c}_{e,\mathfrak{v}_5}^{\min}$ are aggregated to $v_2^{\min}$.

7.1.3. ECU Variants

Similar to minimal variants described above, *ECU variants* v are sets of components. ECU variants are actually being installed in ordered cars and thus have to be a superset of the components in the minimal variant. This means that the installed variant sometimes supports more components than required. Therefore, it is required that these components in that ECU can be deactivated without being visible to the customer.

It is shown how the *costs of an ECU variant* $\mathfrak{c}_v$ can be determined. These costs are composed from *basic VO ECU costs* $\mathfrak{c}_e^{\text{bas,VO}}$ and VO-reducible component costs $\mathfrak{c}_c^{\text{int,VO}}$, as defined above, of all components that are in the current variant:

$$\mathfrak{c}_v = \mathfrak{c}_e^{\text{bas,VO}} + \sum_{\forall c \in v} \mathfrak{c}_c^{\text{int,VO}}. \tag{7.1}$$

The basic VO ECU costs $\mathfrak{c}_e^{\text{bas,VO}}$ have to be taken into consideration for each variant of the according ECU. The costs $\mathfrak{c}_e^{\text{bas,VO}}$ are the basic costs $\mathfrak{c}_e^{\text{bas}}$ (see

Sect. 3.5.1) of an ECU increased by the not-VO-reducible component costs. Thus, the basic VO ECU costs $\mathfrak{c}_e^{\text{bas,VO}}$ can be calculated with

$$\mathfrak{c}_e^{\text{bas,VO}} = \mathfrak{c}_e^{\text{bas}} + \sum_{\forall c \in \mathbf{c}_e} \mathfrak{c}_c^{\text{int,nVO}}. \tag{7.2}$$

In the example (see Tab. 7.2) and with $\mathfrak{c}_e^{\text{bas}} = 20\,€$, $\mathfrak{c}_e^{\text{bas,VO}}$ can be determined with $\mathfrak{c}_e^{\text{bas,VO}} = 20.90\,€$.

7.1.4. Variant Combinations

Above, single ECU variants are accounted. Now, sets and combinations of these variants are examined. One type of set of ECU variants is the so-called *set of possible variants* $\mathbf{v}$. This set contains all variants that make sense. In a trivial case, it contains all variants that can be constructed using the given set of ECU-allocated components $\mathbf{c}_e$. How to do this in detail and how to reduce the number of possible variants, is shown in Sect. 7.4.

For each ECU, a set of ECU variants with limited size can be defined and ordered at the supplier for manufacturing the ordered cars. This set is a subset of the set of possible variants and called *variant combination* $\mathbf{V}$. Each set $\mathbf{V}$ is a solution—valid or invalid—of the VCP. A variant combination is a valid solution of the VCP if for each minimal variant in $\mathbf{v}^{\min}$ at least one ECU variant can be found being a superset:

$$\mathbf{V} \text{ valid: } \forall v^{\min} \in \mathbf{v}^{\min}\, \exists\, v \in \mathbf{V} | v \supseteq v^{\min} \tag{7.3}$$

Supposed, a solution $\mathbf{V}$ is valid, it is interesting to calculate the quality of this variant combination, namely the costs. Besides the average hardware costs of the variant combination, another cost factor is involved influencing the result: In general, a rising number of variants causes rising so-called variant-handling costs. For the sake of simplification, it is assumed that each additional variant causes the same order-rate-independent *handling costs per variant* $\mathfrak{c}_e^{\text{hand}}$. These costs for development, administration, and handling of additional variants can be determined according to past values. This can be difficult, since various divisions of a car manufacturer get in contact with ECUs. Thus, all these divisions have to evaluate their costs caused by the variants.

The *average ECU costs of a variant combination* $\mathfrak{c}_{e,\mathbf{V}}^{\text{avg}}$ depend on the accounted variant combination $\mathbf{V}$. For each valid variant combination $\mathbf{V}$, the corresponding

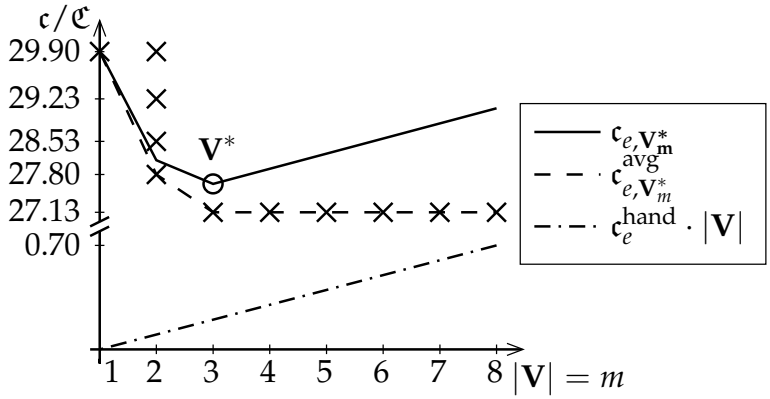

Figure 7.2.: *Average variant-combination-dependent costs on the number of variants*

average ECU costs $\mathfrak{c}^{\text{avg}}_{e,\mathbf{V}}$ can be calculated using the costs of each contained ECU variant $\mathfrak{c}_v$ and the according variant installation rate $p_{\text{i},v}$

$$\mathfrak{c}^{\text{avg}}_{e,\mathbf{V}} = \sum_{\forall v \in \mathbf{V}} \mathfrak{c}_v p_{\text{i},v}. \tag{7.4}$$

The *variant installation rates* $p_{\text{i},v}$ depend on the customer orders. For each minimal variant $v^{\min}$ ordered from customers, the cheapest variant $v \in \mathbf{V}$ that supports all required functions is built in. The value of $p_{\text{i},v}$ is defined by the number of cars built using the variant v divided by the total number of cars.

For each valid variant combination, the *average variant-combination-dependent costs* $\mathfrak{c}_{e,\mathbf{V}}$ can be determined by summing up the average ECU costs, the variant handling costs for the according number of variants, and the external component costs similar to (3.1) with

$$\mathfrak{c}_{e,\mathbf{V}} = \mathfrak{c}^{\text{avg}}_{e,\mathbf{V}} + \mathfrak{c}^{\text{hand}}_e |\mathbf{V}| + \sum_{\forall c \in \mathbf{c}_e} \mathfrak{c}^{\text{ext}}_c p_{\text{o},c}, \tag{7.5}$$

For example if $\mathbf{V} = \{v_1, v_2\}$, $v_1 = \{c_2, c_3\}$, $v_2 = \{c_1, c_2, c_3\}$, and $\mathfrak{c}^{\text{hand}}_e = 0.10\,€$, it follows: $\mathfrak{c}_{v_1} = 20.90\,€ + 2.70\,€ + 3.50\,€ = 27.10\,€$ and $\mathfrak{c}_{v_2} = 29.90\,€$. The ECU variant v_2 is installed in all 15,000 cars of $v_2^{\min}$—the cheaper ECU variant v_1 is sufficient in all 45,000 cars of $v_1^{\min}$ and $v_3^{\min}$. This is resulting in variant installation rates of $p_{\text{i},v_1} = 75\,\%$ and $p_{\text{i},v_2} = 25\,\%$. Thus, the value of the average ECU costs of $\mathbf{V}$ in the example is $\mathfrak{c}^{\text{avg}}_{e,\mathbf{V}} = 27.10\,€ \cdot 75\,\% + 29.90\,€ \cdot 25\,\% = 27.80\,€$. With $|\mathbf{V}| = 2$, this is resulting in $\mathfrak{c}_{e,\mathbf{V}} = 27.80\,€ + 0.10\,€ \cdot 2 = 28.00\,€$.

Usually, there is more than one valid variant combination that contains a defined *number of variants* m. In Fig. 7.2, each X marks the costs $\mathfrak{c}^{\text{avg}}_{e,\mathbf{V}}$ for a variant combination with $|\mathbf{V}| = m$ elements. The minimum of the costs with m elements belongs to the *local optimal variant combination* $\mathbf{V}^*_m$ with exactly $|\mathbf{V}^*_m| = m$

elements. In general, the more variants are managed, the lower the average ECU costs of a variant combination $\mathfrak{c}^{\text{avg}}_{e,\mathbf{V}}$, but the higher the variant handling costs $\mathfrak{c}^{\text{hand}}_{e} \cdot |\mathbf{V}|$. Connecting the cost points of all local optimal variant combinations $\mathbf{V}^*_m$ ($m = 1, 2, \ldots$), the curve $\mathfrak{c}^{\text{avg}}_{e,\mathbf{V}^*_m}$ in Fig. 7.2 can be constructed. The *optimal variant combination* $\mathbf{V}^*$ is the $\mathbf{V}^*_m$ with the m where the resulting curve $\mathfrak{c}_{e,\mathbf{V}^*_\mathbf{m}}$ according to (7.5) has its minimum. Algorithms for determining $\mathbf{V}^*$ are presented in Sect. 7.5. If the optimal variant combination can be determined, the *average ECU costs* $\mathfrak{c}_e$ are

$$\mathfrak{c}_e = \mathfrak{c}_{e,\mathbf{V}^*}. \tag{7.6}$$

7.2. Mathematical Problem Description

This section provides a mathematical description of the variant combination problem. In preparation of Sect. 7.3, where the VCP is transformed into a so-called warehouse location problem, additional variables are introduced.

For a specific minimal variant $v^{\min}$, only those ECU variants v containing at least all components of $v^{\min}$ can be installed. For all these variants, the product of their price $\mathfrak{c}_v$ and the order number $n_{v^{\min}}$ of the minimal variant $v^{\min}$ equals the so-called *minimal-variant-to-ECU-variant costs relation* $\mathfrak{c}_{v^{\min},v}$. It is defined between each minimal variant $v^{\min}$ and each possible variant v. It equals the costs that arise if in all cars that require the minimal variant $v^{\min}$, variant v is installed. For all couples $v^{\min} - v$, where v is not valid for fulfilling the requirements of $v^{\min}$, the corresponding costs relation $\mathfrak{c}_{v^{\min},v}$ is set to a high value M. The high value ensures that the optimisation algorithm does not assign a not allowed variant v to the minimal variant $v^{\min}$. Thus, the costs relation $\mathfrak{c}_{v^{\min},v}$ can be determined with:

$$\mathfrak{c}_{v^{\min},v} = \begin{cases} n_{v^{\min}} \cdot \mathfrak{c}_v & \text{if } v^{\min} \subseteq v, \\ M & \text{else.} \end{cases} \tag{7.7}$$

Furthermore, the variables y_v and $x_{v^{\min},v}$ are introduced. The meaning of these variables in a valid solution of the VCP is:

- $y_v = 1$, if variant v belongs to the current variant combination, $y_v = 0$ otherwise. The set $\mathbf{y}$ contains one y_v for each ECU variant in the set of possible variants $\mathbf{v}$. The set $\mathbf{y}$ is equivalent to a variant combination $\mathbf{V}$.

- $x_{v^{\min},v}$: percentage of the orders $n_{v^{\min}}$ that are satisfied by the installation of v. This value is only continuous during the optimisation. The value of

these variables must be $x_{v^{\min},v} = 100\,\%$ or $x_{v^{\min},v} = 0\,\%$ after the optimisation. The set $\mathbf{x}$ contains all $x_{v^{\min},v}$.

With these variables and identifiers, the VCP can be formulated as follows:

$$\text{minimise } F(\mathbf{x},\mathbf{y}) = \sum_{\forall v \in \mathbf{v}} \sum_{\forall v^{\min} \in \mathbf{v}^{\min}} \mathfrak{c}_{v^{\min},v} \cdot x_{v^{\min},v} + \sum_{\forall v \in \mathbf{v}} \mathfrak{c}_e^{\text{hand}} \cdot y_v \tag{7.8}$$

with the constraints

$$x_{v^{\min},v} \leq y_v \qquad \forall v \text{ and } \forall v^{\min} \tag{7.9}$$

$$\sum_{\forall v \in \mathbf{v}} x_{v^{\min},v} = 1 \qquad \forall v^{\min} \tag{7.10}$$

$$y_v \in \{0,1\} \qquad \forall v \tag{7.11}$$

$$x_{v^{\min},v} \geq 0 \qquad \forall v \text{ and } \forall v^{\min} \tag{7.12}$$

The sets $\mathbf{x}$ and $\mathbf{y}$ are chosen so that the value of $\mathfrak{c}_{e,\mathbf{V}}$ according to (7.5) is minimised. The first summand in (7.8) reflects the average ECU costs of a variant combination $\mathfrak{c}_{e,\mathbf{V}}^{\text{avg}}$ according to (7.4). The second summand reflects the variant handling costs $\mathfrak{c}_e^{\text{hand}} \cdot |\mathbf{V}|$. All other elements of $\mathfrak{c}_{e,\mathbf{V}}$ are not depending on the chosen variant combination and no subject of minimisation.

The constraints (7.9) ensure that orders of minimal variants $v^{\min}$ are only satisfied by ECU variants v that are elements of the variant combination $\mathbf{V}$. For each minimal variant, the corresponding constraint (7.10) guarantees that all orders of this minimal variant are fulfilled. The ranges of the variables are restricted by the constraints (7.11) and (7.12). The optimal variant combination $\mathbf{V}^*$ can be determined from the optimised $\mathbf{y}$. It is the set of all ECU variants v, where the according $y_v = 1$:

$$\mathbf{V}^* = \{v | v \in \mathbf{v}, y_v \in \mathbf{y}, y_v = 1\} \tag{7.13}$$

7.3. Variant Combination Problem as a Warehouse Location Problem

The (uncapacitated, simple) *Warehouse Location Problem* (WLP, [DD96])[1] is similar to the presented VCP. In a WLP, customers with demands n_j for a homogeneous product are given. They are supplied from some warehouses, which are not established yet. There exist some places where warehouses could be established.

[1] This problem is also called the uncapacitated facility location problem or Simple Plant Location Problem (SPLP).

Opening a warehouse at place i causes costs equal to $\mathfrak{c}_i^{\mathrm{wh}}$. The transportation of one unit of the product from the possible place i of a warehouse to customer j costs p_{ij} and the transportation of the whole demand n_j costs $\mathfrak{c}_{ij} = p_{ij} \cdot n_j$. The question is at what places warehouses should be opened and from which opened warehouse, the customers should be delivered, so that the overall costs are minimal. The variables x_{ij} and y_i are introduced for supporting the optimisation process. They are defined similarly to $x_{v^{\min},v}$ and y_v in Sect. 7.2. A more detailed summary of all variables can be found in Tab. 7.4.

According to Domschke and Drexl [DD96], the WLP can be formulated as

$$\text{minimise } F(\mathbf{x},\mathbf{y}) = \sum_i \sum_j \mathfrak{c}_{ij} \cdot x_{ij} + \sum_i \mathfrak{c}_i^{\mathrm{wh}} \cdot y_i \tag{7.14}$$

with the constraints

$$x_{ij} \leq y_i \qquad \forall i \text{ and } \forall j \tag{7.15}$$

$$\sum_{\forall i} x_{ij} = 1 \qquad \forall j \tag{7.16}$$

$$y_i \in \{0,1\} \qquad \forall i \tag{7.17}$$

$$x_{ij} \geq 0 \qquad \forall i \text{ and } \forall j \tag{7.18}$$

The VCP can be seen as a WLP. Therefore, transformations according to Tab. 7.4 have to be performed. Thus, both problems can be described mathematically in the same way.

7.4. Reduction of the Complexity

In the example shown so far, only three components are involved. With three components, the optimal solution can be determined even without computer support. The complexity of the VCP grows very fast with the number of components on the considered ECU.

For determining the complexity of the VCP, it is first shown how many combinations from a set can be formed of its elements. From set with n elements, $\binom{n}{n}$ different combinations of exactly n elements, $\binom{n}{n-1}$ combinations with $n-1$ elements, $\binom{n}{n-2}$ combinations with $n-2$ elements, and so on can be constructed. According to the binomial theorem

$$\sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k = (a+b)^n\,, \qquad a,b \in \mathbb{R},\ n \in \mathbb{N} \tag{7.19}$$

Table 7.4.: *Transformation of a VCP into a WLP*

	variant combination problem		warehouse location problem
$v^{\min}$	minimal variant	j	customer
v	ECU variant	i	possible warehouse location
y_v	equals one, if variant v belongs to the variant combination, zero else	y_i	equals one, if at location i a warehouse has to established, zero else
$x_{v^{\min},v}$	$x_{v^{\min},v} \cdot 100\,\%$ of the orders of component combination $v^{\min}$ is satisfied by the installation of variant v	x_{ij}	$x_{ij} \cdot 100\,\%$ of the demand of the customer j is satisfied by the warehouse at the location i
$\mathfrak{c}_{v^{\min},v}$	costs if variant v is built in all cars that require comp. combination $v^{\min}$	$\mathfrak{c}_{ij}$	costs if warehouse of location i covers the whole demand of customer j
$\mathfrak{c}_e^{\text{hand}}$	costs for developing and handling the additional variant v (unlike WLP here independent from v)	$\mathfrak{c}_i^{\text{wh}}$	costs for building and administration of the additional warehouse i

it is necessary, that

$$\sum_{k=0}^{n} \binom{n}{k} = \sum_{k=0}^{n} \binom{n}{k} 1^{n-k} 1^k = (1+1)^n = 2^n. \tag{7.20}$$

Therefore, from a set of n elements, in total, 2^n different combination can be formed. Given $|\mathbf{c}_e|$ different components, a number of $n = 2^{|\mathbf{c}_e|}$ different variants can be constructed. Thus, $2^n = 2^{2^{|\mathbf{c}_e|}}$ different variant combinations can be found.

In a naiv approach, all thinkable variants are member of the so-called *set of possible variants* $\mathbf{v}$. For example, for an ECU with ten optional components, $2^{10} = 1024$ ECU variants are in the set of possible variants yielding in $2^n = 2^{|\mathbf{v}|} = 2^{2^{10}} \approx 1.8 \cdot 10^{308}$ variant combinations. It is possible to reduce the size of the set of possible variants $\mathbf{v}$. This can significantly reduce the problem size. It is proposed to consider only variants that are equivalent to a minimal variant or the union of two or more minimal variants. Other ECU variants make no sense. They are either not valid for all minimal variants or another ECU variant can be found that fulfils the requirements of exactly the same minimal variants at

Table 7.5.: *Example for the determination of the set of possible variants from a set of minimal variants*

all $\cup v'$	c_1	c_2	c_3	v
$v_1^{\min}$	□	⊠	⊠	$\rightarrow v_1$
$v_2^{\min}$	⊠	⊠	⊠	$\rightarrow v_2$
$v_3^{\min}$	□	□	⊠	$\rightarrow v_3$
$v_1^{\min} \cup v_2^{\min}$	⊠	⊠	⊠	v_2
$v_2^{\min} \cup v_3^{\min}$	⊠	⊠	⊠	v_2
$v_3^{\min} \cup v_1^{\min}$	□	⊠	⊠	v_1
$v_1^{\min} \cup v_2^{\min} \cup v_3^{\min}$	⊠	⊠	⊠	v_2

lower costs. The union of the *power set* $\mathcal{P}$ of all minimal variants defines the set of possible variants. The power set is also called set of all subsets. For example, with $\mathbf{x} = \{1,2,3\}$, it follows $\mathcal{P}(\mathbf{x}) = \{\{1\},\{2\},\{3\},\{1,2\},\{2,3\},\{3,1\},\{1,2,3\},\{\}\}$. Formally, the set of possible variants $\mathbf{v}$ can be determined with

$$\mathbf{v} = \left\{ v | v = \cup v', v' \in \mathcal{P}\left(\mathbf{v}^{\min}\right) \right\}. \tag{7.21}$$

Each *raw ECU variant* v' is a set of minimal variants. The union of this set $\cup v'$ gives the ECU variant v.

Without this reduction of complexity, for the example in Tab. 7.3 with three components, the set of possible variants $\mathbf{v}$ contains a number of $2^3 = 8$ variants. Applying (7.21) yields in three variants. Table 7.5 shows the power set of all minimal variants and the according variants for this example. Redundant variants are printed in grey.

Applied to larger problem instances, this reduction of complexity helps to reduce the problem size significantly. The number of variant combinations that have to be tested if applying an exhaustive search in order to find the optimal variant combination $\mathbf{V}^*$ is reduced. Still, due to the huge number of possible variant combinations, it is often not possible to calculate the corresponding average ECU costs of each combination. Therefore, efficient optimisation algorithms are necessary that do not handle each variant combination separately.

7.5. Algorithms for Solving the Variant Combination Problem

The warehouse location problem is already examined since more than three decades [CDS02]. For the solution of the VCP respectively the WLP there are

exact and heuristic algorithms. Since the VCP can be transformed into a WLP, all algorithms for WLPs can be used for solving VCPs as well.

Several optimisation algorithms—originally developed for the WLP—have been examined and applied to the VCP. Exact algorithms are mostly *Branch & Bound* (B&B) algorithms. The B&B algorithm iterative fixes the problem variables. In the VCP, at each branching point, a $y_v \in \mathbf{y}$ is set to 0 or 1. Estimations for lower and upper bounds are determined considering the decisions already made in the current branch. If an upper bound for a (partially) fixed set of problem variables can be found, no branch, where the lower bound is higher than this upper bound, is interesting anymore. Thus, the number of paths through the tree to be completely discovered can be reduced very fast.

A B&B method with usage of a *simplex algorithm* is presented by Domschke and Drexl in [DD02]. It transforms the sub-problems in corresponding relaxed linear problems and solves them optimal with the simplex algorithm. This gives the lower bounds.

Another B&B algorithm is the so-called *Erlenkotter algorithm*. It has first been introduced by Erlenkotter in [Erl78] and further improved by Körkel [Kör99]. In difference to the simplex algorithm, the sub-problems are not relaxed. Rather, the original sub-problems are solved not optimal by heuristic algorithms. Subsequently, these results are considered as lower bounds. The lower bound determined by the simplex algorithm is more accurate but the determination is more complex. The computational effort of the simplex algorithm rises faster than of the Erlenkotter algorithm. Thus, for bigger problem instances, the simplex algorithm is much slower and it is not considered in the course of this section anymore.

A further class of algorithms is heuristic. A new heuristic approach developed in the course of this work —the so-called *merge algorithm*—is introduced here. The merge algorithm starts with a solution $\mathbf{V}$ containing all minimal variants ($\mathbf{V} := \mathbf{v}^{\min}$). Since in real applications, often, a high number of different minimal variants are ordered by customers, the size of $\mathbf{V}$ normally has to be reduced. Therefore, in each step, exactly two variants are merged. Merging means that these two variants are replaced by a new variant, which is the superset of the two replaced ones. At each step, all possible combinations of two variants are tested. The combination of two variants with the lowest worsening of the average ECU costs is chosen for merging. This method ensures that the solution $\mathbf{V}$ is always valid. The algorithm stops, when no further improvement can be reached anymore.

The merge algorithm forbids keeping solutions worse than the currently known best solution. In order to leave local optima, it makes sense to accept worse solutions in some cases. Thus, *Simulated Annealing* (SA, [KGV83])

Table 7.6.: *Details of the test problems*

test problem	$\lvert\mathbf{c}_e\rvert$	$\lvert\mathbf{v}^{\min}\rvert/2^{\lvert\mathbf{c}_e\rvert}$	type
TP1	8	0.150	random generated example
TP2	8	0.500	random generated example
TP3	9	0.150	random generated example
TP4	6	0.387	real application
TP5	10	0.419	real application

is analysed for usability as well. Simulated annealing works analogue to the way in which a metal cools down into a crystalline structure of minimum energy. In each iteration, SA considers some neighbours of the current solution $\mathbf{V}$. Simulated annealing decides, whether to keep the current solution or one of the new ones probabilistically. This decision is depending on a parameter called temperature. The higher the temperature the more likely a solution worse than the current one is chosen. The temperature is falling, until the system is frozen and a good solution hopefully near or equal to the optimum has been found. The application of SA to the VCP is explained in detail in [Kol05].

The different algorithms—Erlenkotter algorithm, merge algorithm, and SA—are tested with five different *Test Problem*s (TP). An overview is given in Tab. 7.6. The first three test problems (TP1 – TP3) are generated randomly. First, a certain number of components $\lvert\mathbf{c}_e\rvert$ and a percentage of minimal variants in comparison to the number of possible variants $\lvert\mathbf{v}^{\min}\rvert/2^{\lvert\mathbf{c}_e\rvert}$ is specified. The component costs are varying between 1.5 € and 3.5 €. Second, the configurations of the minimal variants are generated randomly so that the specifications are met. Test problems TP4 and TP5 are two real applications of this method. The values $\lvert\mathbf{v}^{\min}\rvert/2^{\lvert\mathbf{c}_e\rvert}$ show the complexity of these problems.

Measures of the execution time (see Tab. 7.7) and the quality of the results are compared in Tab. 7.8. The tests have been performed on an Intel Pentium III 1000 with 512 MB RAM. It can be shown that for smaller problem sizes, SA is slower than the Erlenkotter algorithm. Even for bigger problem sizes (10 components), SA is not significantly faster than the Erlenkotter algorithm. This is probably due to many local extremes of the solution space of a VCP. The neighbour solutions of the few valid solutions are normally quite bad or invalid. It is difficult to leave these local extremes with SA. The merge algorithm is normally much faster than the Erlenkotter algorithm and the SA. Additionally, the quality of results is better than SA. It is proposed to apply the Erlenkotter algorithm as long as the time is sufficient to do so. The amount of necessary time in turn

Table 7.7.: *Time in seconds for calculating the solutions for five different test problems*

algorithm	test problem					average
	TP 1	TP 2	TP 3	TP 4	TP 5	
	s	s	s	s	s	s
merge algorithm	0.15	0.41	0.24	0.61	3.67	1.02
SA	1.23	5.92	3.93	4.71	44.51	12.06
Erlenkotter algorithm	0.43	0.65	9.56	10.48	117.12	27.65

Table 7.8.: *Difference of the solutions in comparison to the optimal one for five different test problems*

algorithm	test problem					average
	TP 1	TP 2	TP 3	TP 4	TP 5	
	%	%	%	%	%	%
merge algorithm	0.14	0.01	0.59	1.59	0.27	0.52
SA	0.40	0.01	2.57	1.58	0.51	1.014
Erlenkotter algorithm	0.00	0.00	0.00	0.00	0.00	0.00

depends on the available computing power. If the time is not sufficient, the application of the merge algorithm is proposed.

7.6. Summary

In order to consider variants of ECUs, an extension to the objective costs from Sect. 3.5 is proposed. Variant optimisation is a promising optimisation on its own. Variant optimisation has been applied to a real ECU with 6 and 10 (optional) components (TP4 and TP5). In comparison to a previously realised manual variant determination, the introduced method for computer-aided optimisation saved over 1 € per car and optimised ECU under the condition that the future orders exactly match the forecast. This method represents a big improvement in comparison to the manual determination of the variants. It saves not only costs but also supports the process of determining the variants of ECUs. According to [Scho6], the Volkswagen group will produce over 5,000,000 in 2006.

Thus, assumed that this method can be applied in a similar manner for only two ECUs in each car of the Volkswagen group, savings of over 10 million € per year can be expected.

8. Conclusion

This chapter summarises the most important results of this thesis. It compares the work to existing approaches. Furthermore, it gives an outlook on future work that will help to further improve the system.

8.1. Summary

Several tasks were defined in the beginning of this work. These tasks have been treated in the course of the work.

A number of objectives and constraints that can be influenced by the allocation of functions have been characterised. For each objective, a formal description shows how to calculate the according quality ratings. During the treatment of the objectives, a database model has been developed for storing all information necessary for the calculation of the quality ratings. A large portion of the data is already spread in different databases of the car manufacturers. The collection of this data and the calculation of quality ratings will improve the process of architecture development significantly by showing drawbacks and disadvantages of architecture proposals in early development phases.

After examining the objectives, an optimisation framework has been developed. This framework allows a flexible extension with arbitrary objectives and arbitrary population based optimisation algorithms.

An application example has been introduced with a realistic complexity. Evolutionary algorithms and ant colony optimisation have been investigated as possible candidates for the optimisation of the according function allocation. For both algorithms, extensions have been proposed that enable applying them to this problem type. In the performed tests, the ant colony optimisation algorithm performed better than the modified evolutionary algorithm. The provided method enables not only to evaluate a given function allocation proposal but also to propose new optimised ones.

In order to reduce the cost estimation of a given allocation further, the so-called variant optimisation has been developed. It is shown how to determine a combination of variants of one ECU differing in hardware options so that a cost minimum is reached. Thereby, past customer orders and order forecasts

are taken into consideration. It has been shown that a considerable amount of money can be saved applying only this optimisation.

8.2. Comparison to Existing Approaches

This section summarises existing approaches and compares them with this work. There are already approaches for the estimation of the quality ratings for different objectives. Park et al. [PCCL94] and Kloske et al. [KS94] published wiring harness optimisation methods. Different resources, like for example pins, memory, and so on are handled by Blickle et al. [BTT98]. In this work, an approach has been introduced allowing the consideration of arbitrary objectives at the same time. Resources are modelled in a general way. To the knowledge of the author, the objectives electrical energy consumption and supplier complexity have not been handled before in the introduced way. If more detailed quality ratings are needed the framework allows an inclusion at any time.

Several general optimisation algorithms are able to consider multiple objectives and combinatorial problems, like SPEA2, NSGA-II, and ACO. Existing constraint handling methods can be used in combination with these algorithms. It is not possible to apply these algorithms to arbitrary problems. There are specialised optimisation algorithms for solving the travelling salesman problem, the warehouse location problem or the multiple knapsack problem with a good performance. Additionally, in the area of hardware and software co-design, work has been done on solving allocation problems. Its focus lies on real time properties, communication issues, and task and bus schedules. To the knowledge of the author, there is no specialised optimisation algorithm for solving the function allocation problem in the form shown here. This work presents a new problem representation enabling the application of general optimisation algorithms to the special optimisation problem. Existing constraint handling methods are combined with SPEA2 and an ACO algorithm. In the course of this work, several new constraint handling techniques have been developed and tested. Furthermore, some drawbacks of existing optimisation methods have been fixed or improved. Orthogonal arrays were applied for the determination of parameters of the optimisation algorithms.

Subsuming the above points, this work combines several fields of work that have not yet been brought together before. Furthermore, a number of additional parts have been developed. Since the amount of data—contained in several databases at all parties in the automotive industry—rises, the results of this work will get even more important in the future.

8.3. Future Work

In the course of this work, it has been shown that the optimisation system can improve the electronic architecture. The optimisation delivers architecture proposals on even quite complex application problems. Nevertheless, there are still open technical issues that would help to further increase the benefit of the system:

- In the future, the number of safety relevant functions will probably further increase. Thus, it is increasingly necessary to regard redundant functions during the design of the electronic architecture. Future work must regard this aspect as well.
- Currently, only the optimisation of the allocation of functions is supported. In the future, it is planned to examine ways of how to support degrees of freedom in the hardware as well. This increases as well the complexity as the potential.
- All parameters of the optimisation algorithms are set statically. It is planned to consider dynamically changing parameters as well in order to further improve the performance of the optimisation algorithms.
- The influence of different application examples on the optimisation results has to be examined.
- In the opinion of the author, there is potential for further development of ACO. Currently, no heuristic information like in the existing approaches is applied for determining the paths of the ants.
- The current system has no graphical user interface. Further work has to be performed in order to make the system user-friendly even for non-experts in optimisation and databases.

The items listed above are technical issues that are intended to further improve the optimisation system. A main problem is still the gathering of the input data for the optimisation. In the future, processes have to be installed that motivate and support all necessary persons within the development process during their design of the optimal electronic architecture for future vehicles. Tailor-made solutions depending on the specific infrastructure of the OEM have to be found. Existing databases have to be connected to the optimisation system and extended for improving the support of the architecture design.

The sum of the open points shows that there is still a long way to go. This work provides not only methods but also a vision. Nevertheless, the example of

variant optimisation shows that it is not always necessary to complete the whole task in one huge step in order to realise benefits. I hope that the whole vision will come true step by step.

A. Parameters of the Optimisation Algorithms

This appendix summarises the parameters of the optimisation algorithms in Cha. 6.

Table A.1.: *Summary of parameters of evolutionary algorithms*

parameter	description
$\mathfrak{p}_{\mathrm{PSF}}$	*Population size factor*. Defines a factor that is multiplied with the number of objectives in order to determine the desired population size.
$\mathfrak{p}_{\mathrm{MR}}$	*Normalised mutation rate*. The probability p_{M} that a mutation is performed is depending on the normalised mutation rate and the number of components $\lvert s\rvert$ with $p_{\mathrm{M}} = \mathfrak{p}_{\mathrm{MR}}/\lvert s\rvert$.
$\mathfrak{p}_{\mathrm{MMR}}$	*Move mutation rate*. In case that a mutation is applied to the current component, this parameter denotes the probability, that the move mutation is performed instead of the exchange mutation.
$\mathfrak{p}_{\mathrm{CR}}$	*Crossover rate*. Denotes the probability that the assignment of a component is crossed with the assignment of the same component of another solution.

Table A.2.: *Summary of parameters of ant colony optimisation*

parameter	description
$\mathfrak{p}_{\text{ERP}}$	*Exploitation random parameter*. A parameter for configuring the probability that the exploitation-supporting selecting mechanism is applied in (6.5).
$\mathfrak{p}_{\text{NCPO}}$	*Number of colonies per objective*. This parameter is multiplied with the number of objectives in order to determine the number of colonies.
$\mathfrak{p}_{\text{SPF}}$	*Sub population factor*. This parameter is multiplied with the number of objectives in order to determine the number of solutions within each sub population.
$\mathfrak{p}_{\text{PER}}$	*Pheromone evaporation rate*. Denotes the amount of evaporation in the pheromone matrix.
$\mathfrak{p}_{\text{CER}}$	*Correlation evaporation rate*. Denotes the amount of evaporation in the correlation matrix.
$\mathfrak{p}_{\text{PSF}}$	*Population size factor*. This parameter is multiplied with number of objectives in order to determine the number of solutions in the global solution set.
$\mathfrak{p}_{\text{UGS}}$	*Update factor global solution set*. Parameter for adjusting the influence of the global solution set on the update of the pheromone and correlation matrices.
$\mathfrak{p}_{\text{ULF}}$	*Update factor local feasible solutions*. Parameter for adjusting the influence of the feasible solutions in the current colony on the update of the pheromone and correlation matrices.
$\mathfrak{p}_{\text{ULN}}$	*Update factor local non-dominated solutions*. Parameter for adjusting the influence of the non-dominated solutions in the current colony on the update of the pheromone and correlation matrices.
$\mathfrak{p}_{\text{FUP}}$	*Factor for update by step*. Multiplied with the values in the pheromone and correlation matrices after each step of the ant in order to reduce the probability for ant following the same path twice.

B. Orthogonal Arrays

This appendix lists the orthogonal arrays used in Sect. 6.3. Each column stands for a specific parameter and each value for certain parameter setting. Each row of the arrays is equivalent to an experiment that has to be conducted.

Table B.1.: *Orthogonal Array $L(6,5)$ [Sloo6]*

0	0	0	0	0	0
0	1	2	4	3	2
0	2	3	1	4	3
0	3	4	2	1	4
0	4	1	3	2	1
1	0	4	4	2	3
1	1	0	1	1	1
1	2	2	3	0	4
1	3	1	0	4	2
1	4	3	2	3	0
2	0	1	1	3	4
2	1	4	3	4	0
2	2	0	2	2	2
2	3	3	4	0	1
2	4	2	0	1	3
3	0	2	2	4	1
3	1	3	0	2	4
3	2	1	4	1	0
3	3	0	3	3	3
3	4	4	1	0	2
4	0	3	3	1	2
4	1	1	2	0	3
4	2	4	0	3	1
4	3	2	1	2	0
4	4	0	4	4	4

Table B.2.: *Orthogonal Array* $L(13,3)$ *[Sloo6]*

0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	2	2	0	1	2	1	0	0
2	0	2	2	2	1	1	0	2	1	2	0	0
0	1	0	1	1	1	2	2	0	1	2	1	0
1	1	1	2	2	0	1	2	1	0	0	1	0
2	1	2	0	0	2	0	2	2	2	1	1	0
0	2	0	2	2	2	1	1	0	2	1	2	0
1	2	1	0	0	1	0	1	1	1	2	2	0
2	2	2	1	1	0	2	1	2	0	0	2	0
0	0	1	0	1	1	1	2	2	0	1	2	1
1	0	2	1	2	0	0	2	0	2	2	2	1
2	0	0	2	0	2	2	2	1	1	0	2	1
0	1	1	1	2	2	0	1	2	1	0	0	1
1	1	2	2	0	1	2	1	0	0	1	0	1
2	1	0	0	1	0	1	1	1	2	2	0	1
0	2	1	2	0	0	2	0	2	2	2	1	1
1	2	2	0	1	2	1	0	0	1	0	1	1
2	2	0	1	2	1	0	0	1	0	1	1	1
0	0	2	0	2	2	2	1	1	0	2	1	2
1	0	0	1	0	1	1	1	2	2	0	1	2
2	0	1	2	1	0	0	1	0	1	1	1	2
0	1	2	1	0	0	1	0	1	1	1	2	2
1	1	0	2	1	2	0	0	2	0	2	2	2
2	1	1	0	2	1	2	0	0	2	0	2	2
0	2	2	2	1	1	0	2	1	2	0	0	2
1	2	0	0	2	0	2	2	2	1	1	0	2
2	2	1	1	0	2	1	2	0	0	2	0	2

List of Figures

List of Tables

Bibliography

[3SO06] 3SOFT GmbH. tresos—More than OSEK. `http://www.tresos.de`, June 2006. 10

[AEE05] AEE Konsortium. Architecture Electronique Embarquée. `http://souheil.zina.free.fr/projets/AEE/aee.html`, December 2005. 5

[Apa05] Apache Software Foundation. Torque—persistence layer, December 2005. `http://db.apache.org/torque/`. 62

[AUT05] AUTOSAR. Automotive Open System Architecture, December 2005. `http://www.autosar.org/`. 5

[BDT99] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, New York, NY, USA, 1999. 19

[Bic06] Bettina Bickel. Anwendung von Schwarmintelligenz auf ein Mehrkriterienoptimierungsproblem mit Nebenbedingungen. *Diploma thesis, Friedrich-Alexander University Erlangen-Nuremberg, Computer Science Department 2*, April 2006. 97

[BIP98] BIPM: Bureau international des poids et mesures. *Le Système international d' unités (SI)*. Stedi, Paris, France, 1998. 33

[BK94] Thomas Bäck and Sami Khuri. An evolutionary heuristic for the maximum independent set problem. In Michalewicz et al. [MSS+94], pages 531–535. 23

[BLTZ03] Stefan Bleuler, Marco Laumanns, Lothar Thiele, and Eckart Zitzler. PISA—a platform and programming language independent interface for search algorithms. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization (EMO 2003)*, Lecture Notes in Computer Science, pages 494–508, Berlin, Germany, 2003. Springer. 61

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley & Sons, New York, NY, USA, 1996. 62, 66

[BP96] George Bilchev and Ian C. Parmee. Constrained optimisation with an ant colony search model. In *Proceedings of the 2nd International Conference on Adaptive Computing in Engineering Design and Control*, pages 145–151, Plymouth, United Kingdom, March 1996. 21

[BS98] James E. Beck and Daniel P. Siewiorek. Automatic configuration of embedded multicomputer systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(2):84–95, February 1998. 9

[BSPF01] J. M. Bull, L. A. Smith, L. Pottage, and R. Freeman. Benchmarking Java against C and Fortran for scientific applications. In *Proceedings of ACM Java Grande/ISCOPE Conference*, pages 97–105, Stanford, CA, USA, June 2001. 62

[BTT98] Tobias Blickle, Jürgen Teich, and Lothar Thiele. System-level synthesis using evolutionary algorithms. *Journal of Design Automation for Embedded Systems*, 3(1):23–58, January 1998. 9, 33, 128

[CAN05] CAN in Automation. Controller area network (CAN)—an overview. `http://www.can-cia.de/can/`, March 2005. 5

[CDM91] Alberto Colorni, Marco Dorigo, and Vittorio Maniezzo. Distributed optimization by ant colonies. In *Proceedings of the European Conference on Artificial Life*, pages 134–142, Paris, France, 1991. 19

[CDS02] John Current, Mark Daskin, and David Schilling. Discrete network location models. In Zvi Drezner and Horst W. Hamacher, editors, *Facility Location—Applications and Theory*, pages 80–118. Springer, Berlin, Germany, 2002. 122

[CFSC01] Roberto Cordone, Fabrizio Ferrandi, Donatella Sciuto, and Roberto Wolfler Calvo. An efficient heuristic approach to solve the unate covering problem. *IEEE Transactions on ComputerAided Design of Integrated Circuits and Systems*, 20(12):1377–1388, 2001. 59

[CFT98] Alberto Caprara, Matteo Fischetti, and Paolo Toth. Algorithms for the set covering problem. Technical Report OR-98-3, DEIS-Operations Research Group, 1998. 8, 58, 59

[Che76] Peter Pin-Shan Chen. The entity-relationship model toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976. 26

[Coe00] Carlos A. Coello. An updated survey of GA-based multiobjective optimization techniques. *ACM Computing Surveys*, 32(2):109–143, 2000. 15

[Coe02] Carlos A. Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11–12):1245–1287, January 2002. 11, 21, 23

[DAPM00] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. A fast elitist nondominated sorting genetic algorithm for multiobjective optimisation: NSGA-II. In *Parallel Problem Solving from Nature VI Conference*, pages 849–858. Springer, 2000. 16

[DD96] Wolfgang Domschke and Andreas Drexl. *Logistik—Standorte*, volume 3. Oldenbourg, München-Wien, Germany-Austria, 4th edition, 1996. 119, 120

[DD02] Wolfgang Domschke and Andreas Drexl. *Einführung in Operations Research*. Springer, Berlin, Germany, 5th edition, 2002. 123

[Deb00] Kalyanmoy Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2–4):311–338, 2000. 23

[Deb01] Kalyanmoy Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Ltd., Chichester, UK, 2001. 14, 15

[DEC06] DECOS. Dependable Embedded Components and Systems, May 2006. `http://www.decos.at/`. 5

[DG97] Marco Dorigo and Luca M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997. 98

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 48

[Ecl05] Eclipse Foundation. Eclipse: Extensible Java IDE. `http://www.eclipse.org/`, December 2005. 62

[EHM99] Ágoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999. 101

[Erl78] Donald Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, 26(6):992–1009, 1978. 123

[ETA05] ETAS GmbH. ETAS—INTECRIO overview. `http://en.etasgroup.com/products/intecrio/`, December 2005. 10

[Fle05] FlexRay Consortium. FlexRay communications system protocol specification version 2.1, May 2005. 5, 46

[FRHW00] Ulrich Freund, Thomas Riegraf, Markus Hemprich, and Kai Werther. Interface based design of distributed embedded automotive software—the TITUS approach. In *VDI-Berichte 1547, VDI Congress Electronic Systems for Vehicles*, pages 105–123, Baden-Baden, Germany, October 2000. VDI Verlag GmbH. 5

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, October 1994. 62

[GJSB05] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification—Third Edition*. Addison-Wesley, Boston, MA, USA, 2005. 62, 66

[GMCH04] Carlos García-Martínez, Oscar Cordón, and Francisco Herrera. An empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria tsp. In *ANTS Workshop*, pages 61–72, 2004. 20

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley Longman, Boston, MA, USA, 1989. 16, 23

[GP99] Peter Gulutzan and Trudy Pelzer. *SQL-99 Complete, Really*, volume 3. CMP Books, 1st edition, March 1999. 26

[GTA99] Luca M. Gambardella, Eric Taillard, and Giovanni Agazzi. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 63–76. McGraw-Hill, 1999. 20

[HAB97] Atidel B. Hadj-Alouane and James C. Bean. A genetic algorithm for the multiple-choice integer program. *Operations Research*, 45:92–101, 1997. 23

[HIS03] HIS—Hersteller Initiative Software, Volkswagen AG. HIS / Vector CAN-Driver Specification V1.0. `http://www.automotive-his.de`, August 2003. 5

[HIS04] HIS—Hersteller Initiative Software. HIS / API IO driver v 2.1.3, 2004. `http://www.automotive-his.de/`. 5

[HK05] Bernd Hardung and Thomas Kollert. Optimisation of the variant combination of electronic control units considering the order history. In Hans D. Haasis, Herbert Kopfer, and Jörn Schönberger, editors, *Operations Research Procedings 2005*, pages 361–366, Bremen, Germany, September 2005. 8, 111

[HKK04] Bernd Hardung, Thorsten Kölzow, and Andreas Krüger. Reuse of software in distributed embedded automotive systems. In *Proceedings of the 4th ACM International Conference on Embedded Software, EMSOFT*, pages 203–210, Pisa, Italy, September 2004. 4

[HKKK05] Bernd Hardung, Thomas Kollert, Gabriella Kókai, and Andreas Krüger. Optimization of variants of electronic control units under consideration of customer orders. In VDI Verein Deutscher Ingenieure [VDI05], pages 619–630. 8, 111

[HNG94] Jeffrey Horn, Nicholas Nafpliotis, and David E. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the 1st IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, pages 82–87, Piscataway, NJ, USA, June 1994. 16, 95

[Hol92] John H. Holland. *Adaptation in Natural and Artificial Systems*. The MIT Press, Cambridge, MA, USA, 1992. 15

[HS96] Frank Hoffmeister and Joachim Sprave. Problem-independent handling of constraints by use of metric penalty functions. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck, editors, *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 289–294, Cambridge, MA, USA, 1996. MIT Press. 22

[HTRE02] Christian Haubelt, Jürgen Teich, Kai Richter, and Rolf Ernst. System design for flexibility. In Carlos Delgado Kloos and Jose da France, editors, *Proceedings of Design, Automation and Test in Europe (DATE'02)*, pages 854–861, Paris, France, March 2002. IEEE Computer Society Press. 111

[HZR03] Robert M. Hubley, Eckart Zitzler, and Jare C. Roach. Evolutionary algorithms for the selection of single nucleotide polymorphisms. *BMC Bioinformatics*, 4(30), July 2003. 93

[IMM01] Steffen Iredi, Daniel Merkle, and Martin Middendorf. Bi-criterion optimization with multi colony ant algorithms. In *Proceedings of the 1st International Conference on Evolutionary Multicriterion Optimization*, pages 359–372, 2001. 20, 99

[ITE05] ITEA. EAST-EEA: Embedded electronic architecture—ITEA-Project-No. 00009. `http://www.east-eea.net/`, December 2005. 5

[JH94] Jeffrey A. Joines and Christopher R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs. In David B. Fogel, editor, *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, pages 579–584, Orlando, FL, USA, 1994. IEEE Press. 23

[KAW02] A. Kurpati, S. Azarm, and J. Wu. Constraint handling improvements for multiobjective genetic algorithms. *Structural and Multidisciplinary Optimization*, 23:204–213, 2002. 21

[KE01] James Kennedy and Russell C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001. 19

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983. 123

[Kol05] Thomas Kollert. Variantenoptimierung von Steuergeräten in der Automobilindustrie. *Diploma thesis, Technical University of Darmstadt, FG Operations Research*, September 2005. 111, 124

[Kow97] Ryszard Kowalczyk. Constraint consistent genetic algorithms. In *Proceedings of the 1997 IEEE Conference on Evolutionary Computation*, pages 343–348, Indianapolis, IN, USA, April 1997. IEEE Press. 22

[Koz92] John Koza. *Genetic Programming: On the programming of computers by means of natural selection*. MIT Press, Camebridge, MA, USA, 1992. 16

[Kör99] Manfred Körkel. *Effiziente Verfahren zur Lösung unkapazitierter Standort-Probleme*. VWF, Berlin, Germany, 1999. 123

[KS94] David A. Kloske and Robert E. Smith. Bulk cable routing using genetic algorithms. In *IEA/AIE '94: Proceedings of the 7th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 427–431, Austin, TX, USA, May 1994. 37, 128

[Köt05] Jens Kötz. Electronic architectures—chances and risks. In VDI Verein Deutscher Ingenieure [VDI05], pages 487–494. 5

[KWEP03] Andreas Krüger, Gerhard Wagner, Nils Ehmke, and Sascha Prokop. Economic considerations and business models for automotive standard software components. In VDI Verein Deutscher Ingenieure [VDI03], pages 1057–1071. 5

[Law00] Wolfhard Lawrenz. *CAN Controller Area Network—Grundlagen und Praxis*. Hüthig, Heidelberg, Germany, 2000. 45

[LIN03] LIN Consortium. LIN specification package revision 2.0. `http://www.lin-subbus.org`, September 2003. 5, 45

[LZT01] Marco Laumanns, Eckart Zitzler, and Lothar Thiele. On the effects of archiving, elitism, and density based selection in evolutionary multi-objective optimization. In *Evolutionary Multi-criterion Optimization (EMO 2001), Lecture Notes on Computer Science 1993*, pages 181–196, Berlin, Germany, March 2001. Springer. 91, 105

[MA94] Zbigniew Michalewicz and Naguib F. Attia. Evolutionary optimization of constrained problems. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 98–108. World Scientific, 1994. 23

[MFM01] Alexandre Mendes, Paulo França, and Pablo Moscato. NPOpt: An optimization framework for NP problems. In *Proceedings of POM2001—International Conference of the Production and Operations Management Society*, pages 82–89, Guarujá, SP, Brazil, August 2001. 61

[MH01] Mercer Management Consulting and Hypovereinsbank. Studie, Automobiltechnologie 2010. Munich, Germany, August 2001. 3, 36

[MH04] Koen Mertens and Tom Holvoet. CSAA: A distributed ant algorithm framework for constraint satisfaction. In Valerie Barr and Zdravko Markov, editors, *Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference*, pages 764–769, Miami Beach, FL, USA, May 2004. 21

[MM99] Carlos E. Mariano and Eduardo Morales. MOAQ: an ant-Q algorithm for multiple objective optimization problems. In *Proceeding of the Genetic and Evolutionary Computation Conference (GECCO-99)*, volume 1, pages 894–901, Orlando, FL, USA, 1999. Morgan Kaufmann. 20

[MOS05] MOST Cooperation. MOST specification rev., 2.4. `http://www.mostnet.de/downloads/Specifications/MOSTSpecifications/`, May 2005. 5, 46, 50

[MQ98] Angel Kuri Morales and Carlos Villegas Quezada. A universal genetic algorithm for constrained optimization. In *Proceedings 6th European Congress on Intelligent Techniques & Soft Computing, EUFIT'98*, pages 518–522, Aachen, Germany, September 1998. 23

[MSS+94] Zbigniew Michalewicz, J. David Schaffer, Hans-Paul Schwefel, David B. Fogel, and Hiroaki Kitano, editors. *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, Piscataway, NJ, USA, 1994. IEEE Press. 139, 147

[MSVCS05] Desire Luc Massart, Johanna Smeyers-Verbeke, Xavier Capron, and Karin Schlesier. Visual presentation of data by means of box plots. *LC GC Europe*, 18(4):215–218, April 2005. 97

[Neu05] Christoph Peter Neumann. Conceptional design of an open framework for optimizing the distribution of hardware and software components in control networks for vehicles. *Diploma thesis, Friedrich-Alexander University Erlangen-Nuremberg, Computer Science Department 2*, June 2005. 61

[Ngu96] Nam-Ky Nguyen. A note on the construction of near-orthogonal arrays with mixed levels and economic run size. *Technometrics*, 38:279–283, August 1996. 102

[NHN03] Thomas Nolte, Hans Hansson, and Christer Norström. Probabilistic worst-case response-time analysis for the controller area network. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03)*, pages 200–207, Washington, DC (relocated from Toronto), USA, May 2003. IEEE Computer Society. 45

[OSF05] Akira Oyama, Koji Shimoyama, and Kozo Fujii. New constraint-handling method for multi-objective multi-constraint evolutionary optimization and its application to space plane design. In R. Schilling, W. Haase, J. Periaux, H. Baier, and G. Bugeda, editors, *Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems (EUROGEN 2005)*, Munich, Germany, September 2005. 24

[OY05] Özgür Yeniay. Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and Computational Applications*, 10:45–56, 2005. 23

[PCCL94] Hisup Park, Marc R. Cutkosky, Andrew B. Conru, and Soo-Hong Lee. An agent-based approach to concurrent cable harness design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 8(30):45–61, 1994. 37, 128

[PK94] Charles C. Palmer and Aaron Kershenbaum. Representing trees in genetic algorithms. In Michalewicz et al. [MSS+94], pages 379–384. 22

[Rec94] Ingo Rechenberg. *Evolutionsstrategie '94*. Frommann-Holzboog Verlag, Stuttgart, Germany, September 1994. 16

[RZE+99] Kai Richter, Dirk Ziegenbein, Rolf Ernst, Lothar Thiele, and Jürgen Teich. Representation of function variants for embedded system optimization and synthesis. In *Proceedings of the 35th Design Automation Conference (DAC)*, pages 517–522, New Orleans, LA, USA, June 1999. 111

[Sch85] J. David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In John J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the 1st International Conference on Genetic Algorithms*, pages 93–100. Lawrence Erlbaum, 1985. 14

[Sch06] Gerd Scholz. Im Fokus: Volkswagen. *Automobil-Produktion*, pages 14–15, March 2006. 125

[SD99] Thomas Stützle and Marco Dorigo. ACO algorithms for the traveling salesman problem. In Kaisa Miettinen, Marco M. Mäkelä, Pekka Neittaanmäki, and J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, pages 163–183. John Wiley & Sons, Chichester, UK, 1999. 19

[SHT05] Thomas Schlichter, Christian Haubelt, and Jürgen Teich. Improving EA-based design space exploration by utilizing symbolic feasibility tests. In Hans-Georg Beyer et al., editors, *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, pages 1945–1952, Washington, DC, USA, June 2005. 9, 16

[SKPR98] S. Schmitz, J. Kruppa, R. Pettit, and C. Roser. Ein vielseitig einsetzbares schlüsselloses Zentralverriegelungssystem (in German). In VDI Verein Deutscher Ingenieure, editor, *VDI-Berichte 1998, VDI Congress Electronic Systems for Vehicles*, pages 259–279, Baden-Baden, Germany, October 1998. VDI Verlag. 77

[Slo06] N. J. A. Sloane. A library of orthogonal arrays. `http://www.research.att.com/~njas/oadir/`, March 2006. 102, 103, 133, 134, 137, 138

[SRHR02] J. E. L. Simmons, J. M. Ritchie, P. O. B. Holt, and G. T. Russell. Immersing the human in the design: Design-for-manufacture of cable harnesses. In *Proceedings of SCI/ISAS 2002 Volume XXII*, Orlando, FL, USA, 2002. 37

[SSRB00] Douglas Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. Wiley & Sons, New York, NY, USA, 2000. Reprinted with corrections April 2001. 62, 64

[SU05] Felix Streichert and Holger Ulmer. JavaEvA—a Java framework for evolutionary algorithms. Technical Report WSI-2005-06, Centre for Bioinformatics Tübingen, University of Tübingen, Germany, 2005. 61

[Tag86] Genichi Taguchi. *Introduction to Quality Engineering: Designing Quality into Products and Processes*. Asian Productivity Organization, Japan, 1986. 102

[TEF+03] Thomas Thurner, Joachim Eisenmann, Ulrich Freund, Roland Geiger, Michael Haneberg, Ulrich Virnich, and Stefan Voget. The EAST-EEA project—a middleware based software architecture for networked electronic control units in vehicles. In VDI Verein Deutscher Ingenieure [VDI03], pages 545–563. 5

[THW95] Ken Tindell, Hans Hansson, and Andy Wellings. Analysing real-time communications: Controller area network (CAN). In *Proceedings of the 15th IEEE Real-Time Systems Symposium*, San Juan, Puerto Rico, December 1995. IEEE Society Press. 9

[TS95] David M. Tate and Alice E. Smith. A genetic approach to the quadratic assignment problem. *Computers and Operations Reseach*, 22(1):73–78, 1995. 22

[UD91] Resit Unal and Edwin B. Dea. Design for cost and quality: The robust design approach. *Journal of Parametrics*, 11(1):73–93, 1991. 102

[VAKV02] Douglas A. G. Vieira, Ricardo L. S. Adriano, Laurent Krähenbühl, and Joao A. Vasconcelos. Handling constraints as objectives in a multiobjective genetic based algorithm. *Journal of Microwaves and Optoelectronics*, 2(6):50–58, December 2002. 94, 95

[VDI03] VDI Verein Deutscher Ingenieure, editor. *VDI-Berichte 1789, VDI Congress Electronic Systems for Vehicles*, Baden-Baden, Germany, September 2003. VDI Verlag GmbH. 145, 148

[VDI05] VDI Verein Deutscher Ingenieure, editor. *VDI-Berichte 1907, VDI Congress Electronic Systems for Vehicles*, Baden-Baden, Germany, October 2005. VDI Verlag GmbH. 143, 145

[Vec05] Vector CANtech Inc. DaVinci tool suite: Functional software design for distributed automotive systems, 2005. `http://www.vector-cantech.com/products/davinci.html`. 10

[VL00] David A. Van Veldhuizen and Gary B. Lamont. On measuring multiobjective evolutionary algorithm performance. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, pages 204–211, La Jolla, CA, USA, June 2000. IEEE Press. 25

[W3C04] W3C. Extensible markup language (XML) 1.0 (third edition), February 2004. `http://www.w3.org/TR/2004/REC-xml-20040204/REC-xml-20040204.xml`. 64

[XMT97] Jing Xiao, Zbigniew Michalewicz, and Krzysztof Trojanowski. Adaptive evolutionary planner/navigator for mobile robots. *IEEE Transactions on Evolutionary Computation*, 1(1):18–28, 1997. 22

[ZLT01] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Eidgenössische Technische Hochschule Zürich (ETH), Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001. 16, 92

[ZT99] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, November 1999. 22, 25

[ZTL+03] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, April 2003. 25

Author Index

Index

Curriculum Vitae

Dipl.-Ing. (FH) Bernd Hardung

03. November 1977	geboren in Neustadt a. d. Aisch
09/1983–07/1988	Grundschule Ipsheim
09/1988–07/1995	Georg-Wilhem-Steller Gymnasium Bad Windsheim
09/1995–07/1997	Wolfgang-Borchert Gymnasium Langenzenn, Abschluss: Allgemeine Hochschulreife
07/1997–04/1998	Ableistung des Grundwehrdienstes
05/1998–09/1998	Werkstudent bei der LGA Nürnberg
10/1998–09/2002	Studium der Nachrichtentechnik an der Georg-Simon-Ohm Fachhochschule Nürnberg, Abschluss Dipl.-Ing. (FH)
10/1999–02/2000	Erstes Praxissemester bei der LGA Nürnberg
12/1999–02/2001	Werkstudent bei der Beneke-Lepsinger Unternehmensberatung in Nürnberg
07/2000–09/2000	Werkstudent bei der Deutscher Supplement Verlag GmbH in Nürnberg
03/2001–09/2001	Zweites Praxissemester bei Siemens Malaysia Sdn. Bhd.
02/2002–09/2002	Diplomarbeit beim Institut 01Plus in Nürnberg
10/2002–08/2006	Doktorand bei der AUDI AG in Ingolstadt
09/2006–heute	3SOFT GmbH