

Anwendung multi-objektiver MaxSAT-Solver für die Optimierung hochgradig konfigurierbarer Produkte

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Noah Frederik Bruns, BSc.

Matrikelnummer 01630029

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Stefan Woltran

Mitwirkung: Senior Lecturer Dipl.-Ing. Dipl.-Ing. Dr.techn. Wolfgang Dvořák

Dipl. Ing. Stefan Willminger

Wien, 21. Oktober 2024

Noah Frederik Bruns

Stefan Woltran

Application of Multi-Objective MaxSAT-Solvers for Optimizing Highly Configurable Products

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Noah Frederik Bruns, BSc.

Registration Number 01630029

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Stefan Woltran

Assistance: Senior Lecturer Dipl.-Ing. Dipl.-Ing. Dr.techn. Wolfgang Dvořák
Dipl. Ing. Stefan Willminger

Vienna, October 21, 2024

Noah Frederik Bruns

Stefan Woltran

Erklärung zur Verfassung der Arbeit

Noah Frederik Bruns, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, habe ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 21. Oktober 2024

Noah Frederik Bruns

Danksagung

An dieser Stelle möchte ich mich bei allen herzlich bedanken, die mich beim Schreiben dieser Arbeit unterstützt haben.

An erster Stelle möchte ich mich bei meinem Betreuer Prof. Stefan Woltran bedanken, der mich durch den gesamten Entstehungsprozess hindurch begleitet hat und sich die Zeit für ausführliches Feedback und die qualitative Betreuung genommen hat.

Danke auch an Stefan Willminger für deine Innovationsbereitschaft und deine Expertise im Bereich Plattform- und Komplexitätsmanagement. Deine Begeisterung für das Thema hat bei mir das Interesse daran erst entfacht!

Weiters möchte ich mich bei meiner Partnerin Daniela Marecek bedanken, die diese Arbeit akribisch auf Formulierungsfehler und Nachvollziehbarkeit hin untersucht hat und ein essenzieller Motivator war, um stetig daran weiterzuschreiben.

Weiters geht ein aufrichtiger Dank an das ganze *Team-Stefan* von *EFS Consulting* für die gemeinsame Aufbereitung des Beispiels der TT-Plattform. Es hat Spaß gemacht, mit euch gemeinsam die verschiedenen Aspekte der Modellautos zu diskutieren, und ich bin immer wieder begeistert von dem vielen Wissen und den spannenden Ideen, die im Kollektiv entstanden sind!

Genauso möchte ich mich bei Guido Witt-Döring bedanken, der sich die Zeit genommen hat, mir zur Verständlichkeit sowie dem Inhalt dieser Arbeit Feedback zu geben.

Last, but not least: Danke auch an meine Familie, besonders meine Mama Debby und meine Großeltern Uta und Ulrich sowie Ernst-August und Margarete, dass ihr mir das Studieren ermöglicht und mich stetig dazu motiviert habt, Neues zu lernen.

Acknowledgements

I would like to take the opportunity at this point to thank everyone who supported me in writing this thesis.

First and foremost, I would like to thank my advisor Prof. Stefan Woltran, who accompanied me throughout the writing process and who took the time to provide detailed feedback and qualitative support.

Many thanks to Stefan Willminger for your innovative spirit and expertise in platform and complexity management. Your enthusiasm for the topic has sparked my interest in the first place!

I would also like to thank my partner Daniela Marecek, who meticulously checked this thesis for formulation errors and in terms of understandability and who was an essential motivator to keep me writing.

Furthermore, sincere thanks goes to the whole *Team Stefan at EFS Consulting* for preparing the example of the TT-Platform. It was a pleasure to discuss the various aspects of model cars with you, and I was amazed by the vast amount of knowledge and exciting ideas that emerged collectively!

I would also like to thank Guido Witt-Döring for your valuable feedback on the comprehensibility and content of this work.

Last but not least, special thanks to my family, especially my mom Debby and my grandparents Uta and Ulrich as well as Ernst-August and Margarete, who have enabled me to pursue my studies and have consistently inspired me to embrace new learning opportunities.

Kurzfassung

Viele Unternehmen und Hersteller, insbesondere in der Automobilindustrie, streben nach dem Ziel, Massenindividualisierung bei gleichzeitiger Beschleunigung der Produktiterationen zu ermöglichen. Zum Bewältigen dieser Herausforderungen haben sich Produktplattformen und intelligente modulare Ansätze mit dem Ziel etabliert, dem Kunden/der Kundin eine größere Auswahl zu bieten und gleichzeitig die Komplexität in der gesamten Wertschöpfungskette minimal zu halten. Feature-basierte Dokumentation (FBD) wurde als De-facto-Standard eingeführt, um eben diese Komplexität zu bewältigen. FBD definiert Features (Merkmale), welche die Varianten eines Produkts beschreiben und ermöglicht basierend auf diesen Merkmalen, eine dynamische Zusammenstellung neuartiger Produktkonfiguration aus einer bestehenden Produktplattform (SuperBoM).

Diese Arbeit präsentiert ein neuartiges Framework, um die Anwendung von MaxSAT-Solvern zur Optimierung hochgradig konfigurierbarer Produktplattformen zu ermöglichen. Verschiedene, oft widersprüchliche Optimierungsziele existieren im gesamten Produktlebenszyklus, die aufgrund der schieren Anzahl an möglichen Produktkonfigurationen oft schwer zu quantifizieren sind. Ein Beispiel ist die *geführte Konfiguration*, die darauf abzielt, den Kunden zu einem Produkt hinzuführen, welches schneller lieferbar sowie günstiger zu produzieren ist oder einen reduzierten Verkaufspreis hat. Darüber hinaus gibt es in der Planungsphase viel Optimierungspotenzial wie die Reduzierung der technischen Lösungen bei der gleichzeitigen Erhöhung des *Market-Fit*, die Erstellung besserer und abgesicherter Preismodelle, oder die Reduzierung des ökologischen Fußabdrucks bei der gleichzeitigen Verbesserung des Deckungsbeitrags.

Um diese Auswertungen zu ermöglichen, wird nach einer allgemeinen Einführung von FBD eine formale Definition gegeben, die gemeinsam mit einer ebenfalls vorgestellten und evaluierten Transformationspipeline zur Reduktion von FBD-Instanzen auf das Erfüllbarkeitsproblem fungiert. Im Kern der Arbeit werden verschiedene moderne MaxSAT-Solver vorgestellt, für die eine Zusammenfassung sowie ein Vergleich der Lösungsansätze gegeben werden. Mit dem Ziel die praktische Effektivität von MaxSAT in diesem Bereich zu evaluieren, dient die Produktplattform der Tamiya TT-RC-Car-Serie als Grundlage für unterschiedliche Experimente. Es werden mögliche Zielkonflikte untersucht, um die resultierenden Kompromisse anhand von Pareto-Fronten zu beschreiben und zu analysieren. Die Experimente werden qualitativ bewertet und ihre mögliche Praktikabilität und Auswirkungen für reale Anwendungsfälle werden diskutiert.

Abstract

Companies strive to achieve mass customization while accelerating product iterations in the modern manufacturing landscape, particularly in the automotive industry. They utilize product platforms and intelligent modular design techniques to tackle this challenge with the aim of delivering a broader range of choices to the customer while reducing the complexities throughout the company – from engineering, logistics and manufacturing to sales. Feature-based Documentation (FBD) has been introduced as a de-facto-standard to deal with the complexity of having highly configurable products. FBD introduces *features* that describe the variations of a product. Based on these features, conditions allow for the dynamic composition of novel product configurations based on an existing product platform (SuperBoM).

This thesis will present a novel framework for applying MaxSAT-Solvers to optimize highly configurable product platforms. Many inherent and often conflicting objectives exist throughout the product lifecycle that are hard to quantify due to the sheer volume of possible configurations. An example is *guided configuration*, intending to direct the customer to a product that is faster to deliver, cheaper to produce or has a reduced sales price. Furthermore, in the planning phase, product optimizations include reducing technical solutions while increasing market fit, creating better pricing models, or reducing the environmental footprint while improving the contribution margin.

To facilitate the research, a general introduction to FBD and a formal definition that serves as the foundation for the consequent steps are presented along a transformation pipeline that reduces FBD-Instances to the satisfiability problem. To approach the core of the thesis, several modern MaxSAT-Solvers will be presented, giving a summary and comparing how the different solvers approach the problem. With the aim of evaluating the effectiveness of MaxSAT in this domain, the product platform of the Tamiya TT RC-Car-Series will serve as a baseline for various experiments. Several exemplary conflicting objectives are explored and the resulting trade-offs are visualized using Parteo-Fronts. The experiments will be evaluated qualitatively, and their possible impact and practicability will be discussed.

Contents

Kurzfassung	xi
Abstract	xiii
Contents	xv
1 Introduction	1
1.1 State-of-the-Art	2
1.2 Related Work	7
1.3 Open Challenges	9
1.4 Main Contribution	9
1.5 Structure of the Thesis	10
2 Background & Preliminaries	13
2.1 Approaching the SAT-Solver	13
2.2 Feature-based Documentation	17
3 State of the Art & Contribution to the Domain	33
3.1 Distinctive, Complete and Consistent	33
3.2 Exclusivity	35
3.3 Module Interface Analysis	37
3.4 Involvement of the Author	40
3.5 EFS Modularity Suite	41
4 Reducing to the Satisfiability Problem	43
4.1 Step 1: Parsing Code Conditions	44
4.2 Step 2: Transformation to CNF	48
4.3 Step 3: The SAT-Solver	50
4.4 Performance Evaluation	50
5 Introducing MaxSAT-Solvers	53
5.1 Problem Definition	53
5.2 State-of-the-art MaxSAT-Solvers	55
5.3 Approximation Techniques for MaxSAT	58
	xv

5.4	Multi-Objective MaxSAT-Solvers	63
5.5	Optimization Objectives in FBD	70
6	Experiments and Evaluation	75
6.1	Implementation Details	75
6.2	Best Fit vs. Sales Price	76
6.3	Best Fit vs. Delivery Time	77
6.4	Sales Price vs. Material Cost	79
6.5	Market Reach vs. Technical Solutions	80
6.6	Market Fit vs. Weighted Technical Solutions	82
6.7	Generalizability of the Experiments	85
7	Summary & Outcome	87
7.1	Conclusio	87
7.2	Outlook & Open Questions	87
A	Module Interface Graph	89
	List of Figures	91
	List of Tables	93
	List of Algorithms	95
	Glossary	97
	Acronyms	99
	Bibliography	101



Introduction

“By 2026, configuration life cycle management will transform 40% of manufacturers, reducing the amount of customer-specific engineering required to deliver products.” - *Gartner Report of Top Strategic Technology Trends in Asset-Intensive Manufacturing for 2023* [MH23]

Configuration Lifecycle Management (CLM) is listed by Gartner’s report as one of the top transforming trends for the manufacturing industry in 2023. CLM describes the management of all the product configurations over the complete lifecycle from concept and engineering till after-sales [MRH18]. This form of management is becoming more and more necessary because the trend of *mass customization* and *composability (modularization)*, make it indispensable for manufacturers to have a proper management and documentation system to keep track of their highly complex and configurable products [MH23]. The goal is shifting from offering a limited set of predefined product configurations to being able to serve customers –that want to buy a common category of a product– with specific adaptations for their custom use-case without the need of time-consuming re-engineering.

To achieve this goal a well documented modular product platform is needed that provides the needed agility to enable the necessary configurability of the product. Modular product platforms are concepts that have been around for a long time in the world of manufacturing [Mar, Muf99]. Along with the mass-producing capabilities given by the Industrial Revolution, it became increasingly viable to produce sub-parts on a large scale and to combine them into a divers product. With this approach, products can be created that better fit the customer’s wishes and can be acquired for cheaper. For companies in the automotive industry, the proper design of their respective platform has become a competitive priority and is seen as a key enable for flexibility, cost reduction and to deal with a global product rollout [Muf99].

One goal of modularity is to achieve the most possible product diversity while using the smallest amount of parts. Figure 1.1 visualizes this goal by schematically showing

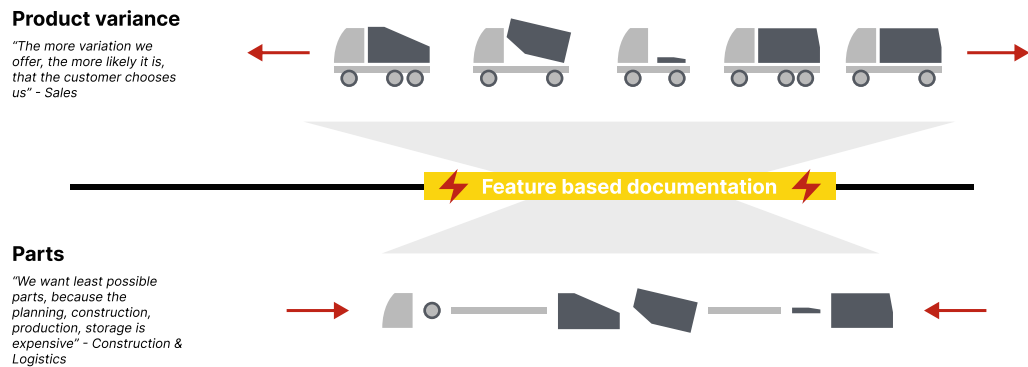


Figure 1.1: Schematic explanation of the goal of modularity and the different interests in a company that are involved (freely adapted from [Man])

different types of truck vehicles. To realize this, we need thorough knowledge about the interfaces of the parts and about their possible combinations as well as a good language to describe the variation. One such language is Feature-based Documentation (FBD). It is the central concept around which this thesis is built and will be thoroughly described in Section 2.2.

FBD is built around logic expressions, which allow for dynamic composition of products based on the customer choices. Due to these inherent logical conditions, satisfiability solving is a key technique to apply in the domain offering a huge variety of possible use-cases. Hence, thesis aims at studying the application of SAT-Solvers and MaxSAT-Solvers and explore different promising applications to analyze, verify, and improve FBD instances.

1.1 State-of-the-Art

Modularization is a widely used concept for creating diverse product platforms that increase the market fit, especially in industrial applications where there is a need for specialized solutions. Based on the experience of the author, currently, many companies create their own, tailored documentation systems to handle the complexity of modular product platforms. They are aided by consulting- and implementation companies which offer solutions to tackle feature based on systems like SAP, Windchill, etc. This leads to a high diversity in solutions, but still the base concept of using features to handle the variations is very common. Many such systems are marketed under the term *CPQ*.

1.1.1 CPQ-Systems

Configure Price Quote (CPQ) is a process that can be implemented by companies that produce customized products. It is an umbrella-term that describes processes that aim at improving the customer's journey from configuration, pricing till the quotation [JAJK20]. Traditionally this process consists of many manual steps especially in companies that work in the B2B market, where supplier agents work out deals with customers in response to a tender. The configuration-process requires a lot of expert knowledge of the agent about the product specifics and may lead to time-consuming clarification steps with the engineering department about the product parts, material costs, etc. This is a bottleneck for many companies because the customer wants good, authoritative answers in a timely manner, otherwise they may choose a competitor company, which tailors to their needs more efficiently.

As a consequence, software systems have come into existence that streamline the CPQ-process by encoding the relevant product knowledge and enabling the sales person or team to authoritatively answer questions about configurability, pricing and availability. And doing that with the click of a button and without the need for extra clarification loops with other subsidiaries of the company.

CPQ is the first step in the value chain, and it should be aligned with the production line and the product itself to fully streamline the customer experience till the delivery [MRH18]. We can address this with a proper modularization of the product [Jan10]. Modularization tries to divide the product into clearly concealed parts and defining interfaces between the parts that aim at enabling the most possible reuse of parts and highest amount of flexibility. This aligns with the paradigm of *separation of concerns*.

Modularization has profound influences about how products are designed and manufactured and induces profound knowledge about all the aspects of the product and its whole lifecycle. *Modular function deployment* is a tool developed to provide a systematic way of approaching modularization and collecting the relevant product knowledge.

1.1.2 Controlling Design Variants

The concept of Modular Function Deployment (MFD) was developed by Anna Ericsson and Gunnar Erixon from the University of Stockholm and Modular Management AB [EE99]. They define a technique to manage, plan and document the modularity of products. The goal is to apply the concept of modularity across the whole company, from sales to through engineering to production to leverage the effects of a streamlined and well-integrated company.

They documented their method in the book *Controlling Design Variants* which is published by the Society of Manufacturing Engineers [EE99]. The following sections will give a broad overview of the method described in the book.

Module Drivers™

To translate the positive effects of modularity into product design, companies have to recognize the driving forces of modularity over the whole product lifecycle. These driving forces have been studied and identified by the Swedish Institute of Production Engineering Research and the Royal Institute of Technology, Stockholm, Sweden through a great number of case studies. In the following, these twelve Module Drivers™ will be described.

- **Carryover**

Some parts of a product do not need to be exposed to any design changes during the whole product lifecycle. Therefore, we do not have to "reinvent the wheel" and can carry that part over to the next generation.

An example is the window crank mechanism. For the customer it is not necessary to know which specific mechanism is used. As long as it fulfills their requirements, it can be carried over to the next product generation.

- **Technology evolution**

Shifts in technology can cause changes in parts that need a refitting to continue to fulfill the customers changing demands. The technological evolution from mechanical to mechatronic or novel materials force adaptations and are the driving forces for such changes. An example is the CD-Player which was faded out by new technologies like Bluetooth or CarPlay.

- **Planned Product Changes**

Parts are planned to be changed or added later in time. Companies may have a plan laid out for the future development of the product. In it, transitions are planned out that require product changes that need to be taken into account in the current design of the product.

- **Different Specification**

Specifications for a product can vary for many reasons like different power outlets for different countries or different regulations.

- **Styling**

Trends or fashion can also influence a product and lead to different designs. The visible parts of the product relevant for styling can be separated into styling modules to simplify refitting to new stylings.

- **Common Unit**

A common unit is a part that can be used for a large part -or ideally the complete product assortment. Good candidates for such parts are ones whose functionality is used by a wide range of customers.

Compared to Carry Over, these parts are not part of the product for a long time and might only serve for one product generation. Carry Over parts however might be used in only a few products, but for a very long time.

- **Process and/or Organization**

To streamline the production process, parts that need similar processes and production steps can be clustered. For example parts that need welding could be combined into one single module.

- **Separate Testing**

Testing and verification to ensure that the product meets the requirements of the specification and relevant laws is a crucial step. The ability to test parts independently before they are used in the final product is very important and may lead to quality improvements.

- **Availability from Supplier**

A very important consideration in the process of creating a product is how much should be produced in-house and what makes sense to buy off the shelf from suppliers. These decisions are very strategic and have to consider prices to buy as well as the costs of failure of the supplier.

- **Service and Maintenance**

Painless service and maintenance is a very important aspect, especially in the field of industrial applications. Parts, ideally those that may require service, are exposed and easy to reach and also clustered by functionality, so that in the case of failure a certain functional module can be swapped.

For example in the Swedish high-speed train X2000 electric wires and hydraulic tubes are assembled in drawers that can easily be swapped and exchanged with pre-assembled boxes.

- **Upgrading**

Building upgradable products can give customers the possibility to extend the lifetime of the product and adapt it to their needs and anticipate technological improvements. For example, many personal computers have the options to upgrade RAM or hard disk space.

- **Recycling**

Environmental aspects are taken into account more and more when developing new products and therefore not only the aspect of production has to be considered, but also how a product can be disassembled and recycled. This can be achieved by limiting the amount of different materials, or keeping fixations between parts simple to disassemble.

These module drivers offer a fundamental tool to make sound and holistic decisions about the modules that a product will have. To accompany these decisions, the MFD-Process was developed as a process for continuous improvement of the modular structure.

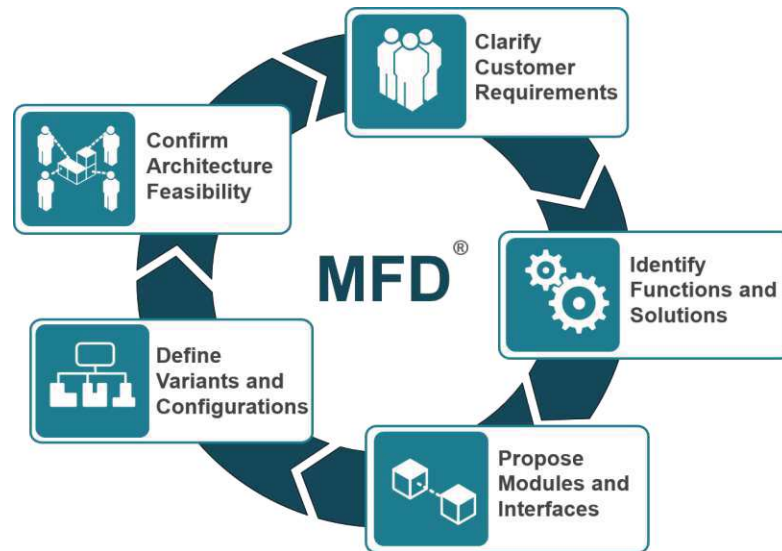


Figure 1.2: Apply Modular Function Deployment (MFD) to create product architectures that meet customer needs, strategic targets and functional requirements. From <https://www.modularmanagement.com/en/mfd> accessed on 14.02.2024 15:07

Modular Function Deployment™

The MFD-Process consists of five major steps as illustrated in Figure 1.2. These steps involve method of structured data collection methods (often matrices) that guide a cross-functional team through the whole process step-by-step.

The first step focuses on deriving the product requirements of the customers demands. The output is a weighted mapping of customer demands to product properties. It matches the *What* with the *How*.

In the second step, different technical solutions are proposed and evaluated along several criteria with the focus of finding the solutions, that fit best with the customer's needs.

Within the third step, the core of the method, the technical solutions are analyzed according to the module drivers and reasons for why certain solutions should be grouped into modules and which should be split up.

In step number four, interfaces for the created modules are defined and analyzed. These interfaces have influence on the composability of the product and on how the production line can be set-up to produce the product. This also involves economical predictions about the expected effects of modularization.

In the last step, a specification is created for each module containing relevant information and from here the implementation of the defined module structure can be triggered. At the same time, the process starts the loop again to further optimize the module structure and stay up-to-date with the customer's needs.

1.2 Related Work

This section will provide an overview of the relevant scientific work related to the topic of this thesis. Plenty of scientific work has been done, out of which three will be highlighted in the following sections.

1.2.1 BA-Thesis

The author has written a bachelor thesis about the *Application of SAT-Solvers in Feature-based Documentation Systems* [Bru20]. In this thesis the author introduced definitions and theorems to verify the correctness of FBD systems utilizing SAT-Solvers. It also suggested that for some tasks in this context, MaxSAT-Solvers are promising tools, but it did not further explore this proposal.

Chapter 3 will revisit this work and introduce new contributions and refinements which were introduced since the bachelor thesis has been written.

1.2.2 Modularity

There are architectural concepts that facilitate modularity. One first proponent was the Architect Albert Farwell Bemis who developed a concept for building houses based on a modular concept [Rus12]. Bemis suggested cooperation with architects, manufacturers and laborers to define common standards for the dimensions of building materials. His "four-inch cubical module" was his approach to have a sound modular system with which to build houses and tackle the issues that the housing industry of 1920 to 1930 was having.

1.2.3 Software Configuration

The Linux operating system has undoubtedly become one of the cornerstones of modern software development and counts as the largest open-source software development project with more than 21 million lines of code, 4,000 developers and more than 440 different companies that contributed [Bha16]. Linux also supports an impressive amount of close to 20 different CPU-Architectures¹.

To handle the diversity of use cases that the Linux kernel is operating in, the developers have placed around 15.000 different feature flags in the code to configure the kernel to the specific needs of a certain application [FBF⁺21]. These feature flags can be used to selectively enable and disable certain functionalities of the codebase.

For large *software product lines*, like the Linux kernel, which offer an excessive degree of variation, it is important to properly model the dependencies and compatibilities [KKS⁺23]. One very prominent way for modelling is the feature diagram which arranges the features in a hierarchical tree structure and defines the constraints between them

¹List of supported CPU-Architectures is available at <https://www.kernel.org/doc/html/v6.3/arch.html> accessed on 24.9.2024 15:49

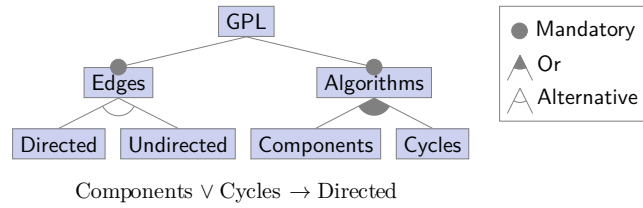


Figure 1.3: Example feature diagram for a graph product-line. Taken from [KKS⁺23]

using annotations on the parent/child relations. In Figure 1.3 the feature diagram of a graph product line is illustrated. The example defines seven different features that describe the different types of Edges (directed, undirected) as well as the algorithms needed to find components or cycles.

All these features have relations with each other. For example, the *Edges* as well as the *Algorithms* are mandatory for the *GPL* feature, as indicated, using the mandatory relation (Figure 1.3). *Edges* have two alternatives, either *Directed* or *Undirected*. For the available algorithms, an *inclusive-or* is used, therefore the Features *Components* or *Cycles* can be selected or also both. Furthermore, we can use a cross-tree constraint to ensure that for the algorithms, directed edges are used.

SAT-Solvers play a crucial role in the analysis of *software product lines* [FBF⁺21]. They are used for many highly relevant tasks [KKS⁺23]:

- **Interactive Configuration** Improving the feedback for the user during a configuration process and taking transitive constraints into account [Jan08].
- **Modularization** Identifying feature model interfaces and the compositional analysis of a model can be reduced to the satisfiability problem [SKT⁺16, KST⁺16].
- **Explanation of Anomalies** Anomalies like redundant constraints, false optional features, dead features, etc. can be expressed formally and verified by the SAT-Solver [FBGR13].
- **Model Checking** Feature interactions that lead to unintended –possibly critical– behavior of a software can be automatically detected [ASW⁺11].
- **Formal Verification** Verifying every product individually in a large software product line does not scale, but by using a meta-product that contains all the features, a complete verification is feasible. [TSAH12].

Comparing FBD to Software Configuration and the feature diagrams, one essential difference is the hierarchial representation of the features. In Feature-based Documentation features are part of groups that do not have inherent hierarchial relations. The relations between the groups are defined using implications separately. The example of the *GPL*

can be modelled with three feature groups, one for directed/undirected and one for each of the algorithms, defining *With* and *Without* the algorithm. The parent structure of *Edges* and *Algorithms* is not represented in FBD. Restrictions can be used to define the relation $\text{Components} \wedge \text{Cycles} \rightarrow \text{Directed}$.

1.3 Open Challenges

Satisfiability Solving and Automated Verification have been studied and proven valuable in the context of software configuration (Section 1.2.3) and mechanical product configuration [Bru20]. These static methods can verify the correctness and analyze huge FBD instances with an innumerable amount of possible configurations.

However, there are new challenges emerging that are not addressable solely by the SAT-Solver. Product creation involves numerous –often conflicting– objectives: we want to reduce the price, increase profit, improve the market fit, reduce delivery times, and many more. These challenges require MaxSAT-Solvers, since these *soft* objectives need to be taken into account alongside the *hard* buildability-restrictions.

The challenge to optimize towards these objectives systematically is to define a proper encoding and facilitate suitable solvers that can optimize towards the given objectives. Because many of these objectives are contradictory in their nature (e.g. reduce the price vs. increase profit) we need to argue about trade-offs and visualize Pareto Fronts to find optimal solutions.

1.4 Main Contribution

In this thesis the author aims at addressing the open challenge of MaxSAT-Solvers in FBD and along the way introduce contributions to the domain and the state-of-the-art methods.

In a first step, in chapter *Background & Preliminaries*, the author contributes the summarization and formal definition of FBD. This is an essential groundwork for the following chapters.

By revisiting the bachelor thesis, in chapter *State of the Art & Contribution to the Domain* the author introduces improvements in the analysis of exclusivity by taking the complete instance –including the restrictions– into account, which makes the method more powerful, since transitive relations of features will be considered. Furthermore, the author has proposed the *Module Interface Analysis* method, which defines a process to systematically identify and optimize modular products by studying their interfaces. The output of the method is a FBD instance that contains the needed technical solutions to cover the intended product variations.

Approaching the SAT-Solvers, in chapter *Reducing to the Satisfiability Problem*, the author has studied the Pratt Parser and the Tseitin Transformation including improvements

introduced by Plaisted and Greenbaum as well as subformulation tracking. Together, these improvements yield a noticeable difference in the performance and parsing verbosity compared to the bachelor thesis [Bru20] as evaluated in *Performance Evaluation*.

Lastly and mainly, the contribution lies in the aspect of MaxSAT-Solving for FBD in Chapter *Introducing MaxSAT-Solvers* and Chapter *Experiments and Evaluation*. Three things are mentionable contributions in this context:

Firstly, the author has surveyed and categorized state-of-the-art to MaxSAT-Solvers, and summarized their algorithms. These include, exact solvers, approximation solvers and multi-objective solvers.

Secondly, the author proposed objectives in FBD and provided their formal definitions. Utilizing the reduction pipeline, these objectives can be added to the MaxSAT-Solver instance as objectives.

Thirdly, experiments are conducted on the TT-Platform to qualitatively evaluate the effectiveness of the solver to optimize toward a given objectives. Furthermore, the trade-offs for the respective conflicting objectives are analyzed and explained.

1.5 Structure of the Thesis

This thesis is structured in five main parts after the introduction.

The chapter *Background & Preliminaries* will explain the background and relevant knowledge that is needed to understand SAT-Solvers and the domain of FBD. To explain FBD, an example product will be introduced that serves as the example in the whole thesis.

Afterwards, *State of the Art & Contribution to the Domain* will take a look at several contributions made to the domain by the author's bachelor thesis and mention improvements. Furthermore, the author will introduce a new process to create a FBD-Instance and also introduce the *EFS Modularity Suite*, which has made a lot of the introduced techniques accessible in a web-tool.

Reducing to the Satisfiability Problem describes how the domain of FBD can effectively be reduced to the satisfiability problem. This chapter will guide through all the steps of the transformation and explain alternatives in the approach at every step.

Chapter *Introducing MaxSAT-Solvers* will explain what MaxSAT-Solvers are and survey several MaxSAT-Algorithms. It will also introduce the Pareto Front and multi-objective optimization for dealing with conflicting optimization targets. Lastly, it will formally define objectives that can be applied to solve various problems in the domain of FBD.

The presented optimization objectives and MaxSAT-Solvers will be put to the test in the Chapter *Experiments and Evaluation*. Several experiments for multi-objective optimization will be conducted on the example product, and the generalizability of the experiments to a larger-scale product will be discussed.

Lastly, the chapter *Summary & Outcome* will summarize the results and contributions of this thesis.

Background & Preliminaries

2.1 Approaching the SAT-Solver

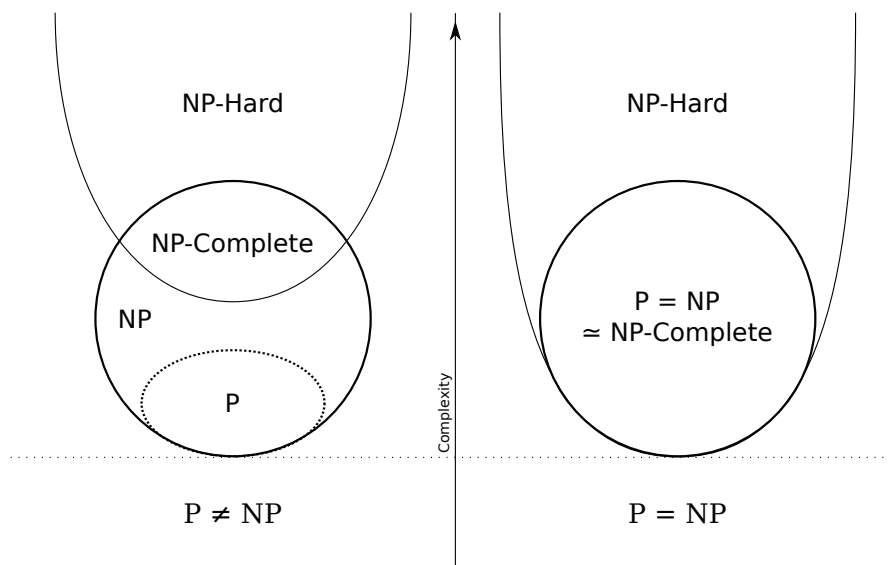


Figure 2.1: Euler diagram for P , NP , NP -complete, and NP -hard set of problems. From https://en.wikipedia.org/wiki/P_versus_NP_problem accessed on 7.3.2024 16:06

In the realm of computational complexity, it is essential to classify algorithms on their needed computational power. One very important differentiation is between \mathcal{P} and \mathcal{NP} [For09]. It addresses the question, whether something that can be verified efficiently (in polynomial time, thus called \mathcal{P}) can also be solved efficiently (\mathcal{NP} , standing for

Non-deterministically Polynomial). So far, there has been no answer to this question, and it stands as one of the big questions for modern-day computer science.

We differentiate two cases, either $\mathcal{P} = \mathcal{NP}$ which would mean that, a problem that can be verified efficiently can also be solved efficiently, or $\mathcal{P} \neq \mathcal{NP}$ which means there are problems that are harder to solve than to verify. For now, we know of many such problems that we can verify efficiently, but do not (yet) know an algorithm that can efficiently solve them. Therefore, we have to submit to the assumption that $\mathcal{P} \neq \mathcal{NP}$ and deal with the complexities that arise from it.

The challenge of computer science often lies on the boundary between \mathcal{P} and \mathcal{NP} . Finding the best possible algorithm for a given problem can be really challenging and hard, especially if the problem is \mathcal{NP} . Fortunately there is a further class that we call \mathcal{NP} -complete. Every problem in \mathcal{NP} can be effectively reduced (converted) to any problem in the \mathcal{NP} -complete class. In other words there is a polynomial time conversion, which we can utilize to convert any \mathcal{NP} problem into a \mathcal{NP} -complete one. The Cook-Levin theorem serves as proof for this fact and uses the problem of Satisfiability Solving as a prime example of a \mathcal{NP} -complete one.

Satisfiability Solving is therefore a cornerstone algorithm that enables the solving of numerous problems. If we face a \mathcal{NP} -Problem in any domain now, we know that there must be a way to convert the problem to the satisfiability problem. This is a very important approach, and we will make use of this fact in many of the following chapters.

2.1.1 Satisfiability Solving

Given a propositional formula ϕ , Satisfiability Solving can be defined as the finding of an assignment A of each variable in ϕ to $\{0, 1\}$ such that ϕ is true. If such an assignment can be found we consider ϕ SAT else ϕ UNSAT [MSLM21].

SAT-Solvers require the formula ϕ to be in Conjunctive Normal Form (CNF). Any arbitrary propositional formula can be converted to CNF. Chapter 4 will describe the reduction steps needed for arbitrary propositional logic.

CNF

A CNF-formula is a conjunction of clauses ω over boolean Variables $X = \{x_1, x_2, \dots, x_n\}$. Each clause is a disjunction of literals, whereby a literal is a variable either in positive x_i or negative $\neg x_i$ form [MSLM21].

Example 1 A simple exemplary CNF formula

$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_3 \vee x_4) \quad (2.1)$$

The CNF can also be represented as set of sets:

$$\phi = \{\{x_1, \neg x_2\}, \{\neg x_2, x_3\}, \{\neg x_1, \neg x_3\}, \{\neg x_3, x_4\}\} \quad (2.2)$$

2.1.2 From DPLL- to the CDCL-Algorithm

Recent years have shown a lot of progress in the domain of SAT-Solvers. [KIMS21] There is a wide consensus about this fact, and it manifested in several SAT competitions year by year, where improved versions of the previous year or even completely new solvers have been released and are able to reach better and better performance scores¹.

One of the first SAT-Algorithms was the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [OC99], which is the core of many modern SAT-Solvers, though many optimizations have been added over the years. The DPLL-Algorithm works by applying unit-propagation until there is no more unit to propagate. Afterwards, it picks a literal and assumes that to either be true or false and recursively calls DPLL for each case. If both cases are UNSAT, then the result is UNSAT.

Algorithm 2.1: DPLL-Algorithm [OC99]

Data: ϕ in CNF Format
Result: SAT or UNSAT

```

1 while exists  $\{u\}$  in  $\phi$  that has length of at most one do
2   |  $\phi \leftarrow \phi|_u$ 
3 end
4 if  $\phi$  is empty then
5   | return SAT;
6 end
7 if  $\phi$  includes an empty clause then
8   | return UNSAT
9 end
10  $u \leftarrow$  choose a literal in  $\phi$ ;
11 if  $DPLL(\phi|_u)$  is SAT then
12   | return SAT;
13 end
14 if  $DPLL(\phi|_{\neg u})$  is SAT then
15   | return SAT;
16 end
17 return UNSAT

```

Unit Propagation

Unit propagation is happening in lines 1-3 in Algorithm 2.1. A unit-clause is a clause which has only one literal, and because the clause must have at least one literal satisfied, we can assume the literal to be satisfied. As a consequence, we can assume this literal to be true and consider all clauses in which the literal occurs as satisfied. In clauses where the literal occurs in the negated form, we can remove the literal. This might lead to new

¹<http://www.satcompetition.org/> accessed on 15.10.2024 22:28

unit clauses, and we can repeat this process till there is no unit clause left. Hence, it is called unit propagation.

CDCL

Conflict-Driven Clause Learning (CDCL) Solvers are an improvement of the DPLL-Solvers since they do not need to copy ϕ for each branch. A lot of research has been done to improve CDCL and many key techniques have been added over the years [MSLM21]:

- Using backtrack search to learn new clauses [SS97].
- Exploring the structure of the implication sequence to find the assignments that are directly responsible for the conflict [SS97].
- Utilizing lazy data structures for deleting clauses that were added during clause learning [MMZ⁺01].
- Reducing the overhead on the decision strategy to make more efficient decisions. [MMZ⁺01].
- Restarting the solver using randomization to mitigate the solver getting stuck in long tails [GSK98].

2.1.3 Prominent Solver Backends

In the world of science as well as the industry, there has been a lot of interest in advancing SAT-Solvers and making them more performant, reliable and simpler to use. This goal has lead to *The International SAT Competition*², an international competition that takes place annually. Numerous SAT-Solver implementations compete against each other to evaluate which one is the best on a wide variety of different problems.

Several prominent solver backends have been the result of this competition. Here is a small overview over what the author has identified as prominent solvers:

- **MiniSat** is a solver developed to be minimalistic, open-source and highly efficient. It performed really well in the 2005 competition and aims at being a good starting point for researchers and developers to get started with SAT.
Website: <http://minisat.se/> accessed on 15.10.2024 22:28
- **Glucose** has ranked top places in different categories over several years since its introduction 2009. It is heavily based on MiniSat in its core but improves the solver by focusing on removing "bad" learnt clauses and thereby enhancing performance.
Website: <https://www.labri.fr/perso/lsimon/research/glucose/> accessed on 15.10.2024 22:28

²The International SAT Competition Web Page <http://www.satcompetition.org/> accessed on 15.10.2024 22:28

- **CaDiCal** is a solver introduced in 2019 and has won first place in the SAT track in that year. CaDiCal is designed to be easy to understand and modify which has made it very influential since many new solvers are based on its implementation, and there is a distinct category of solvers based on CaDiCal in the recent SAT competitions called the *CaDiCaL 1.5.3 Hacks Track*³.

Website: <https://fmv.jku.at/cadical/> accessed on 15.10.2024 22:29

- **Kissat** is a reimplement of CaDiCal in C that focuses on condensing and improving performance. It has won the SAT-Track in 2020. One downside is that it is not yet possible to use it incrementally.

Website: <https://fmv.jku.at/kissat/> accessed on 15.10.2024 22:29

2.1.4 Incremental SAT-Solving

Many of the mentioned solvers support incremental solving, where clauses can be added to the solver in phases [FBS19]. For each *solve* call in a given phase, all the clauses added till this phase are considered. Furthermore, a set of assumption literals \mathcal{A} can be supplied, and all solutions τ will hold $\tau \subset \mathcal{A}$ [NR12, JBNJ22]. This extendable and assumption-based approach offers many possibilities for reusing SAT-Instances for different solver calls in a given problem-space.

Furthermore, solvers can extract unsatisfiable cores, if the formula is unsatisfiable, given the assumptions [JBNJ22]. An unsatisfiable core is a –mostly small but not necessarily minimal– subset of the assumptions $\mathcal{A}_{core} \subset \mathcal{A}$, where the formula is also unsatisfiable, given the assumptions \mathcal{A}_{core} .

Practically, unsatisfiable cores offer comprehensible and traceable reasons for understanding the unsatisfiability of a given problem. Therefore, they are useful for debugging as well as analysis tasks.

2.2 Feature-based Documentation

FBD is a documentation system, designed towards modular product platforms. By systematically describing all the product-variations of the platform in terms of features, it facilitates dynamic composition of novel product configurations from predefined modular components. This empowers companies to offer the customer a high degree of customization, while still keeping the documentation manageable.

The creation of a platform based on FBD follows several steps as outlined in Figure 2.2 [WB23]. The lifecycle starts with the *platform definition* phase where the market is analyzed and segmented. The product positioning can consequently be done to target segments of the market that promise a good market fit. Based on the market knowledge, the base-type and the feature lists of the FBD system are created. This phase is finalized

³CaDiCaL 1.5.3 Hacks track <https://satcompetition.github.io/2023/tracks.html> accessed on 15.10.2024 22:28

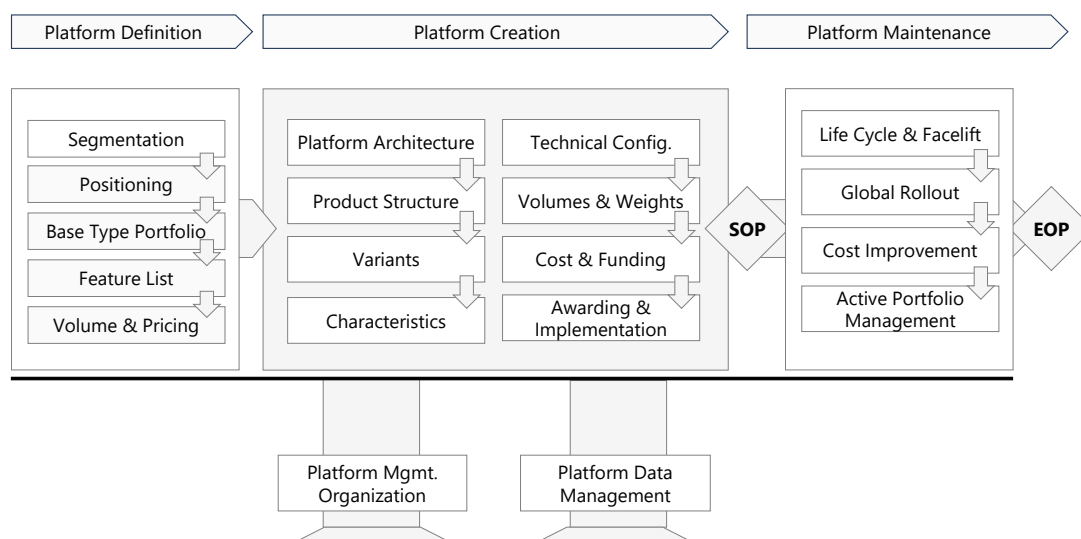


Figure 2.2: Platform Management [WB23]

by planning for volumes and defining the pricing, since now have a concrete definition of what we want to create.

The next step is *platform creation* where we will plan the whole product on the technical side and define all the parts and processes that go into building it. This phase ends with the Start Of Production (SOP) where the first product will be produced for the customer.

After the start of the production, there is still demand for *platform maintenance* because the cost needs to be controlled, regulations be considered, or a facelift is done to slightly upgrade the product to make it more attractive for customers. At the end of this phase is the End Of Production (EOP), where the production of the platform ends.

2.2.1 Defining the Goal

For an Original Equipment Manufacturer (OEM) it is important to be competitive and strive for economic goals [WB23]. These goals may include:

- Serving new markets
- Complying with legislation for emissions, noise, safety and security
- To adapt to increasing customer demands
- To offer new functionality

These challenges force the OEM to constantly extend and adapt the product range to stay ahead on the competition. FBD serves as an enabler for the product platform to

constantly grow and to be extended on a granular level without re-engineering the whole product.

In the product planning process, there are many departments as well as internal and external players involved, each with different objectives of what they want to achieve. These different objectives are the perfect feeding ground for organizational strain. Figure 1.1 illustrates one of those strains.

On the one side, there is the sales department which has the core objective of satisfying the customer and offering the best possible product. Because the better the product fits the customer's needs, the likelier it is that the product is chosen.

On the other hand, there is the sales department. From its perspective the goal is to have as few technical solutions as possible. Every technical solution (in other words part) needs to be developed, tested, constructed and logistically planned, etc. and therefore takes valuable resources that we ought to spend carefully.

The task of variant and complexity management is to deal with these different sides and to weight the advantages and disadvantages of certain product decisions to find a balance between both objectives.

2.2.2 Intro to the Example TT-Platform



Figure 2.3: Lineup of the Tamiya TT-Platform [WB24]

To demonstrate FBD and Modular Product Platforms, *EFS Consulting* has chosen an example product that breaks down many of the aspects and challenges of large product platforms to an understandable and compact version. This example product has been used as the foundation for modularity workshops with different companies from around the world. In the same way, it will also serve in this thesis for different explanations as well as qualitative evaluation of the algorithms.

The chosen product platform is the *Tamiya TT-Platform* for model RC cars which consists of a predecessor, a successor and a derivate of the successor as illustrated in

Figure 2.3. This setup allows us to properly discuss common effects of generation change ($TT01$ to $TT02$) as well as product derivatives ($TT02$ to $TT02B$).

The ratio for choosing model RC cars is that they are functional and not "just toys". They need to properly deal with the mechanical and electrical stresses of driving –often in pretty rough conditions–. They are also extendable and customizable in terms of the wheelbase, motor/ESC configuration, batteries, etc. Therefore, they occupy a middle ground between completely pre-built, fixed product and the limitless configurability of a Lego like system which would be impractical and not purposeful for demonstration in this context.

As an add-on the author has developed an autonomous driving module that extends the TT02 model. The concept was adapted from the F1Tenth Race cars and combined with the NAV2 stack for autonomous driving. Together they provide the ROS2 nodes needed for the hardware interaction as well as the algorithms and controllers to achieve autonomous navigation based on LIDAR data. This also targets the complex world of hardware and software integration.

2.2.3 Features

As the first step, it is important to have a common language to express the variations of a product. This is done by utilizing the features and feature groups.

Each Feature group describes a dimension in which the product can vary. For our TT-Platform such dimensions are the wheelbase, the motor power, clearance, spur width or the Battery size (see Figure 2.4). A feature is a selected value in a feature group. As example for the Feature Group *wheelbase*, a feature would be *257mm*. To order a product, a customer has to select the features that match the given desire/requirement.

Each Feature also has a code assigned to it that represents the variation of change as a concrete value. These codes later serve as variables in the conditions that will be written. In Table 2.1 the exemplary features from Figure 2.4 are assigned to codes. For example the representation of the *wheelbase of 257mm* is *WB_257*

In Table 2.1 a list of all the features groups and their features are given. Altogether, there are 16 groups with 39 codes.

Feature Combination Explosion

Given the Feature list in Table 2.1 we can compute the amount of combinations by multiplying the amount of codes for all feature groups. In the case of this rather simple product, the amount theoretically possible combinations already reaches around 125 thousand (see in Table 2.2). In large scale product platforms, the amount of combinations is often way larger exceeding the 10^{100} possibilities.

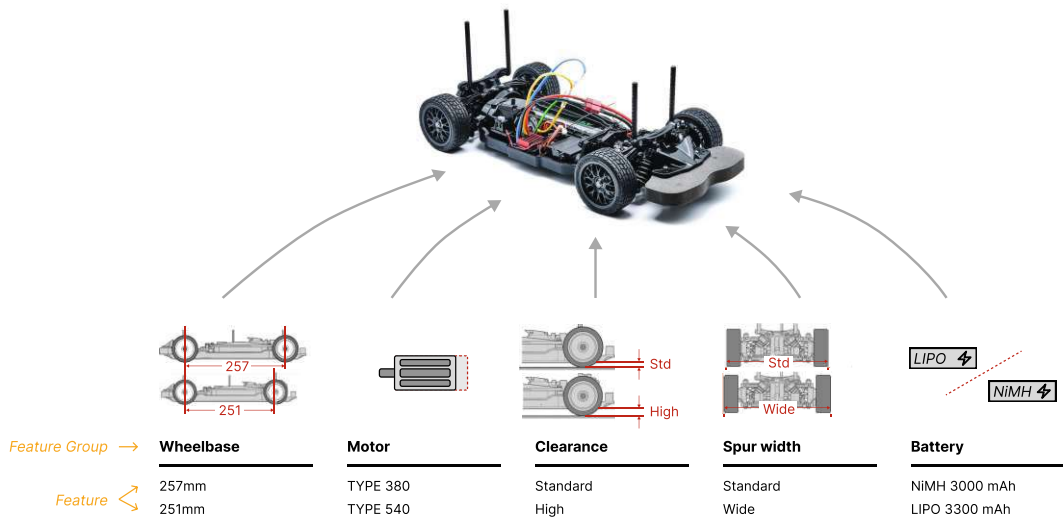


Figure 2.4: Exemplary description of a Feature as a dimension of variation [WB24] (Illustrations taken from the TT-02 Chassis Manual [TAM13])

Configuration Space	
BASE 10	$10^{5.09}$
Total	124 416

Table 2.2: Combination Space

Feature based documentation takes on the challenge of maintaining this huge amount of combinations and enabling a way to build every possible combination, even though the whole configuration space is not practically enumerable. To make this work, Section 2.2.7 will explore a way to dynamically allocate the needed parts and processes, but first, let's take a closer look at the features.

2.2.4 Configurations

A configuration is a selection of features by a customer. It can be represented as a list of feature codes e.g. {WB_257, GC_STD, ..., SV_6V_8K}. This selection of codes is called complete if it contains a code for every feature group. A complete configuration is a prerequisite for a *buildable* product. *Buildable* means that this product can actually be created in a production process and delivered to the customer.

In reality, customers typically select only a subset of features that are important to them and leave out the other ones. This necessitates the automatic *completion* of the

2. BACKGROUND & PRELIMINARIES

TT-Platform Feature List		
Feature Group	Feature Codes	Feature Description
Model Type	TT_01	TT_01_Chassis
Model Type	TT_02	TT_02_Chassis
Model Type	TT_02B	TT_02B_Chassis
Generation	PF_G1	Generation 1
Generation	PF_G2	Generation 2
Usage	ON	Onroad
Usage	OFF	Offroad
Wheelbase	WB_257	Wheelbase 257mm
Wheelbase	WB_251	Wheelbase 251mm
Wheelbase	WB_266	Wheelbase 266mm
Ground Clearance	GC_STD	Ground Clearance std
Ground Clearance	GC_HIGH	Ground Clearance high
Chassis Width	XW_STD	Chassis Width std
Chassis Width	XW_WIDE	Chassis Width wide
Motor Type	M_TYP_540	Motor Type 540
Motor Type	M_TYP_380	Motor Type 380
Motor Type	M_TYP_BRUSHLESS	Brushless Motor
Motor Ratio	RATIO_1961	Gear Ratio 19:61
Motor Ratio	RATIO_2270	Gear Ratio 22:70
Motor Ratio	RATIO_1770	Gear Ratio 17:70
Chassis Damping	XD_AIR	Chassis damping, air
Chassis Damping	XD_OIL	Chassis damping, oil
Suspension Stiffness	XS_SOFT	Suspension stiffness soft
Suspension Stiffness	XS_STD	Suspension stiffness med
Suspension Stiffness	XS_HARD	Suspension stiffness hard
Tire_Wheel	TW_ON_SLICK	Tires; onroad, slick
Tire_Wheel	TW_ON_PROFILE	Tires; onroad, profiled
Tire_Wheel	TW_OFF_SPIKE	Tires; offroad
Body Length	BODY_STD	Body length, std
Body Length	BODY_LONG	Body length, long
Batt Capacity	AK_NIMH_3000	7.2V, 3000mAh (21Wh) - NIMH
Batt Capacity	AK_LIPO_3300	11.1V, 3300mAh (36.63Wh) - LIPO
Servo	SV_6V_8K	Steering servo; 6V/8KG/0,15s
Servo	SV_6V_20K	Steering servo; 6V/25KG/0,14s
Autonomous Control	AI_MANUAL	Manual only control
Autonomous Control	AI_BREAK_ASSIST	Break assist
Autonomous Control	AI_SELF_DRIVING	Full self-driving
Voltage Monitor	WO_VOLTAGE_MONITOR	Without Voltage Monitor
Voltage Monitor	W_VOLTAGE_MONITOR	With Voltage Monitor

Table 2.1: Feature List Table

configuration to ensure that it is *buildable*. This completion process creates opportunities for optimization, which we will leverage using MaxSAT-Solvers in the course of the thesis.

2.2.5 Base-Type

Now that we know in what ways our product platform varies, we have to start defining the actual products that we want to sell to the customer. For this step in product planning, it is really important to properly understand what the product strategy and

market segmentation is and how it affects the products we create.

For the example of the Tamiya RC car, there are several market segments we want to serve. These segments include buggies, race-cars, entry-level cars, etc. Our goal as a company is to serve all those diverse segments with very dissimilar requirements which will lead us to engineer cars they may vary in core technical aspects. Consequently, it makes sense to group them into different base-types and plan each of the base-types separately.

In FBD, base-types share the same feature definitions. They define a subset of features that are allowed for them. Furthermore, a standard value is selected per feature group to deal with optional features. Figure 2.5 visualizes the definition for example base-types by defining S (Standard) and O (Optional) feature selections for each base-type.

These core technical variations can be described by selecting a technical core of base-type features. In the example in Figure 2.5 the technical core is defined as the wheelbase and the motor type. These feature groups are very central to the product. For each feature group in the core, there is a standard feature defined for each base-type and no optional ones are allowed.

A company might have different objectives when creating base-types. Base-types can be defined very broadly to allow many optional values. This would lead to a very diverse product allowing a lot of customer choice. The other option is having a very strong base-type definition which does not allow many options and therefore takes away a lot of choice from the customer.

This decision boils down the product positioning of the company. On the one hand, if the product positioning intends to serve a very specific market segment, it makes sense for the company to have a strong base-type and focus on designing this base-type very well. On the other hand, a company can choose the strategy to target a very broad market spectrum and not specialize very much. In this case, a weak base-type makes sense, leaving a lot of choice for the customer.

Defining the Technical Core

Base-types have a deep technical implication on the product's documentation because every base-type will be documented and planned independently. The goal of the base-type is to separate different product variations, that exhibit significant technical distinctions, so that they are better planned independently.

This can be illustrated by the extremes: If a company opts to take a lot of features into the technical core of the base-type definition, the number of base-types would be very high. Now every base-type has to be planned and maintained independently and therefore the over-arching goal of sharing functionality between different product variances is not fulfilled, and extra management effort is introduced.

The other extreme would be to only have one base-type. This reduces the management effort in the first place, but now there are features that create a lot of technical complexity,




			TT 01 	TT 02 	TT 02B 
technical core Base Type Features	Wheelbase	251	S	-	-
		257	-	S	S
	Motor Type	TYPE 380	S	-	S
		TYPE 540	-	S	-
Customer Choice	Battery	NiMH 3000mAh	S	-	-
		LIPO 3300mAh	O	S	S
	Clearance	Standard	-	S	-
		High	S	-	S
	Spur width	Standard	S	O	S
		Wide	O	S	-
	Suspension Stiffness	Soft	O	S	S
		Hard	S	-	-

Figure 2.5: Exemplary description of a base-type as the core technical variations. *S* stands for *standard* and is the default selection while *O* are *optional* features that can be selected by the customer

and hence the documentation for this single base-type becomes super complex. For a truck, the wheelbase of the tires has many implications on the steering system, the braking system, the power transmission, etc. Because of this, it makes sense to take these feature groups into the base-type definition and plan the different variants independently.

To summarize, the goal should be that the technical core is as small as possible, but still includes features that have a huge influence on many core parts of the product.

2.2.6 Bill of Material (BoM)

To manufacture a product, a list of subparts need to be assembled into the full product. These subparts are documented in the so called BoM, which contains this list of parts. On technical drawings the BoM is normally located on the right lower side of the sheet and describes the parts and relevant information like the exact part number, mechanical characteristics, CAD File references and a whole range of descriptors [Tea02].

The BoM is a very central part of a product lifecycle through all stages of a product's existence. It connects and retrieves its data from several data sources across the company like the Product Lifecycle Management (PLM) system, Computer Aided Design (CAD) resources or different Enterprise Resource Planning (ERP) tools.

2.2.7 Assignment
















































Module	Structural Element	ID	Description	image	TT 01	TT 02	TT 02B
Wheel	Wheel Front	104.02.02/01	2.5" onroad profile wheel				
		104.02.02/02	2.5" onroad slick wheel				
		104.02.02/03	3" offroad profile wheel				
Battery	Battery	301.01.01/01	NiMH, 7.2V, 3000mAh				
		301.01.01/02	LIPO, 11.1V, 3300mAh				
Drivetrain	ESC	401.04.01/01	Electronic Speed Controller 40A				
	Motor	201.01.01/01	Motor Type 540				
		201.01.01/02	Motor Type 380				
	Pinion Gear	201.04.01/01	Pinion gear 17T				
		201.04.01/02	Pinion gear 19T				
		201.04.01/03	Pinion gear 22T				
[...]	[...]	[...]	[...]	[...]			

Figure 2.6: Exemplary description for the assignment of technical solutions on base-types

Now, we can take a look at what happens, when we extend the BoM with the concept of product variation. This is done by creating a superset of all the parts that are needed for the product. Hence this is called the SuperBoM (SuperBoM) or also *150%-BoM* in some contexts⁴. To filter out the parts of the SuperBoM, that are needed for a given variant, a mask must be applied. This mask is just a matrix that assigns subparts to products, which is illustrated in Figure 2.6.

This matrix based assignment can work pretty well for a small product platform, but for bigger platforms, whose amount of variations is diverging, a better solution must be found. This is where the usage condition comes into play. A usage condition is a formula based on predicate logic that describes when a certain part will be assigned to a configuration. The variables used are the features and the operators are And, Or and Not.

⁴The *150%-BoM* is in contrast to the *100%-BoM*, where the *150%-BoM* documents several variations and the *100%-BoM* only one

2. BACKGROUND & PRELIMINARIES
















































Module	Structural Element	ID	Description	Usage Condition	image	TT 01	TT 02	TT 02B
Wheel	Wheel Front	104.02.02/01	2.5" onroad profile wheel	TW_ON_PROFILE				
		104.02.02/02	2.5" onroad slick wheel	TW_ON_SLICK				
		104.02.02/03	3" offroad profile wheel	TW_OFF_SPIKE				
Battery	Battery	301.01.01/01	NiMH, 7.2V, 3000mAh	AK_LIPO_3300				
		301.01.01/02	LIPO, 11.1V, 3300mAh	AK_NIMH_3000				
Drivetrain	ESC	401.04.01/01	Electronic Speed Controller 40A	1				
	Motor	201.01.01/01	Motor Type 540	M_TYP_540				
		201.01.01/02	Motor Type 380	M_TYP_380				
	Pinion Gear	201.04.01/01	Pinion gear 17T	RATIO_1770				
		201.04.01/02	Pinion gear 19T	RATIO_1961				
		201.04.01/03	Pinion gear 22T	RATIO_2270				
[...]	[...]	[...]	[...]	[...]	[...]			

Figure 2.7: Excerpt of the SuperBoM and assignment of technical solutions based on usage conditions

Most commonly, + is used for And, / for Or and – for Not. Furthermore, parentheses can be used to group sub-statements together. As example take a look at the following statement $A + (B / C) + \neg D$ which represents $A \wedge (B \vee C) \wedge \neg D$. Section 4.5 about Pratt Parsing will give a more detailed look into the formulation of the usage conditions and their grammatical structure.

Figure 2.7 extends Figure 2.6 by adding the usage conditions. In this case, the conditions are rather simple. For example the part *104.02.02/01 (2.5" onroad profile wheel)* has the usage condition `TW_ON_PROFILE`, and therefore all configurations that include the `TW_ON_PROFILE` feature, will allocate this part. For larger-scale documentation systems, these condition become more complex and often include several parenthesis layers as well as features from many different feature groups.

In Figure 2.8 the assignment process is illustrated on a higher level. For large product platforms, a further distinction is made, as the Structural Element (SE) is introduced. A SE is an element that serves as a container for subparts out of which it is assembled, and it is itself structured as a BoM. The assignment using usage conditions is happening on the level of the SE. As illustrated in Figure 2.8, each base-type gets SE assigned through a usage condition and each SE has all needed parts assigned in a hierarchical structure. For the rc-car, one such SE could be a LED lamp. The lamp is assigned to the base-type with a usage condition and the parts are assigned through the SEs hierarchical structure.

Each SE is part of a Structural Family (SF), which is a group of different SEs that share the same function but are technical variations. For example, the SF Lamp contains different lamp types like the LED lamp, Halogen Lamp, or the Mini-LED-Lamp.

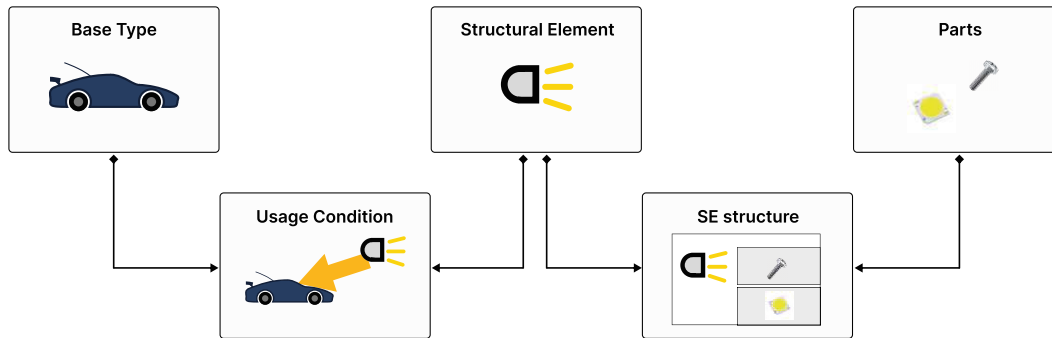


Figure 2.8: Drawing of how the SuperBoM assigns technical solutions and parts in a hierarchical order to possible configurations [WB23]

Making use of the usage conditions, we can now dynamically compute the BoM for different configurations by evaluating the usage conditions of all parts against the chosen codes in the configuration. We have achieved the goal of getting from a customer order (configuration) directly to the products (BoM) and hence the deliverable product. The challenge now is how to write a proper usage conditions. Some requirements that go into this are listed in Section Distinctive, Complete and Consistent, and also a process will be introduced in Section Module Interface Analysis that derives the feature groups, that each module depends on. But first, we need to take a look at one more concept, which is called restrictions.

2.2.8 Restrictions

Restrictions prohibit certain combinations of features. For example, an off-road wheel can only be ordered in combination with the off-road suspension. Since the off-road wheels are too big for the standard suspension, this restriction is caused by a geometrical constraint. Restrictions are expressed as implications, so for example we can say that `TW_OFF_SPIKE` implies `OFF`.

Restrictions can be introduced for several reasons:

- **Technical In Feasibility**

A certain combination is not technically possible because of some interfaces and/or rules, regulations, or norms.

- **Market Segmentation and Brand**

If our brand aims to target a specific market, we want to eliminate other options that do not fit the image of the brand.

- **Economical Viability**

This combination would see only a small amount of customer interest and is therefore not viable.

TT-Platform Restrictions	
Premise	Consequence
TT_01	PF_G1
TT_02/TT_02B	PF_G2
TT_01	ON+GC_STD+-M_TYP_380
TT_02/TT_02B	BODY_STD
TT_01	-AK_LIPO_3300
TT_01	RATIO_1961
TT_02	RATIO_2270
TT_02B	RATIO_1770
WB_266	-XW_WIDE
ON	TW_ON_PROFILE/TW_ON_SLICK
OFF	TW_OFF_SPIKE
ON	WB_257/WB_251
OFF	WB_266
ON	XD_AIR
OFF	XD_OIL
ON	XS_STD
TT_02	ON
TT_02B	OFF+GC_STD
TT_01	AI_MANUAL
TT_02B	AI_MANUAL/AI_BREAK_ASSIST
TT_01	-M_TYP_BRUSHLESS
AI_SELF_DRIVING	M_TYP_BRUSHLESS
AI_SELF_DRIVING	SV_6V_20K
AI_SELF_DRIVING	AK_LIPO_3300
AI_SELF_DRIVING	WB_251
W_VOLTAGE_MONITOR	AK_LIPO_3300

Table 2.3: Example list of base-type to feature restrictions

Types of Restrictions

Restrictions can be separated into two groups [Bru20]:

- **Feature to Feature**

Feature to Feature restrictions are within a feature group and define which features can be combined. For the context of this work we will consider this condition to be an OneHot constraint such that each feature group can have only one feature selected.

- **Base-Type to Feature**

Each base-type can define restrictions. These are written as implications where the premise is just one code and the consequence is an arbitrary code condition as listed in Table 2.3. With this pattern, it is intuitive to understand restrictions because they can be displayed as implication graphs and chains. Furthermore, for writing a restriction, a feature can be assumed, and then it can be argued what must be true for this feature to be buildable. For example, if we want to build a rc-car with WB_266, XW_WIDE cannot be selected.

2.2.9 Formal Definition

To summarize this introduction to FBD this section defines a formalism for a \mathcal{D} instance. The sequence of definitions follows the storyline of the previous chapter.

A \mathcal{D} instance is defined over a tuple of Features, Base-Types, Restrictions and Assignments.

$$\mathcal{D} = \langle F, \{B_0, B_1, \dots, B_n\}, \{R_0, R_1, \dots, R_n\}, \{A_0, A_1, \dots, A_n\} \rangle \quad (2.3)$$

Firstly we take a look at the first element of the tuple, the features F which are defined over a set of feature groups G_0, G_1, \dots, G_n .

$$F = \{G_0, G_1, \dots, G_n\} \quad (2.4)$$

Each feature group G_i represents a dimension of variation and encodes the possible values in this dimension as the codes $c_{i,0}, c_{i,1}, \dots, c_{i,n}$. Each of these codes is unique over the whole instance and occurs in only one feature group.

$$G_i = \{c_{i,0}, c_{i,1}, \dots, c_{i,n}\} \quad (2.5)$$

The base-type B is defined over a identifier b and a subset of codes c of the Feature definition. If c is *Standard* or *Optional* is for this consideration not relevant, the set defines that these choice are possible.

$$B = \{\langle b, c \rangle \mid c \subset \bigcup_i G_i\} \quad (2.6)$$

Restrictions are defined per base-type and code. The base-type and the code can only be configured, if the formula f is satisfied.

$$R = \langle b, c, f(c_0, \dots c_n) \rangle \quad (2.7)$$

The given tuple can be interpreted by using the implication.

$$\phi(R) = (c \wedge b) \Rightarrow f(c_0, \dots c_n) \quad (2.8)$$

Each SE (part) defines a base-type b and a usage condition. If the base-type is selected and the usage-condition is satisfied, then it is allocated, else it is not needed for this configuration.

$$A = \langle b, f(c_0, \dots c_n) \rangle \quad (2.9)$$

The interpretation is that a part will be used, if the base-type b is selected and the condition f is satisfied.

$$\phi(A) = b \wedge f(c_0, \dots c_n) \quad (2.10)$$

To encode the feature groups, we use the OneHot encoding, which is a combination of the At-Most-One and the At-Least-One constraint. It defines that out of the Feature-group G_i always exactly one element must be chosen. The following formulation is a naive implementation. The At-Least-One constraint is trivially implemented using a Or-clause and the At-Most-One is implemented using a pairwise encoding by iterating over all possible combinations of cardinality 2 and defining that they cannot be true together.

$$\Theta(G) = \bigvee_{c_i \in G} c_i \wedge \bigwedge_{c_i \in G} \bigwedge_{c_k \in G, k < i} \neg(c_i \wedge c_k) \quad (2.11)$$

This naive At-Most-One constraint method can be improved by using methods like the *Commander Encoding*, the *Product Encoding*, etc. as outlined and compared in the paper *Empirical Study on SAT-Encodings of the At-Most-One Constraint* [NNKB21].

The interpretation of this instance as a SAT-Problem is given by the following definition:

$$\phi(\mathcal{D}) = \bigwedge_{G_i \in F} \Theta(G_i) \wedge \quad (2.12)$$

$$\bigvee_{b \in B} b \wedge \quad (2.13)$$

$$\bigwedge_{G_i \in F} \left[\bigwedge_{c \in G_i} (c \Rightarrow \bigvee_{\langle b, C \rangle \in B \mid c \in C} b) \right] \wedge \quad (2.14)$$

$$\bigwedge_R \phi(R) \quad (2.15)$$

The first line of the formula combines the OneHot encoding of the all feature groups. The second line is the selection of the base-type. The solver must select at least one base-type. Thirdly, we have the base-type definition which defines that if a code is selected, it must be part of the selected base-type. Therefore, the code implies that at least one base-type must be selected, of which it is a part. Lastly, we have the Restrictions that are all added to the formula given their interpretation (Definition 2.8).

Now the FBD-Instance is created. Depending on the use-case, we can also encode the assignments A into the instance using the interpretation above.

CHAPTER 3

State of the Art & Contribution to the Domain

The author has written a bachelor thesis with the title *Application of SAT-Solvers in Feature-based Documentation Systems* [Bru20] which explored how SAT-Solvers can be used for formal verification and analysis tasks within FBD. It already hinted MaxSAT-Solvers as having possible applications but did not yet elaborate these ideas.

The following section will give a short explanation about core concepts in the bachelor thesis around the SAT-Solver. It will also point out several refinements that have been added to the method in the context of this thesis and the application in practice.

The author's bachelor thesis defined a way to do reductions from code conditions to the satisfiability problem. This reduction will be extended by adding several fundamental improvements that will be described in Chapter 4. These improvements considerably boost the performance since the problem exposed to the SAT-Solvers is reduced in size. Also, for the parsing, the author proposes a method, based on the Pratt Parsing technique, that yields better performance, understandability and addresses imperfections with the original top-down parser.

3.1 Distinctive, Complete and Consistent

The assignment of SEs must adhere to several requirements that make sure that the product generated by the SuperBoM is always buildable and deliverable. All of this can be verified using a SAT-Solver [Bru20]. In the bachelor thesis, the author declared the following definitions, which have been updated to the new formalism.

Consistent

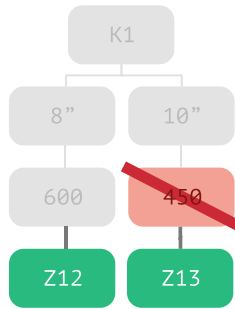


Figure 3.1: Illustration of Consistency

The usage conditions of SEs must be consistent with the restrictions. For example a certain part in the suspension has the condition $WB_266 + XW_WIDE$. This condition can never be true because WB_266 cannot be combined with XW_WIDE . Therefore, the part will never be used and can consequently be eliminated from the product.

Definition 3.1.1. Given an assignment A , it is consistent if $\phi(\mathcal{D}) \wedge \phi(A)$ is satisfiable.

Nevertheless, it has to be considered that documentation systems are dynamic and subject to change. It may be that a certain restriction is only true for a certain time, due to a temporary regulation. Because of that, a computation of all inconsistent SEs might be helpful, but one has to take several factors into account before removing a SE.

Complete

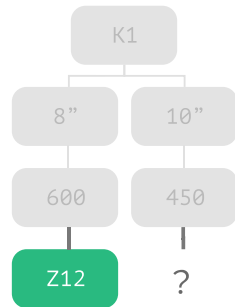


Figure 3.2: Illustration of Completeness

Every possible configuration needs to allocate all the SEs that are needed to build it and may not miss any essential SE. For example, there should not be a configuration without a motor for the rc-car, because the motor is an integral part for the rc-car to work. In other words, the disjunct usage conditions of all SEs in the motor SF must cover all possible configurations – or at least for all configurations where the module is necessary.

Definition 3.1.2. The set of assignments $\{A_0, A_1, \dots, A_n\}$ is complete, if $\phi(\mathcal{D}) \wedge \neg(A_1 \vee A_2 \vee \dots \vee A_n)$ is not satisfiable.

Distinctive



Figure 3.3: Illustration of Distinctiveness

The following definition defines that there is at least one element of each essential SE, and there also must not be more than one. The rc-car only needs one motor to work and not more than one.

Definition 3.1.3. The set of assignments $\{A_0, A_1, \dots, A_n\}$ is distinct if $\bigwedge_{G_i \in F} \Theta(G_i) \wedge A_i \wedge A_j$ is not satisfiable for all pairwise combinations (A_i, A_j) .

3.2 Exclusivity

One essential part of the bachelor thesis is the concept of exclusivity and satisfiability of SEs. The bachelor thesis the author has defined how to compute the exclusivity of a given condition against a premise and evaluated that using a SAT-Solver. Since the bachelor thesis, the concept has been used in many scenarios and has been extended.

One shortcoming of the bachelor thesis was that exclusivity was only considered from a given premise to a condition without taking into account the whole context of feature groups, base-types and restrictions. In practice, it has been shown that computing exclusivity by taking the whole \mathcal{D} instance into account is way more effective and yields up to 10 times the amount of exclusivities.

Given that we look at exclusivity now from more general perspective, the most intuitive way to explain it, is by using set theory. In Figure 3.4 this concept is explained by illustrating the set of all possible configurations.

First, let's take a look at the set of all possible configurations (SE allocation). Each SE is allocated by its usage condition and the base-type. In the set of all possible configurations, these define a subset that is schematically drawn in Figure 3.4.

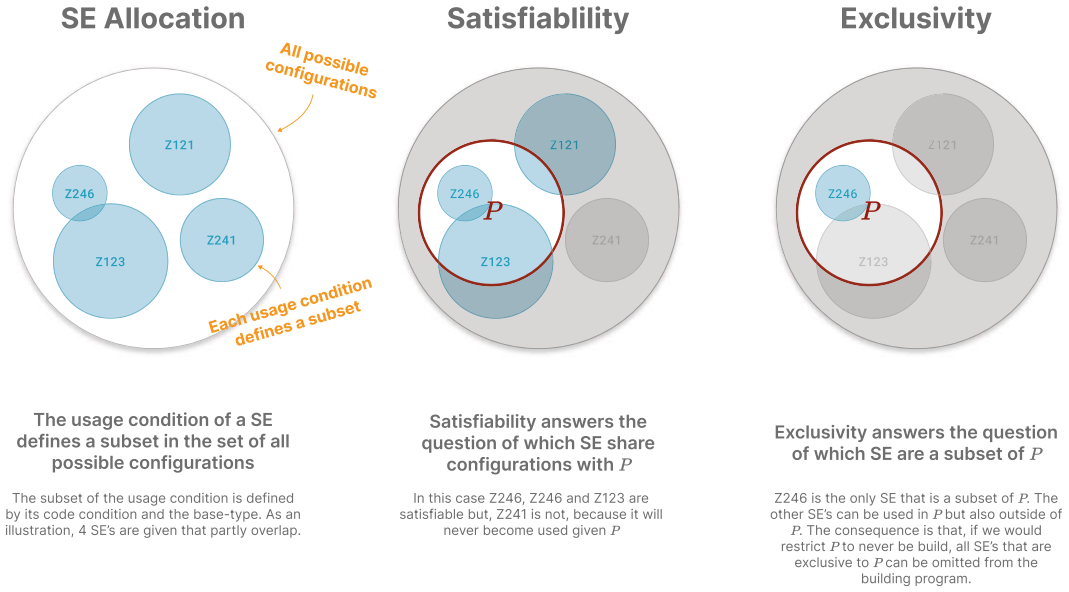


Figure 3.4: Explanation of satisfiable and exclusive SEs based on set-theory

Now we can define a further subset P , which also represents usage condition defining a base-type and code string. We use this as the premise for the following steps. Now, two important questions can be asked given P .

Firstly, which SEs can be allocated to configurations that are also allocated in P ? Speaking in set theory: $A \cap P \neq \emptyset$. There are many real use cases where exactly this question must be asked.

For example, a company decides to build a new plant that only produce rc-cars with the NiMH battery, since handling of the alternative LIPO batteries has many complex requirements for storage and handling. With this method, we can compute all the still allocatable SEs and only do homologation and logistics in that plant for those SEs.

Secondly, which SEs are only (exclusively) allocated given P . In set notation that would be $A \subset P$. This can be used to optimize product platforms by eliminating SEs. For example, if a certain battery and motor combination is not sold very often, then we can compute what SEs are only allocated to this combination, these SEs could consequently be removed from the product portfolio, saving resources in construction, homologation and logistics.

Describing these concepts with Set-theory makes it easier to understand and explain the concept. But there is also a derived formulation in propositional logic.

Definition 3.2.1. We consider A satisfiable, if $\phi(\mathcal{D}) \wedge \phi(P) \wedge \phi(A)$ is satisfiable.

Definition 3.2.2. We consider A exclusive, if $\phi(\mathcal{D}) \wedge \phi(P) \wedge \neg\phi(A)$ is satisfiable.

These theorems can be computed using a SAT-Solver by utilizing the formalism in Section 2.2.9 and applying the reduction pipeline in Chapter 4.

3.3 Module Interface Analysis

This part explains the *Module Interface Analysis* which is a process proposal by the author. It aims at creating the allocations for parts in a FBD starting with the modules and ending with a *consistent*, *complete* and *distinctive* FBD instance. The process follows 6 steps, including 1 optional. The whole process is drafted as a linear process, but in practice, the intention is that there are several refinement steps in which certain findings in a later stage trigger a rework in an earlier stage and trigger a redoing of the consequent steps. The goal is to create a proper decomposition of the product, by utilizing interfaces, such that each dimension variation (feature group) is contained in a minimal amount of modules.

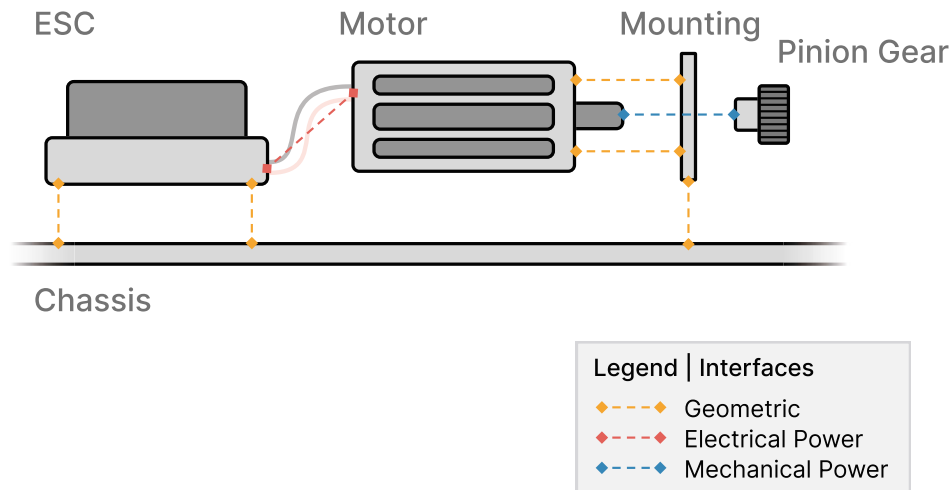


Figure 3.5: Illustration of the interfaces for the example Interface Graph

In Figure 3.5, a schematic view is given over the ESC, the motor, motor mounting and the pinion gear, and the interfaces with which they are assembled. In this example, we differentiate between 3 different interface types:

- *Geometric* interfaces, for parts that need to fit together geometrically.
- *Electrical Power* interfaces where electrical power is conducted.
- *Mechanical Power* interfaces for the transmission of mechanical power.

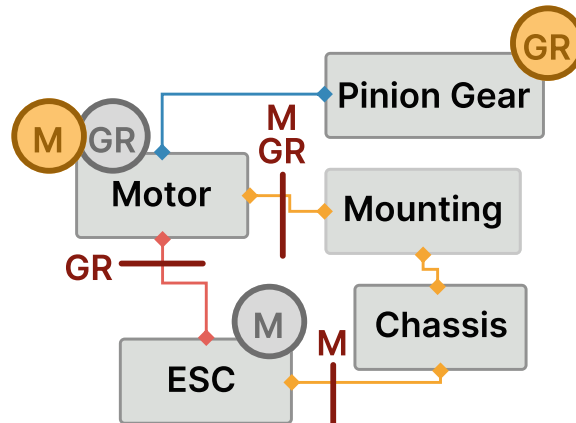


Figure 3.6: Exemplary Module Interface Boundary Graph of Figure 3.5

In Figure 3.6 the schematic view in Figure 3.5 is represented as a Module Interface Boundaries (MIB) graph. In this graph the modules are nodes and each interface is represented as an edge between two nodes. Furthermore, there are annotations for the feature groups that influence each module (M , GR) as well as the influence boundaries (dark red lines), which will be explained in the outline of the process.

The aim of the process is to create a connection between the MFD method and FBD. The starting point for this process is the Module Interface Graph which are an output of the step number four in MFD (Chapter 1.1.2). For the complete car a full *module interface graph* is given in the appendix in Figure A.1.

3.3.1 Outline of the Process

In Figure 3.7 the 6 steps of the process are illustrated and are explained in the following enumeration.

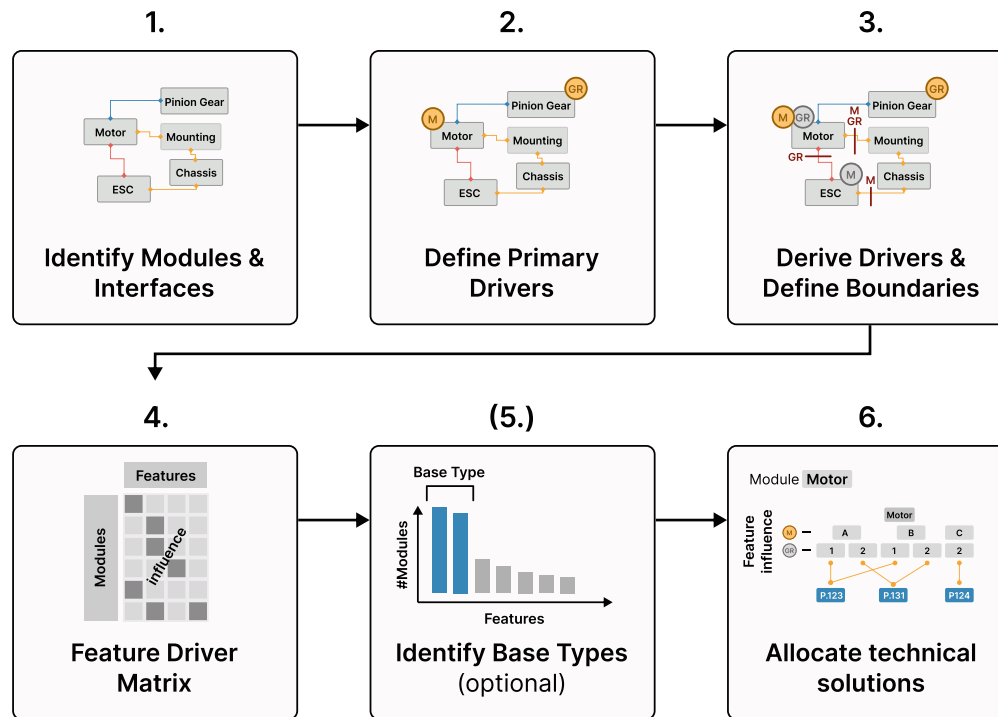
1. Identify Modules & Interfaces

In the first step, the module interface graph is created (or taken from the preceding MFD) by defining all the modules of the product and creating all the interfaces between them.

Depending on the maturity of the project, this graph can be drawn in an abstract high-level manner and or also on a very granular level, if the product is mature enough. Throughout the development process, this graph can be refined and extended whenever needed.

2. Define Primary Drivers

In this step we need to define which module is dependent on which feature group.

Figure 3.7: The 6 steps of the *Module Interface Analysis*

We can define *primary drivers* which represent a feature group that have a direct influence on a module. For example, the motor type will be a primary driver for the motor module and the gear ratio will be a primary driver for the pinion gear. The primary drivers are annotated on the graph using the orange badge.

3. Derive Drivers & Define Boundaries

In the next step we will analyze how the primary drivers will affect the surrounding modules. The core idea is that each driver will propagate along each interface, except, if the interface serves as a boundary for that influence. The result of this step is a MIB graph (Example 3.5).

For example, the *motor* module has a mechanical interface to the pinion gear and a geometrical interface to the motor mounting. Because of that we need to analyze, if the different motor variations will require different motor mountings and pinion gears or if all motor types can be served with a single variant.

If the latter is the case, we have achieved to create an influence boundary, since the motor type will no longer be a cause of variation for the motor holder. This boundary will be annotated using the dark red lines on the interface edges.

Otherwise, if we cannot create one single motor holder through a standardized interface, we need to propagate the influence of the motor type to the motor holder module by creating a derived driver (the gray badges). Now we have to repeat the process and look at all the surrounding modules of the motor holder and discuss the influence along all the interfaces.

4. Feature Driver Matrix

We can now leave the graph representation and compose the Feature Driver Matrix (FDM) directly out of the annotated MIB graph from the step before. The FDM contains all modules in the rows and all the feature groups in the columns. Whenever a feature has a primary or a derived influence on a module, the corresponding cell will contain a 1. The created matrix gives an overview over the influences and is used to check if the created drives are sensible.

5. Identify Base-Types (optional)

As an optional step, the FDM view can be used to analyze the *technical core* of the product by analyzing which feature group has the highest influence on the product. Ordering the feature groups descending by the amount of modules that they drive, the most complex feature groups can be identified. Out of experience, there are mostly a few groups that are standing out compared to the rest. These are candidates for being part of the base-type definition since they have a strong technical influence on the product.

6. Allocate Technical Solutions

As a last step, we need to allocate the corresponding technical solutions to the modules according to the module drivers given the combination of all the features that have influence on the module. This is done by writing the usage conditions for the parts and making sure that the distinctiveness (Definition 3.1.3), completeness (Definition 3.1.2) and consistency (Definition 3.1.1) are taken into account. When engineering new parts for the product, the feature drivers and boundaries need to be taken into account such that the interfaces implement the defined influence boundary. Inconsistencies or issues with implementing a boundary can trigger a modification of the boundary and a refinement of the prior steps.

3.4 Involvement of the Author

For several years the author has worked in a company called *EFS Consulting* which focuses on consulting companies in the automotive industry and has its roots in managing modular platforms for large OEMs all over the world. Over the years, a lot of experience has been gained in many aspects of this concept and the author has successfully introduced several concepts around the SAT-Solver that help in the area of analytics and verification as well as desktop tools that aim at improving visualization and analysis.

3.5 EFS Modularity Suite

The *EFS Modularity Suite*, developed by the author together with a development team at *EFS Consulting*, is a web-based multi-user tool, that is the perfect companion to plan product variations and for creating a FBD instance. It includes automated verification tests to guarantee the consistency, distinctiveness and uniqueness of the created usage conditions and restrictions. Using SAT-Solvers, it can compute Variant Trees to visually explore a certain configuration subspace and reason about the restrictions as well as allocations.

The EFS Modularity Suite is fully cloud hosted and supports multi-user collaboration. In Figure 3.8, an overview over the UI is given. In the main view, the *Planner*-view, we can visualize variant trees for feature groups that can be selected in the panel on the left side. On the right side, the module tree is shown with the module hierarchy and the variants of each module. The activated variants (parts) are added as leaves to the tree based on their usage-conditions. This view allows to simulate the allocation of parts and verify the correctness. Furthermore, we can change the view to show only restricted configurations, or configurations where no variants are allocated.

Besides the *Planner*-view there is also a *Configurator*-view with which a customer-facing configurator can be simulated. Even productive customer facing configurators can be created, which rely on the EFS Modularity Suite to resolve constraints and verify the correctness of configurations.

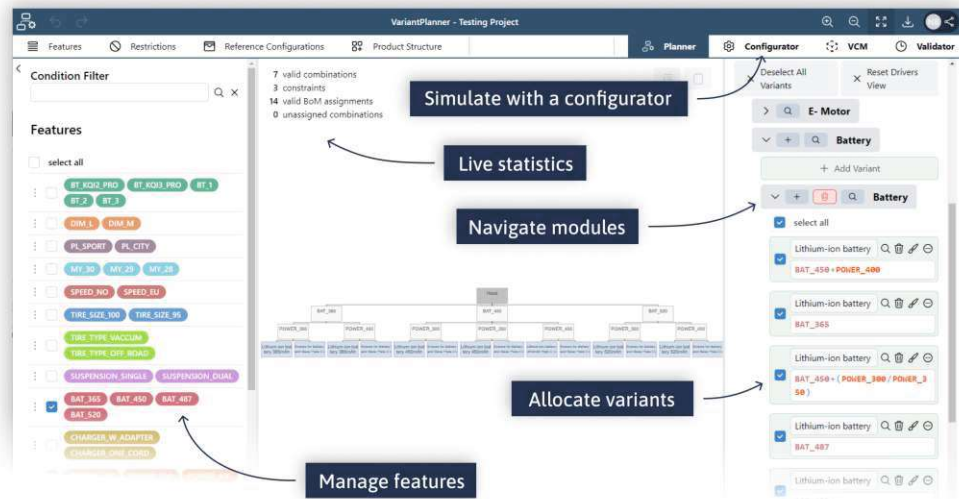


Figure 3.8: Overview over the EFS Modularity Suite

Reducing to the Satisfiability Problem

In Section 2.2, the concept of feature-based documentation and the crucial role that usage conditions and restrictions play in it are explained. These conditions are written in propositional logic with 6 different operators and possibly many layers of parenthesis. To use these formulations in a SAT-Solver, we need a way to transform them into a set of clauses (CNF), because this is the form that SAT-Solvers require.

There are several different methods to approach this and they mostly consist of three main steps: Parsing, Transformation, Solving (see Figure 4.1). The following sections will guide through all the steps and explain different options along the way.

Alongside this thesis, the author has implemented this pipeline in the Rust programming language by creating a custom implementation for the Pratt-Parser and the Tseitin Transformation. The SAT-Solver interface is created using the *rustsat*¹ library. This library provides interfaces to several different SAT-Solvers, including MiniSat, Glucose and CaDiCaL. The subsequent algorithms rely heavily on this transformation pipeline.

A lot of programming languages offer parsing tools that can parse conditions based on abstractly defined grammars. The author has analyzed several of these tools for use in this context, but the effectiveness and simplicity of Pratt Parsers in combination with the authors' interest in how parsers work under-the-hood, has lead to the decision to implement a custom parsing pipeline from the ground up and document the learnings in this thesis. Furthermore, this approach offers performance benefits compared to larger and more complex libraries that are not fine-tuned for the specific use-case.

¹<https://crates.io/crates/rustsat> accessed on 15.10.2024 22:29

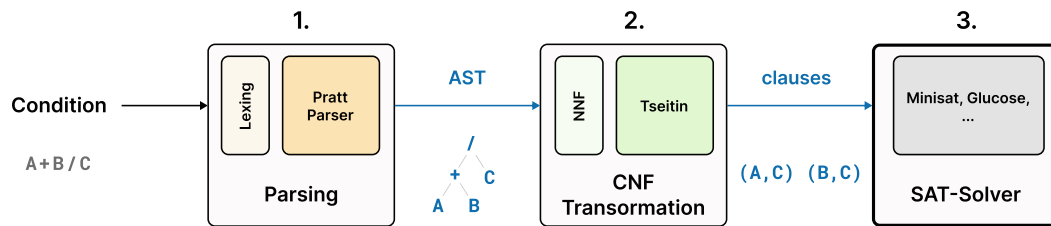


Figure 4.1: Pipeline for reducing code conditions to the SAT-Solver

4.1 Step 1: Parsing Code Conditions

In the first step, the goal is to create a form of representation of the code condition that can be used for further processing. A structure that does exactly that is the Abstract Syntax Tree (AST). An AST is a tree representation where operators are tree nodes and variables or constants are the leaf nodes. Each operator node has a number of children on which its operator is applied on.

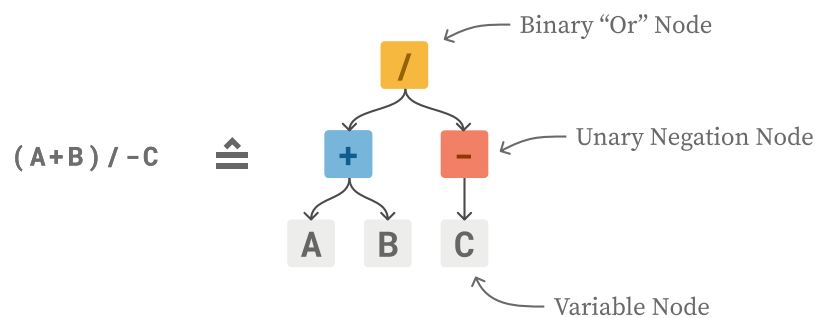


Figure 4.2: Illustrative example of an AST

This concept of ASTs is used widely in many areas of computer science. For example, for creating compilers that parse code into an AST and that start the conversion to machine code from there. The AST representation has the advantage that all the brackets and operator hierarchies, and specialties of the syntax are taken into account and therefore yield an unambiguous way of representing the underlying logic. And because of the recursive nature of the tree structure, it is ideal to implement recursive algorithms that traverse the tree and execute operations on each node. In future steps, we will take advantage of precisely that.

The process of converting a code condition from a string into an AST is called *parsing*, and there are several approaches to tackle this task.

One such technique is Pratt Parsing, developed in 1973 by Vaughan R. Pratt [Pra73].

He described and argued the correctness of his approach in the paper. Several articles have since tried to summarize and explain what Pratt's original ideas were.

One such post is written by Bob Nystrom [Nys11], titled *Pratt Parsers: Expression Parsing Made Easy*, which introduces the concept of the algorithms and compares it to other parsing techniques. Furthermore, Martin Janiczek has written a blog post titled *Demystifying Pratt Parsers* [Jan23] in which he tries to give a improved version of Bob Nystrom's article. He utilizes graphics and examples to aid better understanding of the algorithm.

In the following, the author gives an overview of the Pratt Parser approach by starting with the first step, which is called lexing.

4.1.1 Lexing

In the lexing step, the input string is converted into a stream of tokens. A token is an atom (the smallest possible chunk) of the condition with a defined meaning [Nys11]. In Figure 4.3 this is illustrated. Lexing typically differentiates between several types of tokens. For code conditions, the author has identified 3 main types of tokens: The *Variable Token*, the *Operator Token* and the *Bracket Token*.

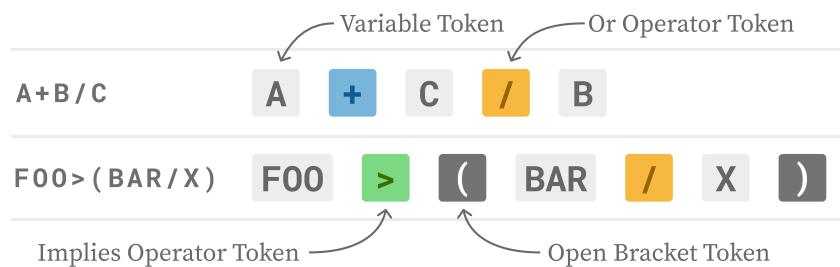


Figure 4.3: Lexing converts a given input string into its atoms

Firstly, the *Operator Token* differentiates between all possible operators. In code conditions we have 6 different operators, as listed in Table 4.1. Operators are always one character long, which makes the implementation of this lexer trivial because no lookahead is needed.

The operators are differentiated between unary and binary. Unary operators bind only to one side while binary operators bind to two sides. For example, in $A+B$: The $+$ is binary, because it binds the A on the left and the B on the right together. The negation in $A+-B$ on the other hand is unary because it binds to B , it expresses: "not B ".

Secondly, a *Bracket Token* represents either an opening or a closing parenthesis. In conditions, we only utilize parentheses, other types of brackets are ignored (square brackets, curly brackets, etc.).

Character	Meaning	Type	Precedence
-	Negation	Unary	6
+	And	Binary	5
^	XOr	Binary	4
/	Or	Binary	3
>	Implies	Binary	2
=	Equals	Binary	1

Table 4.1: Operator table

Lastly, everything that is not any of the above is considered a *Variable Token*. Variables are strings that can take any length.

Separating the lexing step from the parsing step, allows the parser to focus on the structure of the language, abstracting away the specific of the syntax. If another definition of operators would be defined, where "And" is not represented by "+" but rather by "&", we can still utilize the same parser and just need to swap the lexer.

4.1.2 Parsing using the Pratt Parser

After the step of lexing, we are ready to convert the stream of tokens into an AST. To start this process, let's take a look at one of the challenges of generating ASTs. Operators have different precedence as listed in Table 4.1. The precedence defines which operator is binding *stronger*, the higher the number the higher the "binding-force". [Jan23] For example, let's take a look at Figure 4.4. If we do not take precedence into account, option 1 and option 2 would both be valid interpretations, even if they are not logically equivalent.

As seen in the operator Table 4.1, the + (*And*) operator has a higher precedence number and therefore it binds stronger than the / (*Or*) operator. Hence, Option 2 is correct.

The Parser must be able to take the precedence table into account when parsing to represent the right execution hierarchy in the ASTs.

The Pratt Parser approaches this the following way:

It defines a function `pratt(limit, tokens)` [Jan23] that takes the tokens as well as a limit as input. It starts iterating the tokens given the precedence limit. When it reaches an operator that exceeds the maximum precedence, it returns and will stop parsing.

The precedence of the top level parsing is the maximum (0). Because of this we can call the parser routine with `pratt(0, tokens)` [Jan23].

In Figure 4.5 the internal program flow of the algorithm is illustrated. The parsing function always starts with parsing the prefix/left sided operator, since the start will always be a variable, either after a bracket or at the beginning of the input. This step is *Parse a prefix expr into left* in Figure 4.5.

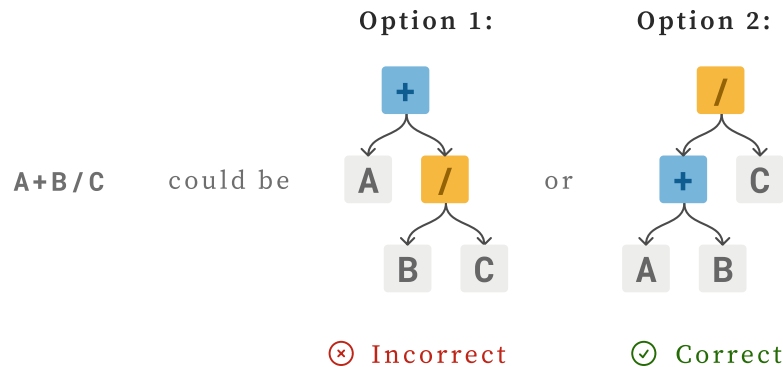


Figure 4.4: The two possibilities of interpreting a condition are used to show why precedence is needed

From there on we can iterate the stream token by token in a loop. The first thing that happens in the loop is a check if the next token is an operator or any other symbol. If it is not an operator, it will return the current left value. Else it reads out the precedence of the operator and continues, if the precedence is higher than the limit, or exits returning the current left value. If it continues, it parses the next operator by calling itself recursively with the precedence of the current operator as limit. If it returns, it creates a new binary operator with the current left and the result of the recursion as right. This is then the new left, and the loop starts again.

After the top level function call returns, we need to check that there is nothing left in the token stream. Otherwise, this would mean that there were too many closing brackets. Conversely, if a subroutine reaches the end of tokens, there are too many opening brackets.

4.1.3 Implementation by the Author

The author opted to implement the Pratt Parser in Rust based on the article by Alex Kladov titled *Simple but Powerful Pratt Parsing* [Kla20]. Kladov describes a very effective way to implement a Pratt Parser in Rust and utilize Rust's powerful type system to keep the code very clean, short and understandable.

The ratio for choosing Rust in the first place is the availability of open source multi-objective MaxSAT-Solver as well as abstract bindings for SAT-Solvers. These well-designed interfaces offer interoperability and are perfectly compatible with the created parser.

The core function of the author's implementation works with only 39 lines of code. Older implementations by the author, which utilized top-down parsing approaches needed several hundred lines of code. Therefore, the Pratt Parser is a huge simplification. It is also able to generate helpful parsing error messages for different kinds of syntax errors.

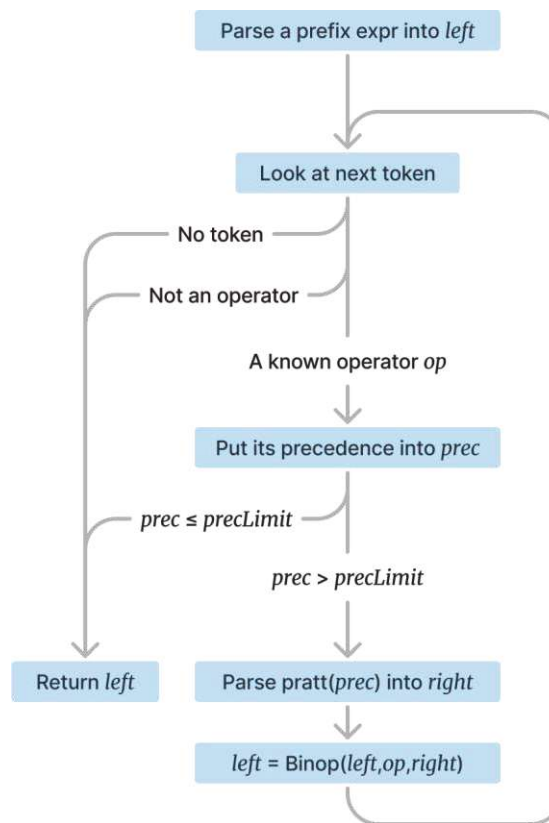


Figure 4.5: Pratt Parsing algorithm overview. From <https://martin.janiczek.cz/2023/07/03/demystifying-pratt-parsers.html> accessed on 13.3.2024 at 16:49

4.2 Step 2: Transformation to CNF

There are several ways of converting arbitrary logic into conjunctive normal form that have been developed and improved over the years. The following sections will explain several of these methods.

4.2.1 Distributive Transformation

The Distributive Transformation applies the law of distributivity until the formula is in the correct form [KKS⁺23]. This approach can trivially be proven correct, because it only applies algebraic reformulations. However, it suffers one problem, which is the exponential growth of formula size. There are several ways to deal with this, like the idempotence, the absorption, or the Quine-McCluskey algorithm, but these approaches are still limited and have exponential time complexity.

4.2.2 Tseitin Transformation

Another approach is the Tseitin Transformation which was proposed by Tseitin in 1983 [Tse83] as a new approach for CNF conversion. It replaces each subformulation ϕ with a boolean variable x_ϕ , which can be used as a shortcut to refer to ϕ . The new variable x_ϕ is tied to the original subformulation using the equivalence operator $x_\phi \equiv \phi$ [KKS⁺23]. With this approach, we do not suffer the issue of an exponential explosion of formulas.

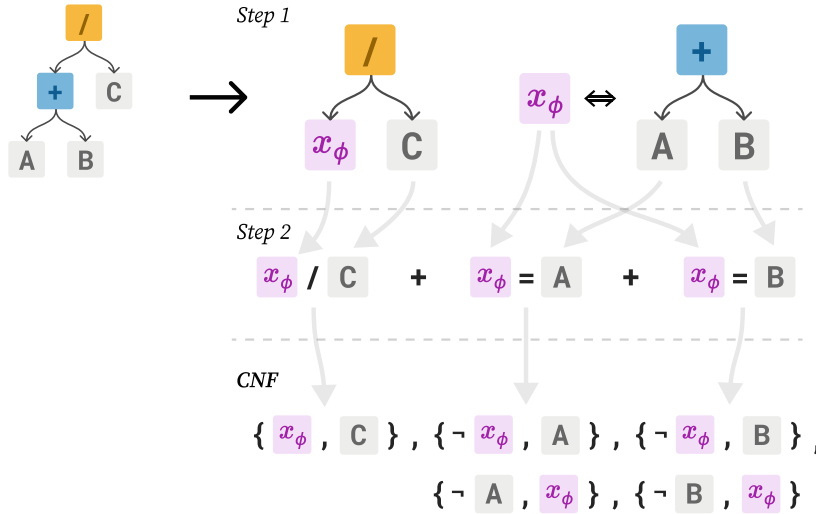


Figure 4.6: Tseitin concept explanation

In Figure 4.6 the concept is illustrated: In step one the variable x_ϕ is introduced and now replaces the $A+B$ subtree of the top *Or* node since x_ϕ is equal to $A+B$. In step two we can utilize distributivity and lift the verb $|+$ operator. This leads to x_ϕ being equal to A and equal to B . In the last step we convert each of the equivalences into two clauses which represent two implications: x_ϕ implies A and A implies x_ϕ . Using these steps, we have now arrived at a CNF representation.

The Tseitin Transformation is not *equivalent*, but *equi-satisfiable*. Given ϕ , *equi-satisfiable* defines that either ϕ and the transformed ϕ' are satisfiable or none of them is [KKS⁺23]. The transformation is not strictly *equivalent* since it introduces new variables and therefore deviates in terms of assignment.

4.2.3 Plaisted-Greenbaum Transformation

A few years later, Plaisted-Greenbaum [PG86] further improved the Tseitin Transformation and found out that a lot of clauses could be dropped by taking the polarity of subformulations into account [KKS⁺23]. Because of that, we can encode the ties between the x_ϕ variables as implication $x_\phi \implies \phi$ and drop the equivalence $x_\phi \equiv \phi$ constraint.

The Formulation is first converted into Negative Normal Form (NNF). The NNF can be reached by pushing all negations downwards to the variables by recursively applying *De Morgan's Theorem*. For example, $\neg(A \wedge B)$ can be converted into the equivalent formulation $(\neg A \vee \neg B)$. Applying this reformulation on all negations recursively yields a formulation, where negations are only applied on variables. Hence, the polarity of the subformulations is always positive, and the implication $(x_\phi \implies \phi)$ is sufficient.

4.2.4 Subformulaion-tracking

One important aspect of the Tseitin Transformation is the ability to reuse subformulations. If a subformulation occurs several times, we can reuse the x_ϕ , which can reduce the number of clauses in many applications drastically.

In Rust the Subformulation-Tracking is implemented by deriving the *Hash* trait on the enum that represents a node in the AST. This enables us to keep track of nodes using the *HashMap* provided by the standard library. Whenever the recursive Tseitin implementation considers a new node, it will first query the *HashMap*. If successful, the recursion is stopped and the found literal is returned, otherwise, the standard Tseitin procedure is continued.

One limitation of this method is, that the ordering of the children in a binary operator will influence the hash value of the node. Because of this two semantically equivalent nodes will not be considered equal by the *HashMap*. This can be mitigated by sorting the children of a given node.

4.3 Step 3: The SAT-Solver

The final step of the pipeline is to add the clauses created by the previous step to the solver. Solvers have a function to add clauses to the instance. During the transformation step, whenever the transformation yields a new clause, this function is utilized to add all clauses to the instance.

After the instance is created, we can use the *solve* functionality of the solver to check whether it is *SAT* or *UNSAT*. Furthermore we can take advantage of the incremental solver interface of some solvers to apply assumptions and retrieving UNSAT-Cores as explained in Section 2.1.4.

4.4 Performance Evaluation

Given the explained transformation methods, we will now compare their performance to one another. First, as a baseline the classical Tseitin Transformation. Secondly, Plaisted-Greenbaum's improvements. And lastly adding the Subformulation-Tracking capability to the classical Tseitin and the Plaisted-Greenbaum's version.

The dataset is a closed source and originates from the automotive industry. For this dataset, a Distributive Transformation would be infeasible.

In Figure 4.7, the reduction of clauses and variables is listed given the different techniques. Comparing the baseline approach (classical Tseitin) to Plaisted-Greenbaum, the amount of variables is identical, but the amount of clauses is reduced to around 60%. This is reasonable, since Plaisted-Greenbaum does use the same variables to reference subformulations, but adds fewer clauses because it only adds one implication clause per subformulation. The subformulation tracking further reduces the amount of clauses and variables by another 50%.

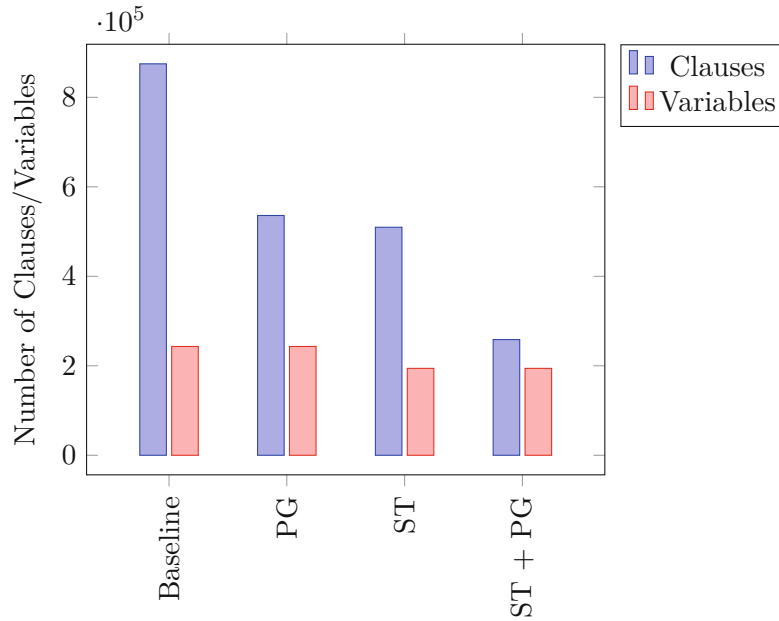


Figure 4.7: Comparing the effects of the transformation optimizations in terms of Execution Duration. *PG* stands for Plaisted-Greenbaum optimization and *ST* stands for Subformulation-Tracking

The reductions lead to reduced execution time of the SAT-Solver seen in Figure 4.8. The Plaisted-Greenbaum optimization reduces the wall-clock time by around 30 seconds (11%). Subformulation tracking reduces it by 122 seconds, which is 45% of the baseline time. Therefore, subformulation tracking has a way bigger and very significant impact on execution time. With both optimizations in place, the execution time compared to the baseline is reduced to the half.

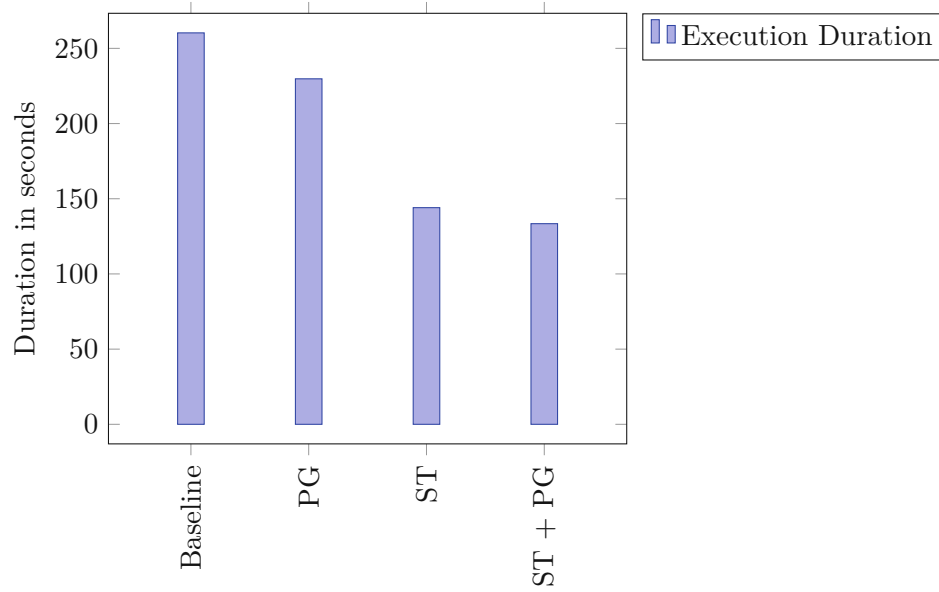


Figure 4.8: Comparing the effects of the transformation optimizations in terms of clause and variable count. *PG* stands for Plaisted-Greenbaum optimization and *ST* stands for Subformulation-Tracking

Introducing MaxSAT-Solvers

Maximum Satisfiability (MaxSAT) is a fundamental optimization problem that extends the classical SAT-Problem. Compared to SAT, MaxSAT tries to find an assignment of variables that satisfies the maximum number of clauses, even if satisfying all clauses is impossible.

A lot of theoretical work has been put forward to analyze the complexity and approxability of MaxSAT [IMMS19]. MaxSAT is known to be computationally expensive, with wall-time durations that often exceed several hours. Therefore, many approximation algorithms have been developed that compute solutions in reasonable time with bounds on the quality of the solution. In terms of computational complexity MaxSAT is *NP-hard*, since the SAT-Problem can be reduced to MaxSAT.

5.1 Problem Definition

Similar to the SAT-Problem, MaxSAT is defined over a set of clauses, that are a disjunction of literals, but in contrast to the SAT-Problem, not all clauses have to be satisfied all the time. The following section will explain the different categories and give examples for each of them.

5.1.1 Weighted MaxSAT

The weighted MaxSAT-Problem adds a non-negative weight to each clause [Stü18]. The goal is to maximize the sum of the weights of all the clauses that are satisfied by the found assignment.

Example 2 A simple exemplary weighted MaxSAT-Problem

$$\phi = \{(\neg x_1 \vee \neg x_2, w_1), (\neg x_2 \vee \neg x_3, w_2), (\neg x_1 \vee \neg x_3, w_3), (x_1 \vee x_2 \vee x_3, w_4)\}$$

Definition 5.1.1. The objective is to maximize over the assignment x

$$\text{maximize}_x \quad \sum_C \begin{cases} w_i & C \text{ is satisfied by } x \\ 0 & \text{else} \end{cases}$$

5.1.2 Partial MaxSAT

A partial MaxSAT-Problem differentiates between two sets of clauses [Stü18]. One set is called *hard clauses* ϕ_{HARD} , these clauses have to be satisfied, and the other is called *soft clauses* ϕ_{SOFT} . As many as possible of the latter should be satisfied.

Example 3 A simple exemplary partial MaxSAT-Problem

$$\begin{aligned} \phi_{HARD} &= \{(x_1 \vee x_2 \vee x_3)\} \\ \phi_{SOFT} &= \{(\neg x_1 \vee \neg x_2), (\neg x_2 \vee \neg x_3), (\neg x_1 \vee \neg x_3)\} \end{aligned}$$

Definition 5.1.2. The objective is to maximize over the assignment x

$$\begin{aligned} \text{maximize}_x \quad & \sum_{C \in \phi_{SOFT}} \begin{cases} 1 & C \text{ is satisfied by } x \\ 0 & \text{else} \end{cases} \\ \text{subject to} \quad & x \text{ satisfies } \phi_{HARD} \end{aligned}$$

5.1.3 Weighted Partial MaxSAT

Weighted Partial MaxSAT is the combination of the ones before. It defines hard clauses and soft weighted clauses. The goal is to maximize the sum of the satisfied soft clauses while satisfying all hard clauses. Typically, when speaking about MaxSAT, this type of Problem is considered [IMMS19].

Example 4 A simple exemplary weighted partial MaxSAT-Problem

$$\begin{aligned} \phi_{HARD} &= \{(x_1 \vee x_2 \vee x_3)\} \\ \phi_{SOFT} &= \{(\neg x_1 \vee \neg x_2, w_1), (\neg x_2 \vee \neg x_3, w_2), (\neg x_1 \vee \neg x_3, w_3)\} \end{aligned}$$

Definition 5.1.3. The objective is to maximize over the assignment x

$$\begin{aligned} \text{maximize}_x \quad & \sum_{C \in \phi_{SOFT}} \begin{cases} w_i & C \text{ is satisfied by } x \\ 0 & \text{else} \end{cases} \\ \text{subject to} \quad & x \text{ satisfies } \phi_{HARD} \end{aligned}$$

5.1.4 Minimum and Maximum Dualism

In order to maximize the amount of satisfied clauses, many solvers aim to minimize the number of unsatisfied clauses, as these objectives are functionally equivalent.

We can practically use this dualism to utilize a maximizing solver to solve a minimizing problem. Any instance can be reformulated by inverting the polarity of the soft clauses. This effectively converts our goal of minimizing the amount of satisfied soft clauses to maximizing their violation.

To invert the polarity of a soft clause, we can do the following:

- **Unit clause:** Negate the literal
- **Non-unit clause:** Introduce a new auxiliary variable that is equivalent to the original clause within the hard clauses, and flip this literal.

In the following Example 5, the maximization objective of Example 4 has been converted to a minimization objective:

Example 5 The minimization problem from Example 4 expressed as maximization problem.

$$\begin{aligned}\phi_{HARD} &= \{(x_1 \vee x_2 \vee x_3), (x_a \equiv \neg x_1 \vee \neg x_2), (x_b \equiv \neg x_2 \vee \neg x_3), (x_c \equiv \neg x_1 \vee \neg x_3)\} \\ \phi_{SOFT} &= \{(\neg x_a, w_1), (\neg x_b, w_2), (\neg x_c, w_3)\}\end{aligned}$$

5.2 State-of-the-art MaxSAT-Solvers

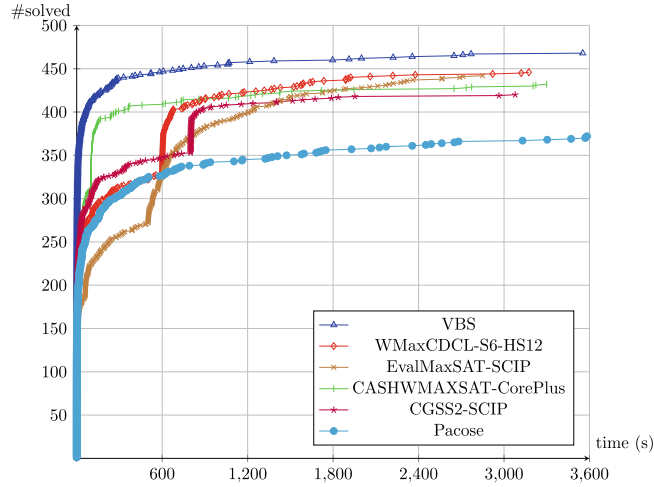
For several years, an international MaxSAT evaluation has been taking place with the aim of comparing the latest improvements in MaxSAT-Solvers against each other¹. There are two categories of solvers [MBJN23]: *Exact Solvers*, which ensure to deliver a global optimum, and *Anytime Solvers*, which aim at delivering an as good as possible solution. The score of the latter is based on the quality measured by the distance of their best found solution, to the currently known maximum. Each of the categories is further split into a weighted and an unweighted track. The results for the 2023 weighted exact track are shown in Figure 5.1.

VBS stands for *virtual best solution*, and it is determined by taking the best-performing solution for each individual benchmark and combining them into an aggregate score. Figure 5.1 shows, that the distance between the best solvers and the *VBS* is small. This indicates that the competing solvers are able to perform well across various instances and are not specialized to one instance type. One very promising solver is called WMaxCDCL-S6-HS12, which will be examined in further detail in the following chapter.

¹<https://maxsat-evaluations.github.io/> accessed on 14.6.2024 at 14:27

Detailed Results

Weighted



- Only one version of each solver is shown in the cactus
- Gap between best solver and VBS is small

14 / 31

Figure 5.1: Detailed Results on the weighted exact track of the 2023 MaxSAT competition [MBJN23]

5.2.1 WMaxCDCL

One of the best performing solvers in 2023 was the WMaxCDCL-S6-HS12 solver (see in Figure 5.1). It is a variant of the WMaxCDCL-Solver which already participated in 2022 [CLL⁺22]. This new variant starts by executing the SCIP-Solver for 10 minutes. If unsuccessful, the MaxHS algorithm is executed for 20 more minutes before it starts the main WMaxCDCL routine with the input of an initial Upper Bound from the MaxHS execution.

The core algorithm is MaxCDCL which has two variants, the standard MaxCDCL and WMaxCDCL which extends the standard algorithm to handle weighted instances [CLL⁺23]. MaxCDCL is based on the CDCL algorithm used in SAT-Solvers. It alternates between decisions and executes unit propagation as well as clause learning. At some points in the execution, it will calculate a *lower bound* (LB) for the number of soft clauses that will be falsified by the current assignment, given the hard clauses. When this new *lower bound* cannot be improved ($LB \geq UB$), a soft conflict is detected and conflict analysis is executed which triggers clause learning. Furthermore, if no soft conflict is detected, but $LB = UB - 1$, all non-falsified soft clauses can be considered as hard clauses and consequently unit propagation is executed. The *lower bound* computation is based on UNSAT-Cores over the soft clauses. For every detected local core, the *lower bound* can be increased by one.

The extension to weighted solvers lead the developers to rewrite a lot of the code, because the weighted variant utilizes a lot more data structures. The contribution to the *lower bound* is different depending on the minimum weight among the clauses of the core. To solve this, the solver will make virtual copies of the soft clause so that their weights can be split on several local cores. Furthermore, the step of hardening had to be changed, since the hardening of a soft clause can lead to falsify new soft clauses which in turn changes the bound – which leads to more hardening. To solve this, WMaxCDCL utilizes a fix point propagation loop which alternates between unit propagation and hardening.

A detailed explanation of the algorithm can be found in the descriptions of the authors [LXC⁺21a, LXC⁺21b]. An implementation is available at the MaxSAT Evaluation 2023 Website.

5.2.2 Relationship with ILP

MaxSAT is closely related to the Integer Linear Programming (ILP) problem which is in fact an alternative formulation of the MaxSAT-Problem. A weighted partial MaxSAT-Problem can be expressed as ILP in the following manner [AG13].

Given a weighted partial MaxSAT-Problem as a set of clauses where clauses, 1 to m are weighted and the clauses $m + 1$ till $m + m'$ are hard (having weight ∞). Formally, $\{(C_1, w_1), \dots, (C_m, w_m), (C_{m+1}, \infty), \dots, (C_{m+m'}, \infty)\}$.

Now we can separate the soft and hard clauses into two sets: $h = (\bigcup_{i=m+1}^n C_i)$ for the hard clauses and $s = (\bigcup_{i=1}^m \text{CNF}(\bar{b}_i \rightarrow C_i))$ for soft clauses. b_i is used as an auxiliary variable that implies the clause. An implication is sufficient instead of a full equivalence, since in the case that the clause is satisfied, the solver will set b_i to 0 to optimize towards the minima. With the $\text{CNF}(\phi)$ we can convert $b_i \rightarrow C_i$ into the CNF.

Given this definition, the ILP-Problem can be stated in the following manner:

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^m w_i \cdot b_i \\ & \text{subject to} && \sum_{l_i \in C_j} \begin{cases} x(l_i) & l_i \\ -x(l_i) & \bar{l}_i \end{cases} > \sum_{l_i \in C_j} \begin{cases} 0 & l_i \\ -1 & \bar{l}_i \end{cases}, \forall C_j \in (s \cup h) \\ & && 0 \leq x_i \leq 1, x_i \in \text{var}(s \cup h) \end{aligned}$$

The problem starts by defining the optimization goal which is to minimize the weight of all soft clauses multiplied by the auxiliary variable for the clause b_i . If b_i is 1, it represents the clause C_i to be unsatisfied. By minimizing the sum of unsatisfied clauses, we maximize the sum of satisfied ones.

The optimization goal is subject to an inequality that is created for each clause in s and h . $x(l_i)$ stands for the ILP variable that represents the literal l_i . The inequality is

created by taking the sum of $x(l_i)$ for positive literals and $(1 - x(l_i))$ for negative literals. Reformulating this inequality to have only variables on the left, we end up with the negated count of negative literals on the right side of the inequality.

Lastly, the bounds are defined such that all the variables are boolean, either 0 or 1.

Example 6 Given the weighted partial MaxSAT-Problem: $\{(x_1 \vee x_2, 2), (x_1 \vee \overline{x_2}, 3), (\overline{x_1} \vee x_2, \infty), (\overline{x_1} \vee \overline{x_2}, \infty)\}$ the ILP problem would look like this:

$$\begin{array}{ll}
 \text{minimize} & 2b_1 + 3b_2 \\
 \text{subject to} & x_1 + x_2 + b_1 > 0; \quad \backslash \quad \overline{b_1} \rightarrow (x_1 \vee x_2) \\
 & x_1 - x_2 + b_2 > -1; \quad \backslash \quad \overline{b_2} \rightarrow (x_1 \vee \overline{x_2}) \\
 & -x_1 + x_2 > -1; \quad \backslash \quad \overline{x_1} \vee x_2 \\
 & -x_1 - x_2 > -2; \quad \backslash \quad \overline{x_1} \vee \overline{x_2} \\
 & 0 \leq x_1 \leq 1; \\
 & 0 \leq x_2 \leq 1; \\
 & 0 \leq b_1 \leq 1; \\
 & 0 \leq b_2 \leq 1;
 \end{array}$$

Comparing IPL-Solvers to MaxSAT-Solvers, they yielded inferior results in the 2023 MaxSAT competition (Figure 5.2). The ILP based *CPLEX*-Solver was able to solve close to 300 problems in time, whereas the best MaxSAT-Solvers reached close to 450 solved problems.

5.3 Approximation Techniques for MaxSAT

Since the complex nature of MaxSAT and the intractability of finding the global optimum for many use-cases, there are several approximating techniques that calculate a locally optimal solution on a best-effort basis with a certain guarantee of distance to the global optimum [Stü18]. This distance is defined by the *approximation ratio* $v(x, s)$, which is defined in the following manner:

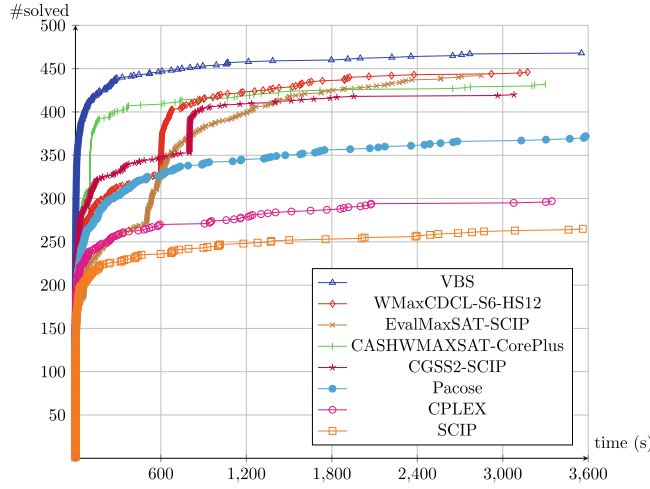
Definition 5.3.1. An *approximation ratio* of a solution $s \in S(x)$ takes the ratio of the value of solution $v(x, s)$ to the globally optimal solution $v_{best}(x)$

$$r(x, s) = \frac{v(x, s)}{v_{best}(x)} \quad (5.1)$$

For calculating the approximation ratio of an algorithm instead of only one solution, we have to consider the worst case ratio over all instances x .

How well do ILP solvers perform by themselves?

Weighted



► ILP solvers by themselves are not competitive with MaxSAT solvers

16 / 31

Figure 5.2: Detailed results on the weighted track of the 2023 MaxSAT competition with ILP-Solvers [MBJN23]

Definition 5.3.2. An *approximation ratio* of an algorithm is defined over the worst-case ratio of all instances x

$$r_A = \sup\{r \leq 1 : r(x, s_A(x)) \geq r \text{ for all instances } x\} \quad (5.2)$$

Known Algorithms and their Approximation Bounds

One of the first approximation algorithms is by Johnson [Joh74], who found a $\frac{1}{2}$ -approximation algorithm (outlined in Section 5.3.1). Jianer and Friesen further improved the bound of this algorithm to $\frac{2}{3}$ and proved that this bound is tight [CFZ97].

There are also some $\frac{3}{4}$ -algorithms. For example, Yannakakis found an algorithm which preprocesses the MaxSAT-Instance and converts it into a *network flow problem* which can be solved by non-trivial network flow algorithms [Yan94]. Furthermore, Gormans and Williamson found a $\frac{3}{4}$ -algorithm that combines the Johnson algorithm with a randomized rounding approach that is applied on a linear programming representation of the instance [GW94].

In consequent years, these bounds were further improved up to 0.7968, introducing hybrid algorithms and Semidefinite Programming Techniques [Stü18]. Currently, the algorithm with the highest bound is by Avidor, Berovitch and Zwick, which is an in several steps improved version of Gormans and Williamson [ABZ06].

5.3.1 Johnson's Greedy algorithm

Greedy algorithms are baseline approximation algorithm for many optimization problems. They are called *greedy* because they iteratively make decisions based on a currently local heuristic and never roll back any decision after it is made. Due to their incremental approach they –in most cases– have a polynomial runtime.

Johnson has created an algorithm that approaches the MaxSAT topic in this greedy manner [Joh74]. The Algorithm 5.1 (originally called *B2*) iterates all literals in the CNF and decides an assignment based on a local heuristic.

As the first step of the algorithm, all clauses are assigned to some virtual weight based on the amount of literals they contain and their original weight $w_c 2^{-|c|}$. The virtual weight of a clause is inversely proportional to its number of literals, as more literals offer more potential ways to satisfy the clause. Later we will make use of the power of 2 formulations because we can multiply the current weight by 2, when we remove a literal from the clause to update the clauses virtual weight.

Next, we initialize 4 sets. The sets S_{SAT} and L_{SAT} are initialized to empty. These sets will later contain the found literal assignments L_{SAT} and the satisfied clauses S_{SAT} . S_{LEFT} and L_{LEFT} on the other hand contain the clauses that still need to be considered.

At this point, we can start with the main loop (Line 5-25 in Algorithm 5.1). The iteration executes the following steps, as long as there are literals in L_{LEFT} to consider. At each iteration we first pick a random literal y from the leftover literals and create two sets of clauses S_y and $S_{\bar{y}}$ that represent all clauses with y and \bar{y} respectively. Now we sum the virtual weights of both these set of clauses and decide y to the assignment that yields a better result. The consequences of this assignment are now executed, and the sets are adapted to represent this decision. All clauses which contain the opposite of the assignment, will be multiplied by 2 to incorporate the removal of one option to satisfy the clause.

Theorem 1. [Stü18] *For MaxSAT-Instances with $k \geq 1$ literals per clause, Algorithm 5.1 has an approximation ratio of $r(n) \geq \frac{2^k - 1}{2^k}$*

The approximation ratio for this algorithm is strongly dependent on the number of literals per clause. The more literals in the smallest clause, the higher the approximation bound. For a MAX-3-SAT-Problem, the approximation would be $\frac{7}{8}$. A detail proof of this bound can be found at [Stü18].

5.3.2 Simulated Annealing

Simulated Annealing is an optimization algorithm based on local search that is able to escape local optima [NJ10]. Its conceptual origin stems from thermodynamics, where a crystalline structure is heated up and then slowly cooled, until a regular crystalline

Algorithm 5.1: Johnson's Greedy Algorithm [Stü18]

Input : Set of clauses S
Output : Set of literals L_{SAT} and set of satisfied clauses S_{SAT}

```

1 for all  $c \in S$  do
2   |  $w(c) \leftarrow w_c 2^{-|c|}$ 
3 end
4  $S_{SAT} \leftarrow \emptyset, L_{SAT} \leftarrow \emptyset, S_{LEFT} \leftarrow S, L_{LEFT} \leftarrow$  literals in  $S$ ;
5 while  $L_{LEFT} \cap S_{LEFT} \neq \emptyset$  do
6   | Let  $y \in L_{LEFT}$  such that  $y$  occurs in  $S_{LEFT}$ ;
7   |  $S_y \leftarrow$  clauses in  $S_{LEFT}$  containing  $y$ ;
8   |  $S_{\bar{y}} \leftarrow$  clauses in  $S_{LEFT}$  containing  $\bar{y}$ ;
9   | if  $\sum_{c \in S_y} w(c) \geq \sum_{c \in S_{\bar{y}}} w(c)$  then
10    |  $L_{SAT} \leftarrow L_{SAT} \cup \{y\}$ ;
11    |  $S_{SAT} \leftarrow S_{SAT} \cup S_y$ ;
12    |  $S_{LEFT} \leftarrow S_{LEFT} \setminus S_y$ ;
13    | for all  $c \in S_{\bar{y}}$  do
14    |   |  $w(c) \leftarrow 2w(c)$ ;
15    | end
16    | else
17    |  $L_{SAT} \leftarrow L_{SAT} \cup \{\bar{y}\}$ ;
18    |  $S_{SAT} \leftarrow S_{SAT} \cup S_{\bar{y}}$ ;
19    |  $S_{LEFT} \leftarrow S_{LEFT} \setminus S_{\bar{y}}$ ;
20    | for all  $c \in S_y$  do
21    |   |  $w(c) \leftarrow 2w(c)$ ;
22    | end
23    | end
24    |  $L_{LEFT} \leftarrow L_{LEFT} \setminus \{y, \bar{y}\}$ ;
25 end
26 return  $L_{SAT}, S_{SAT}$ ;

```

lattice structure, free of imperfections, is formed. For this to occur, the cooling must be sufficiently slow to allow the forming of a perfect structural layout.

Comparably, the simulated annealing algorithm defines a cooldown schedule t_k , a function that approaches 0. At the beginning of the execution of the algorithm, t_k is big, such that the algorithm is encouraged to explore the possible solution space. As the cooldown is progressing, the algorithm gradually shifts its emphasis from exploration to exploitation. This allows it to focus on refining and optimizing a promising local solution.

In Algorithm 5.2 the overall concept is outlined. The goal of the algorithm is to find the global optimum ω^* such that $f(\omega^*) < f(\omega)$ for all $\omega \in \Omega$. It runs in two nested loops. The outer loop is the cooldown loop, which progresses the cooldown function, and it

is terminated, if a certain termination condition is met (e.g. temperature sufficiently low, good enough solution reached, ...). Inside this loop there is another loop that is executed M_k times. This loop samples a neighbor solution ω' to the current solution ω . The delta in terms of cost $\Delta_{\omega,\omega'}$ is calculated. If the delta is negative, meaning that ω' is better in terms of cost than ω it is directly selected as the new ω , else ω' is selected as the new ω , based on a probability that is influenced by the current temperature t_k .

The temperature is reduced in discrete steps t_k while also the repetition schedule M_k is reduced. The repetition schedule defines how many samplings occur in each given cooldown step k .

Algorithm 5.2: Simulated Annealing Algorithm [NJ10]

```

1  $\omega \in \Omega$  ; // sample random solution  $\omega$ 
2  $k \leftarrow 0$ ;
3  $t_k \leftarrow$  temperature cooling schedule;
4  $M_k \leftarrow$  repetition schedule;
5 repeat
6    $m \leftarrow 0$ ;
7   repeat
8      $\omega' \in N(\omega)$  ; // sample neighbor solution
9      $\Delta_{\omega,\omega'} = f(\omega') - f(\omega)$ ;
10    if  $\Delta_{\omega,\omega'} \leq 0$  then
11       $\omega \leftarrow \omega'$  ; // select better solution
12    else
13       $\omega \leftarrow \omega'$  with probability  $e^{\Delta_{\omega,\omega'}/t_k}$ ;
14    end
15     $m \leftarrow m + 1$ ;
16  until  $m = M_k$ ;
17   $k \leftarrow k + 1$ 
18 until stopping criteria is met;
19 return  $\omega$ ;
```

Simulated Annealing in MaxSAT

One of the first applications simulated annealing on the MaxSAT- and the SAT-Problem was proposed by Spears [Spe93] in 1998 to tackle large SAT-Problems that were not feasible to be solved with traditional solvers. For each temperature step, the algorithm iterates all variables and flips them individually. For each flip, the delta (Δ) of satisfied clauses is calculated. This delta together with the temperature is supplied to a logistic probability function to accept or reject the flip.

5.4 Multi-Objective MaxSAT-Solvers

Over the lifecycle of the product we frequently encounter the challenge of balancing several competing objectives. For example, we want to build the cheapest, but also the highest quality product, or we want to buy a cheap electric car with the best possible range.

Formally, we can express a multi-objective optimization instance over the functions $o_1, o_2 \dots o_n$ [BF17].

Definition 5.4.1. A multi-objective optimization problem is defined over several (conflicting) objectives.

$$\min_x [o_1(x), o_2(x), \dots, o_n(x)] \quad (5.3)$$

In terms, of MaxSAT we express each objective $o_n(x)$ as a separate set of soft clauses.

Pareto Front

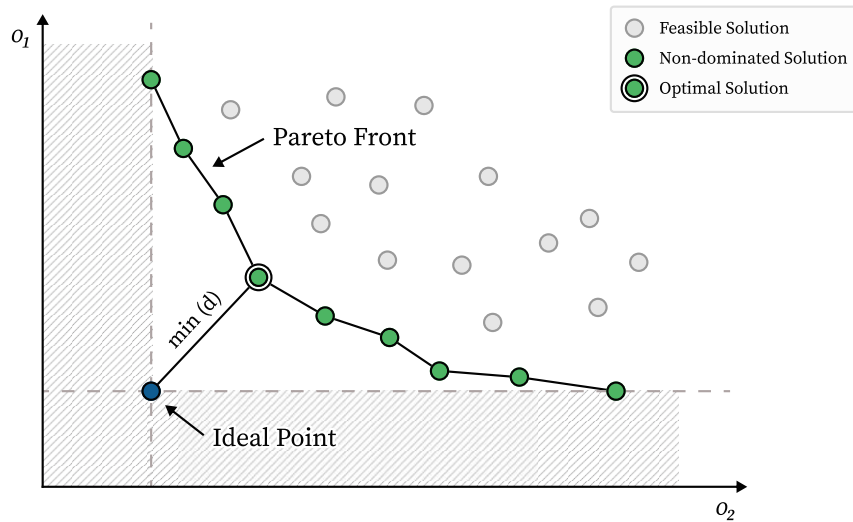


Figure 5.3: Illustration of the Pareto Front with the contradictory optimization objectives o_1 and o_2 (Graphic adapted from [BF17])

When optimization problems involve conflicting objectives, there is not a single best solution [BF17]. Instead, there is a set of equally good solutions, each with different trade-offs between the objectives. These equally good solutions are called non-dominated solutions, since there is no other solution, that improves one objective without sacrificing another. Together they form what is known as a Pareto Front (Figure 5.3).

A Pareto Front contains at least one ideal solution, which is the solution with the minimal distance to the ideal point. The ideal point is defined by the individual maximum for

each objective. The ideal solution is the best solution, when the trade-offs are considered equally. With different weightings on each objective, the ideal-solution can change.

Example 7 In Figure 5.4 an exemplary Pareto Front of the Battery Electric Vehicle (BEV) passenger car market as of December 2020 is given. The chart shows two objectives that a customer usually will have, when choosing a new vehicle. On the one hand, the price should be minimal (y-axis). While, on the other hand, the maximum driving range should be as high as possible (x-axis). The axes are oriented, such that the ideal point is in the origin of the axes (on the left lower in the chart).

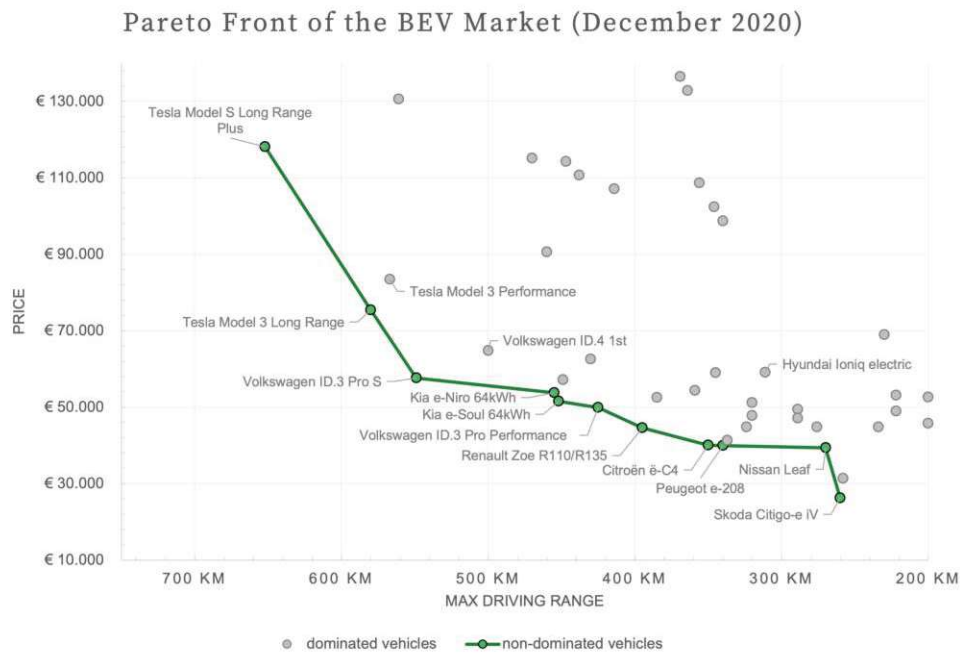


Figure 5.4: An exemplary Pareto Front of the electric passenger car market until the 2nd December 2020 (Dataset [HK21])

BEV-Models that are non-dominated are shown in green. For example, the *Tesla Model 3 Long Range* is non-dominated, since a cheaper alternative like the *Volkswagen ID.3 Pro S* will sacrifice a few km of range and an alternative with more range, like the *Tesla Model S Long Range Plus*, will sacrifice on price. A dominated solution on the other hand, like the *Tesla Model 3 Performance* is dominated by the *Tesla Model 3 Long Range*, since the latter is cheaper **and** has a longer range.

Each optimum for a single objective, is always a boundary for the Pareto Front. For example the *Skoda Citigo-e iV* is absolutely the cheapest vehicle and any other solution will sacrifice on the price, hence, this is a non-dominated solution. For the *Tesla Model S Long Range Plus* the same argument can be made for the range. Therefore each Pareto Front will always include one solution, which maximizes each single objective.

Even though each non-dominated solution is an equally valid solution, some are less prominent than others. For example, the *Nissan Leaf* has only a little more range than the *Skoda Citigo-e iV*, but is several thousand euros more expensive, also the *Peugeot e-208* is only a bit more expensive, while offering around 40 km more range. In this situation, the Pareto Front is showing a concave curve. On the other hand the *Volkswagen ID.3 Pro S* is an very outstanding solution, which results in a lot of trade-offs in the direction of both objectives. In this case, the Pareto Front shows a convex curve. Comparatively, around the *Citroën ë-C4* and *Peugeot e-208* the curve shows a slight convex behavior. This is why, when making a choice, solutions in convex parts of the curve are of particular interest.

The ideal solution for this case would be the *Volkswagen ID.3 Pro S*, since it has the lowest distance to the ideal-point. The ideal point is located at 652km and 26.256€.

5.4.1 Preliminaries

There are several algorithms that are able to solve multi-objective MaxSAT-Problems. This thesis will explore *P-MINIMAL* and *BIOPTSAT*. But before we can start introducing these algorithms, we first need to introduce the concept of Totalizers.

Totalizers

To encode bounds into the SAT-Instance, we can utilize the totalizer formulation [JBNJ22]. Given a bound in a range of $k = 1, \dots, n$ and a set of n literals L , $\text{TOT}(L, k)$ defines a set $\{\langle L < 1 \rangle, \dots, \langle L < k \rangle\}$ of literals so that each literal l_k limits the amount of satisfied literals in L . Given the assignment τ satisfies $\text{TOT}(L, k)$, then $\tau(\langle L < b \rangle) = 1$ if $\sum_{l \in L} \tau(l) < b$.

Incremental totalizers allow for the increasing of the bound k , without the need to rebuild the existing totalizer. This can be achieved by defining the formalism such that $\text{TOT}(L, k) \subset \text{TOT}(L, k')$ and $\text{TOT}(L, k) \subset \text{TOT}(L \cup L', k)$ hold for any set L' and $k' > k$.

5.4.2 *P-MINIMAL* Algorithm

The *P-MINIMAL* solver was developed to solve the Multi-Objective Discrete Optimization Problem (MODOP) over a Constraint Satisfaction Problem (CSP), which is corresponding to a multi-objective MaxSAT-Problem [SBTLB17, JBNJ22]. The algorithm enumerates all *P-MINIMAL* models and efficiently blocks away already dominated solutions by using only one clause per previously found model.

The *P-MINIMAL* algorithm introduces a totalizer for every objective function $o_x \sim \text{TOT}_x \sim \text{TOT}(L_x, k_x)$. The range of this totalizer is defined from the lower to the upper bound of the optimization function by setting $k_x = lb(o_x), \dots, ub(o_x)$, where $lb(o_x)$ and $ub(o_x)$ represent the lower and the upper bound of o_x respectively. With the aid of the totalizers, each boundary b for an objective function has a corresponding literal

$l_{x,b} = \langle L_x < b \rangle$, which represents that the value for the objective function o_x must be smaller than b .

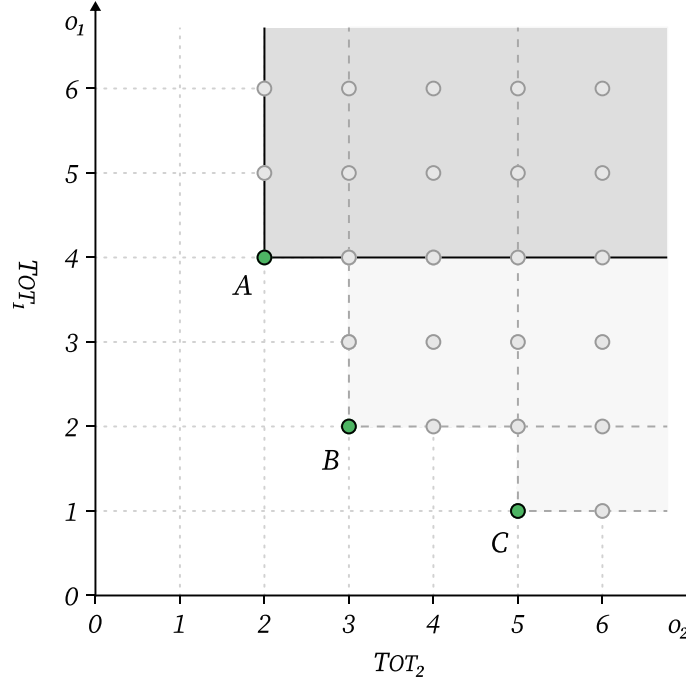


Figure 5.5: Regions pruned by blocking clauses (adapted from [SBTLB17])

Algorithm 5.3: *P*-MINIMAL Algorithm (adapted from [SBTLB17]²)

Input: A Multi-objective MaxSAT-Instance $\Omega = \langle (o_1, \dots, o_m), \phi_{HARD} \rangle$

Output: The Pareto Front of Ω

```

1  $P \leftarrow \text{TOT}(o_1) \wedge \dots \wedge \text{TOT}(o_m)$ ;
2  $\mathcal{Y}_N \leftarrow \emptyset$ ;
3 while  $M_P = \text{FINDP-MINIMALMODEL}(\phi_{HARD}, P)$  do
4    $\mathcal{Y}_N \leftarrow \mathcal{Y}_N \cup \{M_P\}$ ;
5    $\phi_{HARD} \leftarrow \phi_{HARD} \wedge \text{BLOCK}(M_P)$ ;
6 end
7 return  $\text{DECODE}(\mathcal{Y}_N)$ ;

```

The main routine of the *P*-MINIMAL Algorithm 5.3 uses the totalizers to iteratively execute the procedure to find a new solution and then to block all the solutions that are dominated by the found solution. In Figure 5.5 the blocked region is visualized for the

²Changed formalism to fit this paper and the MaxSAT-Problem

solution A as the gray region. Similarly, B and C define a region of dominance, that renders all solutions in this region as dominated.

For example, given the solver found a solution τ which achieves $o_1(\tau) = 4$ and $o_2(\tau) = 2$ (Point A in Figure 5.5), every solution with $o_1(\tau') \geq 4 \wedge o_2(\tau') \geq 2$ is dominated by τ and therefore not part of the Pareto Front. To encode the constraint that blocks the dominated region, we can add a condition that states that $\neg(o_1(\tau') \geq 4 \wedge o_2(\tau') \geq 2)$. By utilizing the totalizers in combination with DeMorgan's law we can add the clause $\langle L_1 < 4 \rangle \vee \langle L_2 < 2 \rangle$ as constraint to the hard clauses (Algorithm 5.3 line 5). Generally, Definition 5.4.2 defines the clause that is needed to block the dominated region as the function $\text{BLOCK}(\tau)$.

Definition 5.4.2. $\text{BLOCK}(\tau)$ defines a constraint that blocks all solutions that are dominated by τ

$$\text{BLOCK}(\tau) = \bigvee_{1 \dots m} \langle L_m < o_m(\tau) \rangle \quad (5.4)$$

Algorithm 5.4: FINDP-MINIMALMODEL

Input: ϕ_{HARD}, P

Output: A non-dominated solution M_P

```

1  $\tau \leftarrow \text{GETSOLUTION}(\phi_{\text{HARD}} \wedge P, \emptyset)$  ;
2  $M_P \leftarrow \emptyset$  ;
3 while  $\tau \neq \emptyset$  do
4    $M_P \leftarrow \tau$  ;
5    $\tau \leftarrow \text{GETSOLUTION}(\phi_{\text{HARD}} \wedge P, \{\langle L_m < o_m(\tau) \rangle \mid 1, \dots, m\})$ ;
6 end
7 return  $M_P$ ;

```

The function $\text{FINDP-MINIMALMODEL}(\phi_{\text{HARD}}, P)$ yields a new non-dominated solution. It starts by getting a random solution τ and then sets the assumptions to $\{\langle L_m < o_m(\tau) \rangle \mid 1, \dots, m\}$, which restricts the temporary search-space to be a solution that is dominating τ . If such a solution is found, it is considered the new τ and the process is repeated, else the solution is non-dominated and τ yielded as new non-dominated solution.

The main loop of the P -MINIMAL algorithm utilizes $\text{FINDP-MINIMALMODEL}(\phi_{\text{HARD}}, P)$ to find a new non-dominated solution M_P . The found solution is added to the set of non-dominated solution $\mathcal{Y}_{\mathcal{N}}$ and a corresponding blocking clause is added. This loop is executed until no further non-dominated solution can be found. This leads to the termination of the algorithm and a final decoding of the list of all found solutions $\mathcal{Y}_{\mathcal{N}}$.

5.4.3 BIOPTSAT Algorithm

BIOPTSAT is an optimization algorithm specialized on the bi-objective (2-dimensional) case. Functionally it is related to *P-MINIMAL* and also utilizes totalizers in combination with assumptions to iteratively enumerate all pareto-optimal solutions [JBNJ22, JBNJ24].

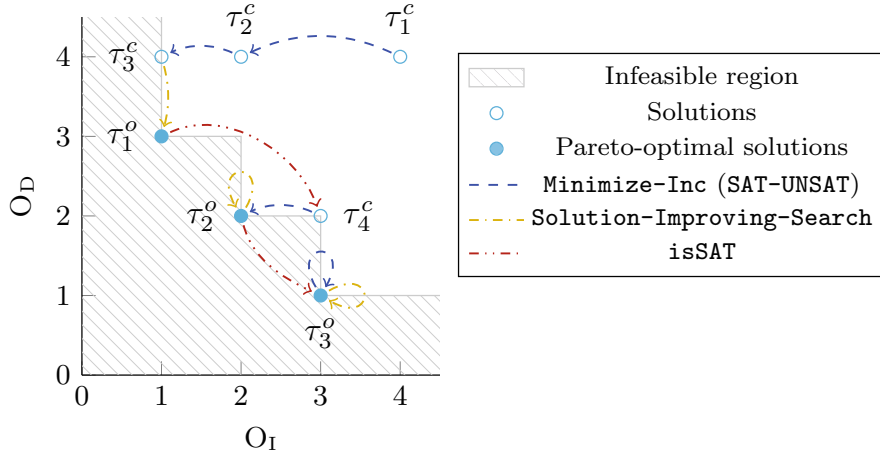


Figure 5.6: Search Trace of BIOPTSAT [JBNJ24]

The algorithm, as outlined in Algorithm 5.5, starts by sampling a solution τ from the hard clauses ϕ_{HARD} and then continues by optimizing each objective function in separate steps. As boundaries, b_D is initialized as ∞ and b_I as 0.

In Figure 5.6, the search trace of BIOPTSAT for an example problem is illustrated. The first solution τ_1^c is improved in the direction of O_I , using the MINIMIZE-INC function with the bound of b_D . In the first execution, the bound is ∞ and therefore the global optimum for O_I will be reached. In the example, this yields τ_3^c .

The next step, SOLUTION-IMPROVING-SEARCH, is employed to optimize the direction of O_D , such that objective O_I cannot get worse than the found bound b_I . The resulting solution (τ_1^o) must be non-dominated and consequently is yielded as solution. To escape the local minima of (τ_1^o), the Sat-Solver is invoked with the assumption that it must find a solution that improves in the direction of O_D since every new non-dominated solution must satisfy $\langle O_D < b_D \rangle$. This process is repeated, till the optimum for O_D is reached.

The subroutine MINIMIZE-INC is executing a single objective MaxSAT routine that can deal with assumptions. In the paper [JBNJ22] 5 different classical MaxSAT algorithms are proposed and analyzed to utilize in this task.

The second subroutine, SOLUTION-IMPROVING-SEARCH, employs the SAT-Solver in combination with the totalizers to improve the existing solution towards a non-dominated, pareto-optimal solution. This is done by iteratively decreasing a bound k starting from $k = O_D(\tau)$ while invoking the SAT-Solver with the assumption $\{\{\langle O_D < k \rangle\}\{\langle O_I \leq b_I \rangle\}\}$. The smallest possible k , which is still SAT, is yielded as the found non-dominated optima.

Algorithm 5.5: BiOPTSAT: MaxSAT-based bi-objective optimization [JBNJ22]**Input:** CNF formula ϕ_{HARD} , objectives O_I and O_D **Output:** Either one or all Pareto-optimal solutions corresponding to each Pareto point of ϕ_{HARD}

```

1  $\tau \leftarrow \text{GETSOLUTION}(\phi_{HARD}, \emptyset)$  ;
2 if  $\tau = \emptyset$  then
3   | return no solutions
4 end
5  $b_D \leftarrow \infty, b_I \leftarrow 0$  ;
6 while  $\tau \neq \emptyset$  do
7   |  $(b_I, \tau) \leftarrow \text{MINIMIZE-INC}(b_D, O_I(\tau))$  ;
8   |  $(b_D, \tau) \leftarrow \text{SOLUTION-IMPROVING-SEARCH}(b_I, O_D(\tau))$  ;
9   | yield  $\tau$  ;
10  |  $\tau \leftarrow \text{GETSOLUTION}(\phi_{HARD}, \{O_D < b_D\})$ 
11 end

```

5.4.4 Comparison

In Figure 5.7 from [JBJ24], a comparison of the search traces for *P-MINIMAL*, BiOPTSAT and LOWERBOUND is given. LOWERBOUND is an extension to *P-MINIMAL* that starts setting an upper bound on each objective and executes *P-MINIMAL* inside this bound. During the execution, the bound is iteratively expanded, so that the algorithm covers the whole Pareto Front.

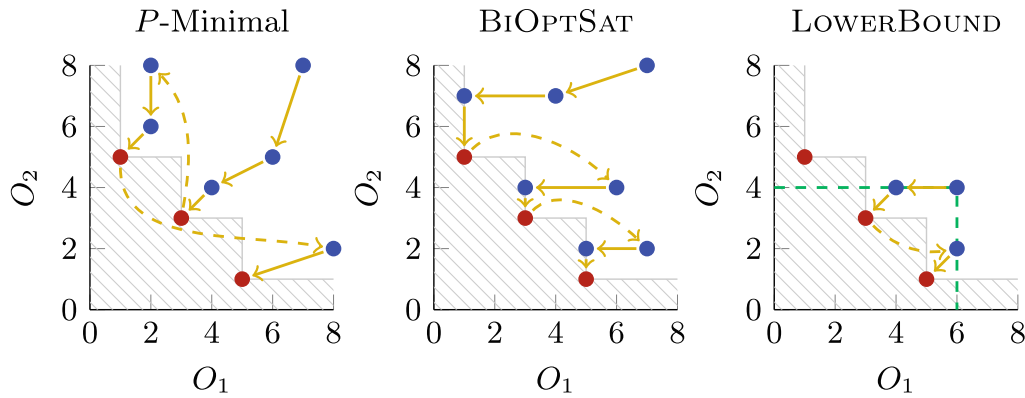


Figure 5.7: Comparison of search traces for P-MINIMAL, BiOPTSAT and LOWERBOUND, [JBJ24]

5.5 Optimization Objectives in FBD

FBD offers several possible ways for applying MaxSAT-Solvers as well as multi-objective MaxSAT-Solvers. With the reduction pipeline of Chapter 4 and the formalism of Section 2.2.9, we can transform the restrictions and features into hard clauses. Adding onto that, the soft clauses can represent different (conflicting) optimization objectives. The following subsections will define several optimization objectives and describe them formally, afterwards these objectives will be put into action in different experiments in the next section.

In a practical example, these objectives could be used as part of a configuration process for the customer. In highly complex products with several hundred feature groups to select, the customer typically only selects a few of the features. The optimization towards various optimization objectives can afterwards be done on the basis of Pareto Fronts analysis. Furthermore, during the configuration steps, each next choice can be evaluated in terms of the optimizations objectives, to steer the customer towards given optima.

In addition to the configuration process, these objectives provide a method for product analysis, for example the configuration with the worst profit margin can be found by comparing the objective to minimize *sales price* to the objective of maximizing the *material cost*.

5.5.1 Best Customer Fit

Customers can use features to tailor the product to their specific needs. Each customer's choice is represented by a configuration, which may be complete or partial, if they have no preference for a particular feature group. While optimality (price, delivery time, etc.) and feasibility (restrictions) may vary across configurations, our primary objective is to fulfill the customer's wishes as closely as possible.

Given a customer configuration, either partial or complete, denoted by $C = \{c_1, \dots, c_n\}$, we can express the objective of achieving the best customer fit by the minimization of the deviation from C .

Definition 5.5.1.

$$\phi_{best\ fit} = \{(\neg c, 1) \mid c \in C\}$$

The optimization function utilizes a penalty-based approach, to punish the solver for every feature that it deviates from the customers' selection, thus it will force the optimizer to fit the product to the customers wishes as closely as possible. Potential trade-offs with respect to other optimization criteria, can later be argued in the Pareto Front.

One of the challenges with features is how to encode the relevance of different feature groups. Not every feature group is equally relevant. For example, the motor type is, generally speaking, a selection, that is highly important to the customer. Compared to

that, the customer typically does not hold strong opinions about the type of side mirror. But this relevance is hard to measure objectively since it differs for every use-case and customer. For the sake of this thesis we will consider each feature group equally relevant.

5.5.2 Material Cost

To compute the material cost for a given configuration, we need to sum the material cost of each allocated part A in the FBD instance \mathcal{D} .

We define the price of a given part A as $\$(A) \in \mathbb{R}^+$. Since MaxSAT-Solvers are defined over discrete integer values, we need to define a bucketing function $\sigma(x) : \mathbb{R}^+ \rightarrow \mathbb{N}$ that returns integers and with which we find the integer cost of each given part. The aim of the bucketing function is to roughly maintain the additive property: $\sigma(x) + \sigma(y) \approx \sigma(x + y)$.

A possible bucketing function is given here. This function depends on the maximum price and creates k equally sized buckets using rounding.

$$\sigma_S^f(x) = \left\lfloor \frac{f(x) \cdot k}{\max_{v \in S} f(v)} \right\rfloor$$

Definition 5.5.2. The optimization objective for the material cost is defined as:

$$\phi_{\text{material cost}} = \{(\phi(A), \sigma_{A \in \mathcal{D}}^{\$(A)}) \mid A \in \mathcal{D}\}$$

This concept can be extended beyond material cost by integration production process data from the manufacturing BoM (for example the required processes, consumables, etc). With the adding of this data we could achieve a more comprehensive optimization model.

5.5.3 Sales Price

The sales price of configurations commonly is defined by prices assigned to specific feature choices. For example, the selection of the large motor or the premium convenience package have a higher price point assigned to them.

More broadly, the price is defined by a set of conditions that add a certain price point, if a condition is satisfied. The cumulative sum of all of these price points is the sales price.

Definition 5.5.3. Each price point is defined as a tuple in the set of all price points K .

$$K = \{\langle f(c_0, \dots, c_n), p \rangle, \dots\}$$

Similar as in the objective before we need a bucketing function $\sigma_K(x)$ to discretize the values with respect to the maxima for the set K .

Definition 5.5.4. The sales price optimization objective can be defined as:

$$\phi_{sales\ price} = \{(f, \sigma_K(p)) \mid \langle f, p \rangle \in K\}$$

5.5.4 Delivery Time

The time from order to fulfillment should be as short as possible. Practically, there are many reasons for delivery time and large Machine Execution System (MES) systems are used, to optimize the overall ability of a company to produce products fast and efficiently. However, the most central part of the delivery time is the availability of needed subcomponents. To encode this availability as an optimization objective, we need to know the availability for every part A in the FBD instance \mathcal{D} . We can express the availability in several categories and assign a cost to them as seen in Table 5.1. These are exemplary categories, for other scenarios the categories may differ.

Availability Category	cost	$\alpha(A)$
In Stock	0	a_0
Backorder ≥ 1 day	1	a_1
Backorder ≥ 3 days	3	a_3
Backorder ≥ 10 days	10	a_{10}
Backorder ≥ 30 days	30	a_{30}

Table 5.1: Example Categories for the part availability

For every part we know the current availability $\alpha(A)$ expressed in the literals a_0 to a_{30} . It is significant to consider that the delivery time of any given configuration is defined by the worst case availability of any part needed to build the configuration, since the parts must be available for assembly. Therefore the following encoding is proposed:

Definition 5.5.5. The optimization objective for the delivery time is defined as:

$$\begin{aligned} \phi_{hard} = & [\forall A \in \mathcal{D} \phi(A) \Rightarrow \alpha(A)] \wedge \\ & (d_0 \Rightarrow \neg a_1 \wedge \neg a_3 \wedge \neg a_{10} \wedge \neg a_{30}) \wedge \\ & (d_1 \Rightarrow \neg a_3 \wedge \neg a_{10} \wedge \neg a_{30}) \wedge \\ & (d_3 \Rightarrow \neg a_{10} \wedge \neg a_{30}) \wedge \\ & (d_{10} \Rightarrow \neg a_{30}) \wedge \\ & (d_{30} \Rightarrow \top) \end{aligned}$$

$$\phi_{delivery} = \{(d_0, 0), (d_1, 1), (d_3, 3), (d_{10}, 10), (d_{30}, 30)\}$$

The literals d_0 to d_{30} stand for the requested worst case delivery time. For any d to be satisfied, the part delivery times (a_0 to a_{30}) cannot be worse. This condition is expressed through hard clauses in ϕ_{hard} . Every delivery objective d is assigned to the corresponding cost in the soft clauses of the optimization objective $\phi_{delivery}$.

5.5.5 Standardization

Product line diversity and complexity, which is often growing exponentially, can significantly strain a business's resources and operational efficiency. This diversity is driven by both, deliberate choices and lack of guidance towards standards. Therefore, many companies strive to reduce the complexity and unnecessary diversity of their product offering by incentivizing customers towards more standard configurations. Especially mitigating the creation of unintentionally novel configurations that could have been served with already existing configurations.

If a customer still wants to deviate from a given standard, companies tend to allow that for an extra cost, if the added technical complexity is acceptable. Consequently, the definition of the standardization is very similar to the sales price, but still considered as a separate aspect of optimization, since these aspects can be in conflict.

Definition 5.5.6. Each standard is defined as a tuple in the set of standards \mathcal{S} . A standard tuple consists of a condition and a weight to represent the importance of the standard.

$$\mathcal{S} = \{ \langle f(c_0, \dots, c_n), w \rangle, \dots \}$$

Definition 5.5.7. The standardization optimization objective can be defined as:

$$\phi_{standardize} = \{ (f, w) \mid \langle f, w \rangle \in K \}$$

5.5.6 Environmental Sustainability

A commitment to environmental responsibility throughout the whole product lifecycle, from sourcing materials to sustainable production and energy efficient products, is an important aspect for businesses. This ambition is driven by customer demand, regulatory compliance and brand reputation. The pressure to reduce the carbon footprint and to minimize the environmental impact is ever-increasing. Particularly in the EU, where stringent regulations to promote sustainability and to combat climate change have been implemented, but also in the rest of the world, companies are trying to optimize their products to operate more environmentally friendly.

To evaluate the environmental aspects of production and source materials, we want to analyze how negative the influence of a particular material used in a product is on the

environment. A measure called $CO_2 - eq$ (CO_2 -equivalence, also known as CO_2e), has been created, that, based on scientific evaluations, serves as a lookup table for the carbon footprint of different source materials measured relative to the effects of CO_2 [ACF⁺15]. One such data source is the *GaBi* database³, which is one of the leading databases for Life Cycle Assessment (LCA) worldwide. We can make use of this database and encode the $CO_2 - eq$ data into the MaxSAT-Solver instance to optimize the product towards an environmentally friendly product choice.

Definition 5.5.8. The optimization objective for $CO_2 - eq$ reduction is defined as:

$$\phi_{CO_2} = \{(\phi(A), \sigma_{A \in \mathcal{D}}^{CO_2 - eq}(A)) \mid A \in \mathcal{D}\}$$

Beyond $CO_2 - eq$, there are other measurements that can be considered, like the energy consumption for production steps, the waste generation of certain parts (battery) and the reusability of components, etc.

5.5.7 Product Weight

Another optimization objective is the total weight of the final product. Depending on the use-case of the product, it might be a technical requirement to keep the product as light as possible, e.g. for the airplane industry or highly performant race cars. Given that we have weight information about the weight of each part A given as $g(A)$, we can encode this optimization objective analog to the material cost.

Definition 5.5.9. The optimization objective for the product weight is defined as:

$$\phi_{weight} = \{(\phi(A), \sigma_{A \in \mathcal{D}}^g(A)) \mid A \in \mathcal{D}\}$$

³<https://ghgprotocol.org/gabi-databases> accessed on 15.10.2024 22:29, accessed on 2.7.2024, 8:30

Experiments and Evaluation

To qualitatively analyze the concept and evaluate the performance, the author has conducted several experiments on the dataset of the TT-Plattform. The following sections will present the resulting Pareto Fronts and discuss the results.

The used algorithm for most experiments is the *P*-MINIMAL-Algorithm. For later experiments that are more complex and the computation time exceeded 1 second, the performances of *P*-MINIMAL and BiOPTSAT are compared.

6.1 Implementation Details

In the course of this thesis, the author has implemented a versatile transformation pipeline for FBD instances as mentioned in Section 4.1.3. The combination of this pipeline and a *Rust* implementation of several multi-objective MaxSAT-Solvers by Christoph Jabs (called *scuttle*¹) has formed a capable analysis and optimization toolkit for FBD. The *scuttle* repository includes implementations for *P*-MINIMAL and BiOPTSAT as well as data structures that are needed to deal with MaxSAT-Instances.

The authors' implementation includes an instance loading mechanism that can build SAT- and MaxSAT-Instances directly from product documentation databases including the *EFS Modularity Suite* toolkit. This implementation will be used for the following evaluation. It provides several methods to load the instances and convert the relevant metadata into objectives for the solver, based on the transformation pipeline explained in Chapter Reducing to the Satisfiability Problem.

¹<https://github.com/chrjabs/scuttle> accessed on 15.10.2024 22:29

6.2 Best Fit vs. Sales Price

This experiment evaluates how a customer can be guided towards a cheaper variant of the product by changing least possible desired configuration choices. The context for such an optimization strategy can be a guided configuration process, where the customer can define their desired features and a schooled sales representative can lead a negotiation on what features are essential and which can be dropped to achieve given price targets.

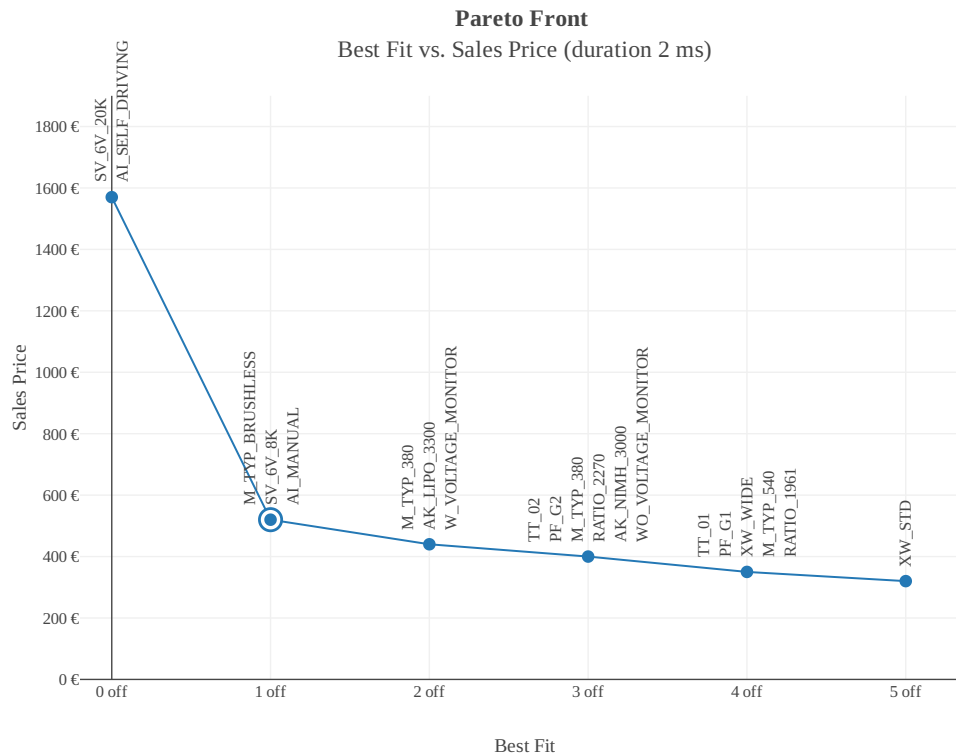


Figure 6.1: Pareto Front of best fit vs. sales price in the TT-Platform

In Figure 6.1 the customer's choices (Table 6.1) are evaluated against the sales prices for each feature (as listed in Table 6.2). The goal is to minimize the price while maximizing the customer fit. The optimal configuration in terms of the customers fit, is including all desired features. It costs 1.560€, since the hardware needed for full self-driving is relatively expensive. The cheapest option on the other hand is taking the old model (TT_01) without any add-on features (like autonomous driving, brushless motor, etc.). The Pareto Front shows the trade-off from the best fit to the cheapest option in 5 steps along a convex curve.

The first choice that is dropped for the customer is the self-driving feature, since it is by far the most expensive feature. The next trade-off is choosing a brushed motor

Selected Features
TT_02
M_TYP_BRUSHLESS
XW_WIDE
AI_SELF_DRIVING
W_VOLTAGE_MONITOR

Table 6.1: Selected features by the customer

Feature / Condition	Price
PF_G1	230€
PF_G2	300€
OFF	80€
WB_257	10€
GC_HIGH	20€
XW_WIDE	30€
M_TYP_380	50€
M_TYP_540	70€
M_TYP_BRUSHLESS	130€
XD_OIL	50€
-XS_STD	70€
TW_ON_PROFILE	20€
BODY_LONG	20€
AK_NIMH_3000	20€
AK_LIPO_3300	50€
SV_6V_20K	50€
AI_SELF_DRIVING	1000€
AI_BREAK_ASSIST	80€
W_VOLTAGE_MONITOR	10€

Table 6.2: Sales price of features

(M_TYPE_380) over the more expensive brushless variant –which also requires a more expensive motor controller. Continuing these trade-offs, we arrive at the absolute cheapest option which is the TT_01 model with the standard 540 motor and XW_STD.

Out of the perspective of a customer, the ideal point would be to choose the brushless motor with the manual driving, an NIMH 3000mAh battery and the 8kg Servo at a price point of 520€. This choice is considering most customers choices, but at a reasonable price point.

A sales representative based on this Pareto Front can lead a negotiation with the customer by comparing the ideal configuration to the cheaper alternatives and discussing the trade-offs in the given application. Since AI_SELF_DRIVING is one of the configurations with a lot of prestige for the company and also a high profit margin, the goal of the sales representative would be to highlight the advantages of self-driving. But if the customer is not willing to pay the extra price, the suggested alternative would be the presented optional solution.

6.3 Best Fit vs. Delivery Time

The next experiment will be best fit vs. delivery time. Our goal is to offer the customer a deliverable product, that can be shipped in a timely manner. The delivery time is a

6. EXPERIMENTS AND EVALUATION

central aspect of the sales department and reducing it is a highly important objective for a company [SFH⁺18]. This can be done by streamlining the production and improving the logistics –and also by leading the customer towards a configuration, that can be shipped faster, because all the parts are already available in stock.

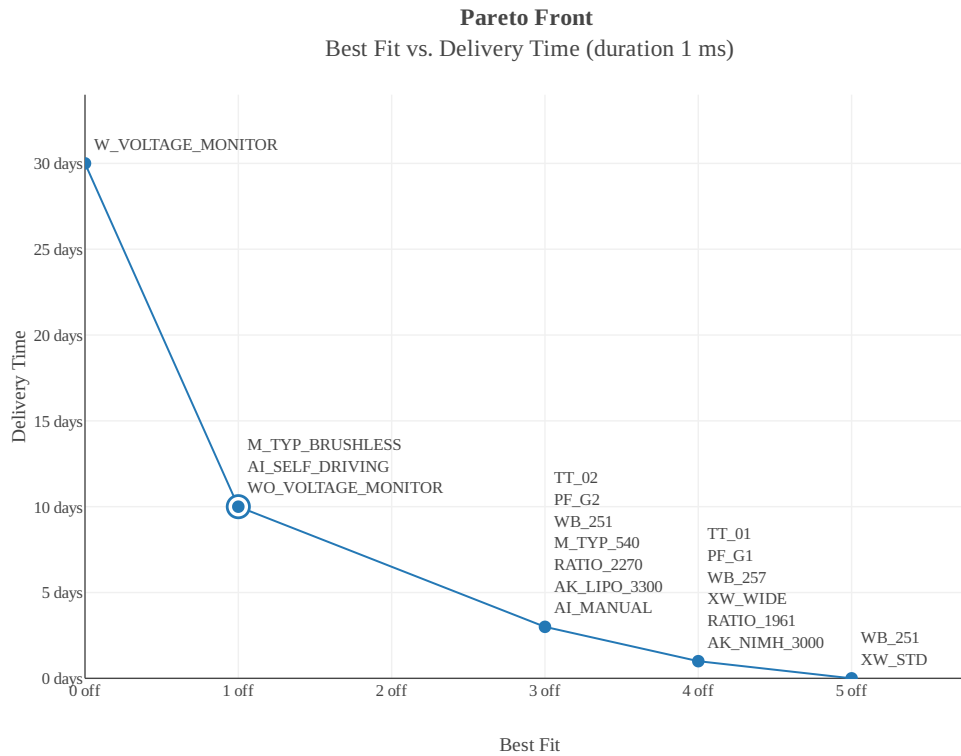


Figure 6.2: Pareto Front of best fit vs. delivery time in the TT-Platform

The customer choices in this experiment are the same as for the previous example (Table 6.1). In Figure 6.2, the trade-offs for the choices are displayed against an exemplary per-part delivery time. Currently, the parts needed for the voltage monitor are not in stock and have a back-order time of at least 30 days. Therefore, delivering the choice of W_VOLTAGE_MONITOR leads to 30 days in delivery. But, when trading off the voltage monitor, there is an option for a 10-day delivery with a brushless motor and manual driving. Trading off other choices, we arrive at a delivery time of 0 days –everything in stock–, but without considering any choices of the customer. The optimal configuration (optimal point) would be to drop the W_VOLTAGE_MONITOR and keeping the AI_SELF_DRIVING with the M_TYP_BRUSHLESS.

For the company to reach the given target of improved delivery time, this Pareto front makes the trade-offs visible. Based on it, a sales representative can lead the negotiation

with the customer to see how the change of the given configuration can lead to a better delivery time.

For highly configurable products with the amount of features exceeding hundreds of features, finding the best way to slightly change the choice, while improving the delivery time. Achieving this is close to impossible without intelligent tools, that have access to the documentation system and the current stock data.

6.4 Sales Price vs. Material Cost

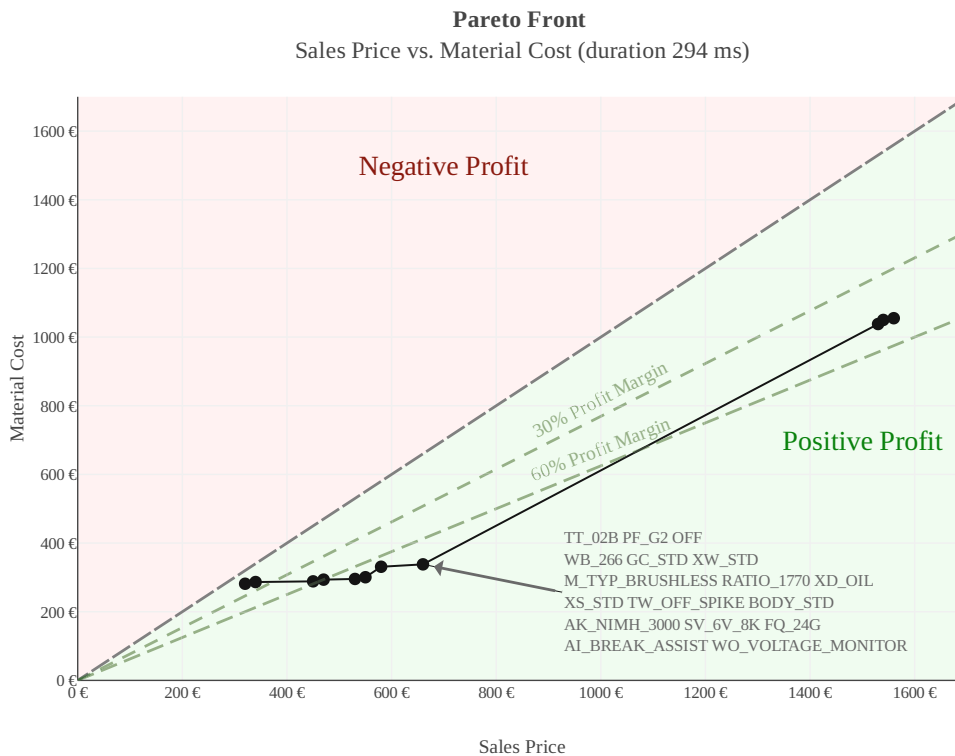


Figure 6.3: Pareto Front of sales price vs. material cost in the TT-Platform

When preparing the pricing for a product, a challenging task is deciding how certain features should be priced. Generally speaking, the price should cover all the costs for material and process plus a profit margin. Furthermore, customers are generally more willing to pay for extra add-on features, therefore we want to have increased profit for non-standard configurations. Designing a proper pricing model on a highly configurable product creates additional challenges, because we need to make sure it is profitable on all possible configurations.

In Figure 6.3 a Pareto Front is given with the objective to minimize the sales price and maximize the material cost. In other words, the solver aims to find the worst case product in terms of profit margin from the perspective of the company. The dotted gray line separates the negative profit area from the positive profit. Given the pricing in Table 6.2, the profit is positive for every configuration. Starting with the cheapest configurations, we see a profit margin of around 15%. The more special features are selected, the higher the profit margin will be, with around 45% margin for the full self-driving configurations.

The Pareto Front shows a convex behavior, which represents the fact that we incur extra profit for add-on features, and the customers' willingness to pay extra for add-on features is taken into account.

In a practical case of this experiment, this evaluation would happen during or towards the end of the technical development phase, where we have a good knowledge of the costs of each part. Given these material costs, the prices for the features can iteratively be adjusted to align the Pareto Front towards the desired profit margins and curvature.

6.5 Market Reach vs. Technical Solutions

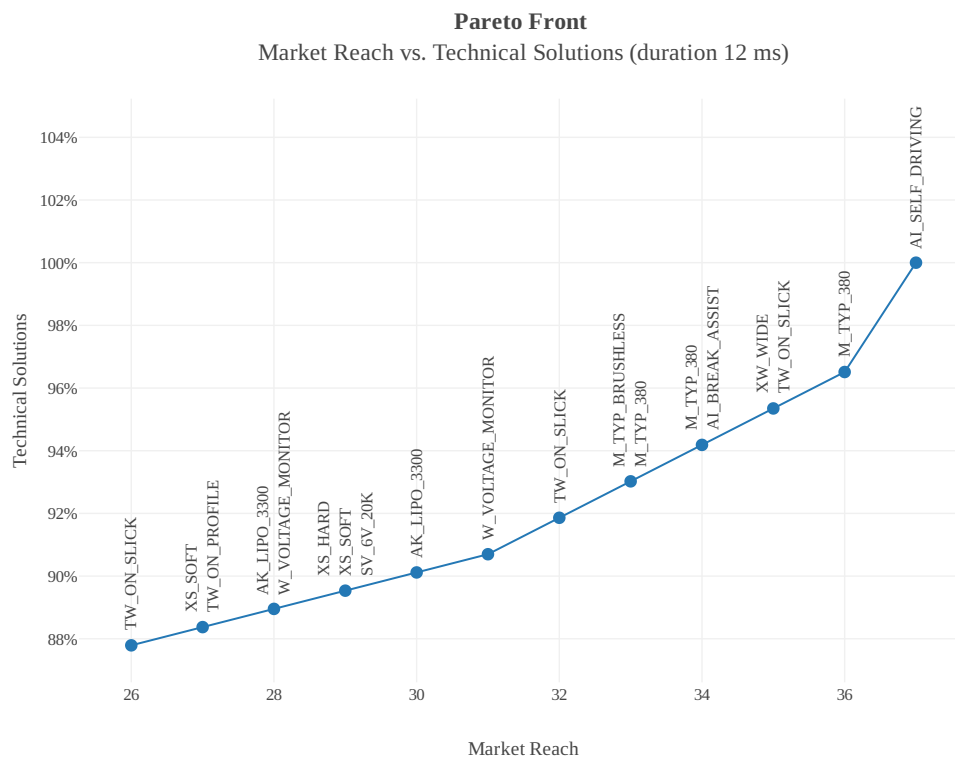


Figure 6.4: Pareto Front of market reach vs. technical solutions in the TT-Platform

In Figure 1.1 one of the goals of modularity is explained as the aim to offer the highest possible product variance while reducing the parts necessary to deliver the products to a minimum. These objectives can be encoded by MaxSAT-Solvers utilizing two conflicting objectives.

The first objective is the market reach objective which values each feature that is selectable with the weight of 1.

Definition 6.5.1. The optimization objective for market reach is defined as:

$$\phi_{\text{market reach}} = \{(\phi(c_i), 1) \mid c_i \in \mathcal{D}\}$$

We want to allow the solver to select several features at the same time to be true, such that the solver creates a mask of the features. For this we have to change the definition of our instance \mathcal{D} and replace the OneHot constraint with an at-least-one constraint. Now the solver can select several features of out a feature group. The base-type features in this case is not part of the optimization, since they each exhibit a huge amount of technical solutions that would overshadow the discussion about the complexity of features.

The second objective is the amount of technical solutions, that is needed. Because several features can be selected by the solver, we slightly have to change the usage conditions, since we do not want to have negations in the condition. To remove the negations, we can replace every negated code by the disjunction of the other codes in the given feature group. For example, if a condition ϕ contains $\neg \text{M_TYP_540}$ we can replace convert it to ϕ' by replacing $\neg \text{M_TYP_540}$ with $(\text{M_TYP_380} / \text{M_TYP_BRUSHLESS})$. If the solver decides to allow all motors, ϕ' will still be true. The correctness of this substitution can be argued by the OneHot constraint.

Definition 6.5.2. The optimization objective for the reduction of technical solutions is defined as:

$$\phi_{\text{technical solutions}} = \{(\phi'(A), 1) \mid A \in \mathcal{D}\}$$

The curve is slightly convex, which signifies, that there are complex features that yield a higher amount of technical solutions to be implemented. The minimal amount of market reach is at 24 features and 83% of the technical solutions. This is the minimum, dropping any technical solution would result in not buildable configurations.

On the high side of the market reach, we can see the `AI_SELF_DRIVING` feature which requires the highest amount of technical solutions. Identifying the amount of technical solutions for a given feature can be done using the exclusivity analysis as given in Section 3.2. Given the company wants to reduce the complexity of the product, the discussion traverses the curve starting at the most complex features on the right side, till a balance

is reached somewhere on the Pareto front. This balance incorporates the trade-offs the company is willing to take.

But for objectively deciding the trade-offs, a central question is, how big the market is behind a given feature. There might be features that are technically complex, but are ordered very often, and therefore would be economically viable. The next experiment will address the shortcomings of this experiment by encoding the customer interest and also the effective complexity by weighting the parts based on their technical complexity.

6.6 Market Fit vs. Weighted Technical Solutions

In Figure 6.5 the *customer interest* vs. *technical complexity* goal for a company is illustrated in a matrix with the dimensions of technical complexity, given in the amount of exclusive parts, and customer interest as the sales volume. Trivially, the obvious target would be to only sell products with low technical complexity and high customer interest, while not selling technically complex products with low customer interest.

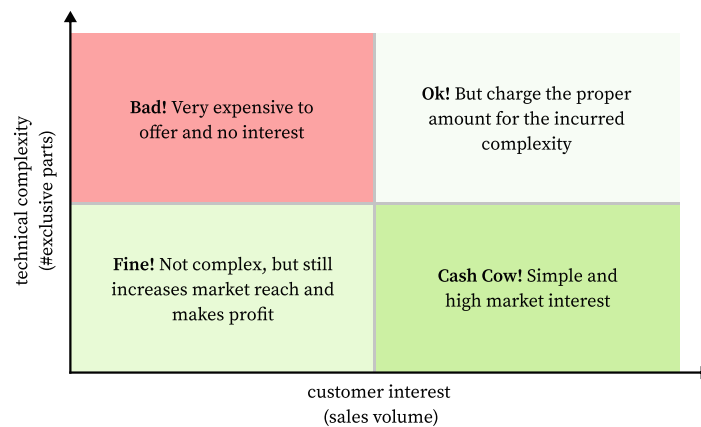


Figure 6.5: Identifying the target for a company's complexity vs. customer interest

The case of high customer interest and high technical complexity is more delicate. For this case, we need to make sure that we price the features such that we incur an adequate extra price for the complexity that covers the cost added by engineering, testing, logistics, management, etc. for these complex parts. This can be done by using the objectives from the previous experiment to create an adequate pricing model.

The other case of low complexity and low customer interest can be considered *fine*, because all necessary parts exist, we only need to compose them in a specific manner for a few customers. Given that we have properly documented the product in a FBD instance, creating this special variant poses no challenge. The only consideration is if it is possible to increase the price for these variants, because there might be less competition for these market segments.

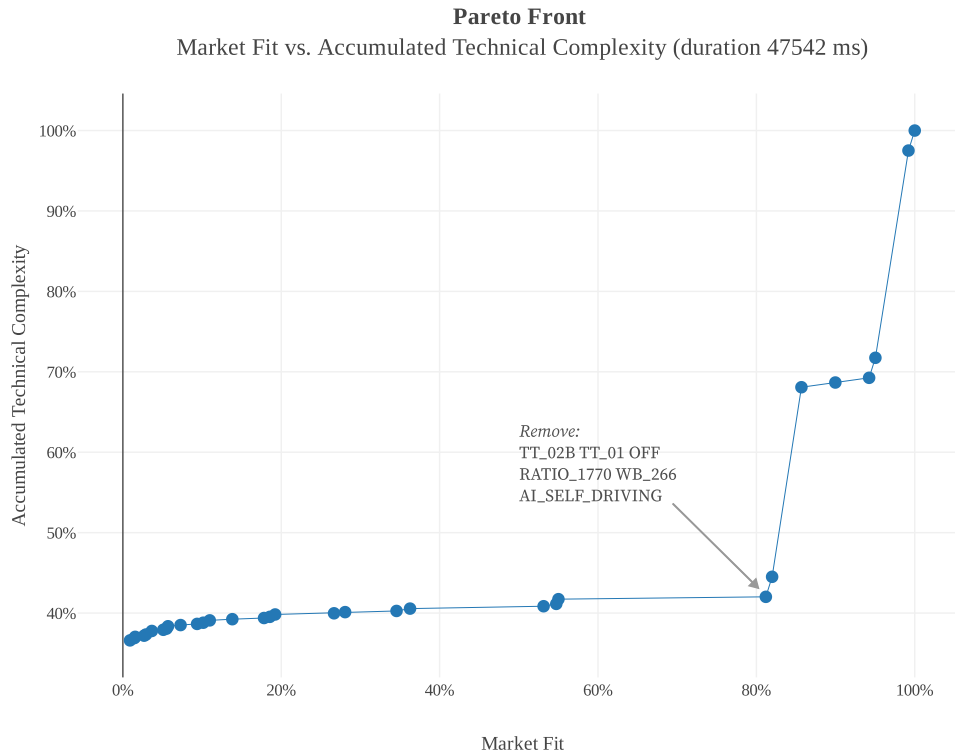


Figure 6.6: Pareto Front of market fit vs. accumulated technical complexity in the TT-Platform

For this experiment, instead of targeting special features, we encode complete customer configurations weighted by the sales volume. The technical solutions on the other hand will be weighted by the effort needed in the engineering and production phase. The author has created an index that considers the engineering and production effort for each part (e.g. a screw that can be bought off-shelf compared to a specially formed plastic part needs custom engineering and custom in-house processes).

The result of the evaluation is, that there is a very clear optima, at 80% market fit and only 40% accumulated technical complexity of the technical solutions. This optimum drops the following features of the portfolio:

This would only keep the TT_02 base-type and drop the TT_01 and TT_02B. In our fictional order history, the amount of people ordering not the standard base-type TT_02 is very low. Therefore, the optimizer takes the chance to drop the old base-type and the buggy variant. Such kind of distributions are very common in the industry.

For a company, phasing out the old TT_01 model for sure would be a good decision. The buggy variant (TT_02B) as well as the autonomous driving feature (AI_SELF_DRIVING)

Dropped Features
TT_02B
TT_01
OFF
RATIO_1770
WB_266
AI_SELF_DRIVING

Table 6.3: Features dropped for optimal point (80% market fit and 40% accumulated technical complexity)

are features that require a lot of technical complexity and are bought less often, because of the smaller market and the higher price point. A company in this case has to consider the business case for both of these variants and also, if the generated prestige is worth the technical complexity.

For each step along the Pareto Front from the optimal point toward 100% market fit, the consideration for the dropped features can be repeated, yet the penalty on the market side for removing the features is decreasing. A company can decide to target any option along this way considering its market side product strategy. By reducing the technical complexity, the price of the product can be lowered, and the more competitive pricing can, in-return, increase the market share and profit –making up for the lost customers in the first place–. This discussion sheds light onto one essential challenge of a modular product platform, which is their tendency to become more complex over time while not necessarily increasing the market fit.

Performance Evaluation

In terms of performance, this experiment requires a lot more computation time compared to the previous one. The computation time is around 77 seconds for P-MINIMAL and 107 for the BIOPTSAT algorithm (see Table 6.4).

P-MINIMAL	47.542 seconds
BIOPTSAT	101.410 seconds

Table 6.4: Comparing the performance of different multi-objective solvers

P-MINIMAL outperforms BIOPTSAT by taking less than half the wall-clock computation time. The most probable explanation in this case is that the many very close non-dominated solutions along the Pareto Front lead the BIOPTSAT to take more time in switching between the different one-directional optimization mode compared to the more direct approach of P-MINIMAL.

For both algorithms, the solver first executed the core boosting routine [JBJ24] to preprocess and reformulate the instance.

6.7 Generalizability of the Experiments

The TT-Platform has 39 features and around 170 parts. In the industry, many products are equivalently complex, but there are also a lot that exhibit a way higher configurability. Based on the author's experience, many highly configurable product lines exist, with over a thousand features and several hundred thousand parts. The truck industry, for example, is particularly known for its complexity due to many applications with specific requirements –from long-haul tractors to construction site concrete mixers–.

Scaling the experiments to products of high complexity will pose a challenge. Still, experiments Best Fit vs. Sales Price and Best Fit vs. Delivery Time ran in a few milliseconds in the TT-Platform example and therefore should be feasible to execute in reasonable time in a larger-scale scenario. Also, adequate levels of abstraction (dropping parts that are irrelevant, focusing on only a certain module, concentrating parts, ...) may reduce the complexity and increase the speed. The author has conducted validation tests in a highly complex product that confirm the feasibility.

The experiment of Sales Price vs. Material Cost is a bit more compute-intensive than the previous one, but still it executed in around 300 milliseconds. Scaling this experiment to more complex products is expected to be feasible as well.

In the experiments for optimizing the product offering, the computation time of the first (Market Reach vs. Technical Solutions) is sufficiently low (12 milliseconds) such that the execution in a highly complex product is feasible. On the contrary, the second experiment (Market Fit vs. Weighted Technical Solutions) is infeasible in a more complex product due to the already high computation time (47 seconds) in the TT-Platform. To reduce the computation time, the weighting of the customer interest can also be achieved differently. Similar to the first example, the individual features can be optimized, while the sales volumes of each individual feature is defined as the weight. This simplification should yield an acceptable loss of accuracy while drastically reducing the necessary computation time.

Furthermore, approximation techniques, like simulated annealing or greedy algorithms, may yield good enough results while keeping the computation times reasonable.

CHAPTER 7

Summary & Outcome

The goal of this thesis was to investigate the domain of FBD and to explore the application of the MaxSAT-Solver in the domain. This application has been evaluated in several experiments that yield insights into an example product platform and its configurability.

7.1 Conclusio

The conclusion is that there are several use-cases in which MaxSAT-Solvers are an effective tool to optimize product platforms and individual products towards given objectives. The resulting Pareto Fronts offer valuable insights into the trade-offs between the different objectives. The application is manifold from product configurators that suggest trade-offs to the customer, for choosing a cheaper or faster deliverable product, to the analysis tasks of discussing the pricing of the product and also of analyzing the product's complexity.

Furthermore, the suggested framework is highly adaptable, since new objectives can be defined and processed using the reduction pipeline. Combined with the *EFS Modularity Suite*, the objectives can be loaded directly from the information in the products' documentation system.

As an extension of the conclusions about applying multi-objective MaxSAT, the proposed methods in Section Exclusivity and Module Interface Analysis offer viable capabilities. These can profoundly enhance the way FBD-Systems are analyzed and how the discussion about the module interfaces can systematically improve the way that we think about modularity and the composability of highly configurable products.

7.2 Outlook & Open Questions

While this thesis explored several possible applications, there are many more waiting in the world of product manufacturing. MaxSAT-Solvers have the potential to address

7. SUMMARY & OUTCOME

a wide range of objectives in this context. The open question is how effective the existing solvers are and how the algorithms scale to large scale platforms with extremely high configurability. Furthermore, a central question is how practically the discussion of the resulting trade-offs can be integrated into organizational processes and different subsidiaries of a manufacturing company. The existing tool-set around the *EFS Modularity Suite* poses a capable platform to deliver these evaluations and allow discussions to take place in a company around a shared documentation system.

APPENDIX **A**

Module Interface Graph

In Figure A.1 an interface graph over the complete rc-car is given with the following Interfaces (refined version of [WB24]):

- G** Geometrical interface
- F** Functional Interfaces
- D** Data Interfaces
- P** Power Interfaces
- M** Mechanical Interfaces
- E** Environmental Interfaces

A. MODULE INTERFACE GRAPH

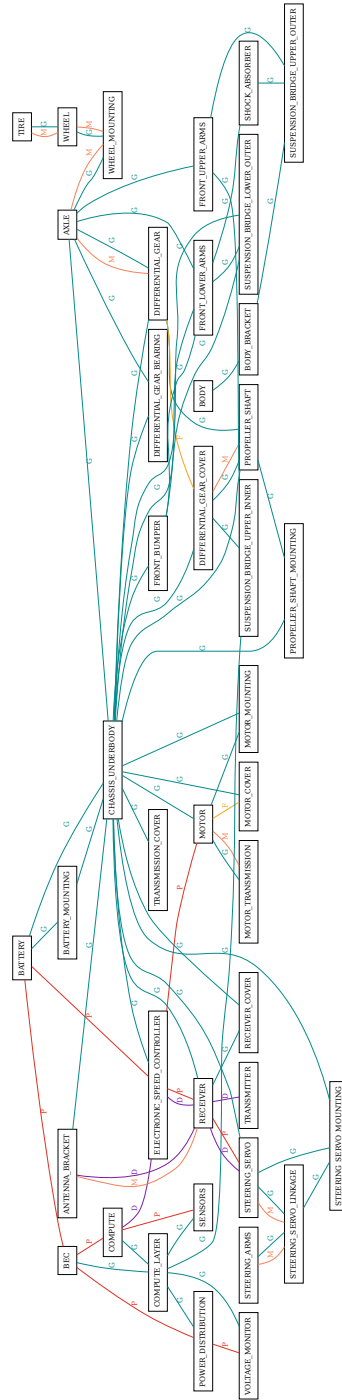


Figure A.1: Exemplary Module Interface Graph for the complete rc-car (refined version of [WB24])

List of Figures

1.1	Schematic explanation of the goal of modularity and the different interests in a company that are involved (freely adapted from [Man])	2
1.2	Apply Modular Function Deployment (MFD) to create product architectures that meet customer needs, strategic targets and functional requirements. From https://www.modularmanagement.com/en/mfd accessed on 14.02.2024 15:07	6
1.3	Example feature diagram for a graph product-line. Taken from [KKS ⁺ 23]	8
2.1	Euler diagram for P, NP, NP-complete, and NP-hard set of problems. From https://en.wikipedia.org/wiki/P_versus_NP_problem accessed on 7.3.2024 16:06	13
2.2	Platform Management [WB23]	18
2.3	Lineup of the Tamiya TT-Platform [WB24]	19
2.4	Exemplary description of a Feature as a dimension of variation [WB24] (Illustrations taken from the TT-02 Chassis Manual [TAM13])	21
2.5	Exemplary description of a base-type as the core technical variations. <i>S</i> stands for <i>standard</i> and is the default selection while <i>O</i> are <i>optional</i> features that can be selected by the customer	24
2.6	Exemplary description for the assignment of technical solutions on base-types	25
2.7	Excerpt of the SuperBoM and assignment of technical solutions based on usage conditions	26
2.8	Drawing of how the SuperBoM assigns technical solutions and parts in a hierarchical order to possible configurations [WB23]	27
3.1	Illustration of Consistency	34
3.2	Illustration of Completeness	34
3.3	Illustration of Distinctiveness	35
3.4	Explanation of satisfiable and exclusive SEs based on set-theory	36
3.5	Illustration of the interfaces for the example Interface Graph	37
3.6	Exemplary Module Interface Boundary Graph of Figure 3.5	38
3.7	The 6 steps of the <i>Module Interface Analysis</i>	39
3.8	Overview over the EFS Modularity Suite	41
4.1	Pipeline for reducing code conditions to the SAT-Solver	44
		91

4.2	Illustrative example of an AST	44
4.3	Lexing converts a given input string into its atoms	45
4.4	The two possibilities of interpreting a condition are used to show why precedence is needed	47
4.5	Pratt Parsing algorithm overview. From https://martin.janiczek.cz/2023/07/03/demystifying-pratt-parsers.html accessed on 13.3.2024 at 16:49	48
4.6	Tseitin concept explanation	49
4.7	Comparing the effects of the transformation optimizations in terms of Execution Duration. <i>PG</i> stands for Plaisted-Greenbaum optimization and <i>ST</i> stands for Subformulation-Tracking	51
4.8	Comparing the effects of the transformation optimizations in terms of clause and variable count. <i>PG</i> stands for Plaisted-Greenbaum optimization and <i>ST</i> stands for Subformulation-Tracking	52
5.1	Detailed Results on the weighted exact track of the 2023 MaxSAT competition [MBJN23]	56
5.2	Detailed results on the weighted track of the 2023 MaxSAT competition with ILP-Solvers [MBJN23]	59
5.3	Illustration of the Pareto Front with the contradictory optimization objectives o_1 and o_2 (Graphic adapted from [BF17])	63
5.4	An exemplary Pareto Front of the electric passenger car market until the 2nd December 2020 (Dataset [HK21])	64
5.5	Regions pruned by blocking clauses (adapted from [SBTLB17])	66
5.6	Search Trace of BiOPTSAT [JBNI24]	68
5.7	Comparison of search traces for P-MINIMAL, BiOPTSAT and LOWERBOUND, [JBNI24]	69
6.1	Pareto Front of best fit vs. sales price in the TT-Platform	76
6.2	Pareto Front of best fit vs. delivery time in the TT-Platform	78
6.3	Pareto Front of sales price vs. material cost in the TT-Platform	79
6.4	Pareto Front of market reach vs. technical solutions in the TT-Platform	80
6.5	Identifying the target for a company's complexity vs. customer interest	82
6.6	Pareto Front of market fit vs. accumulated technical complexity in the TT-Platform	83
A.1	Exemplary Module Interface Graph for the complete rc-car (refined version of [WB24])	90

List of Tables

2.2	Combination Space	21
2.1	Feature List Table	22
2.3	Example list of base-type to feature restrictions	28
4.1	Operator table	46
5.1	Example Categories for the part availability	72
6.1	Selected features by the customer	77
6.2	Sales price of features	77
6.3	Features dropped for optimal point (80% market fit and 40% accumulated technical complexity)	84
6.4	Comparing the performance of different multi-objective solvers	84

List of Algorithms

2.1	DPLL-Algorithm [OC99]	15
5.1	Johnson’s Greedy Algorithm [Stü18]	61
5.2	Simulated Annealing Algorithm [NJ10]	62
5.3	P -MINIMAL Algorithm (adapted from [SBTLB17] ¹)	66
5.4	FINDP-MINIMALMODEL	67
5.5	BIOPTSAT: MaxSAT-based bi-objective optimization [JB NJ22]	69

Glossary

Battery Electric Vehicle (BEV) An electric vehicle that utilizes a Battery as the primary power source. 64

Bill of Material (BoM) A list of all the parts that are in a product given in a hierarchical order [Tea02]. 24, 25, 27

Configuration Lifecycle Management (CLM) Configuration Lifecycle Management describes the management of the all the product configurations over the complete lifecycle from idea, engineering till after-sales [MRH18]. 1

Configure Price Quote (CPQ) CPQ stands for *Configure Price Quote* and is umbrella term that describes processes that aim at improving the customer's journey from Configuration, Pricing till the Quotation [JAJK20]. 3

End Of Production (EOP) EOP is the planned end of the production. 18

Modular Function Deployment (MFD) Modular Function Deployment (MFD) is a recognized method for conceptualizing modular products by taking a holistic view of which functions modules need to implement and what requirements (functional and non-functional) they need to satisfy. The aim of the method is to produce a well-designed modular product that improves the handling in the whole lifecycle from production till after sales. [EE99]. 3, 6, 38

Negative Normal Form (NNF) In the NNF, only *And* and *Or* are operators. Negations are only applied on variables and not on operators. 50

Original Equipment Manufacturer (OEM) a company that makes parts and products for other companies which sell them under their own name or use them in their own products [WC11]. 18, 40

Structural Element (SE) A Structural Element is a functional part that is used to build a product. It can either be a single part or a subassembly. 26, 30, 33–36, 91

Structural Family (SF) A Structural Family defines a group of Structural Elements. This grouping is done on a functional level, such that all elements serve a similar function. 26, 34

Start Of Production (SOP) The SOP marks the beginning of the mass production of the product. 18

SuperBoM (SuperBoM) A superset of all the BoMs of all possible product variations [WB24]. 25, 33

Acronyms

- AST** Abstract Syntax Tree. 44, 46, 50, 92
- CAD** Computer Aided Design. 25
- CDCL** Conflict-Driven Clause Learning. 16
- CNF** Conjunctive Normal Form. 14, 43, 49, 57
- CSP** Constraint Satisfaction Problem. 65
- DPLL** Davis-Putnam-Logemann-Loveland. 15, 16
- ERP** Enterprise Resource Planning. 25
- FBD** Feature-based Documentation. 2, 7–10, 17–19, 23, 29, 31, 33, 37, 38, 41, 70–72, 75, 82, 87
- FDM** Feature Driver Matrix. 40
- ILP** Integer Linear Programming. 57–59, 92
- LCA** Life Cycle Assessment. 74
- MES** Machine Execution System. 72
- MIB** Module Interface Boundaries. 38–40
- MODOP** Multi-Objective Discrete Optimization Problem. 65
- PLM** Product Lifecycle Management. 25

Bibliography

- [ABZ06] Adi Avidor, Ido Berkovitch, and Uri Zwick. Improved approximation algorithms for MAX NAE-SAT and MAX SAT. In Thomas Erlebach and Giuseppe Persinao, editors, *Approximation and Online Algorithms*, pages 27–40, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [ACF⁺15] Laura Ausberg, Andreas Ciroth, Silke Feifel, Juliane Franze, Kaltschmitt Martin, Inga Klemmayer, Kirsten Meyer, Peter Saling, Liselotte Schebek, Jana Weinberg, and Christina Wulf. *Umweltbewertung für Ingenieure: Methoden und Verfahren*, chapter Lebenszyklusanalysen, pages 203–314. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
- [AG13] Carlos Ansótegui and Joel Gabàs. Solving (Weighted) Partial MaxSAT with ILP. In *Integration of AI and OR Techniques in Constraint Programming*, 2013.
- [ASW⁺11] Sven Apel, Hendrik Speidel, Philipp Wendler, Alexander von Rhein, and Dirk Beyer. Detection of feature interactions using feature-aware verification. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering, ASE '11*, page 372–375, USA, 2011. IEEE Computer Society.
- [BF17] Facundo Bre and Víctor Fachinotti. A computational multi-objective optimization method to improve energy efficiency and thermal comfort in dwellings. *Energy and Buildings*, 154, 08 2017.
- [Bha16] Swapnil Bhartiya. Linux is the largest software development project on the planet: Greg Kroah-Hartman, 2016. <https://www.cio.com/article/241071/linux-is-the-largest-software-development-project-on-the-planet-greg-kroah-hartman.html> accessed on 21.10.2024 at 10:00.
- [Bru20] Noah Bruns. Application of SAT-Solvers in Feature-based Documentation Systems. Bachelor’s thesis, Technical University Vienna, 2020.

- [CFZ97] Jianer Chen, D.K. Friesen, and Hao Zheng. Tight bound on johnson's algorithm for max-sat. In *Proceedings of Computational Complexity. Twelfth Annual IEEE Conference*, pages 274–281, 1997.
- [CLL⁺22] Jordi Coll, Shuolin Li, Chu-Min Li, Felip Manyà, Djamal Habet, and Kun He. MaxCDCL and WMaxCDCL in MaxSAT Evaluation 2022. In *MaxSAT Evaluation 2022: Solver and Benchmark Descriptions*, Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, 2022.
- [CLL⁺23] Jordi Coll, Shuolin Li, Chu-Min Li, Felip Manyà, Djamal Habet, Mohamed Sami Cherif, and Kun He. WMaxCDCL in MaxSAT Evaluation 2023. In *MaxSAT Evaluation 2023: Solver and Benchmark Descriptions*, Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, 2023.
- [EE99] Anna Ericson and Gunnar Erixon. *Controlling Design Variants: Modular Product Platforms*. Society of Manufacturing Engineers, 1999.
- [FBF⁺21] Patrick Franz, Thorsten Berger, Ibrahim Fayaz, Sarah Nadi, and Evgeny Groshev. Configfix: Interactive configuration conflict resolution for the linux kernel. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 91–100, 2021.
- [FBGR13] Alexander Felfernig, David Benavides, José Galindo, and Florian Reinfrank. Towards anomaly explanation in feature models. In *CEUR Workshop Proceedings*, volume 1128, pages 117–124, 08 2013.
- [FBS19] Katalin Fazekas, Armin Biere, and Christoph Scholl. Incremental inprocessing in sat solving. In *International Conference on Theory and Applications of Satisfiability Testing*, 2019.
- [For09] Lance Fortnow. The status of the P versus NP problem. *Communications of the ACM*, 52(9):78–86, September 2009.
- [GSK98] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, page 431–437, USA, 1998. American Association for Artificial Intelligence.
- [GW94] Michel X. Goemans and David P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7(4):656–666, 1994.

- [HK21] Bartłomiej Hadasik and Jakub Kubiczek. Dataset of electric passenger cars with their specifications. Mendeley Data, 2021. <https://data.mendeley.com/datasets/tb9yrptydn/2> accessed on 19.10.2024 at 11:03.
- [IMMS19] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. Rc2: an efficient maxsat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 11:53–64, 09 2019.
- [JAJK20] Michelle Jordan, Gunnar Auth, Oliver Jokisch, and Jens-Uwe Kühl. Knowledge-based systems for the Configure Price Quote (CPQ) process - a case study in the IT solution business. *Online Journal of Applied Knowledge Management*, 8:17–30, 09 2020.
- [Jan08] Mikoláš Janota. Do SAT solvers make good configurators? In *Software Product Lines Conference*, pages 191–195, 01 2008.
- [Jan10] Mikoláš Janota. *SAT Solving in Interactive Configuration*. PhD thesis, University College Dublin, 2010.
- [Jan23] Martin Janiczek. Demystifying pratt parsers. Blog Post, <https://martin.janiczek.cz/2023/07/03/demystifying-pratt-parsers.html> accessed on 19.3.2024 at 14:15, 2023.
- [JBJ24] Christoph Jabs, Jeremias Berg, and Matti Järvisalo. Core Boosting in SAT-Based Multi-objective Optimization. In Bistra Dilkina, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research—21st International Conference, CPAIOR 2024, Uppsala, Sweden, May 28–31, 2024, Proceedings, Part II*, volume 14743 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2024.
- [JBNJ22] Christoph Jabs, Jeremias Berg, Andreas Niskanen, and Matti Järvisalo. MaxSAT-Based Bi-Objective Boolean Optimization. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:23, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [JBNJ24] Christoph Jabs, Jeremias Berg, Andreas Niskanen, and Matti Järvisalo. From Single-Objective to Bi-Objective Maximum Satisfiability Solving. *Journal of Artificial Intelligence Research*, 80, August 2024.
- [Joh74] David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, dec 1974.
- [KIMS21] Stepan Kochemazov, Alexey Ignatiev, and Joao Marques-Silva. Assessing Progress in SAT Solvers Through the Lens of Incremental SAT. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing – SAT 2021*, pages 280–298, Cham, 2021. Springer International Publishing.

- [KKS⁺23] Elias Kuitert, Sebastian Krieter, Chico Sundermann, Thomas Thüm, and Gunter Saake. Tseitin or not Tseitin? The Impact of CNF Transformations on Feature-Model Analyses. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, ASE '22*, New York, NY, USA, 2023. Association for Computing Machinery.
- [Kla20] Alex Kladov. Simple but Powerful Pratt Parsing. Blog Post, <https://matklad.github.io/2020/04/13/simple-but-powerful-pratt-parsing.html> accessed on 19.3.2024 at 17:46, 2020.
- [KST⁺16] Sebastian Krieter, Reimar Schröter, Thomas Thüm, Wolfram Fenske, and Gunter Saake. Comparing algorithms for efficient feature-model slicing. In *the 20th International Systems and Software Product Line Conference*, pages 60–64, 09 2016.
- [LXC⁺21a] Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Combining Clause Learning and Branch and Bound for MaxSAT. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, Montpellier (Online), Best Paper Award, France, 2021.
- [LXC⁺21b] Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Boosting branch-and-bound MaxSAT solvers with clause learning. *AI Communications*, 35:1–21, 12 2021.
- [Man] Modular Management. What is palma? Youtube https://www.youtube.com/watch?v=A_7O3LNIzpM accessed on 18.9.2023, 12:50.
- [Mar] Tobias Martin. All You Need to Know About Modularization. webpage. <https://www.modularmanagement.com/blog/all-you-need-to-know-about-modularization> accessed on 18.9.2023, 12:00.
- [MBJN23] Ruben Martins, Jeremias Berg, Matti Järvisalo, and Andreas Niskanen. Maxsat evaluation 2023, 2023. <https://maxsat-evaluations.github.io/> accessed on 12.6.2024, 16:30.
- [MH23] Alexander Hoeppe Marc Halpern, Sudip Pattanayak. Configuration life cycle management. Gartner Report Top Strategic Technology Trends in Asset-Intensive Manufacturing for 2023, 2023.
- [MMZ⁺01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th Annual Design Automation Conference, DAC '01*, page 530–535, New York, NY, USA, 2001. Association for Computing Machinery.

- [MRH18] Anna Myrodia, Thomas Randrup, and Lars Hvam. Configuration lifecycle management – an assessment of the benefits based on maturity. In *Proceedings of the 20th International Configuration Workshop*, CEUR Workshop Proceedings, pages 119–124. University of Hamburg, 2018.
- [MSLM21] Joao Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning sat solvers. In *Handbook of Satisfiability*, 2021.
- [Muf99] Moreno Muffatto. Introducing a platform strategy in product development. *International Journal of Production Economics*, 60-61:145–153, 1999.
- [NJ10] Alexander G. Nikolaev and Sheldon H. Jacobson. *Handbook of Metaheuristics*, chapter Simulated Annealing, pages 1–39. Springer, 2010.
- [NNKB21] Van-Hau Nguyen, Van-Quyet Nguyen, Kyungbaek Kim, and Pedro Barahona. Empirical Study on SAT-Encodings of the At-Most-One Constraint. In *The 9th International Conference on Smart Media and Applications, SMA 2020*, page 470–475, New York, NY, USA, 2021. Association for Computing Machinery.
- [NR12] Alexander Nadel and Vadim Ryvchin. Efficient sat solving under assumptions. In *International Conference on Theory and Applications of Satisfiability Testing*, 2012.
- [Nys11] Bob Nystrom. Pratt parsers: Expression parsing made easy. Blog Post, <https://journal.stuffwithstuff.com/2011/03/19/pratt-parsers-expression-parsing-made-easy/> accessed on 19.3.2024 at 14:20, 2011.
- [OC99] Ming Ouyang and Vasek Chvatal. *Implementations of the DPLL algorithm*. PhD thesis, Rutgers University, USA, 1999.
- [PG86] David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.
- [Pra73] Vaughan R. Pratt. Top down operator precedence. In *Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '73, pages 41–51, New York, NY, USA, 1973. Association for Computing Machinery.
- [Rus12] Andrew L. Russell. Modularity: An interdisciplinary history of an ordering concept. *Information & Culture: A Journal of History*, 47:257 – 287, 2012.
- [SBTLB17] Takehide Soh, Mutsunori Banbara, Naoyuki Tamura, and Daniel Le Berre. Solving multiobjective discrete optimization problems with propositional minimal model generation. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming*, pages 596–614, Cham, 2017. Springer International Publishing.

- [SFH⁺18] Sara Shafiee, Alexander Felfernig, Lars Hvam, Poorang Piroozfar, and Cipriano Forza. Cost benefit analysis in product configuration systems. In Alexander Felfernig, Juha Tiihonen, Lothar Hotz, and Martin Stettinger, editors, *20th International Workshop on Configuration 2018 (ConfWS'18)*, volume 2220 of *CEUR Workshop Proceedings*, pages 37–40. CEUR-WS, 09 2018. Configuration Workshop 2018, ConfWS 2018 ; Conference date: 27-09-2018 Through 27-09-2018.
- [SKT⁺16] Reimar Schröter, Sebastian Krieter, Thomas Thüm, Fabian Benduhn, and Gunter Saake. Feature-model interfaces: the highway to compositional analyses of highly-configurable systems. In *Proceedings of the 38th International Conference on Software Engineering, ICSE '16*, page 667–678, New York, NY, USA, 2016. Association for Computing Machinery.
- [Spe93] William M. Spears. Simulated Annealing for Hard Satisfiability Problems. *Cliques, Coloring, and Satisfiability*, 26:533–558, 10 1993.
- [SS97] João P. Marques Silva and Karem A. Sakallah. GRASP — a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD '96*, page 220–227, USA, 1997. IEEE Computer Society.
- [Stü18] Herwig Stütz. Algorithms for MAX-SAT. Master's thesis, Graz University of Technology, 2018.
- [TAM13] TAMIYA INC., 3-7 Ondawara, Suruga-Ku, Shizuoka 422-8610 Japan. *TT-02 Chassis*, 11053610 edition, 2013. Available at <https://www.tamiya.com/cms/english/rc/rcmanual/tt02.pdf> accessed on 19.10.2024, 11:30.
- [Tea02] The NEMI Perfect BoM Team. In search of the perfect bill of materials (bom). Technical report, National Electronics Manufacturing Initiative, Inc., 2002.
- [TSAH12] Thomas Thüm, Ina Schaefer, Sven Apel, and Martin Hentschel. Family-based deductive verification of software product lines. *SIGPLAN Not.*, 48(3):11–20, sep 2012.
- [Tse83] G. S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.
- [WB23] Stefan Willminger and Noah Bruns. Feature Based Documentation Intro. PowerPoint slides, 2023.
- [WB24] Stefan Willminger and Noah Bruns. EFS Business Consultancy Modularity Experience WS. PowerPoint slides, 2024.
- [WC11] H. Waterhouse and R. Combley. *Cambridge Business English Dictionary*. Cambridge University Press, 2011.

- [Yan94] M. Yannakakis. On the approximation of maximum satisfiability. *Journal of Algorithms*, 17(3):475–502, 1994.