

DISS. ETH NO. 22878

**A METHODOLOGY FOR SUPPORTING DESIGN GRAMMAR DEVELOPMENT AND
APPLICATION IN COMPUTATIONAL DESIGN SYNTHESIS**

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

CORINNA KÖNIGSEDER

Dipl.-Ing. Univ., Technische Universität München

born on 10.08.1985

citizen of Germany

accepted on the recommendation of

Prof. Dr. Kristina Shea

Prof. Dr. Jonathan Cagan

2015

Abstract

In the last decades researchers have developed various methods enabling computers to solve design synthesis problems using engineering design grammars. Successfully applying grammar-based Computational Design Synthesis (CDS) methods is challenging for several reasons. Problem-specific knowledge has to be represented formally through models describing designs and grammar rules that define design transformations. For a fully automated CDS process, evaluation routines have to be provided to enable the comparison of generated designs and a search algorithm is required to guide the synthesis process. The success of CDS methods usually depends on both the development of a suitable grammar and the selection and tuning of an appropriate search algorithm since the two are strongly coupled. Nowadays, there is limited methodological support for human designers during grammar development and selection and tuning of a search algorithm for CDS.

This thesis investigates a methodology to support human designers in the CDS process. The focus is on supporting grammar development and application through automatically generating and analyzing data to provide the designer with information on grammar rules, search algorithms and their interrelations. This can increase the understanding of the CDS process which in turn can give rise to improved grammars and beneficial combinations of grammar rules and search algorithms that lead to better-quality synthesis results. As a long term goal, this can help to bring the benefits of CDS methods to industrial applications.

The methodology consists of four methods that allow human designers to systematically analyze the CDS process. First, the Grammar Rule Analysis Method analyzes developed rules and the influences they have on synthesized designs considering objectives and design characteristics. Second, the Network-based Rule Analysis Method supports the analysis of application conditions of rules and the analysis of sequences of rule applications, e.g., to identify beneficial or counterproductive rule sequences. Third, the Relation Visualization method enables the analysis of both the grammar itself and how search algorithms explore and exploit the design space. Finally, the Search Strategy Comparison Method supports human designers in selecting a strategy that decides whether to apply topologic or parametric rules throughout the synthesis process. For each method, a process to generate data, analyze it and provide visualizations for the human designer is defined inspired by research in visual analytics. Information on interactions among grammar rules, rule sequences, changes in objectives and design characteristics, and the search process are presented to the human designer. A software prototype is developed to evaluate the methods and show their potential on different case studies. The presented methodology is integrated in the process of grammar rule development and contributes to supporting grammar development, selection of a search algorithm and refinement of the search process.

The results of the case studies show not only the potential of the methodology to support the human designer in grammar development and application, but also to increase the understanding of links between problem representation, search and the explored design space.

Zusammenfassung

In den letzten Jahrzehnten wurden verschiedene Methoden erforscht, welche Computer dazu befähigen mit Hilfe von Grammatiken Entwicklungsaufgaben im Ingenieursbereich zu bewältigen. Methoden der computerbasierten Synthese (CDS, engl. *Computational Design Synthesis*) erfolgreich einzusetzen ist aus mehreren Gründen eine Herausforderung. Problem-spezifisches Fachwissen muss formal beschrieben werden. Dies geschieht mittels Modellen, die Produkte darstellen, und mit Hilfe von Grammatikregeln, welche mögliche Umwandlungen der Produkte beschreiben. Um den CDS-Prozess vollständig zu automatisieren, sind einerseits Bewertungsmöglichkeiten erforderlich, um die generierten Produkte miteinander vergleichen zu können, andererseits Suchalgorithmen, die den Prozess der Produktsynthese steuern. Der Erfolg von CDS-Methoden hängt meist sowohl von der Entwicklung einer geeigneten Grammatik, als auch von der Auswahl und Feinabstimmung eines passenden Suchalgorithmus ab, da sich beide Faktoren gegenseitig stark beeinflussen. Derzeit existiert kaum methodische Unterstützung für Entwickler während der Grammatikentwicklung und der Auswahl und Einstellung von Suchalgorithmen für CDS.

Die vorliegende Dissertation erforscht eine Methodik, um den menschlichen Entwickler im grammatik-basierten CDS-Prozess zu unterstützen. Der Fokus liegt dabei darauf, den Entwickler durch Bereitstellung von Informationen bei der Entwicklung und Anwendung von Grammatikregeln zu unterstützen. Diese Informationen können das Verständnis der verschiedenen Komponenten im CDS-Prozess erhöhen und dadurch verbesserte Grammatiken und erfolgversprechende Kombinationen von Grammatikregeln und Suchalgorithmen zur Folge haben, die wiederum hochwertige Entwicklungsergebnisse erzeugen. Als langfristiges Ziel kann dies dazu führen, dass die Vorteile von CDS-Methoden auch im industriellen Umfeld genutzt werden können.

Die Methodik besteht aus vier Methoden die es dem menschlichen Entwickler ermöglichen, den CDS-Prozess systematisch zu analysieren. Erstens analysiert die Grammatikregelanalyse-Methode (*Grammar Rule Analysis Method*) systematisch welchen Einfluss die entwickelten Regeln auf die Zielgrößen und Merkmale der generierten Produkte haben. Zweitens ermöglicht die netzwerk-basierte Regelanalyse-Methode (*Network-based Rule Analysis Method*) eine Untersuchung der Anwendungsvoraussetzungen von Regeln und die Analyse von Regelsequenzen, um zum Beispiel förderliche oder hinderliche Regelsequenzen zu identifizieren. Drittens ermöglicht es die Zusammenhangvisualisierungs-Methode (*Relation Visualization Method*) einerseits die Grammatik zu analysieren und andererseits darzustellen wie Suchalgorithmen den Lösungsraum erkunden und ausschöpfen. Schlussendlich unterstützt die Suchstrategievergleichs-Methode (*Search Strategy Comparison Method*) den Entwickler dabei, eine geeignete Strategie auszuwählen die während des Syntheseprozesses entscheidet ob topologische und parametrische Regeln angewandt werden soll. Inspiriert von Forschung im Gebiet der visuellen Analyse ist für jede Methode ein Prozess definiert, um Daten zu erzeugen, zu analysieren und dem Entwickler Visualisierungen bereitzustellen.

Informationen zum Zusammenspiel von Grammatikregeln, Regelsequenzen, Änderungen der Zielgrößen oder Produktmerkmale und zum Suchvorgang werden dem Entwickler zur Verfügung gestellt. Ein Softwareprototyp wurde entwickelt, um die Methoden zu evaluieren und demonstriert deren Potenzial an Hand verschiedener Fallstudien. Die vorgestellte Methodik ist in den Grammatikentwicklungsprozess integriert und stellt einen Beitrag dar, die Grammatikentwicklung und die Auswahl sowie die Feineinstellung des Suchalgorithmus zu unterstützen.

Die Ergebnisse der Fallstudien zeigen das Potenzial der Methodik auf, den menschlichen Entwickler während der Grammatikentwicklung und -anwendung zu unterstützen. Zusätzlich können die Zusammenhänge zwischen der Problemformulierung, der Lösungssuche und dem erkundeten Lösungsraum besser verstanden werden.

Acknowledgments

First of all, I would like to thank my doctoral advisor Prof. Kristina Shea for all her support and guidance through the last years. She taught me how to become a good researcher, be a critical reviewer and significantly helped me to improve my scientific writing skills. Especially, I want to thank her for the inputs in the research meeting we had, her confidence in my work and for giving me the opportunity to deviate from my planned research direction and find my own path.

Further, I want to thank Prof. Jon Cagan for his interest in my research and being my second examiner and Prof. Jürg Dual for acting as chair person at my defense. I thank them both for their precious time and motivation to examine my research.

I also like to thank Prof. Udo Lindemann, Dr. Markus Mörtl and all former colleagues in Munich for the friendly atmosphere that always made me feel part of the overall Institute of Product Development and made me enjoy working there. I also thank Prof. Matt Campbell for co-supervising my thesis for the time I was in Munich and for the fruitful discussions and programming advice.

My time as a PhD student wouldn't have been such a precious experience without all the colleagues I had the honor to work with. The "VPD-guys" I thank for the warm welcome in the group and the help and advice throughout the years. You made me feel accepted and part of a great team right from the beginning. The "EDAC-guys" I want to thank for the numerous conversations that influenced my research, the inspiring coffee breaks at the roof terrace and the amazing Fondue-evenings. Especially, I would like to express my gratitude to Tino Stankovic from whom I learnt so much, ranging from crazy algorithms to general survival strategies, for his friendship, patience and encouragements. Similarly, I want to thank Clemens Münzer for surviving the last three years in the same office with me and for being such a great colleague and friend. I enjoyed the honest and often diverse discussions and am sure they greatly improved my research. Further, I like to thank (in alphabetical order) Allie for spreading her optimism, Benjamin for the exceptional cooking receipt recommendations, Bettina for her open ear and all the help at various occasions, Eugen for greatly supporting the running team, Fritz for kindly testing my software prototype, Jochen for the helpful discussions of FEM methods, Jung for generating outstanding images, Luca for understanding things wordlessly, Martin for supporting my first steps in Swiss German, Merel for the impressive introduction to kick-boxing, Paul for his advice on endurance in sports and academia and Tim for the exciting conference vacation.

Last but not least, I want to thank my friends and my family for their permanent support through the last years. Thanks for all your encouragement to follow this path. I especially thank my parents Renate and Gerhard and my brothers Daniel and Alexander for always believing in me and supporting me in so many ways. A huge thank you goes to Sandra, my sister-in-law, for proofreading the manuscript of this thesis and for all the cheering ups. Finally and most importantly, I thank my boyfriend Max for his love and his immeasurable support, encouragement and patience with which he helped me through this challenging time. He did an incredible job and found just the right balance between taking care of things so that I can concentrate on my research and tearing me away from the computer from time to time so that I didn't lose sight of the many other important aspects of life.

The following publications are part of the work presented in this thesis:

- [1] Königseder, C. and Shea, K. (2015). "Comparing Strategies for Topologic and Parametric Rule Application in Automated Computational Design Synthesis", *Journal of Mechanical Design*, **138**(1), pp. 011102-1 - 011102-12.
- [2] Königseder, C. and Shea, K. (2015). "A Method for Visualizing the Relations Between Grammar Rules, Performance Objectives and Search Space Exploration in Grammar-Based Computational Design Synthesis", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), Boston, MA, USA.
- [3] Königseder, C., Stanković, T., and Shea, K. (2015). "Improving Generative Grammar Development and Application Through Network Analysis Techniques", International Conference on Engineering Design (ICED), Milano, Italy.
- [4] Königseder, C. and Shea, K. (2014). "Systematic Rule Analysis of Generative Design Grammars", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **28**(3), pp. 227-238.
- [5] Königseder, C. and Shea, K. (2014). "Analyzing Generative Design Grammars", in *Design Computing and Cognition '14*, J. S. Gero and S. Hanna, eds., Springer International Publishing, pp. 363-381.
- [6] Königseder, C. and Shea, K. (2014). "Strategies for Topologic and Parametric Rule Application in Automated Design Synthesis using Graph Grammars", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), Buffalo, NY, USA.
- [7] Königseder, C. and Shea, K. (2014). "The Making of Generative Design Grammars", Workshop on Computational Making, International Conference on Design Computing and Cognition (DCC), London, UK.
- [8] Königseder, C., Shea, K., and Campbell, M. I. (2012). "Comparing a Graph-Grammar Approach to Genetic Algorithms for Computational Synthesis of PV Arrays", in *CIRP Design 2012* A. Chakrabarti, ed., Springer London, Bangalore, India, pp. 105-114.
- [9] Kumar, M., Campbell, M. I., Königseder, C., and Shea, K. (2012). "Rule Based Stochastic Tree Search", in *Design Computing and Cognition '12*, J. S. Gero, ed., Springer Netherlands, pp. 471-587.
- [10] Königseder, C. and Shea, K. (2015). "Visualizing Relations Between Grammar Rules, Objectives and Search Space Exploration in Grammar-based Computational Design Synthesis (currently under review)", *Journal of Mechanical Design*.
- [11] Königseder, C., Stankovic, T., and Shea, K. (2015). "Improving Design Grammar Development and Application Using Transition Graphs (currently under review)", *Design Science Journal*.

© Corinna Königseder, 2015

Parts of the following chapters are published with permission from the copyright holders:

Chapter 4: © Cambridge University Press, 2014, originally published as Königseder, C. and Shea, K. (2014). "Systematic Rule Analysis of Generative Design Grammars", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **28**(3), pp. 227-238, published with permission from Cambridge University Press.

Chapter 4: © Springer, 2014, originally published as Königseder, C. and Shea, K. (2014). "Analyzing Generative Design Grammars", in *Design Computing and Cognition '14*, J. S. Gero and S. Hanna, eds., Springer International Publishing, pp. 363-381, published with permission from Springer.

Chapter 5: © Design Society, 2015, originally published as Königseder, C., Stanković, T., and Shea, K. (2015). "Improving Generative Grammar Development and Application Through Network Analysis Techniques", International Conference on Engineering Design (ICED), Milano, Italy, Paper Number 195, published with permission from the Design Society.

Chapter 6: © American Society of Mechanical Engineers (ASME), 2015, originally published as Königseder, C. and Shea, K. (2015). "A Method for Visualizing the Relations Between Grammar Rules, Performance Objectives and Search Space Exploration in Grammar-Based Computational Design Synthesis", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), Boston, MA, USA, Paper Number DETC2015-46761, published with permission from the ASME.

Chapter 7: © American Society of Mechanical Engineers (ASME), 2014, originally published as Königseder, C. and Shea, K. (2014). "Strategies for Topologic and Parametric Rule Application in Automated Design Synthesis using Graph Grammars", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), Buffalo, NY, USA, Paper Number DETC2014-34691, published with permission from the ASME.

Chapter 7: © American Society of Mechanical Engineers (ASME), 2015, originally published as Königseder, "Comparing Strategies for Topologic and Parametric Rule Application in Automated Computational Design Synthesis", *Journal of Mechanical Design*, **138**(1), pp. 011102-1 - 011102-12.

Parts of the following chapters are submitted for publication in scientific journals:

Chapter 5: Parts are submitted to the *Design Science Journal* (Design Society).

Chapter 6: Parts are submitted to the *Journal of Mechanical Design* (ASME).

Contents

- Abstract.....iii**
- Zusammenfassungv**
- Acknowledgmentsvii**
- Contents.....xi**
- 1 Introduction..... 1**
 - 1.1 Current Situation and Motivation1
 - 1.2 Objectives and Expected Contributions3
 - 1.3 Research Methodology4
 - 1.4 Structure of the Thesis5
- 2 Background..... 7**
 - 2.1 Engineering Design Process.....7
 - 2.2 Computational Design Synthesis (CDS)8
 - 2.2.1 Framework for CDS.....9
 - 2.2.2 CDS using Grammars 10
 - 2.2.3 Applications for CDS Methods with Grammars..... 10
 - 2.2.4 Grammar Interpreters 11
 - 2.2.5 Challenges for CDS Using Grammars..... 12
 - 2.3 Related Research Areas..... 17
 - 2.3.1 Optimization 17
 - 2.3.2 Compiler Design 18
 - 2.3.3 Visualization 18
 - 2.3.4 Method and Tool Evaluation 22
 - 2.4 Summary..... 23
- 3 Method Overview25**
 - 3.1 Grammar Rule Analysis Method..... 26
 - 3.2 Network-based Rule Analysis Method 26
 - 3.3 Relation Visualization Method 26
 - 3.4 Search Strategy Comparison Method 27
- 4 Grammar Rule Analysis Method28**
 - 4.1 Motivation for a Grammar Rule Analysis Method 28
 - 4.2 Method..... 29
 - 4.2.1 Data Generation 30
 - 4.2.2 Data Analysis 30
 - 4.2.3 Visualization and Interpretation of Analysis Results 31

Contents

4.3	Case Study: Gearbox Synthesis.....	33
4.3.1	Introduction to the Gearbox Synthesis Case Study.....	33
4.3.2	Gearbox Rule Sets to Validate GRAM.....	38
4.3.3	Application of GRAM to the Gearbox Rule Sets A – D.....	40
4.3.4	Results	40
4.4	Discussion	45
4.5	Summary.....	46
5	Network-based Rule Analysis Method	49
5.1	Motivation for Network-based Rule Analysis.....	49
5.2	Method	51
5.3	Case Study 1 (Gearbox Synthesis): Analyzing LHSs of Rules	54
5.3.1	Generation of Designs	54
5.3.2	Results	55
5.4	Case Study 2 (Tile Puzzle): Learning and Reusing Rule Sequences	62
5.4.1	Understanding the Small Scale Problem	63
5.4.2	Applying Knowledge on the Large Scale Problem	64
5.4.3	Results	66
5.5	Discussion	68
5.6	Summary.....	69
6	Relation Visualization Method	73
6.1	Motivation for Visualizations in CDS	73
6.2	Method.....	74
6.2.1	Example to Demonstrate the Method	75
6.2.2	Implementation Details.....	78
6.3	Case Study 1: Bicycle Frame Synthesis.....	78
6.3.1	Introduction to the Bicycle Frame Synthesis Case Study	78
6.3.2	Search Algorithms	81
6.3.3	Scenarios	84
6.3.4	Results	84
6.4	Case Study 2: Gearbox Synthesis.....	89
6.4.1	Results	89
6.5	Discussion	94
6.6	Summary.....	98
7	Search Strategy Comparison Method	101
7.1	Method.....	102
7.1.1	Burst Algorithm	103
7.1.2	Strategies for Rule Type Selection.....	105
7.1.3	Metrics for Comparing the Strategies	107

7.2	Case Study 1: Gearbox Synthesis.....	108
7.2.1	Generation.....	108
7.2.2	Evaluation.....	109
7.2.3	Guidance.....	109
7.2.4	Results	110
7.3	Case Study 2: Bicycle Frame Synthesis	114
7.3.1	Generation.....	115
7.3.2	Evaluation.....	115
7.3.3	Guidance.....	115
7.3.4	Results	116
7.4	Discussion	124
7.5	Summary.....	126
8	Implementation	130
8.1	Generic Framework for CDS	130
8.2	Data Generation Options	131
8.3	Data Analysis Options.....	133
8.4	Data Visualizations	133
8.5	Usage of the Framework	134
8.6	Discussion	136
9	Discussion and Future Work	138
9.1	Comparison of the Methods	138
9.2	Modified Process for Grammar Development and Application	139
9.3	Generality of the Methodology.....	141
9.4	Research Contributions	143
9.4.1	Methodology for Supporting Design Grammar Development and Application.....	143
9.4.2	Supporting Human Designers During Grammar Development.....	143
9.4.3	Supporting the Selection of the Search Algorithm.....	144
9.4.4	Supporting the Refinement of the Search Process	144
9.4.5	Providing a Software Prototype to Support CDS.....	145
9.5	Limitations and Future Work.....	146
10	Conclusions.....	148
	References.....	151

1 Introduction

“It is characteristic of the search for [design] alternatives that the solution, the complete action that constitutes the final design, is built from a sequence of component actions. The enormous size of the space of alternatives arises out of the innumerable ways in which the component actions, which need not be very numerous, can be combined into sequences” [12]. This quote from Herbert A. Simon in his famous book *The Sciences of the Artificial* in 1969 describes the generation of design alternatives. It is part of a discussion whether new reasoning methods are required for design synthesis or whether standard logic of declarative statements are sufficient. Design is described as a process of search in a space of design alternatives.

In 1956 already, Herbert A. Simon and Allen Newell developed the Logical Theorist [13], one of the first artificial intelligence (AI) programs. Since then, numerous researchers attained great achievements in supporting human designers through computer applications within and outside of the field of AI. Designing products using Computer-Aided Design (CAD) Systems, performing simulations to analyze product behavior before parts are manufactured and using Computerized Numerically Controlled (CNC) machines to manufacture parts are just a few examples, showing how the use of computers can support product development and manufacturing.

The engineering design process is frequently described to consist of the four phases: task clarification, conceptual design, embodiment design and detail design [14]. During conceptual design, different product concepts are investigated and a decision on a principle solution is taken. The following phases further elaborate the principle solution. Most of the currently used computational tools focus on embodiment design and detail design. Even though far-reaching decisions are taken during conceptual design, the computational support in this phase is limited.

The need to generate numerous possible concept solutions and the lack of detailed knowledge of design requirements and constraints make it challenging to develop support tools for conceptual design. These tools have to be capable of modeling a vast number of designs, ideally be unbiased and be able to either assess different generated concepts or present them to the human designer in a form that allows selecting the most promising concepts from the generated ones.

1.1 Current Situation and Motivation

Design synthesis is one of the most challenging parts in the generation of new products and for a long time it was seen as solely the task of humans to conduct this creative step. Since 1956, when Newell and Simon [13] presented the “Logic Theorist”, a computer program that solves proofs based on heuristic methods that have been observed in human problem solving activities, researchers have demonstrated repeatedly that computers are also capable of carrying out design synthesis tasks. Different approaches are explored to enable

the computer to design automatically and to develop tools that allow an interactive process in which computers and human designers can develop design solutions together. Computational Design Synthesis (CDS) is a research area aimed at supporting human designers with computational synthesis methods. CDS methods enable the computer to generate valid or even innovative solutions for engineering tasks. One approach is the use of generative grammars to capture engineering knowledge formally so that it can be used in computational design systems. For example, graph grammars use graphs consisting of nodes and edges to represent possible designs. Often nodes represent components of a design, while edges represent their connectivity. Grammar rules describe transformations of the graph representation of a design to generate new designs.

CDS methods have many benefits, e.g., they enable the generation of a vast number of design alternatives. Even though time has to be invested to generate a formal description of the design problem, this effort is paid off when generating several designs, as the computer is then capable to generate numerous designs in a short amount of time. While some bias might be implemented in the grammar rules when designers formulate their engineering knowledge, the computer, per se, is not biased by personal preference or past experience and thus capable of comparing designs objectively. This allows for the generation of novel or even creative designs. Fixation effects [15] which often exist for human designers, can be decreased when using formal methods such as grammars in CDS [16]. Modeling problem-specific knowledge formally in the computer leads to greater transparency [17]. It further eases the comparison of different concepts, if they can be evaluated computationally. Another benefit of formal modelling is that the developed model and the formalized knowledge can be stored for future reuse and for documentation [18].

Despite the benefits of CDS approaches, they are still not widely used. Hesitation in industry and research exists for several reasons. First of all, the field of CDS is still relatively new and has not reached common engineering practice. Also, at most universities, engineering design education does not comprise computational approaches to synthesis. Furthermore, diverse knowledge is required to successfully apply CDS methods to an engineering design task. In a first step, the problem-specific knowledge has to be formalized, requiring the human designer to understand concepts of knowledge formalization, e.g. the concepts of vocabulary and rules in case of using grammars. Besides understanding formal modelling approaches for representing the design task at hand, the human designer also requires basic knowledge of search and optimization algorithms to successfully combine the developed rules with an algorithm that guides the synthesis process to purposeful designs. To date there is only little support (see Section 2.2 for more details) for human designers in these tasks of developing a grammar to represent design transformations and combining it with search and optimization algorithms.

This thesis investigates methods to support human designers in the CDS process using grammars. The focus is on developing a methodology to support the human designer during grammar development and grammar application. The use of a systematic process to generate data, analyze it and provide visualizations for a human designer is examined. Using

such a process, information on interactions among grammar rules, rule sequences, changes in objectives and characteristics of the generated designs and the search process can be gained. This information can be provided to the human designer in visual form. A software prototype is developed to test the applicability of the developed methods. It automatically generates and analyzes data and provides visualizations for the human designer. Interpreting the visualizations can increase the designer's understanding of different components of the CDS process. This in turn can give rise to improved grammars and beneficial combinations of grammar rules and search algorithms and lead to improved synthesis results. As a long term goal, this can help to bring the benefits of CDS methods to industrial applications.

1.2 Objectives and Expected Contributions

The first research question that this thesis addresses is:

Preliminary research question: *How can the human designer be supported during CDS using design grammars?*

This question is, however, far too broad to be answered in this thesis. More specific research questions are derived in Chapter 2 based on a literature review. The research focus is narrowed to analyzing grammar rules and their performance after application.

The **subordinate goal** of this thesis is then to support the human design using grammar-based CDS methods through systematic analysis and information visualization. The visualizations shall provide information on rules and rule sequences, the objectives and design characteristics of synthesized designs and the search process.

In this thesis, a methodology of four methods is presented that all pursue this goal. Sub-goals are to support the human designers during grammar development (G1), selection of an appropriate search algorithm (G2) and refinement of the search (G3) to steer the synthesis process towards purposeful designs. The overall goal and sub-goals are presented in Figure 1-1. Under the assumption that visualization of information improves understanding, the long-term goal is to increase the designer's understanding of interdependencies between the formal representation of a design problem using grammar rules and the search process. This would enable the human designer to have more informed interactions with grammar-based CDS methods which in turn can lead to improved synthesis results.

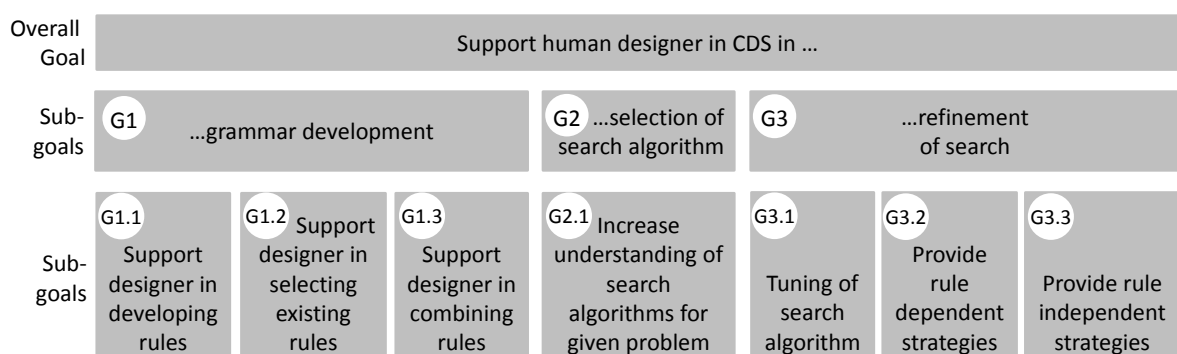


Figure 1-1 Goals of the methods in this thesis.

The following four contributions are expected:

The first expected contribution is to **support the human designer during design grammar development** through systematic analysis and visualization of how grammar rules change objectives and design characteristics of the generated designs. When the human designers compare their mental model of how rules influence objectives and design characteristics to the visualized information, deviations can be identified. The designer can then either change the grammar rules to be in accordance with the mental model or vice versa.

The second expected contribution is **supporting the human designer to select an appropriate search algorithm**. For a given grammar, the search algorithm's progression is analyzed. The human designer is provided interactive visualizations of the search process. This facilitates the understanding of how the search algorithm explores and exploits the design space. Examples for interactions are filtering the visualized content to show specific aspects, varying the level of detail that is presented or replaying chronological sequences in animated visualizations. These interactions enable one to obtain information on how specific grammar rules and search algorithms generate designs. Understanding the characteristics of a search algorithm to explore or exploit the design space when combined with a specific grammar supports the human designer in reasoning about successful combinations of grammar rules and search algorithms.

The third expected contribution is **supporting the refinement of the search process**. Understanding grammar rules and search algorithms in depth enables the human designer to further tune the CDS process. One aspect of this is tuning the search algorithm by setting appropriate algorithm parameter values, e.g. specifying regular restarts of the synthesis process. Another aspect is defining advanced strategies for modifying topologies and parameters of designs during synthesis. Lastly, beneficial sequences of rules can be identified. With the presented methods, the human designer is able to identify how changes that are made to refine the synthesis process can reflect on the generated designs.

The last expected contribution in this thesis is a **software prototype** that supports the human designer working on grammar-based CDS methods. The software prototype is developed to enable an evaluation of the developed methods. Ideally, it supports grammar development and application as well as the analyses presented in this thesis. Interfaces to simulation software to support design evaluations are also preferable. Providing such a software prototype permits to show the potential of the presented methodology.

1.3 Research Methodology

The thesis is structured according to the main steps of the Design Research Methodology (DRM) as defined by Blessing and Chakrabarti [19]. An overview of the DRM and references to the chapters of this thesis in which the respective phases of the DRM are addressed, is given in Figure 1-2. In this chapter, the research task is identified. The existing situation of CDS using grammars is further analyzed in Chapter 2 through a literature review. Based on the increased understanding of the situation the research questions are refined. A

methodology to support designers using grammar-based CDS methods is developed (Chapter 3). It consists of four distinct methods that are presented and evaluated in Chapters 4, 5, 6 and 7. In this thesis only an initial descriptive study II is conducted. The practical applicability of the developed methodology is evaluated based on case studies. The presented methodology for supporting the designer is discussed in Chapter 9 along with future research directions. As suggested by Blessing and Chakrabarti [19], the research is not conducted in a strictly linear process, but includes several iterations and parallel execution of research in different phases. In the remainder of this thesis, the focus is on the outcomes rather than on a detailed description of how the phases of the DRM are addressed. References back to the research methodology are given where appropriate.

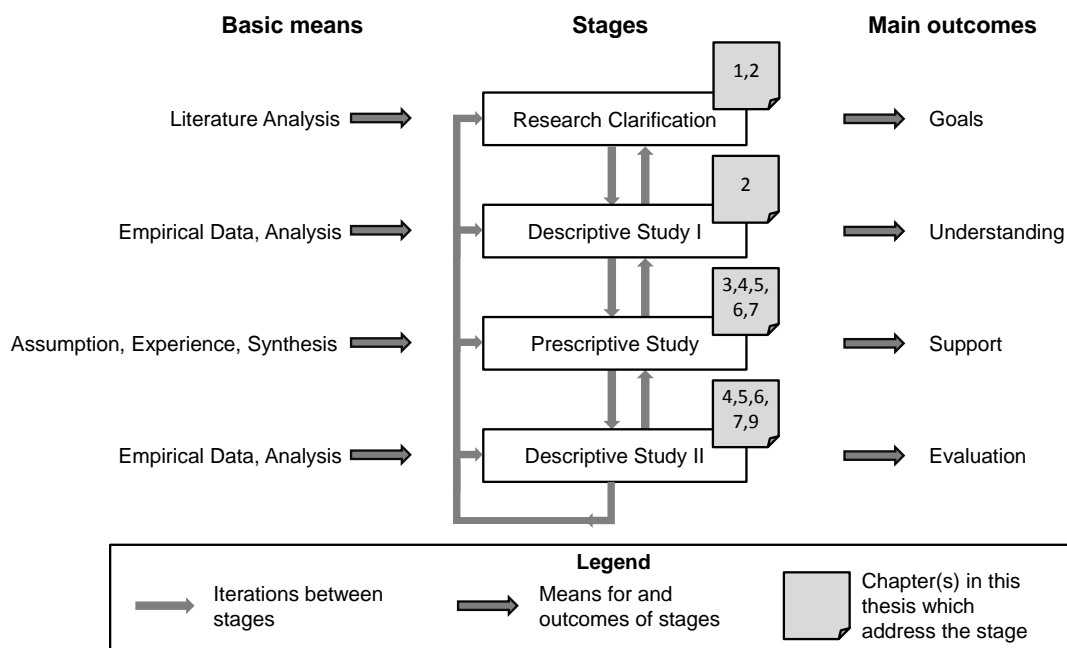


Figure 1-2 Research methodology used to structure this thesis (adapted from [19]).

1.4 Structure of the Thesis

The motivation for a methodology for grammar-based CDS is presented in this chapter. The structure of the remainder of this thesis is as follows:

In **Chapter 2**, the engineering design process and different CDS methods are reviewed as means to support conceptual and embodiment design. Grammatical approaches to CDS are discussed in detail since the scope of this thesis is to support human designers using grammar-based CDS methods. Recent applications and grammar interpreters are presented and the interactions between human designers and the computational method are addressed. Challenges for grammar development and grammar application are identified. Background information on various research areas is given as far as required to understand the concepts used within this thesis.

In **Chapter 3**, the methodology for supporting grammar-based CDS is introduced. It consists of four different methods that are introduced to give an overview of the presented research. The intention behind this chapter is to give the reader enough information to understand

the differences between the presented methods such that he or she is able to select an appropriate method for a given problem and to guide the reader to the relevant chapter where the method is then explained in detail.

The method chapters (Chapters 4, 5, 6 and 7) have an analogous structure. First, the motivation for and purpose of the method are presented. Then the method itself is presented in detail. The application on different case studies demonstrates each method's use, followed by a discussion and summary of the chapter.

In **Chapter 4**, the Grammar Rule Analysis Method (GRAM) is presented that allows human designers to analyze their grammar rules in a systematic way. The method is presented along with an engineering design task, the synthesis of a gearbox. This case study is also used in Chapter 7. Details on the gearbox case study are introduced in Section 4.3.

Chapter 5 introduces a network-based method to analyze rules in detail. It enables the human designer to a) analyze application conditions of rules and b) analyze sequences of rule applications. The method is targeted at supporting grammar development and rule application. It is based on the analysis of a network of rule applications where designs are represented as nodes while edges are used to represent transitions, in form of rule applications, between nodes, or designs.

Chapter 6 introduces a method for visualizing relations between grammar rules, performance objectives of generated designs, and search space exploration. It includes advanced rule analysis to support grammar development. Visualizations of the progression of search algorithms enable the human designer to recognize differences between various search algorithms and thereby support the selection of a search algorithm for a given design task.

Chapter 7 presents a method to analyze and compare different strategies for applying topologic and parametric rules. Topologic rules change the connectivity of components of a design whereas parametric rules change attributes of the components, e.g. mass, material or position of a component. Understanding how strategies for topologic and parametric rule applications influence the generated designs allows selecting an appropriate one for a given design task. This can lead to a faster convergence of the synthesis process towards superior designs or to generate more diverse designs.

Chapter 8 provides details on the implementation of a software prototype to evaluate the presented methods. A generic framework is developed to support CDS during grammar development and application. The available options for data generation, analysis and visualization as well as possible user interactions are presented.

Chapter 9 discusses research contributions and limitations of the presented methodology. The generality of the method is discussed and directions for future research are outlined.

Chapter 10 concludes the thesis by highlighting the main contributions and limitations of the methodology for supporting grammar-based CDS.

2 Background

Related work is presented in this chapter to give the reader the required background information to understand the challenges, benefits and limitations of the presented research. In Section 2.1 a short overview of the design process and the use of CDS methods in the conceptual and embodiment design phase is given. Different CDS methods are presented as a means to support conceptual design in Section 2.2 to provide a broad overview of existing methods. An introduction to grammatical approaches for CDS is then given for which the methods in this thesis are developed. Applications of grammar-based CDS methods are presented and challenges for these methods are identified in a literature review. Based on these challenges, the research task is refined. The methodology developed in this thesis combines concepts from different research fields such as optimization, compiler design and visualization research. An overview of relevant topics in these areas is presented in Section 2.3. Methods to evaluate research results are reviewed in Section 2.3.4 to identify appropriate measures for theoretically evaluating the research in this thesis. Section 2.4 gives a summary of the chapter and presents the refined research questions and the method for evaluating the research results.

2.1 Engineering Design Process

According to Pahl et al. [14] the “main task of engineers is to apply their scientific and engineering knowledge to the solution of technical problems, and then to optimize those solutions within the requirements and constraints set by material, technological, economic, legal, environmental and human-related considerations” [14]. Four main phases are defined in [14] for the planning and design process. They are shown on the right in Figure 2-1. In the planning and task clarification phase, information about the task, i.e. the requirements are specified. In the conceptual design phase the essential problems are identified, function structures are established and working principles and structures are searched. Those are then combined to develop concept variants. The developed concepts are then evaluated against the criteria specified in the requirements list and one or several principle solutions are selected. Once principle solutions are found, preliminary layouts are developed. Often several preliminary layout variants are developed and evaluated against technical and economic criteria. Based on the evaluation results, a definitive layout is defined, often incorporating aspects from different individual variants. The outcome of the embodiment design phase is the definitive layout. In the detail design phase the geometric arrangement, dimensions and tolerances of all individual parts are specified and the product documentation is generated. Many other descriptions of the design process exist. In this thesis only the process by Pahl et al. [14] is presented because its four main phases are incorporated in most other processes in similar forms. The research presented in this thesis addresses the conceptual design phase and part of the embodiment design phase as shown in Figure 2-1. The development of preliminary layouts is addressed. The evaluation of

preliminary layouts is only addressed considering several technical criteria. Economic criteria and the definition of a definitive layout are beyond the scope of this thesis.

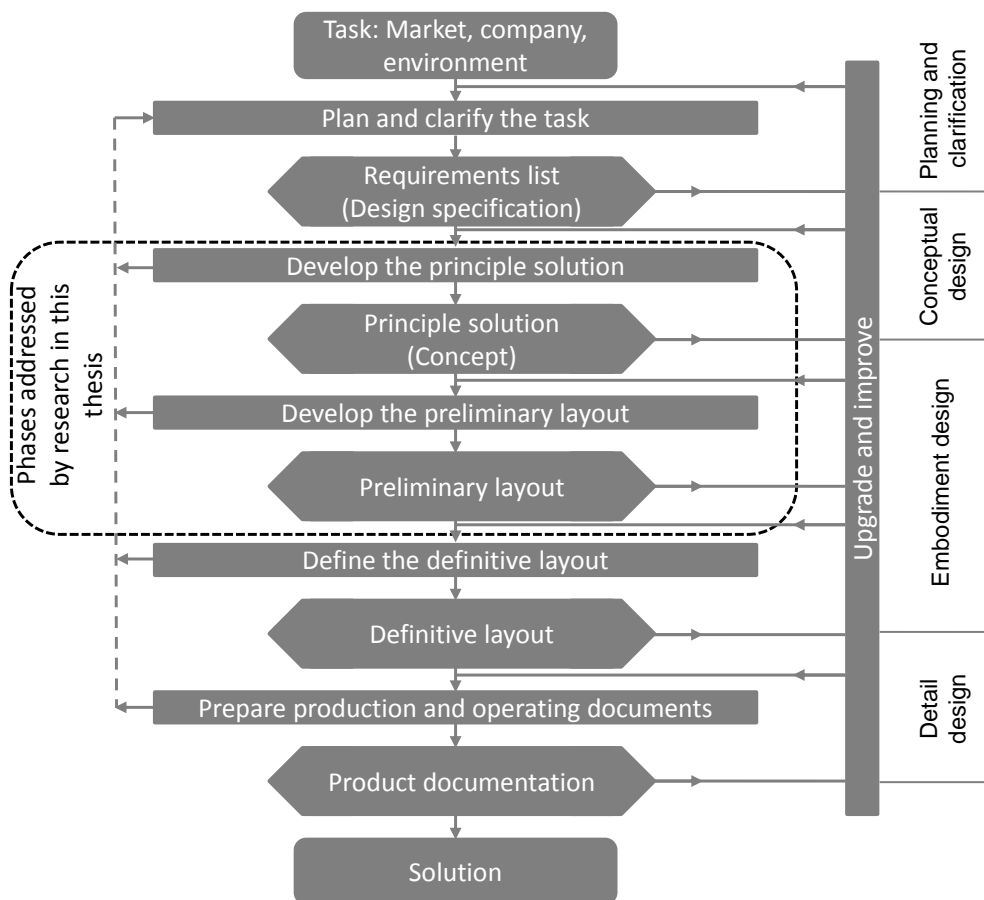


Figure 2-1 Steps in the planning and design process according to Pahl et al. [14] (adapted from [14]).

2.2 Computational Design Synthesis (CDS)

Computational Design Synthesis is a research area that develops guidelines, methods and tools for supporting the generation of novel and creative, but also routine designs. Two major benefits of CDS methods are mentioned in [20]. First, the use of computer-based methods permits to overcome restrictions of human designers such as limited knowledge or design fixation. The computer is not biased per se and explores directions the human designer would probably not consider and thus computational methods have a chance to explore novel designs. Second, computer-based methods can support routine design by automating tedious tasks. Chakrabarti et al. [20] outline three major themes of synthesis: function-based synthesis, grammar-based synthesis and analogy-based synthesis. Other approaches exist, e.g. Münzer et al. [21] define compatibility constraints between components in a metamodel and synthesize design concepts by transforming the metamodel into a boolean satisfiability problem. Its solutions constitute design concepts that are conform with the compatibility constraints. For a broader view of the research field, the reader may refer to [20, 22, 23].

2.2.1 Framework for CDS

Different frameworks for CDS can be found in literature. Shea and Starling [24] present a framework for performance-based parametric design synthesis (Figure 2-2) and describe an iterative process consisting of the steps investigate, generate, evaluate and mediate.

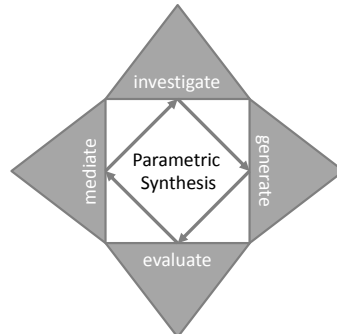


Figure 2-2 Framework for performance-based parametric synthesis (reproduced from [24]).

Cagan et al. [25] define a similar generic framework for CDS. An overview of the generic CDS process is given in Figure 2-3. First, the designer describes the design problem formally at the required level of detail. After the representation is defined, the CDS process consists of the three phases generate, evaluate and guide. In the generation phase, a new design is created by modifying an existing design solution. The resulting design is then assessed using either a simulation tool or some heuristics to evaluate the design's objectives and constraints. Based on the evaluation results, the synthesis process is either continued with this newly generated design or restarted from a previously found design alternative. Search algorithms are commonly used to guide the synthesis process and starting from an initial design the steps generate, evaluate and guide are continued until either no further modifications are possible or a stopping criterion is met. After interpreting the final designs, the designer can rephrase the problem formulation to synthesize different final results. This process is iterative and often the problem formulation and parameter settings of the search algorithm are varied until desired designs are generated. The terminology of this thesis is used as defined in [25]. In the following, the focus is on grammar-based synthesis.

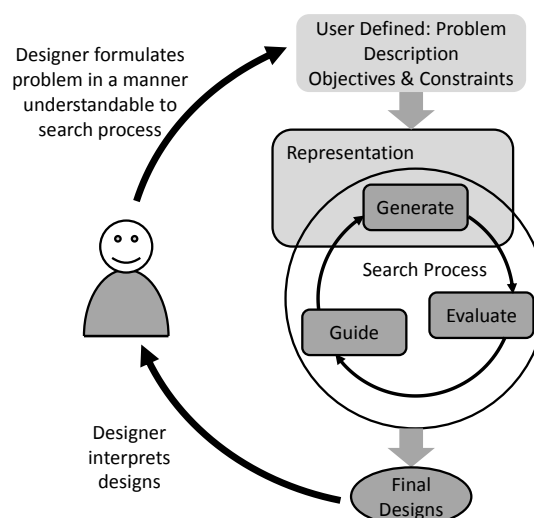


Figure 2-3 The generic CDS process (reproduced from [1]).

2.2.2 CDS using Grammars

In grammatical approaches to CDS, designers develop a grammar to represent a desired design language. It consists of a vocabulary, usually describing design components or subsystems, as well as a set of grammar rules. These rules describe design transformations, that are defined by a left-hand-side (LHS) and a right-hand-side (RHS), i.e. $LHS \rightarrow RHS$. The LHS defines where the rule can be applied in a design and the RHS defines how the design transformation modifies the design. Using graph grammars for design synthesis, each design can be described using a graph representation that consists of nodes and edges. For example, nodes can represent components while edges describe functional or spatial relations between the nodes [26].

Gips and Stiny [26] propose the use of production systems (grammars) for design tasks in 1980. Since then, research on shape grammars has become an active research area in architecture and engineering design. Besides shape grammars, also different types of grammars are used in engineering design. Examples are parallel grammars [27], spatial grammars [28-31] and graph grammars [17, 32-34]. A more extensive overview of grammatical approaches to engineering design can be found in [35, 36].

2.2.3 Applications for CDS Methods with Grammars

Three applications for grammatical approaches in design generation in architecture and engineering design are presented in the following sections. The first one is targeted at solving particular engineering problems using generative grammars. The second aims to formally represent designs to capture style in order to classify designs or generate new ones in a certain style. The third is on identifying commonalities and differences between designs, e.g. for product platform development.

Previous work in the field of CDS using generative grammars includes the synthesis of mechanisms [34] and epicyclic gear trains [37], the automated synthesis of gear boxes [38] and sheet metal design [39]. Besides assisting synthesis, generative grammars can support the design process, e.g. by flexibly providing knowledge that was previously stored in paper-based design catalogs [40] and reusing concepts from extracted design knowledge [41]. Additionally, not only the synthesis of products can be supported with grammatical approaches but also the synthesis of technical processes [42]. For a review of approaches to CDS see, for example Chakrabarti et al. [20]. More recent work on CDS using grammars includes the development of a shape grammar based on historic aircrafts that is able to generate new, feasible aircraft designs [43], a graph grammar to synthesize passive brachiating robots [44], and a spatial grammar to generate 3D solid models for flexible fixture devices [28]. Most of the described applications are in the engineering design domain and focus on solving a particular design task.

Especially in architecture, but also in engineering design, grammars are also used to understand and capture style. Formally representing a style enables, e.g., to recreate new artifacts following a given style. Examples are research to understand brand identity, like of

Harley Davidson motorcycles [45] or Buick cars [46] and research to modify grammars to adapt to new style requirements, e.g. for mobile phones [47].

Another usage of grammars in engineering design is presented by Siddique and Rosen [48]. They present a framework for generating a product platform architecture using a graph representation for functions and structure. Sub-graph isomorphisms are detected to identify core functions of a product family and graph grammar rules are developed to map from function to structure level. Steps to a) develop a new product family, and b) communize existing products to a product family are identified and a coffeemaker example from [49] is used to demonstrate these.

2.2.4 Grammar Interpreters

In the last years, several grammar interpreters have been developed for spatial and graph grammars. They allow the formulation of grammar rules as well as their application but vary in complexity. Examples for graph grammar interpreters are GrGen.NET, boogie, GraphSynth and Design Compiler 43. GrGen.NET (abbreviated with GrGen in the following) is a domain neutral, open source graph rewriting system developed for software designers [50]. GrGen is highly expressive due to the use of attributed typed directed multigraphs and multiple inheritance for edges and nodes. It is also faster than its competitors due to the use of heuristics for finding graph matches [50]. These two advantages lead to the use of GrGen in multiple disciplines, e.g. in computer linguistics, in compiler design or in computational biology [51]. Booggie [52] is an object-oriented graph grammar interpreter based on GrGen. It provides graphical user interfaces to develop grammar rules and aims to bring the use of CDS methods in everyday practice through more efficient and effective knowledge formalization [53]. GraphSynth, an open source graph grammar interpreter, used for example in [39], enables designers to develop and apply graph grammar rules. Developed rules can either be applied automatically on a random match of the rule on the graph or the match can be selected interactively by the human designer. More sophisticated control mechanisms, e.g. search algorithms, can be implemented through c# plugins. Design Compiler 43 [54] is a platform for design synthesis using graph grammars. Graph grammars can be defined on a generic layer and transformed to a domain specific layer. This transformation is done in an interpretation step in which different models, e.g. for analyzing or visualizing synthesized designs, can be generated based on the graph representation. Spatial grammar interpreters are developed, e.g. by Hoisl and Shea [55] and Chau et al. [56]. Hoisl and Shea [55] present a visual and interactive grammar interpreter using three-dimensional labels. Chau et al. [56] present challenges for shape grammar interpreters and present a shape grammar implementation enabling three dimensional shape recognition for shapes consisting of rectilinear and curvilinear basic elements. Grasl and Economou [57], [58] present a graph grammar based approach to implement shape grammars. More detailed overviews on grammar interpreters and their applications can be found in [20, 31].

Sound grammar interpreters are a prerequisite for successfully using grammars in CDS. Research on grammars and the development and implementation of efficient grammar

interpreters advanced the use of grammars in diverse application areas. The question of how to develop and apply a grammar for a given design task is, however, seldom addressed. Some of the presented grammar interpreters aim to support the human designer, e.g. through providing intuitive user interfaces for grammar development. With Booggie and GraphSynth, for example, LHS and RHS of a grammar rule can be defined visually. For more difficult rule application conditions or to change attribute values, however, the rule designer has to program code in both systems. Research on systematically supporting human designers during rule development and application is still in an early stage. Challenges for CDS approaches using grammars and research addressing these are discussed in the following section.

2.2.5 Challenges for CDS Using Grammars

Although many successful applications exist, the use of CDS methods is still challenging, mainly due to the numerous aspects that have to be considered when developing and using CDS methods. A successful application of CDS methods using grammars requires a) a meaningful representation of designs and the design task at hand, b) a careful formulation of grammar rules to synthesize new designs, c) problem-specific design evaluations, and d) the selection of an appropriate algorithm to guide the synthesis process and search the solution space. As there is no one-fits-all solution to design synthesis, the human designer has to take various decisions when setting up a CDS method. Cagan et al. [25] describe this as finding a “complex balance between representation, generation, and search of a design space in pursuit of original design solutions” [25].

In [20], key issues for generative grammars are presented as follows:

- “(1) supporting designers to articulate grammars (i.e., vocabulary and rules) in software implementations,
- (2) defining ways to evaluate implementations, including identifying a set of benchmark problems,
- (3) better integration of generative grammar implementations with other software, e.g. CAD and CAE and
- (4) more methodological support for users in the process of defining a grammar since this process can also lead to better understanding of a design problem.”[20]

This thesis supports the human designer in finding this balance (as described in [25]), through a methodology that analyzes and visualizes aspects of representation, generation and search. It further aims to contribute to solving some of the key issues for generative design grammars (as described in [20]).

In the following, the challenges for a) grammar development, and b) grammar application are elaborated as indicated in literature. This helps to identify the need for more methodological support.

Chase [59] defines different steps for grammar development and application. Grammar development consists of defining the representation, the control mechanism (guidance) and

the grammar rules. The grammar application is divided into determination of a rule, determination of the object on which the rule is applied and determination of the matching conditions. Five scenarios are defined for possible user control of these four steps. In a sixth scenario, the computer controls all four steps. An overview is given in Figure 2-4.

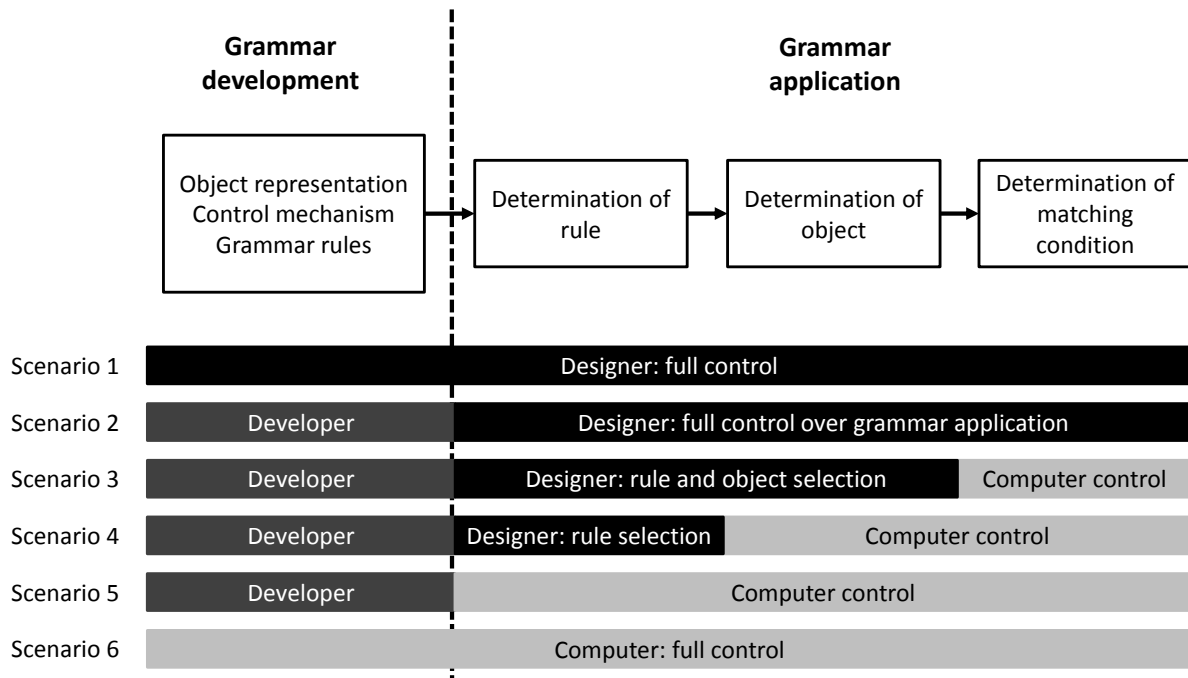


Figure 2-4 Scenarios for user control in grammar based design (adapted from [59]).

In this thesis, the focus is on scenarios 1 - 5, i.e. the scenarios where human designers are involved in either developing a grammar, applying it, or both. Scenario 6, where the computer has full control of grammar development and application, is not considered. The case studies used in the thesis can be assigned to scenario 5, i.e. the grammar system is developed by a human designer. Grammar application is then performed using the computer and is controlled through various search algorithms. The presented research is, however, not restricted to scenario 5 but can be similarly used to support scenarios 1 - 4.

2.2.5.1 Grammar Development

Although design grammars are often developed to formalize and structure the design of products and processes, the process of designing grammars themselves is often rather unsystematic and “treated, to a large extent, in an ad hoc manner with regard to design, implementation, transformation, recovery, testing, etc.” [60]. Zheng and Chen state that “sound and systematic methods and techniques are needed for grammarware to move from hacking to engineering” [61]. Both of these observations come from computer science, where formal grammars for compiler design and other applications are widely used and the research area of grammar engineering and testing has evolved. In design, Knight [62] stated that “it is the designing of a grammar that resembles what a designer does. The development of rules for designs requires the same kind of intelligence, imagination, and guesswork as the development of designs in a conventional way” [62]. Although various methods have been developed for the conventional design process, little attention is given

to design grammar rule development so far. Designing a grammar is seen as an iterative process, “a distillation of practice and experience in a particular domain” [63]. The iterative nature of rule development is a commonality among most publications covering the development process of grammars and is described in [20, 64, 65]. Chakrabarti et al. [20] mention an iterative process for the development and application of generative grammars including an iterative loop back to the modification of vocabulary and rules after designs have been generated. Recent approaches to support rule development either give advice to a human designer on how to develop [62, 66, 67] and manually test a grammar [68] or generate grammar rules automatically [69, 70]. For the latter, extensive research is also conducted in other fields, e.g., grammar induction and improvement [71] for natural language processing. For engineering design grammars, however, no research is known to the author that supports rule development through systematic and automated rule analysis.

In the **mechanical engineering domain**, most publications on CDS methods using grammars describe the grammar rules but give little to no hints on how these rules were developed. Exceptions exist and three of them are presented in the following.

- Shea [72] presents a shape grammar for aperiodic spatial tiling. First, a basic grammar (according to [62]) is developed based on mathematical descriptions for a spatial tiling and is tested. In a second step, the basic grammar is relaxed to an unrestricted grammar (according to [62]). The unrestricted grammar is more expressive and its generative power is demonstrated by the generation of various designs.
- Li et al. [37] describe how a graph grammar for epicyclic gear trains (EGT) is adapted to generate new designs. Motivated by the analysis of gear train designs that cannot be represented with an existing grammar, the authors of this paper modify the existing grammar rules. Two assumptions for EGT design are relaxed in the grammar rules and permit the generation of additional Ring-Plate-Type Cycloid Drive mechanisms. The authors state that this “process of reconciling new EGT designs and the ability of the grammar to generate them models the traditional iterative design process” [37]. They further emphasize the “need to concentrate on grammar design when designing with grammars” [37].
- Chase and Liew [73] present a framework for modifying designs using Function-Behaviour-Structure (FBS) models and grammar adaptation. Designs are modeled using a FBS representation in graph format. Rules are selected from a library of rules storing engineering knowledge. When requirements for the design are modified, the design can be adapted. This is done based on the FBS representation of the design, the modified requirements and the library of rules. Appropriate rules are selected to adapt the design such that it fits the modified requirements.

In the **architectural domain**, more support for developing grammars exist. Ibrahim et al. [65] extend the shape grammar development and application process defined by Chase [59] for a workshop in a first year architectural design studio. In this work, the improvement of the grammar is considered, however, no systematic method is given to support the analysis of the developed grammars. Oster and McCormack [74] present an explicit method for the development of shape grammars. In this method, engineering knowledge and design requirements from the customer can be captured in rules. The method is intended to

support the development of a shape grammar during new product development. The authors state that there is no need to analyze existing products as in most other processes. Drawbacks of the method are mentioned by the authors and are a focus on the customer as opposed to considering also the company's interests. Further, the method follows a linear process that contradicts with the iterative nature of grammar development as described in literature and found in practice. Oster and McCormack [74], therefore, present an interesting approach to support grammar development but there remain several issues that are still to be solved, e.g. giving the designer feedback on the developed grammar which is a central issue in an iterative development process.

Different frameworks and guidelines exist that describe how grammars can be changed to adapt the synthesis process, e.g. to generate designs that conform with a certain style or to generate designs that fit modified requirements. Knight [75] describes three processes for transforming (shape) grammars:

- Rule addition
- Rule deletion
- Rule change

More recent work by Chase and Ahmad [76] adds:

- Merging grammars (composite grammars)

“Composite grammars” are developed by merging different grammars. A framework for adaptations in shape grammar practice is presented by Al-kazzaz and Bridges [77]. They give an overview of research on shape grammar practice using adaptation methods in the architecture and engineering design domain and compare them using their proposed framework.

To summarize, the lack of support for grammar design was discussed more than a decade ago [64, 78] and still it is one of the major drawbacks of grammatical design approaches [20]. In mechanical engineering and architecture, the need for systematically developing and testing grammars is articulated [20, 31, 37, 59]. For example, McKay et al. [31] note that “there is a need for more methodological support for guiding a user in the design of a grammar.” It could be argued that at least some of the methods to support shape grammar development in the architectural domain could be used to support engineering design grammars. While general principles can be adopted, there exist differences between shape grammars in architecture and grammars in engineering design. In architecture, shape grammars are often formulated as basic grammars [62] and applied as pencil and paper grammars. In mechanical engineering, also other types of grammars, e.g. graph grammars, are used as unrestricted grammars. They are of type 0 in Chomsky's hierarchy [35]. This means that they are powerful mechanisms but have to be carefully designed and applied to generate useful designs. Knight [62] stated for shape grammars that “In general, the less restricted and more powerful a shape grammar is, the less can be predicted about how it behaves and what it generates” [62]. Unrestricted grammars are appropriate for many

engineering applications because of their expressiveness. The challenge, therefore, is to support the development of these grammars which is not supported by the methods proposed to date. This situation motivates research into a methodology for supporting human designers in rule development through methods that consider different grammar types and the iterative nature of the (rule) development process.

2.2.5.2 Grammar Application

In [62], Knight stated: “While some surprise is desirable, and a strength of grammars because it opens up new, unimagined design possibilities, some control over the outcomes of rules is also desirable.”

In this thesis, scenarios are observed where the grammar application is controlled by the computer. This is done using search and optimization algorithms for guiding the synthesis process. In this context, the exploration of designs is often considered as a search process through the design space.

Numerous algorithms are developed to support the rule application process. Examples are shape annealing [79], agent based approaches [80, 81], genetic algorithms [8], grammatical evolution [82] or protocol-based multi-agent systems [83]. Besides developing efficient search algorithms for CDS, other research focuses on relieving the designer from tuning search algorithms through machine learning methods [84], by providing generic generate and test type algorithms [85] that require only little tuning, or through generic frameworks for CDS [84].

Some researchers explore solution spaces without the use of search algorithms but through random or enumerative application of rules, as in [34, 52]. When the design space is limited or the search process is guided, e.g. through the use of labels, search algorithms might not be required. This means the need for criteria for the selection of a search algorithm depends on the problem at hand and the grammar rules representing knowledge. This dependency between representation and search is mentioned in several publications (see, e.g., [34, 67]). “Different algorithms work better in different types of space characteristics. Information about the search space characteristics could help in preparing better guiding strategies or selecting appropriate algorithms and heuristics” [25]. Identifying the characteristics of a search space is challenging and sometimes even impossible. Researchers often stick to algorithms they know well and tune them to perform well on the problem at hand.

Further research exists to ease the CDS process of which two are mentioned in the following. Schotborgh [86] presents prescriptive methods to build a knowledge model representing expert knowledge in rules. Poppa et al. [87] present a method to reduce the number of design concepts that are presented to a human designer. These approaches are, however, beyond the scope of this thesis for the following reasons. Schotborgh [86] tackles issues that are of interest for scenario 6 (see Figure 2-4). This scenario is excluded from the research in this thesis since it does not involve the human designer. The method described in [87] is

relevant once designs are generated, i.e. after the rule application, and is therefore also not addressed.

To summarize, the interplay between a developed grammar and a search algorithm is crucial. However, to the knowledge of the author, there exists no commonly accepted method that gives advice on how to select an appropriate search algorithm. This situation motivates research into a methodology for supporting human designers in understanding relations between grammar rules and search algorithms. This information can then be used to select an appropriate algorithm for a given design problem.

2.3 Related Research Areas

To support the human designer in grammar-based CDS approaches, concepts from multi-objective optimization, compiler design and software visualization are combined in one methodology. In the following, basic information is given on these fields to understand the concepts adapted within this thesis. The sections are kept short intentionally to not overwhelm the reader with details that are unessential within the scope of this thesis. References to further literature are given for readers who want to gain a deeper understanding of the respective research fields.

2.3.1 Optimization

CDS is often used in the ideation process and is described as a search through the design space. The terms exploration and exploitation (see [25] for a definition in the context of CDS) are used throughout this thesis to describe how the design space is explored. Exploration refers here to generating new, previously undiscovered designs or making large parametric variations. It can be seen as going into breadth in the idea generation process, i.e. trying to synthesize new concepts. Exploitation, by contrast, can be related with going into depth, i.e. further elaboration of an already explored design. This means only minor changes in topology or smaller parametric variations are conducted.

The goal of the synthesis processes is not necessarily to find one optimal design, but to generate a population of diverse designs. Additionally, in most engineering tasks, designs have to meet several constraints and are evaluated with respect to different, typically competing, objectives. These two aspects make a posteriori multi-objective algorithms a reasonable choice. A posteriori methods give no preferences to the different objectives and generate a set of non-dominated designs, also called a Pareto set. A design A is said to dominate a design B when a) A is no worse than B in all objectives, and b) A is better than B in at least one objective. The Pareto set represents all non-dominated designs, i.e. designs that are equally good in the multi-objective sense. When using multi-objective algorithms, often the designs of the Pareto set are stored in an archive. When a new design A is added to the archive, an update routine is triggered to remove designs that are then dominated by A from the archive (see Figure 2-5). For surveys on multi-objective optimization in engineering see, e.g. [88, 89].

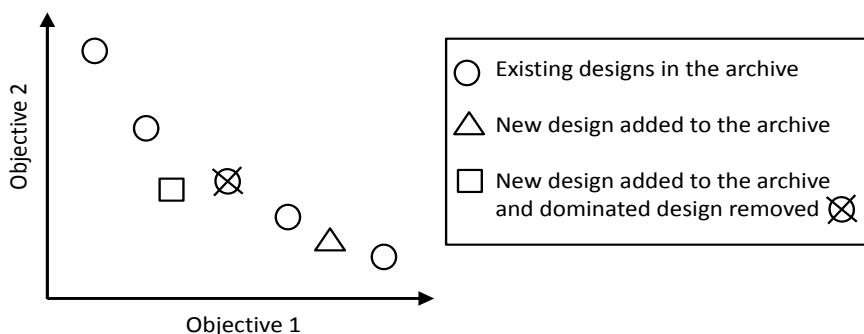


Figure 2-5 Archive evolution.

2.3.2 Compiler Design

In the area of compiler design, grammars are designed formally to automatically translate implemented source code from a higher programming language into machine code and a whole research area of grammar engineering has evolved dealing with the development of grammars. Grammars “are the essential formalism for describing the structure of programs in a programming language.” [90] Two concepts from compiler design are adapted in this thesis: a) the state representations as used to describe finite automata (FA) and transition graphs, and b) data flow analysis techniques as used in intermediate code optimization through analyzing data flows in graph representations.

FA are recognizers that either accept or reject a given input string. Each FA consists of a set of states including a start state and one or more final states, a set of input symbols (the input alphabet) and a transition function that defines the next states for each state and each symbol. [91] FAs can be represented by “transition graphs, where the nodes are states and the labeled edges represent the transition function. There is an edge labeled a from state s to state t if and only if t is one of the next states for state s and input a .” [91]. For more detailed information on finite automata and compiler design, the reader may refer to [90-92].

2.3.3 Visualization

Several research fields are concerned with the analysis of complex data, such as classical statistics or data mining [93]. The benefit of visualizations is hard to express using quantitative metrics [93], however, visual examples as the one described by Anscombe [94] (see also Figure 2-6) motivate the use of visual representations. Data for four different data sets with the same standard output from statistical regression analysis (mean of x 's, mean of y 's, equation of regression line, regression sum of squares, etc.) are plotted. The visualizations demonstrate the capabilities of the human eye to discover structure and patterns in data that are not captured with classical models.

Card et al. [95] define visualization as “the use of computer-supported, interactive visual representations of data to amplify cognition.” This is assumed to be the most accepted definition of visualization in the information visualization research field and motivates the use of information visualization techniques to amplify the understanding of the CDS process.

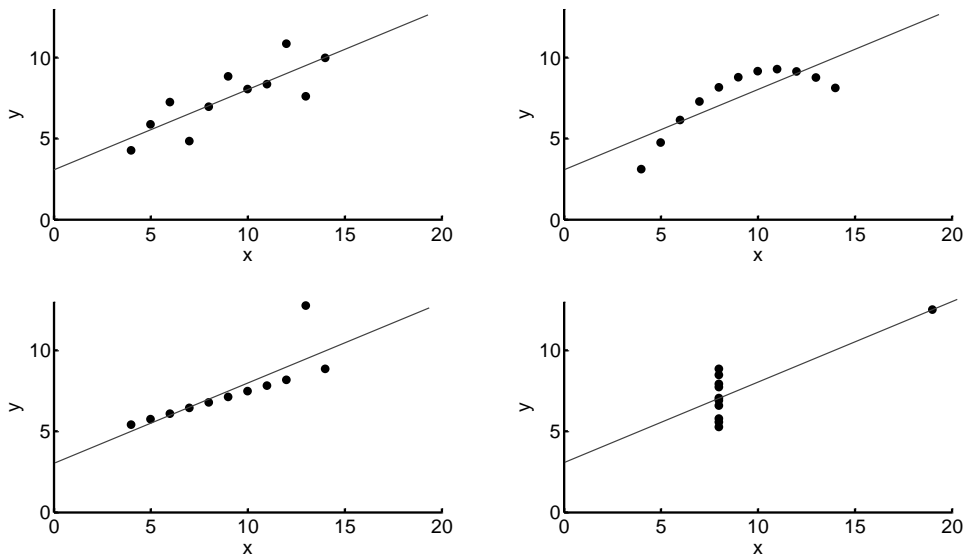


Figure 2-6 Visualizations of four data sets with the same standard output from statistical regression (recreated from data in [94]).

2.3.3.1 Research in Visualization

Keim et al. [96] distinguish among three purposes for visualization: presentation, confirmatory analysis and exploratory analysis. Visualizations for the presentation of facts are usually fixed a priori and their aim is primarily to communicate analysis results. For confirmatory analysis, visualization is used to either accept or reject hypotheses about the presented data. In exploratory analysis, the analyst explores the data without having a hypothesis. It is a search for implicit and potentially useful information in the data. Various research fields address visualization with different purposes. They are briefly introduced in the following subsections.

Scientific Visualization and Information Visualization

Nagel [97] discusses differences between scientific visualization and information visualization (often abbreviated as InfoVis) and shows where these two distinct research fields could benefit from each other. “While Scientific Visualization techniques are used for the clarification of well-known phenomena, Information Visualization techniques are used for searching for interesting phenomena.” [97]

Card et al. [95] give definitions for the two fields:

- “Scientific Visualization: the use of interactive visual representations of scientific data, typically physically based, to amplify cognition.
- Information Visualization: the use of interactive visual representations of abstract, non-physically based data to amplify cognition.” [95]

According to these definitions ([95, 97]), the research field of information visualization is better suited to visualize aspects of the CDS process in this thesis. Visualizations support the human designer to better understand the CDS process. This can include, e.g., relations

between problem representation and search algorithms that are not known in general or to the designer.

Visual Analytics

A new research field emerged from information visualization and automated data analysis taking human-computer interaction into account [98]. Thomas and Cook [99] coined the term “Visual Analytics” for this interdisciplinary field in 2005. A definition of visual analytics and its goals are defined by Keim et al. [100] as follows:

“Visual analytics combines automated analysis techniques with interactive visualizations for an effective understanding, reasoning and decision making on the basis of very large and complex data sets.

The goal of visual analytics is the creation of tools and techniques to enable people to:

- Synthesize information and derive insight from massive, dynamic, ambiguous, and often conflicting data.
- Detect the expected and discover the unexpected.
- Provide timely, defensible, and understandable assessments.
- Communicate assessment effectively for action.” [100]

Figure 2-7 presents a model for the visual analytics process according to [98]. The process combines visual analysis, automatic analysis and human interactions with the goal to gain knowledge from data.

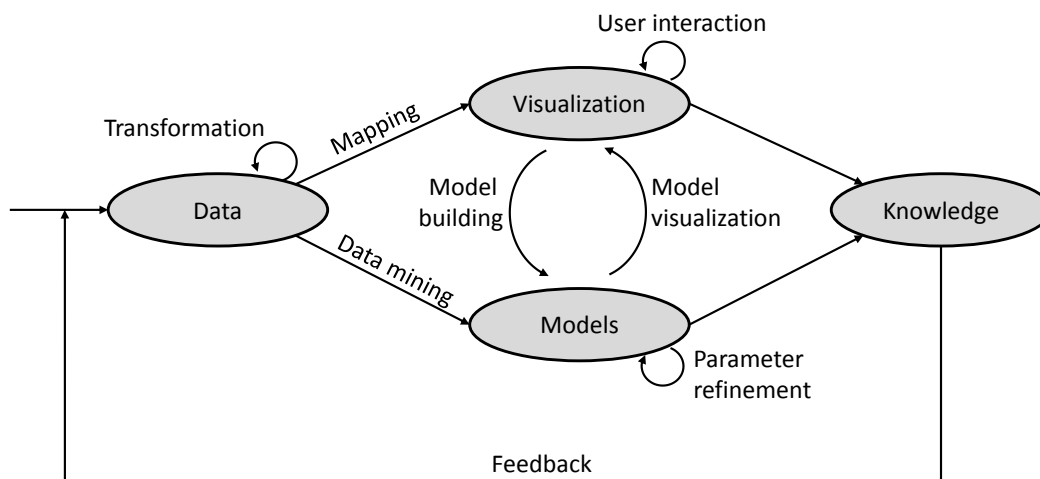


Figure 2-7 Model of the Visual Analytics process (adapted from [98]).

Data is usually preprocessed and can stem from several sources. The human designer can select either automated or visual analysis methods to learn about the data. For the visual analysis methods, the data is mapped to visualizations. Through interactions with the visualized data, like focusing on different regions in the data set or using different representations, the analyst can gain information. This information can be used to build models for automated analysis of the data. Models can also be built based on the original data using data mining techniques. Once models are built, the analyst can interact with

them to understand more about the data, e.g. by changing parameters or selecting different analysis algorithms. Findings can be verified through model visualization. “In the visual analytics process, knowledge can be gained from visualization, automatic analysis, as well as the preceding interactions between visualizations, models, and the human analyst. The feedback loop stores this knowledge of insightful analyses in the system, and contributes to enable the analyst to draw faster and better conclusions in the future” [98].

Software and Algorithm Visualization

Visualizations are also used in computer science for several decades. A movie by Knowlton [101] in 1966 is seen as the first attempt to visualize the behavior of software using animation techniques. Others followed, like the famous “Sorting out Sorting” movie [102], and were mainly developed for teaching. Main applications of software visualizations are research, teaching and program development [103]. Visualizations can, e.g. be *static* or *dynamic* (usually called animation then) and be generated *live* (during program execution) or *post-mortem* (visualization is generated from data collected during the program execution) [104]. Kerren and Stasko [105] give a short overview of the history and more recent advances of algorithm animation, a subfield of software visualization, with a focus on teaching. In [106] statistical evidence is given that algorithm animation aids in understanding, when employed properly. Messac and Chen [107] motivate to use visualizations of the optimization progress to increase its effectiveness and see the potential of algorithm visualization to make the human “understand the problem at hand, rather than discovering the intricacies of the optimization method” [107]. They present a method to visualize the progression of multi-objective optimization in real-time. Software visualization is usually done in a three step process of data acquisition, analysis and visualization. In interactive visualizations, the user can control at least one of these steps [108].

2.3.3.2 Current Visualization in CDS

Visualization is already used in CDS research and presented in publications. When describing algorithms for guiding the CDS process in research papers, usually static control flow graphs (see, e.g., [109]) are used for algorithms adapted from optimization. For tree-based search algorithms, the progression of the algorithm is often represented by showing the explored search tree for different stages of the process (see, e.g., [110]). In many publications on CDS applications, the progression of the synthesis process is shown either by metrics for convergence, or by showing the evolution of the Pareto set. This is usually done by presenting different plots for different stages of the synthesis process, e.g. at predefined iteration numbers. A drawback of these approaches is that even though the overall progression of the Pareto set is represented, there is no direct information on the algorithm’s behavior. It is not visible how the modification of a given design leads to a design with improved objective function values, or to a design with a new topology. Further, representing only archived designs, it cannot be determined, how often the algorithm explores but rejects inferior designs. Several approaches track information on how often the application of individual rules leads to accepted and rejected designs and use this

information for more intelligent search strategies [111]. These approaches are, however, directed at improving the synthesis results while the methodology presented in this paper aims at improving the understanding of algorithm and rules in the CDS process. One example where the visualization of synthesis processes is discussed can be found in [112]. The authors describe a scheme for transforming graph structures to numerical values and vice versa. A simplified transportation network synthesis task is chosen to demonstrate the use of their proposed scheme in an engineering context. A possible use of this scheme is to visualize the search tree that is explored by a synthesis algorithm based on the numerical values of the generated network structures during synthesis. The authors state that when an interactive mapping between the numerical values and graph structure representations would exist, this would “enhance the ability of the user to compare actual progress against expected and thus to identify problems or gaps in their understanding” [112]. Even though the topic of the publication is more general, this example shows the awareness of the authors that and how visualizations of the synthesis process can support designers’ understanding by comparing expected and actual performance visually.

Chakrabarti et al. [20] postulate that more attention should be given to education in the use of grammars for CDS. The successful application of visualization in the research domains described above motivates the use of visualization within this thesis.

2.3.4 Method and Tool Evaluation

To evaluate the research presented in this thesis, appropriate evaluation approaches are required. The following section describes current evaluation methods in the visualization research and the design community.

2.3.4.1 Visualization Research

Evaluation of visualization is a challenging issue. For **information visualization** systems, Plaisant [113] mentions that usability studies or controlled experiments are helpful to understand the potential and limitations of information visualization systems. More extensive evaluation would be desirable, however, this is a challenging issue. “By its very nature, by its very purpose, InfoVis presents fundamental challenges for identifying and measuring value. For instance, how does one measure insight? How does one quantify the benefits of an InfoVis system used for exploring an information space to gain a broad understanding of it?” [93].

Fekete et al. [93] discuss why it is challenging or even impossible to quantify the value of information visualization using metrics. The user of an information visualization system “simply may be examining the data to learn more about it, to make new discoveries, or to gain insight about it. The exploratory process itself may influence the questions and tasks that arise” [93].

In 1993, Price et al. [104] presented an influential taxonomy of **software visualization**. In contrast to previously existing taxonomies that are usually elaborated based on common characteristics of only a small amount of tools, their taxonomy is built on an established

black-box model of software. The proposed taxonomy is well-cited and suggests besides *scope, content, form, method, and interaction*, also the category *effectiveness* on the highest level. The authors mention, however, that *effectiveness* is a highly subjective measure and that there is only little research in the area of software visualization addressing the evaluation of software visualizations. They state that one reason for the few studies performed on evaluating software visualization is “the poor state of the art in software psychology, where there are no reliable methods for comparing programming environments” [104]. As future research challenges in software visualization they mention that the form in which information is presented to the user requires “empirical evaluations and proper psychological studies to determine which techniques are effective” [104]. More than twenty years later, the evaluation of software visualizations is still an issue. Seriai et al. [114] conducted a systematic mapping study which shows that there is still a lack in scientific evaluation of software visualization tools. Most researchers use only quantitative case studies, although the need for qualitative and quantitative evaluations is discussed [115, 116]. Further, evaluated methods are hardly compared to other software visualization tools (in 77% of the publications no comparison is found) and although the software visualization tools are developed to support human users, the majority of studies (70%) are done without human participants. The authors conclude that “the domain still has some way to go to reach the necessary maturity in validating its claims” [2].

2.3.4.2 Computer-aided Design Tool Research

Bracewell et al. [117] present a methodology for research into computational design tools based on the DRM [19]. The authors distinguish between two forms of method validation in Computer-aided engineering Design (CaeD) tool research, namely theoretical and experimental validation. The authors state that theoretical validation can be sufficient for some cases, e.g. “automated routine design tools, [where] there are explicit and commonly agreed criteria to assess the merit of the method” [117]. Software prototypes are, however, required to assess the effectiveness of a computational method in a practical sense.

2.4 Summary

In this chapter, an introduction to engineering design and CDS methods as approaches to support the conceptual design phase is given. The focus is then on grammar-based approaches to CDS and applications. To formulate more specific research questions, a literature review is carried out. Challenges for CDS using grammars are identified and the need for supporting the human designer in CDS using grammars is derived. Two research questions are formulated to support grammar development and application:

Research question 1: *How can the human designer be supported during grammar rule development?*

Research question 2: *How can the human designer be supported to understand interrelations between grammar rules and search algorithms during rule application?*

Research areas that contribute to address these research questions within this thesis are reviewed. These include optimization, compiler design and research fields concerned with visualization. A design decision is made to use visualization techniques to support the human designer in CDS. The current state-of-the-art for evaluating research on methods and tools is presented for visualization and computer-aided design tools. Defining meaningful evaluation methods is a recent issue in visualization research. In visual analytics, where automated analyses are combined with visualizations and human-computer interaction, it is argued that, to date, the value of visual analytics can only be expressed by success stories or subjectively by users of developed systems. Quantifiable metrics, however, do not exist. The decision to use visualization in this thesis, therefore, implicates that evaluating the research results is delicate. It is decided to conduct a theoretical evaluation based on case studies, one of the most common evaluation methods in software visualization [114]. These include tests of the practical applicability of the developed software prototype which is demanded for computational design tool research [117].

3 Method Overview

Figure 3-1 gives an overview of the methodology developed in this thesis. It consists of four methods which are briefly introduced in the following. Each method is represented by a comb in Figure 3-1. The comb structure symbolizes the individual methods but also their integration in a methodology. Further, the comb structure can be extended easily since the developed methodology does not claim to be complete.

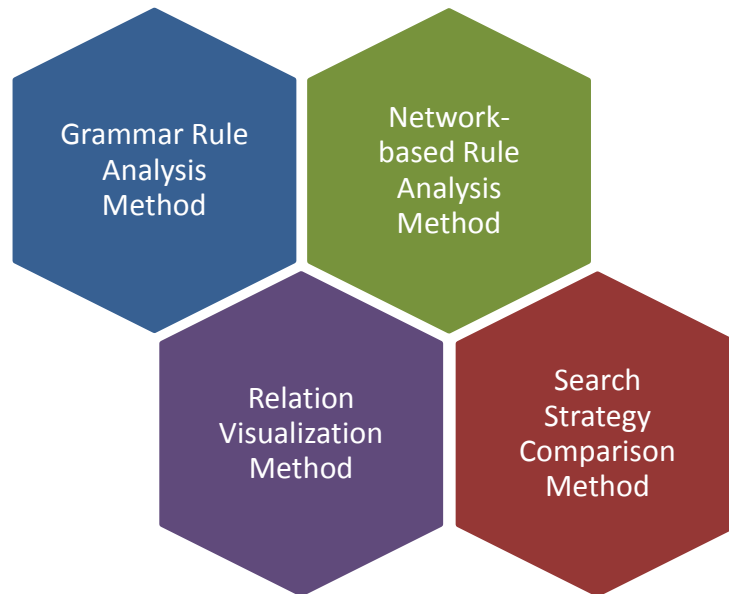


Figure 3-1 Methodology for supporting design grammar development and application.

The methods are presented in detail in separate chapters:

- A Grammar Rule Analysis Method (see Chapter 4)
- A Network-based Rule Analysis Method (see Chapter 5)
- A Relation Visualization Method (see Chapter 6)
- A Search Strategy Comparison Method (see Chapter 7)

All methods have in common that data is generated in a first step. Then, either during design generation or in a separate post-processing step the data is analyzed. Finally, the analysis results are presented to the human designer to provide insights on the synthesis process and the generated designs.

Although these three steps are common for all four methods, the purpose of the methods and the generated visualizations of analysis results vary. The following sections give an overview of each method to enable the reader to understand conceptual differences between the methods.

3.1 Grammar Rule Analysis Method

The Grammar Rule Analysis Method (GRAM) is aimed at supporting the human designer during grammar development. The focus is on the individual grammar rules. GRAM generates data by randomly applying rules starting from an initial design and accepting each rule application, i.e. no feedback from the evaluation is considered. For each rule, GRAM analyzes how the design's objectives and characteristics are changed due to the rule application. Charts are generated automatically and visualize these changes. GRAM gives a qualitative overview of each individual rule's performance and can be used to debug grammar rules. When using GRAM, expected contributions are improved grammar rules according to the designer's intention. Additionally, different implementations of rules can be compared and the designer is provided insights into the search space the grammar rules explore.

3.2 Network-based Rule Analysis Method

The Network-based Rule Analysis Method supports the development and application of grammar rules. The method generates designs in a first step, using either random generation or exhaustive generation of designs for a given number of rule applications. Then, the different designs are analyzed and synthesized to form a transition graph, or network. This network visualizes how the rules (edges) transform the designs (nodes). Network analysis identifies rules and rules sequences that lead to the same designs, rules sequences that have no effect on designs, or pairs of rules where one rule undoes what the other did. Additionally, interactive tools allow the designer to analyze the generated designs and transitions between them. These tools can, e.g., be used to analyze the effect that the location where a rule is applied has on the generated designs. Expected contributions are better rules through an improved rule development process and a more efficient application of sequences of rules. This is due to an increased understanding of rules and their application conditions and the identification of sequences that should be preferred or avoided.

3.3 Relation Visualization Method

The Relation Visualization Method is aimed at supporting the human designer in understanding relations between rules, generated designs and their objective function values. It further visualizes how search algorithms explore the search space for a given rule set. Therefore, it supports the human designer in the development of grammar rules and in selecting an appropriate search algorithm to guide the CDS process. The Relation Visualization Method analyzes designs that are generated when grammar rules are applied using a search algorithm. For each rule application, the changes in objectives and design characteristics are calculated. Further, the transitions between designs are analyzed and a transition graph is synthesized. The user is provided different visualizations that represent the changes in objectives and design characteristics for each rule. Interactive visualizations

permit the human designer to replay the synthesis process and visualize when different designs are generated during the synthesis process and how their performance values change. This facilitates the human designer's understanding of how the search algorithm explores the space. The Relation Visualization Method, therefore, supports understanding of search algorithms, understanding the interplay between a given rule set and an algorithm, as well as understanding the design problem at hand.

3.4 Search Strategy Comparison Method

The Search Strategy Comparison Method analyzes strategies that decide when to apply topologic and parametric rules during the CDS process. Various strategies for topologic and parametric rule applications exist but designers are often not aware of how different strategies influence the synthesis process. The Search Strategy Comparison Method is a means to analyze and compare different strategies using defined criteria. The CDS process is conducted for different strategies and the generated designs are tracked regularly and analyzed in a post-processing step. The human designer is then presented different metrics that allow a comparison of the different strategies. Using the Search Strategy Comparison Method permits a more informed selection of a strategy for topologic and parametric rule application for a given design task which can lead to a faster convergence of the search process or more diversity in the generated designs.

4 Grammar Rule Analysis Method

The use of generative design grammars for CDS has been shown to be successful in many application areas. The development of advanced search and optimization algorithms to guide the synthesis process is an active research area with great improvements in the last decades. The development of the grammar rules, however, often resembles an art rather than a science. Poor grammars drive the need for problem-specific and sophisticated search and optimization algorithms that guide the synthesis process towards valid and optimized designs in a reasonable amount of time. Instead of tuning search algorithms for superior grammars, the research presented in this chapter focuses on supporting the human designer during grammar development. The research aims to answer the following research questions:

Research question 4.1: *How can we systematically investigate the relations between grammar rules, objectives and design characteristics in CDS methods?*

Research question 4.2: *In this context, what information do grammar developers need to develop and refine grammars?*

Research question 4.3: *How can this information be visualized?*

A short motivation for grammar rule analysis is given in Section 4.1. The generic Grammar Rule Analysis Method (GRAM) is then presented in Section 4.2. Section 4.3 validates GRAM on a gearbox synthesis case study. Four different graph grammars for automated gearbox design are analyzed and compared using GRAM. In Section 4.3.4, the results are presented and discussed in Section 4.4. In Section 4.5, the method is summarized, research questions are revisited and the key contributions are presented.

4.1 Motivation for a Grammar Rule Analysis Method

The goal of the method described in this chapter is to take a step in the direction of systematically assisting the rule development process. It is meant to support designers of grammars by giving feedback on the performance of their developed rules through a set of visualizations, produced from systematic rule testing, that the designer can interpret and adjust the grammar rules accordingly. Throughout this thesis the term performance is used to describe how rules impact designs, e.g. how they change design characteristics and objectives. Testing the rules during or after the development process and before they are embedded in a more complicated design synthesis process enables designers to obtain an increased understanding of each rule's performance and to validate them. In 1956, Chomsky stated that a grammar "gives a certain insight into the use and understanding of a language" [118]. GRAM focuses on enabling these insights to allow the human engineer to design better grammar rules.

4.2 Method

GRAM analyzes a grammar under development in a systematic way to give feedback on how rules perform. GRAM is presented in Figure 4-1. Dark grey boxes represent steps that are carried out automatically in the current implementation, light grey boxes represent steps that can be automated in the future.

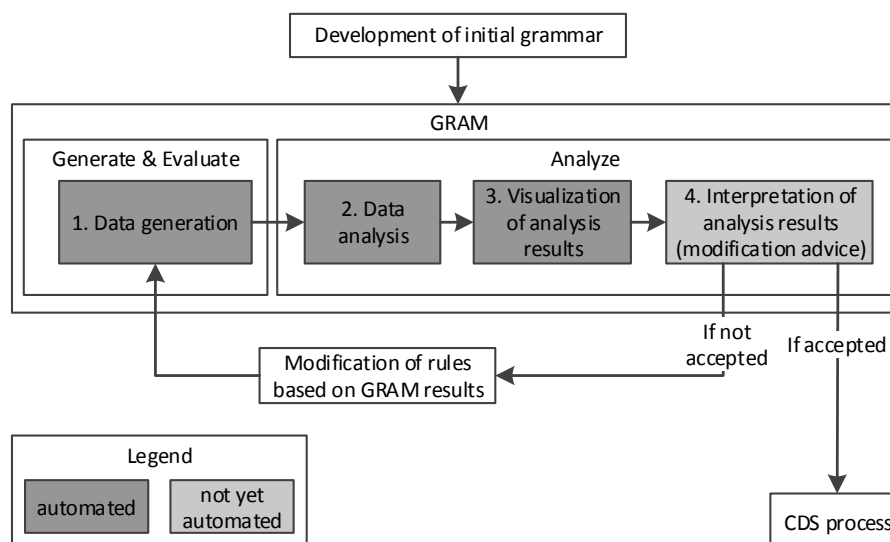


Figure 4-1 Grammar Rule Analysis Method (GRAM) to analyze grammar rules for CDS based on the extended SG process shown in [31].

Individual rule performances (Q1) as well as the performance of the whole rule set (Q2-Q6) are assessed such that the rule designer is able to answer the following questions when interpreting the results:

- Q-1. What impact does each rule have on each objective?
- Q-2. How probable are the applications of each rule?
- Q-3. What solution space do the rules define?
- Q-4. Does the rule set favor certain designs in design generation?
- Q-5. How many valid designs are generated?
- Q-6. How many different designs are generated?

GRAM has a defined way to generate and analyze data. Information from the data analysis is visualized and interpreted by the designer to gain a better understanding of the grammar itself. The different steps in GRAM are described in more detail in the following and a schematic representation of the GRAM steps 1-3 is shown in Figure 4-2 to accompany these descriptions. GRAM is best illustrated using an example, so a grammar for gearbox synthesis is used here that is further introduced in Section 4.3.1. GRAM supports the analysis for any number of objectives, design characteristics and rules but for the sake of clarity it is shown here for this reduced example.

4 Grammar Rule Analysis Method

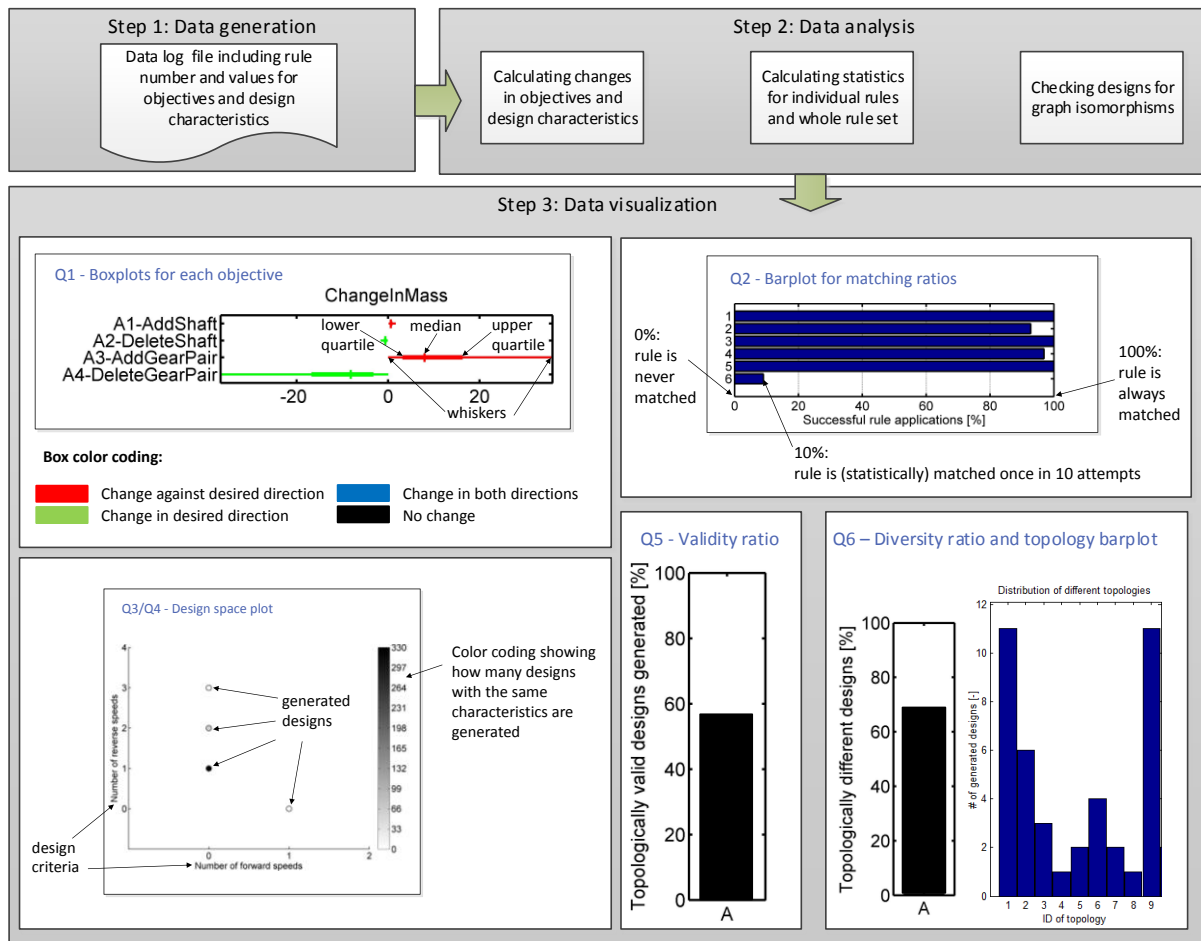


Figure 4-2 Schematic overview of the GRAM steps 1 - 3.

4.2.1 Data Generation

To analyze grammar rules, a variety of data, i.e. objectives and design characteristics, is required. The data is generated using a simple generate-and-test process. It starts with an initial design. A rule is selected randomly from all implemented rules. It is applied, the generated design is evaluated and the data is stored. The generated design resulting from this rule application is taken as the basis for the next iteration. It is not a generate-and-test search process as the design resulting from the rule application is always used as the starting point for the next rule application regardless of impact on design objectives. In most engineering applications there are multiple objectives and it is recommended to store the metrics for each objective individually. Here, also constraints formulated as soft constraints, i.e. penalty functions, can be included. Design characteristics are commonly system characteristics, e.g. the number of components and component types.

4.2.2 Data Analysis

For all data, the change in the objectives is calculated to analyze the performance of each individual rule. The generated designs are analyzed to identify topologically equivalent designs to be able to represent the design space and to identify if the rules favor certain

topologies, i.e. generate them multiple times. Additionally, some basic statistical models are built to prepare the visualization and support the interpretation.

4.2.3 Visualization and Interpretation of Analysis Results

Five different diagram types are presented in Figure 4-2 to visualize the data obtained in the analysis. For a rule set of n_r rules and an analysis of n_o objectives using n_d design characteristics, the following diagrams are generated: Q1) n_o boxplots with n_r boxes each, Q2) one barplot with n_r bars, Q3/Q4) one n_d -dimensional design space plot, additional boxplots if required, Q5) one ratio for valid designs and Q6) one ratio for different designs, one additional barplot if required. The diagrams are explained below and important issues for their interpretation are discussed.

Q1 - General performance analysis using boxplots for each objective. For each objective a diagram is generated showing how it is influenced by each rule (given on the y-axis). The user defines the desired direction of change derived from the problem formulation and a color coding gives a quick overview if the rule changed the objective in the desired direction or not. The red (medium grey) color indicates a change against the desired direction and the green (light grey) color a change in the desired direction. Blue (dark grey) boxes show that changes in both directions are possible and black (black) is used to represent rules that have no influence on an objective. The whiskers, defined by the thin line, represent the maximum and minimum value of the dataset excluding outliers, i.e. data points more than 3/2 away from the lower or upper quartile. The box spans from the lower quartile to the upper quartile showing also the median. Using this diagram, the engineer can visualize the performance of each rule considering each objective separately. Interpreting these diagrams, the designer has to consider that changes against the desired direction, e.g. increasing an objective rather than decreasing it, can be valuable for design synthesis. This means that changes against the desired direction do not automatically identify inferior rules that should be removed. In contrast, it encourages the rule designer to think also about the sequences in which rules can be applied and to consider combining these sequences to create more specific rules to facilitate the generation of beneficial designs.

Q2 – Bar plots to represent matching ratios for each rule. For more detailed information on a rule's applicability, matching ratios are calculated and visualized. Throughout this thesis, the matching ratio of a rule is defined as the number of LHS matches of a rule divided by the number of attempts to apply this rule. This ratio defines how likely it is for a rule to be applied with a matching ratio of 100% meaning a rule can always be applied while a matching ratio of 0% represents a not applicable rule. From the matching ratios, the rule designer can reason about the LHSs of the rules. This often helps to explain the design space that is generated with the rule set. Rules that have a very low application probability are only rarely applied. In grammars with unbalanced rule application probabilities, i.e. some rules are applied very often, others very rarely, the rule designer can, for example, consider formulating the LHSs of rarely applied rules differently to allow their application more often. Additionally the use of search strategies or predefined sequences for the CDS process can be

helpful to improve the rule's application. The interpretation of matching ratios is dependent on the rule design as well as the search and optimization algorithm used later for design synthesis. When using intelligent search methods, it may not be required to ensure higher matching ratios for all rules, whereas when using simple generate-and-test type algorithms, this may be more helpful to explore the design space.

Q3/Q4 - Visualization of the design space. To show the size of the design space, a matrix with the dimensions of the design characteristics 1 and 2, is presented. Each point in the space indicates that a design with, e.g., x elements of design characteristic 1 and y elements of design characteristic 2 exists. The color indicates how often a design with the respective characteristics is generated. This plot gives an indication about how the rules are used to generate the design space. The color can be used to identify solutions in the design space that are favored by the rules ("hot spots"). When continuous design characteristics are required or when the user is interested in additional information on objectives, the x - and y -axes can be made continuous or boxplots for each objective and design characteristic can be made in addition to the design space representation. The visualized space is generated using random generation without feedback. The rule engineer has to consider this when interpreting the results. It can happen that the space is larger than intended, e.g. when the designer allows invalid designs and plans to use penalty functions and an optimization algorithm for the CDS process and these penalized designs are not removed from the design space yet. On the other hand, it can also happen that the generated design space is small because certain rules undo what previous rules did. To derive useful measures to improve a rule set, the rule designer has to consider not only the space explored and the favored designs during the data generation process in GRAM, but also the search and optimization process that will be used.

Q5 – Validity ratio. The validity ratio is defined here as the number of valid designs divided by the number of total designs generated. The validity of a design is defined by the designer and can be, for example, the necessity to have a connection between two components, e.g. a connection between the input and the output shaft in the gearbox example. The validity ratio gives feedback on the probability that the analyzed grammar generates valid designs with simple generate-and-test type algorithms. The lower the validity ratio is, the more intelligent the guidance has to be to lead the grammar rule application to produce feasible designs. On the other hand, a low validity ratio does not mean that a grammar necessarily produces inferior results compared to a grammar with a validity ratio of 1, i.e. generating only valid designs. In some cases it is required to generate invalid intermediate designs to be able to eventually transform an invalid design into a valid one. It is the designer's choice to decide whether or not invalid designs should be allowed during design generation for a specific problem formulation and to interpret the validity ratio accordingly.

Q6 – Diversity ratio. The diversity ratio is defined as the number of valid and topologically different designs generated during the data generation phase divided by the number of all valid designs generated during the data generation phase. A high diversity ratio means that the grammar generates topologically different designs with a high likelihood, i.e. the design

space is more easily explored than when having a lower diversity ratio and generating the same designs repeatedly. The rule designer has to be aware of the fact that the diversity ratio reflects only the design space explored during the data generation process. In most cases the entire design space is unknown and, when using parametric rules, can be infinite. To further analyze the generated designs, additional bar charts can be used to indicate for each topologically unique design, how often it is generated. It permits, e.g., to distinguish between two grammar rule sets with the same diversity ratio in more detail. Assume two rule sets that both generate 100 valid designs with five different topologies. The first one generates four of these topologies only once and the fifth topology in all other cases. The second rule set generates each of the five topologies 20 times. Both rule sets have a diversity ratio of 0.05. However, there is obviously a difference in how they explore different topologies that the human designer might be interested in. Bar charts provide this information and can therefore give further insights into the tendency of the grammar rules towards exploring and exploiting different topologies.

Using these diagrams, designers can check if the grammar represents the intended design language and interpret the relative ease of generating known, intended designs and they can further improve the grammar considering the analysis results.

4.3 Case Study: Gearbox Synthesis

To show the applicability of the proposed method, GRAM is applied to four different grammars for automated gearbox synthesis. Since the gearbox case study is used throughout this thesis to validate the different methods, it is introduced once in the following in more detail. The following chapters then refer back to this description and only mention differences or deviations where applicable.

4.3.1 Introduction to the Gearbox Synthesis Case Study

Gearbox design using generative grammars is an established CDS problem and research has been carried out by several scientists [27, 34, 38, 119-122]. In this thesis, all grammars for gearbox synthesis are formulated and implemented as graph grammars consisting of a metamodel and a rule set to synthesize designs. The rule set can contain both topologic and parametric rules.

4.3.1.1 Metamodel

A metamodel describes all elements that can be used as building blocks within a generative grammar [52], also known as vocabulary. For the gearbox case study, the metamodel consists of three different node types for gears, shafts and the bounding box and one directed edge type to connect nodes. An overview of the metamodel is given in Figure 4-3. All nodes have parameters to specify the components they represent, e.g. diameter, gear width and position for gears. As an initial design of the gearbox, an input and output shaft are given including their distinct position. Additionally, a bounding box defines the spatial boundaries in which the gearbox has to fit. An edge between a gear and a shaft indicates

that the gear is mounted on the shaft and an edge between two gears indicates that the two gears are in mesh. Thus, each path in the graph starting from the input shaft and going to the output shaft indicates a possible power flow path through the gearbox, i.e. it represents one speed. The internal representation uses directed edges between nodes to identify the power flow from input to output shaft.

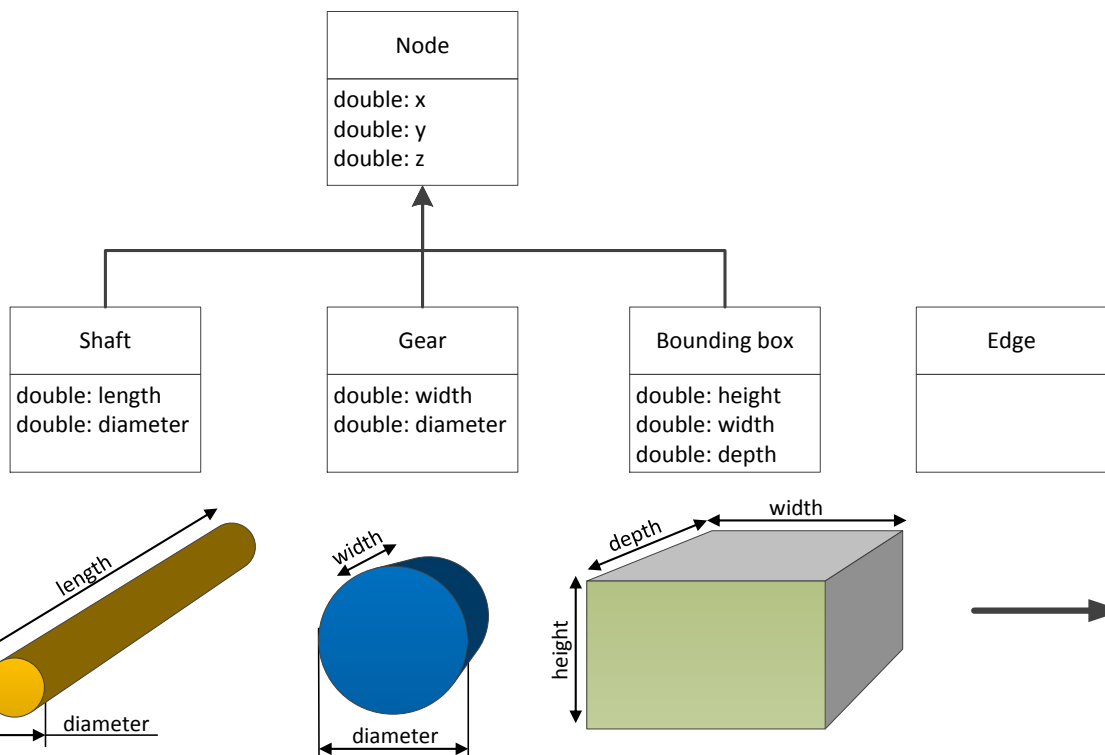


Figure 4-3 Metamodel for the gearbox case study.

4.3.1.2 Implementation of the Rule Sets

Throughout this thesis, the rule sets for synthesizing gearbox designs are implemented as graph grammars using GrGen. Different rule sets are used in Chapter 4. All other sections using the gearbox synthesis task as a case study make use of the same rule set which is why this rule set is explained in more detail in the following.

The rule set consists of five topologic and five parametric rules as illustrated in Figure 4-4. It is an extension of the rule set described in [38] including rules 9 and 10 to shorten and lengthen shafts.

Topologic Rules

The following topologic rules add, delete or replace nodes in the graph and their corresponding edges.

Rule 1 - Create a new shaft: A new shaft and two new gear pairs are created to connect the new shaft to two existing shafts.

Rule 2 - Delete a shaft: A shaft is deleted including all gears connected to it. In case the deletion of the gear set creates dangling nodes in the graph, i.e. nodes that cannot be included in any path from input to output shaft, these nodes are deleted as well.

Rule 3 - Create a new gear pair: Two shafts are selected and a new gear pair is created to connect them.

Rule 4 - Delete a gear pair: The chosen gear pair is deleted. Similar to Rule 2, dangling nodes in the graph are deleted.

Rule 5 - Replace a gear pair: A gear pair is deleted and a new shaft and two new gear pairs are created to replace the previously existing connection between the two shafts.

Parametric Rules

Parameters such as diameter and spatial position of a gear or shaft are stored as attributes within the respective node. Parametric rules change these parameters.

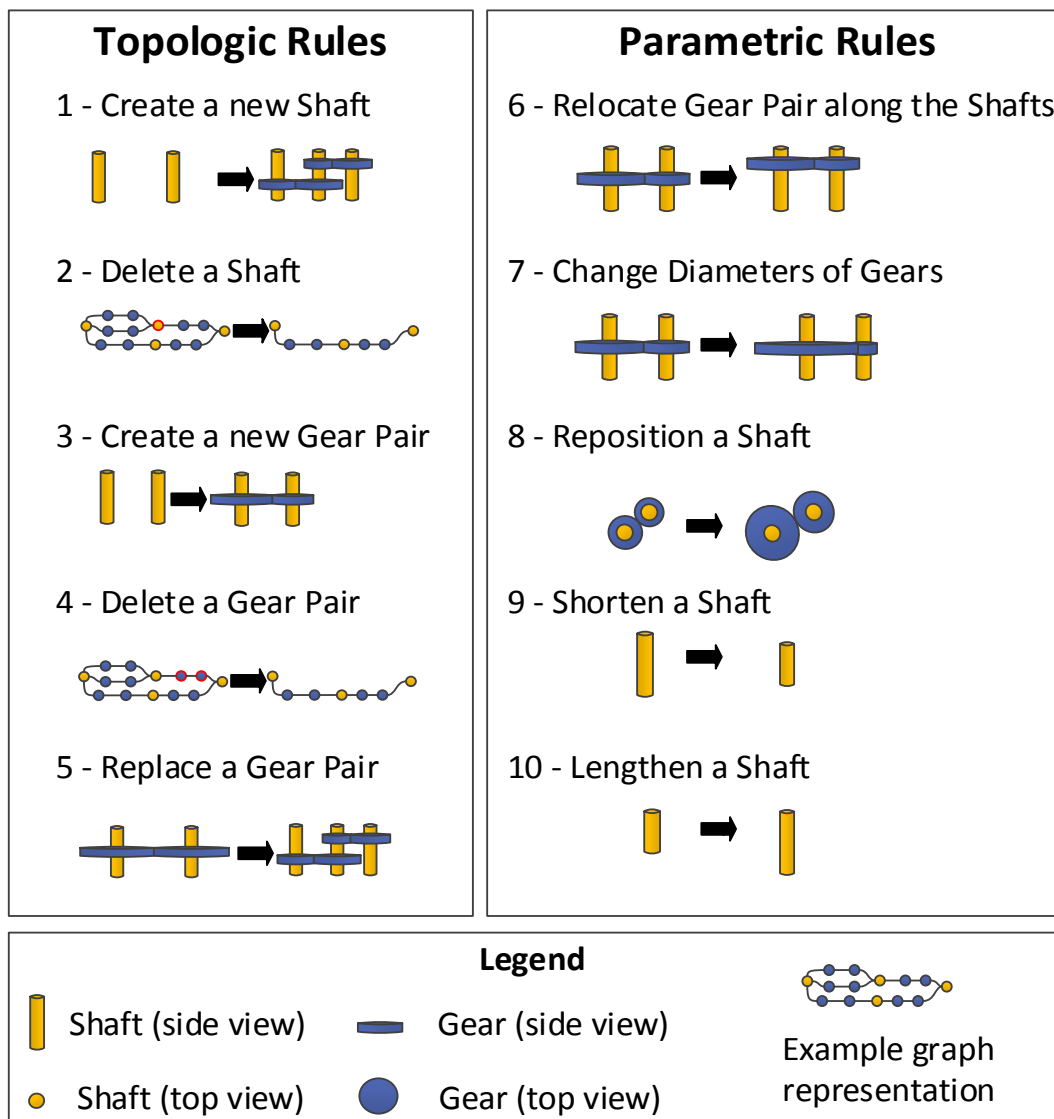


Figure 4-4 Schematic representation of the gearbox synthesis rule set.

Rule 6 - Relocate gear pair along the shafts: The position of gears along their respective shafts is changed.

Rule 7 - Change diameters of gears: The gear diameter is changed.

Rule 8 - Repositioning a shaft: The position of shafts in the x-y plane is changed. The diameters of affected gears are adapted accordingly such that the gears mesh.

Rule 9 - Shorten a shaft: The selected shaft is shortened.

Rule 10 - Lengthen a shaft: The selected shaft is lengthened.

The rule set is developed to only generate topologically valid designs when rules are applied to a valid initial gearbox design. This means that at least one connection between input and output shaft of the gearbox exists and that there are no mechanical deadlocks.

4.3.1.3 Evaluation of Gearbox Designs

The objectives defined in this case study are a) the total mass of the components and b) the amount of collision, a metric calculated based on axial and radial overlap of all components.

All possible power flow paths are identified by analyzing the graph using a depth first search. Doing so, the **number of speeds** can be calculated. The direction of speeds is analyzed by counting the number of gear pairs on the power flow paths.

The gear ratios are calculated accordingly multiplying the ratios of each gear pair on a power flow path. The **ratio error** is then calculated as suggested in [38]:

$$\text{ratioError} = \max(\text{abs}(\log_{10}(a/b))) \quad (1)$$

where a is a vector of the actual gear ratios, b is a vector of the desired gear ratios and the division (a/b) is done element wise.

A **metric for collisions** between components is calculated considering the radial and axial overlap between components, i.e. between a) gears and shaft, b) gears and gears, or d) components and the bounding box. All calculated overlaps are added to create one metric. For further information and the exact formula, please refer to [38].

The **mass** of the design is calculated by adding the individual component masses that are defined by the components' geometry and their specific density.

For this case study, a **valid design** must have at least one speed, i.e. there must be at least one path in the graph connecting the input and output shaft. Figure 4-5 visualizes this definition of valid designs for the case study.

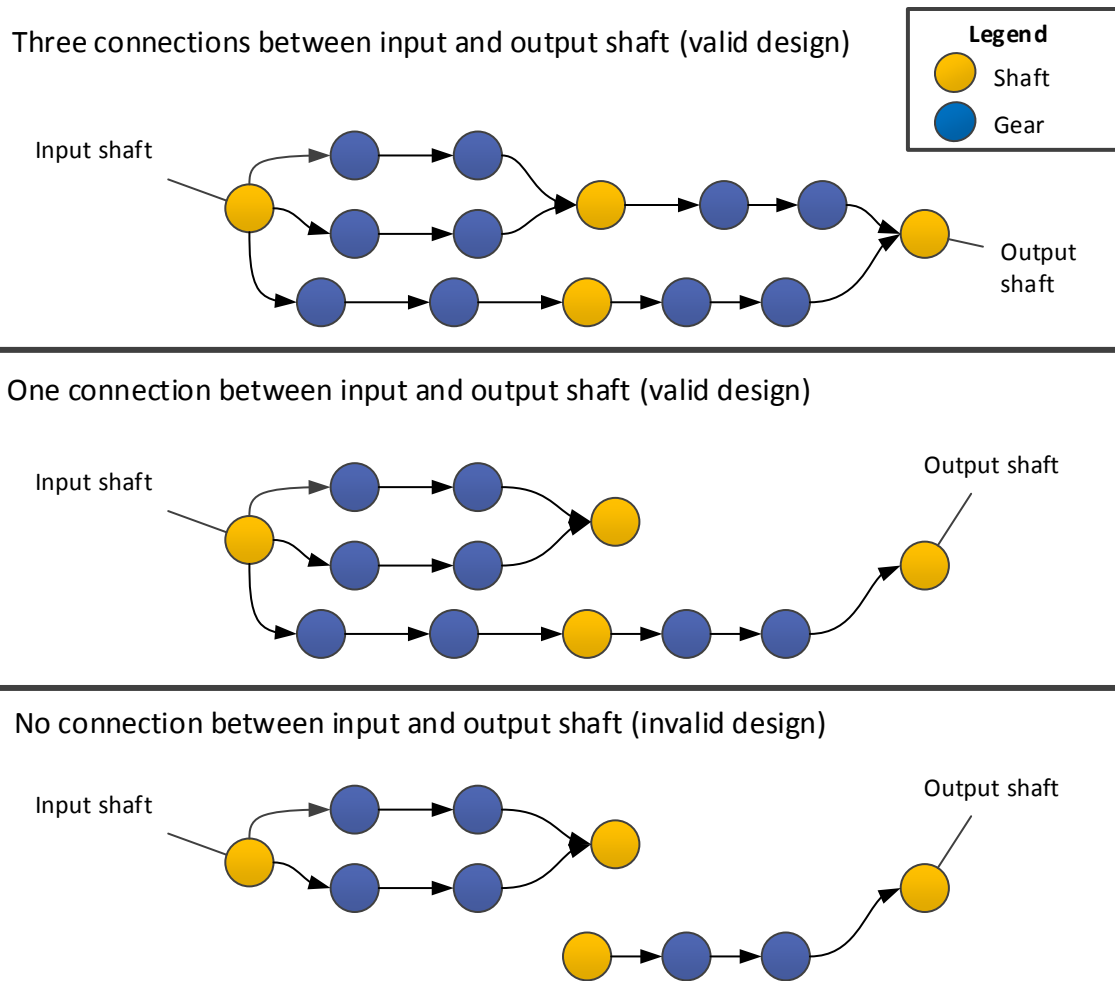


Figure 4-5 Three examples to show how valid designs, i.e. designs having a connection between input and output shaft, are defined.

A small example of a sequence applying three different rules to an initial design is given in Figure 4-6 (top), also showing the graph, a 3D representation for the designs generated and the corresponding objective values for mass and collisions and the design characteristics number of forward speeds and number of reverse speeds (bottom).

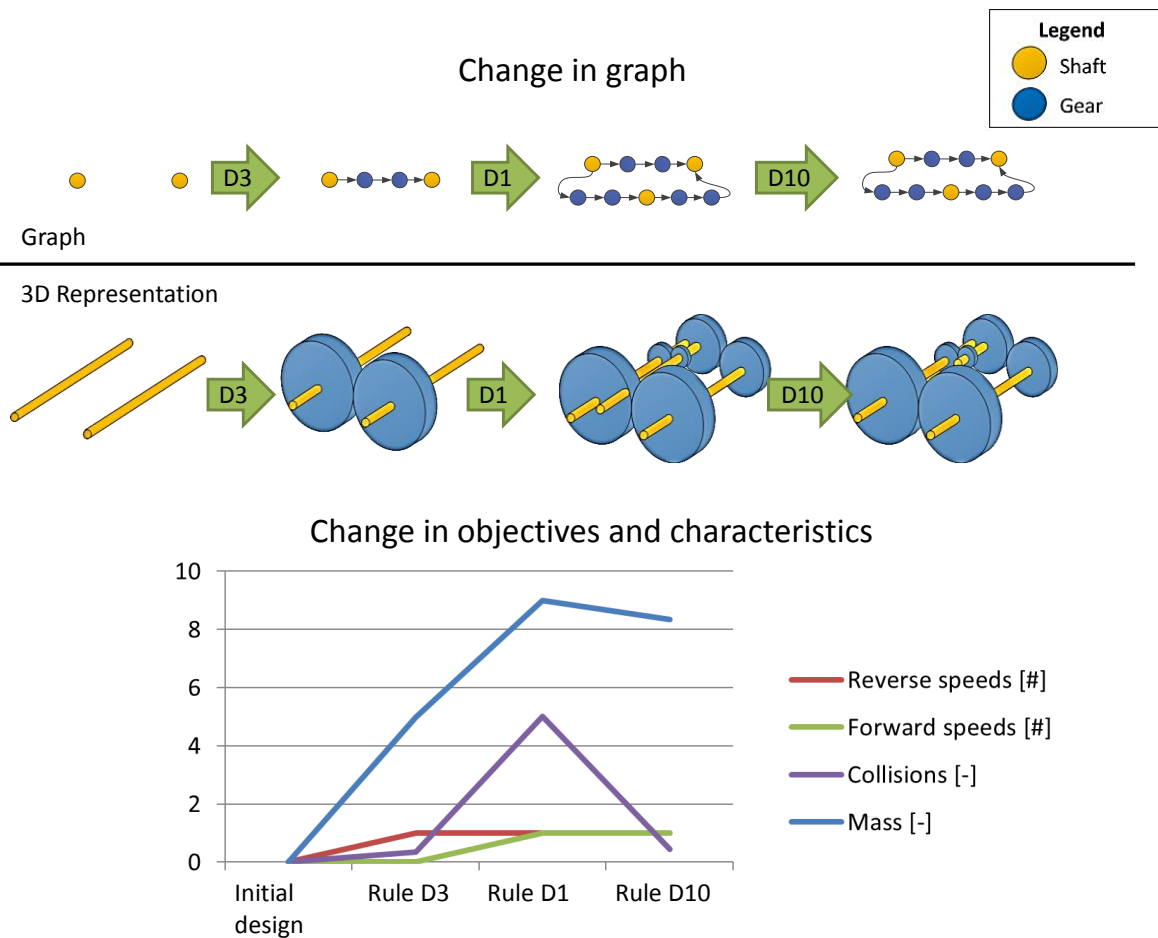


Figure 4-6 Example of applying a sequence of three different rules; changes in the graph and a 3D representation (top), the respective objective values and design characteristics (bottom).

4.3.2 Gearbox Rule Sets to Validate GRAM

To validate GRAM, four different rule sets are implemented as graph grammar rules in GrGen. An overview is given in Figure 4-7. The schematic images for the rules are for visualization purposes only. The schematic graph representations for rules C2, C4, C6, D2, D4 and D6 represent the basic idea of the rule, but not the exact LHS matches. Nodes of the LHS are, however, marked in red (medium grey) in the example graphs. The first two rule sets are based on the work by Starling and Shea [27, 120] and consist of four (rule set A) and 21 (rule set B) rules respectively. Rule set B is an extension of rule set A, adding several rules to change the dimensions and position of gears and shafts and two additional rules that add and remove components. Rule sets A and B were originally developed to generate watches and a winding mechanism in a camera, i. e. requiring only one speed. The third rule set (rule set C) is based on the work by Lin et al. [38] for automotive gearboxes. It considers only parallel shafts that extend to the width of the bounding box and consists of nine rules that are more sophisticated than those of rule set B in both their LHS and RHS. Rule set D is an extension of rule set C. It adds two rules to change the length of shafts and is different from rule set C in that the LHS of several rules account for changed lengths of shafts. Rule sets C and D were originally developed for automated gearbox synthesis, i.e. considering multiple speeds. Rule sets C and D were developed to generate valid designs with every rule

application as long as the initial design is valid. This decision was made by the rule designer to ensure design evaluation after every rule application.

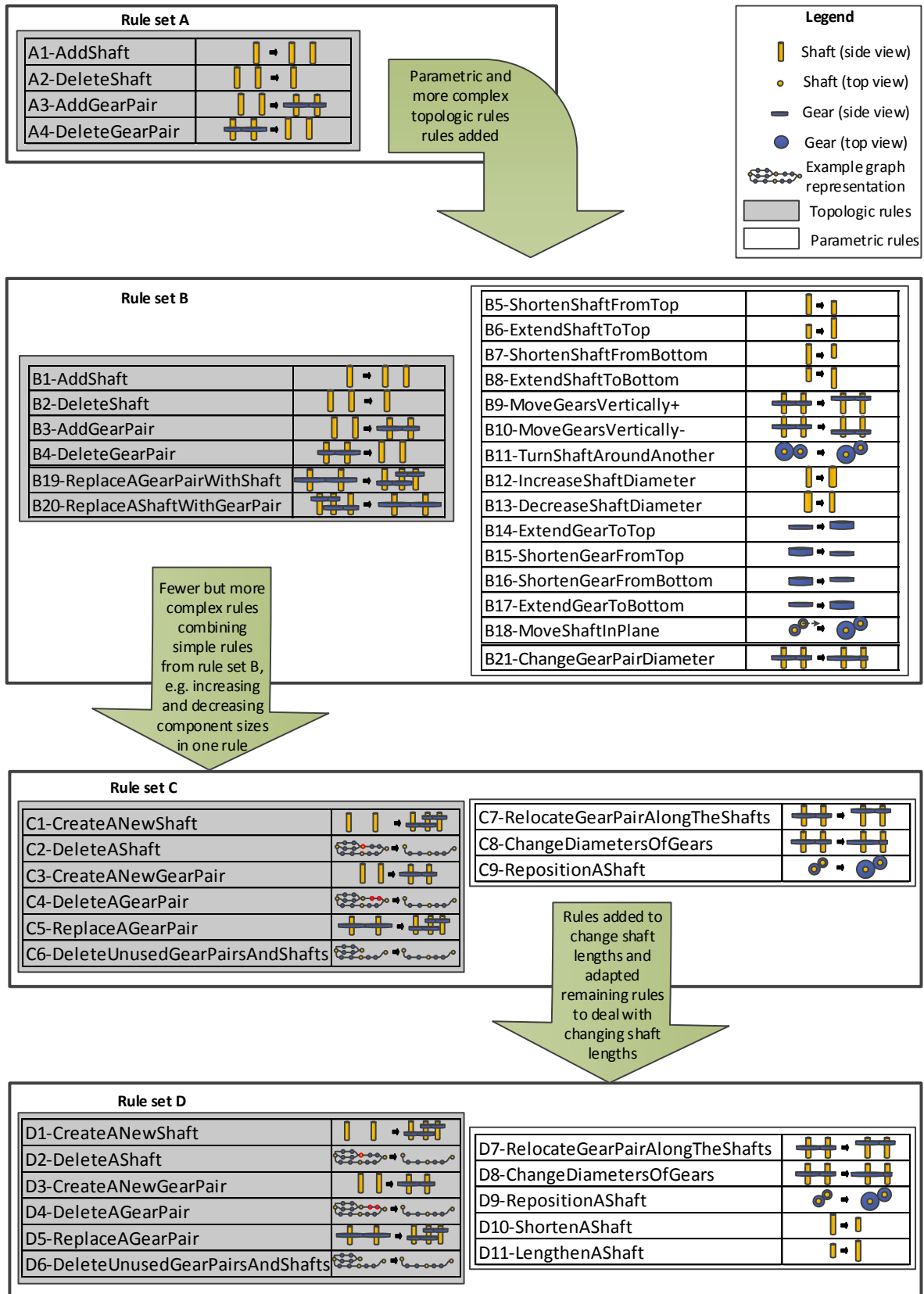


Figure 4-7 Overview of all rule sets organized by their type (topologic or parametric). Rule number (consisting of the rule set label and a number), name and a pictorial description are given as well as the main differences between the grammars.

4.3.3 Application of GRAM to the Gearbox Rule Sets A – D

The data generation is conducted with 50 times 1,000 rule applications for each rule set. **Data generation** is carried out using a gearbox synthesis system developed by the authors based on GrGen, an open source graph rewriting tool [50] (<http://www.grgen.net>). The objectives defined in this case study are a) the total mass of the components and b) the amount of collision, a metric calculated based on axial and radial overlap of all components (see [38] for the exact formula). The number of forward speeds and the number of reverse speeds are defined as design characteristics. The initial design is a bounding box containing an input and an output shaft. **Data analysis** and **visualization** are carried out using Matlab and the graph isomorphism check to identify topologically identical designs is carried out using GrGen.

4.3.4 Results

The results from the case study are presented below. Interpreting the analysis results, the questions Q-1 to Q-6 can be answered.

Q1 – What impact does each rule have on each objective?

The influence of each of the rules in rule sets A-D is shown in Figure 4-8. Adding components (rules A1 and A3) always increases (red color in boxplot) mass and collisions, deleting them (rules A2 and A4) reduces (green color in boxplot) both objectives.

This performance can also be clearly seen for the first four rules in rule set B, however there are also rules that have no influence on an objective (black color in boxplot) or can either increase or decrease an objective value (blue color in boxplot). Looking at the influence of each rule on collisions, it can be seen that although some rules do not influence the mass of a design, as for rules B9 and B10, where gears are moved in the z-direction, the collisions are influenced by each of them.

Rules C1-C4 in rule set C show a do-undo behavior, where rules C1 and C3 add mass and collisions by adding components, rules C2 and C4 reduce both objectives. The similarity between rule C5 and rule B19 that both replace a gear pair can be seen in the boxplots. Comparing rules A1-A4 and B1-B4 respectively, to rules C1-C4, a difference in the magnitude of the change in both objectives can be seen. This stems from the different implementation in rule set C. Rule A1, for example, only adds a single shaft, whereas rule C1 adds a shaft and connects it to the existing design with a gear pair, i.e. more components are added with one rule application which results in bigger changes of both mass and collisions.

Results from rule sets C and D look very similar except for the additional rules D10 and D11 that shorten and lengthen shafts to reduce mass and collisions (D10) or to give the possibility to connect two shafts with a gear pair (D11). However, there is an additional difference between rules C6 and D6, as rule D6 has no influence on either mass or collisions. If the rules are implemented correctly, this should not occur. In this case it stems from a careful implementation of rule set D ensuring that after every rule application no dangling nodes

remain in the design, so this rule from rule set C, intending to repair designs, is not necessary any more.

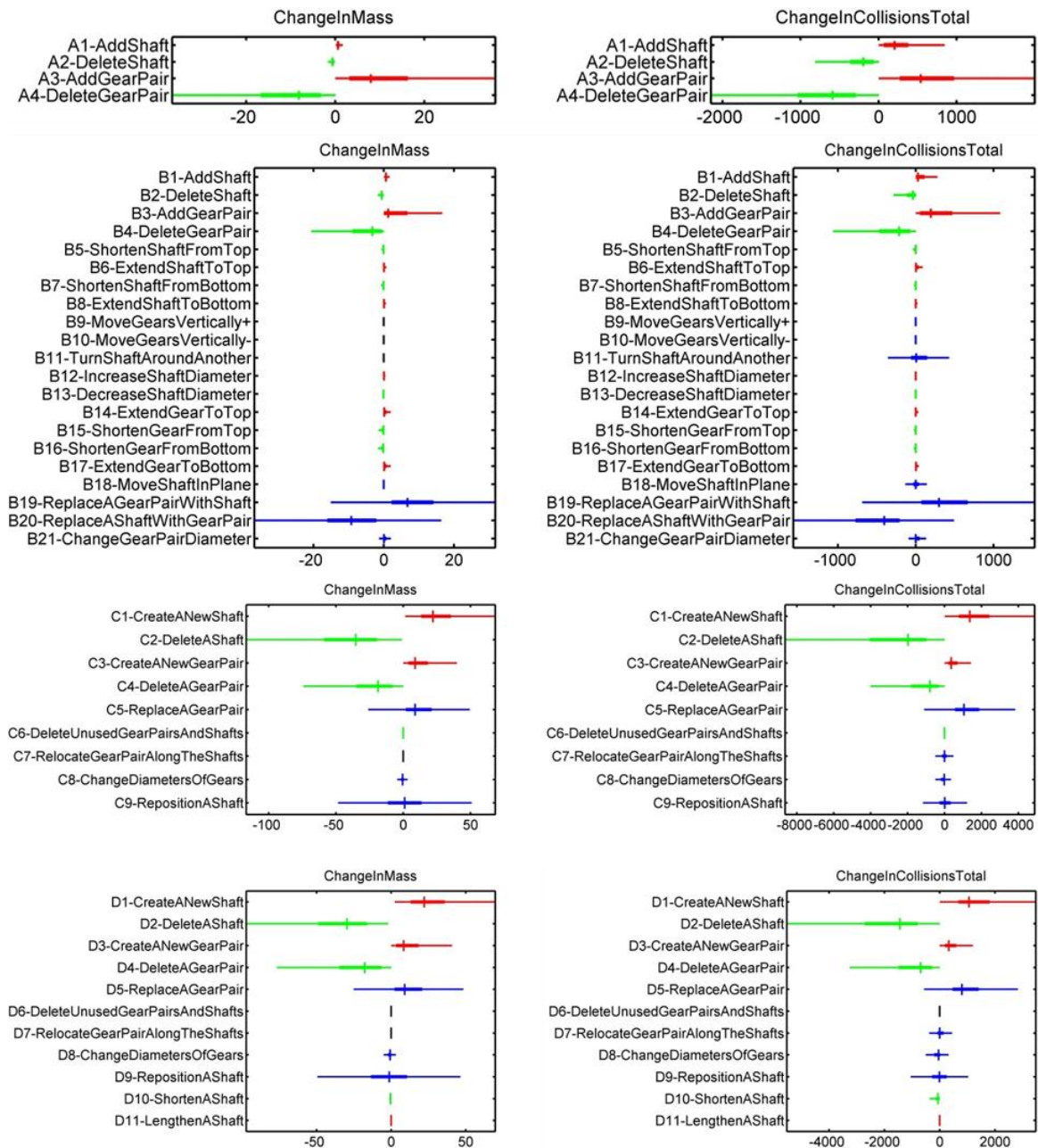


Figure 4-8 Influence of rules on the objectives mass and collisions for rule sets A–D (top to bottom).

Q2 - How probable are the applications of each rule?

For all rules in rule sets A-D, matching ratios are represented as horizontal bars in Figure 4-9. Rules with simple LHSs are applied more frequently than those with more restrictive LHSs caused by constraints, e.g. on parameters of the nodes, or component relations, i.e. more complex sub graphs. This can be seen, for example, in rule set A where the rule to add a shaft (A1) and to connect two shafts via a gear pair (A3) can always be applied, whereas rule A4, which removes a gear pair between two shafts, is rarely matched because there are more rules that add and delete shafts than there are to create gear pairs (rule A3). So, randomly applying rules, the probability to apply rule A4 is lowered just by the fact that

there are more rules that prohibit its application than there are to enable it. Having more rules that influence the LHS of this rule lead to more matches. This can be seen in the plot for rule set B, where the exact same rule (B4) is applied with an almost three times higher matching ratio, due to one more rule in the rule set that generates a LHS match (B19).

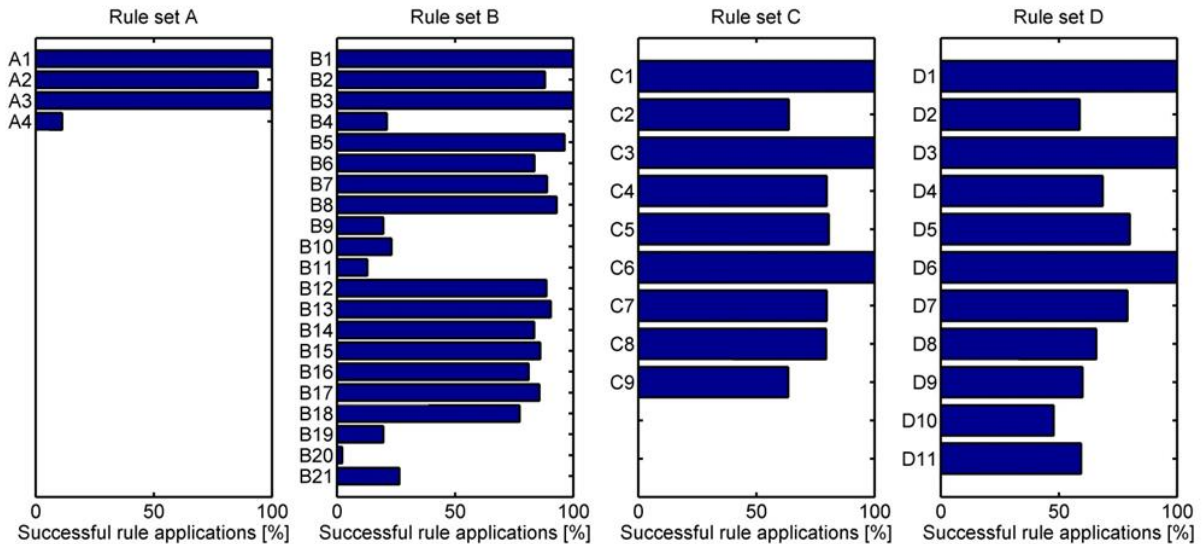
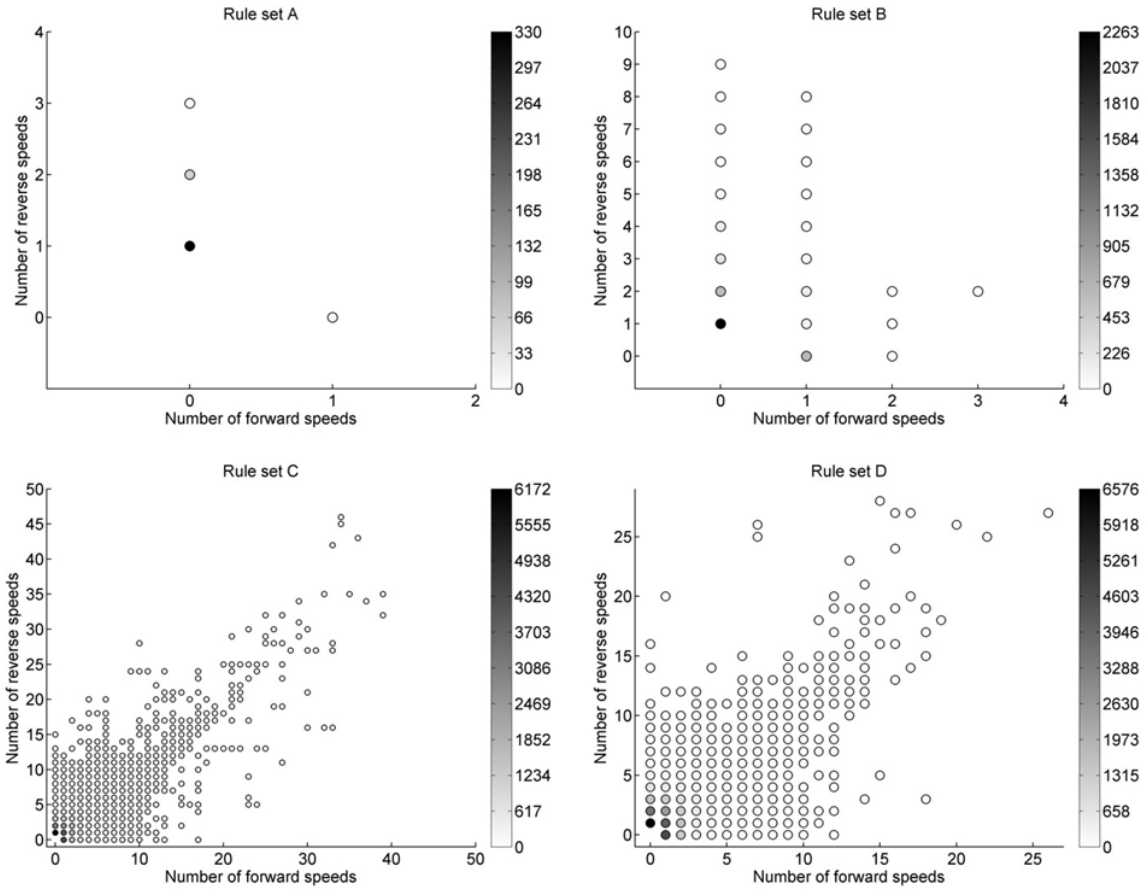


Figure 4-9 Percentages of successful rule matches for rule sets A–D.

Q3 - What solution space do the rules define?

Figure 4-10 shows the design space generated by each of the rule sets in 50 runs applying 1,000 rules selected randomly. The two design characteristics, i.e. the number of forward and reverse speeds in a design, are plotted against each other. The design spaces of rule sets A and B are small. This can be explained by the simple grammar rules that are more dependent on an anticipated intelligent guidance and often do not produce good designs when applied randomly. Rule set B generates more speeds in general, which can be explained by the additional rule to connect shafts, i.e. rule B19. Rule sets C and D, with their rules developed to perturb, but not destroy, existing solutions, generate designs with higher numbers of speeds. Comparing the two, rule set C generates more designs with a higher number of speeds. This can be explained by the higher fraction of topological rules in rule set C that leads to more changes in topology when rules are applied randomly and thus makes it possible to explore the design space more in this respect. Additionally, rule set D has more restricted LHSs of its topological rules allowing, for example, only shafts that have an axial overlap to be connected via a gear pair.



4

Figure 4-10 Plots of the design spaces generated by rule sets A-D. The color indicates how often a design with this speed configuration was generated in 50,000 rule applications.

To compare not only topologic, but also parametric aspects of the design space, the average mass and collisions metrics for all designs generated by the four rule sets are visualized in Figure 4-11. These representations support the points discussed. Rule set A generates many designs with a high number of components, thus leading to high mass and collisions. Rule set B has lower values for both due to rules that allow also parametric changes. The same effect can be observed with rule sets C and D, where the addition of more parametric rules in rule set D, i.e. rule D10 to shorten shafts, leads to lighter designs with less collision.

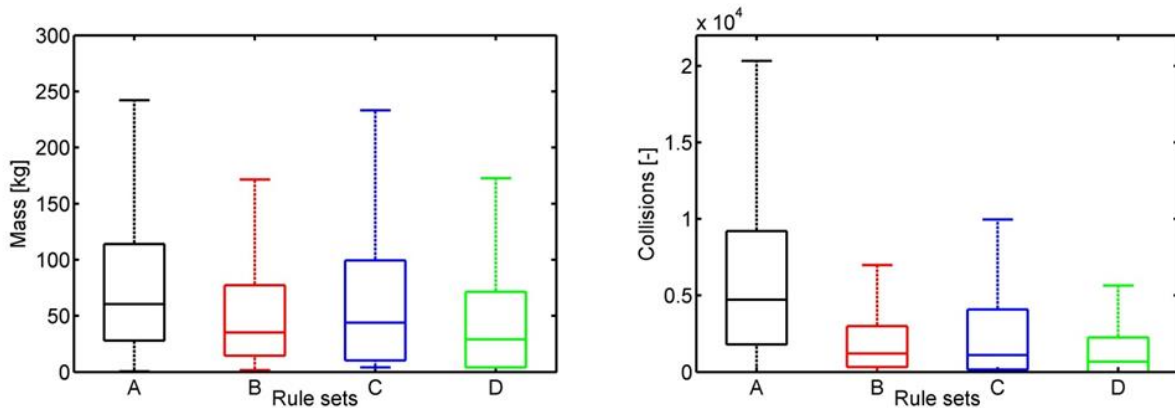


Figure 4-11 Average mass and collisions of all generated designs for rule sets A-D.

Q4 - Does the rule set favor certain designs in design generation?

From the coloring of generated designs in Figure 4-10 it can be seen that all four rule sets favor designs with few forward and reverse speeds. Rule sets A and B generate designs with a high number of reverse speeds, e.g. by directly connecting input and output shaft via a gear pair (rule A3, B3). Rule sets C and D do not only favor reverse speeds, but also generate designs with high numbers of forward speeds due to rules that easily introduce forward speeds when applied to input and output shaft (rule C1, D1) or change the direction of an existing speed (rule C5, D5).

Q5 - How many valid designs are generated?

On the left of Figure 4-12, the validity ratio is given. It can be seen that this ratio increases from rule set A to B to C as the number of rules to connect shafts grows. Rule set D produces fewer topologically valid designs than rule set C, which is caused by a slightly lower probability to change a topologically invalid design into a valid design. This is due to its smaller portion of topological rules and their reduced chance of application due to their more restrictive LHSs. Rule sets C and D are developed to generate valid designs only when applied to a valid design. For this case study, however, the initial design for all rule sets is invalid, as it contains only the input and the output shaft, causing the generation of invalid designs for rule sets C and D.

Q6 - How many different designs are generated?

On the right of Figure 4-12, the diversity ratio is shown. Comparing the rule sets underlines what has been found in the design space diagrams. Rule sets A and B produce many designs of the same topology. Rule set C has the most topologically different solutions and rule set D produces fewer different solutions.

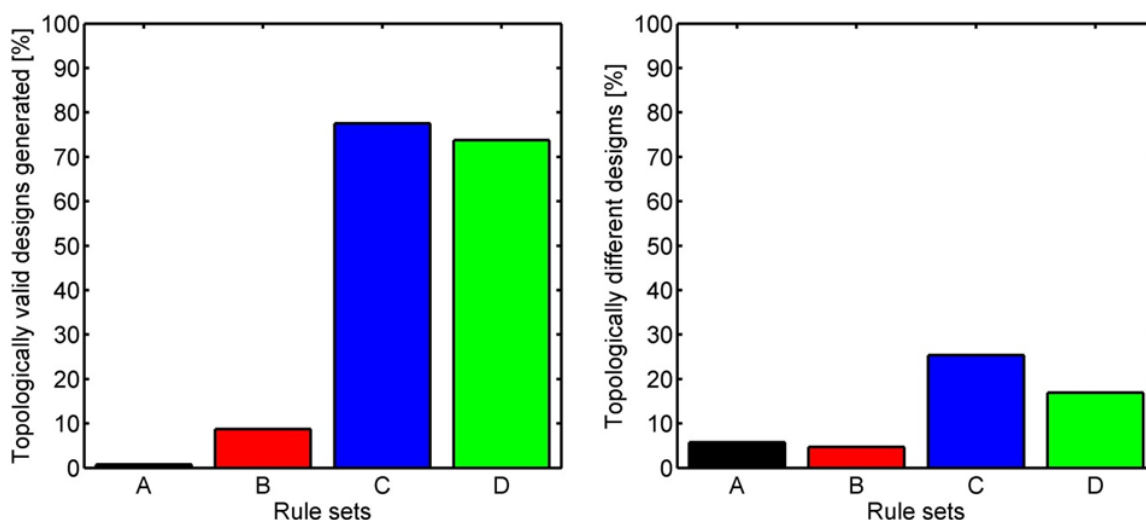


Figure 4-12 Validity ratio (left) and diversity ratio (right) for rule sets A-D.

4.4 Discussion

The research presented in this chapter focuses on supporting the rule development process. It is expected that through a systematic analysis of the rules during the rule design, “better” grammars can be developed that lead to a more successful synthesis process. The understanding gained in the analysis using GRAM can also deliver important insights about the search space that can be considered in tuning sophisticated search algorithms. In contrast to the work by Vale and Shea [84] where statistics are collected and rule sequences are defined during the CDS process, GRAM enables not only to reuse insights gained before the CDS process but also to analyze the grammar itself and to improve it based on the results.

The case study shows that GRAM is capable of supporting designers to analyze their developed rule set and the described design language can be tested against the intended language by comparing rule performances. The finding that rule sets A and B generate designs with fewer speeds and rule sets C and D generate designs with more speeds, for example, reflects the purpose for which the rule sets were originally developed, i.e. generating single speed gear trains for rule sets A and B and generating gearboxes with multiple speeds for rule sets C and D, respectively. Non-influential rules can be detected to reduce the rule set, e.g. rule D6. No unintended performance was discovered in this case, which might be explained by the long history of improving and further developing the grammars for this case study. The fact that rule D6 (*delete unused gears and shafts*) can be removed from the rule set is a result of the careful implementation of the gearbox rules in this rule set such that no unattached shafts and gears are generated. This was a useful discovery due to GRAM and shows a secondary use of the method for rule set debugging. The design space representation allows statements to be made about the ambiguity of the design grammar regarding the defined design characteristics. The case study shows, for example, that the grammar is ambiguous as designs with the same design characteristics are achieved several times. This is also reflected in the diversity ratio which, taking the point of view that rule sets with few rules are superior, could be calculated differently as the number of topologically different designs divided by the number of rules. For the case study, this would give similar results (rule set A: 0.014, rule set B: 0.008, rule set C: 0.042, rule set D: 0.028) as the diversity ratio defined in GRAM.

Further, GRAM provides support for rule debugging. If, for example, an error occurred in the implementation of rule A1 (*add shaft*) such that a shaft is removed, GRAM would show this unintended performance of the rule in the boxplots. Similarly, GRAM helps to identify rules that are never applied, e.g. through the matching ratios, or that have no influence on any of the objectives, e.g. through the boxplots. This visual feedback on the rules enables the rule designer to find errors in the implementation and identify starting points for improving the quality of the developed rule sets.

Although the case study uses graph grammars, any type of generative design grammar can be analyzed using GRAM. Further, the method is independent of rule type and definition, i.e. simple vs. knowledge-intensive and topologic as well as parametric grammar rules. As

feedback is given based on each rule's performance with respect to defined objectives, it is necessary that intermediate as well as final designs can be evaluated.

Issues to be tackled are related to the visualization for large-scale problems as well as to automating the interpretation. The visualization of the design space becomes difficult when more than two design criteria are defined. This is a known issue and research topic also in other domains and new visualization techniques have to be investigated. Considering all objectives separately facilitates a thorough analysis of the grammar rules. However, for problems with multiple rules and objectives, the number of diagrams to interpret rises. An automated interpretation of the data instead of a visualization for the rule designer is one approach to tackle this issue.

Possible extensions to GRAM are to analyze not only the performance of individual rules but also rule sequences to enable a better understanding of rule sets and how sequences of rules impact design criteria. This issue is addressed in Chapter 5. Another extension to GRAM is exploiting the knowledge gained in the rule analysis and reusing these findings to automatically generate strategies for more intelligent grammar rule application. One approach is analyzing the performances of sequences of rules and learning favorable ones that can be re-used. This issue is also addressed in Chapter 5. All of these directions aim to increase the understanding of the developed grammar and further extend the idea to minimize the effort of tuning the search algorithm to the design problem while increasing the quality of the synthesis results.

4.5 Summary

GRAM is a method to support the human designer in the development of generative grammar rules for CDS. The method focuses on a systematic analysis of the rules in the development phase rather than during their application within a search algorithm after the rules are developed. GRAM facilitates gaining in-depth knowledge of the rules' performance, their relations to objectives, constraints and design characteristics, and their interaction. Further, it is possible to find errors in the implementation of the rules through easily readable feedback. Superior synthesis results as well as less effort to adapt search algorithms due to better understanding of the solution space defined by the grammar rules are expected. Using a systematic data generation process, data for the analysis is generated, analyzed and visualized in defined diagrams. The interpretation of these diagrams using predefined questions helps to identify if, and which parts of the rule set, need improvement. With GRAM, future grammar rule developers are given a means to reason about their specific grammar rule implementations in a systematic way. Research questions 4.1 - 4.3 can be answered as follows:

Research question 4.1: *How can we systematically investigate the relations between grammar rules, objectives and design characteristics in CDS methods?*

GRAM presents a generic and systematic method to generate data, analyze it and visualize the information to the human designer. As shown in the case study, the method can be used for different grammars and it is also successfully applied to analyze grammar rule sets besides those presented within this thesis. GRAM is formulated generically and can in theory be used to analyze also other types of grammars, e.g. shape grammars. GRAM is one possible method to support rule developers that can further be improved and extended. Other methods exist and extensions to GRAM are presented within this thesis.

Research question 4.2: *In this context, what information do grammar developers need to develop and refine grammars?*

In Section 4.2, six questions are formulated that can be answered when using GRAM. These questions are based on the author's experience in grammar development, general statements in research papers on grammar development and discussions with researchers in the area.

Research question 4.3: *How can this information be visualized?*

GRAM defines visualizations that show individual rule performances considering objectives and design characteristics. Cognitive aspects of humans' visual perception (see, e.g., [123]) are considered in the choice of the visualizations to balance easy readability and information content that is represented. An example are the color-coded boxplots to visualize individual rule performances. The color coding allows a quick qualitative overview if the rule changes the design in the desired direction. The boxplot then gives more detailed information on the rule's performance. Using the same shape (rectangle of the box in the boxplot) but changing length (box and whiskers) and color (to indicate whether or not the change is in the desired direction) generates a notation that can be processed efficiently [123].

Besides answering research questions 4.1 – 4.3, the following research contributions are achieved through GRAM:

- **Contribution 1:** Criteria are defined to assess the performance of individual grammar rules and of a set of rules.

To date there are no commonly accepted criteria defining what constitutes a “good” grammar. With GRAM, a set of criteria to assess grammar rules are proposed. These can serve as a starting point to further elaborate criteria for grammar rule assessment through discussions in the research community.

- **Contribution 2:** With GRAM, a systematic method is developed to analyze grammar rules independent of a search algorithm.

GRAM defines how grammar rules can be analyzed systematically in a three step process of data generation, analysis and visualization. Providing criteria and a systematic method to assess grammar rules according to these criteria can facilitate future research on grammar rule development.

- **Contribution 3:** The process for grammar rule development is enhanced.

GRAM is integrated into the grammar rule development process. The iterative nature of the rule development process is discussed by several researchers. In practice, rule developers conduct ad-hoc tests on their developed grammar rules before changing them. Adding a process step in which developed grammar rules are analyzed systematically to the grammar development process reflects this approach and can encourage designers to use systematic rule analysis instead of trial and error testing to improve the developed grammar.

Besides the research contributions described above, the human designer is supported in CDS when using GRAM as shown in Figure 4-13. GRAM supports the human designer in the rule development process. In particular, rule development is supported through visualizing information on the rules (sub-goal G1.1). Rule selection (sub-goal G1.2) is supported since GRAM represents a means to compare different grammar rules in a systematic way.

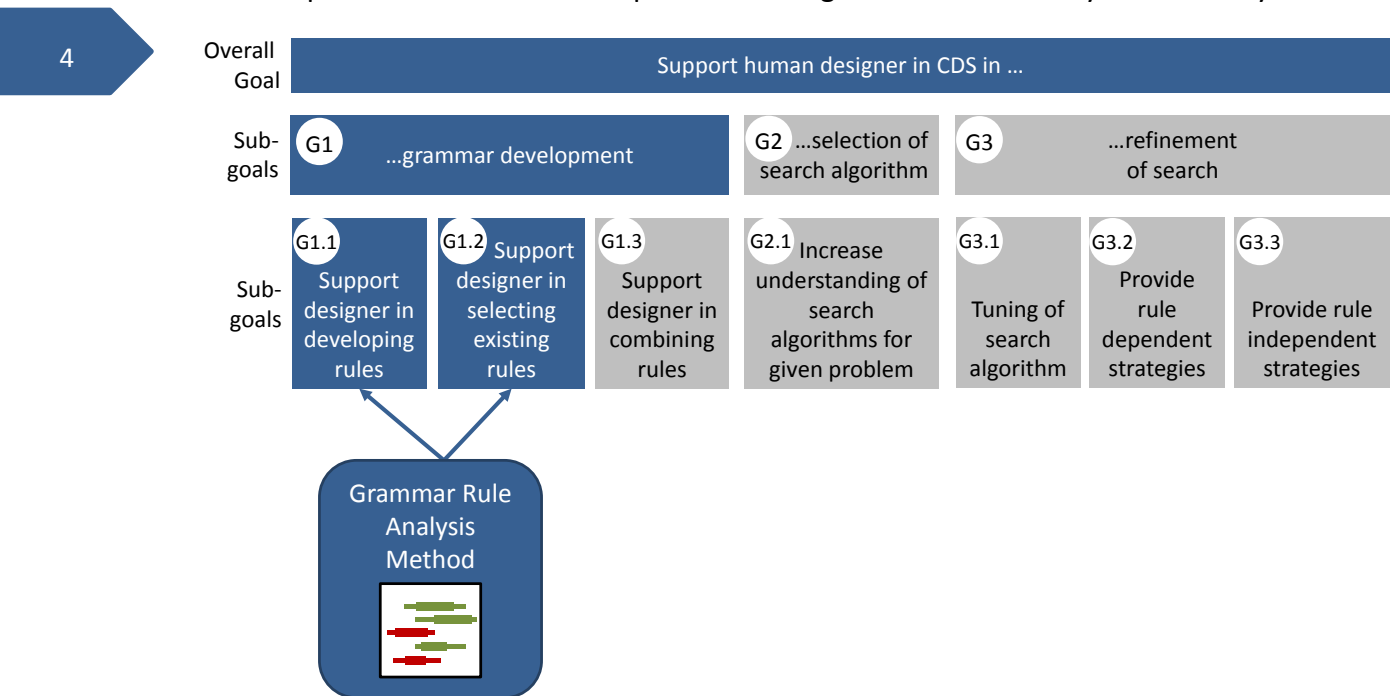


Figure 4-13 Positioning of the Grammar Rule Analysis Method with respect to the goals of the thesis.

The grammar development process is supported through the following contributions:

- **The human designer is given a means to systematically assess grammar rules.** This enables them to reason about developed rules and to compare different grammar rules to each other. This can improve the quality of developed grammar rules and ease the selection of grammar rules among existing ones.
- **Generic visualizations are defined that present the performance of the analyzed grammar rules to the human designer.** The visualizations give an overview of the different performance criteria for each individual rule and the developed rule set.
- **A generic software prototype is developed.** It generates data, analyzes it and visualizes information on rule performances for the defined criteria.

5 Network-based Rule Analysis Method

With GRAM (Chapter 4), the human designer is given a means to analyze individual grammar rules during development. Understanding how each rule influences objectives and design characteristics has shown helpful to understand characteristics of a grammar rule set during grammar development. More detailed information on individual rules, e.g. beneficial locations for their application, but also on sequences of rules can further support the rule development and application process. In this chapter, research is presented that aims to answer the following questions:

Research question 5.1: *How can grammar rules in CDS be systematically analyzed to identify the effect of a rule's application location on the generated designs?*

Research question 5.2: *How can grammar rules in CDS be systematically analyzed to identify sequences of rules that are beneficial or counterproductive?*

In the following, motivation for a network-based analysis of rules is given (Section 5.1) and the Network-based Rule Analysis Method to support grammar development and rule application is presented (Section 5.2). Section 5.3 demonstrates how the method can be used to analyze LHSs of rules in detail using a gearbox synthesis case study. In Section 5.4, the case study demonstrates how the method is used to analyze a grammar for a sliding tile puzzle. Knowledge learnt on small scale is successfully applied to solve a larger scale problem. The results (Section 5.4.3) show the feasibility of the method and its generality is discussed (Section 5.5). A summary is given in Section 5.6, where also the achieved contributions are revisited.

5.1 Motivation for Network-based Rule Analysis

Grammatical approaches have been successfully applied in numerous engineering design disciplines [20], e.g. in electrical engineering, architecture and mechanical engineering as well as in natural language processing, e.g. for automated language translation or speech recognition. These different areas, all emerging from the formal study of grammars, evolved in different directions. In the area of compiler design, grammars are designed formally and a whole research area of grammar engineering has evolved. In architecture, VLSI design and engineering, grammars are often used to develop product concepts in the early stages of the design process and are often formulated less mathematically. In some areas, grammars are mainly used as pencil and paper grammars, i.e. they do not require computational implementations. Engineering design grammars are often developed for very specific use cases and no commonly accepted criteria for “good” grammars have been specified by the engineering design community.

While with formal grammars, as in compiler design, many algorithms exist to analyze properties of the language that is described by a grammar, such analyses are usually not

done when developing engineering design grammars. As an example, one can look at research on the automated synthesis of gearboxes, a popular CDS task. Studies have been carried out by several researchers [38, 119] and different grammars as well as algorithms to guide the synthesis process exist. Most researchers develop their own grammar for the problem instead of reusing previously developed ones and the grammar development process is usually not documented. Being presented only the final versions of different developed grammars and the synthesis results, it is not obvious for the designer which grammar or which particular rules are preferable for gearbox synthesis and why. Further, none of the publications investigates in depth how the grammar explores the space. This makes it difficult to fully understand the importance and influence of single grammar rules on the synthesis process. Several researchers mention the need for more support in the development of engineering design grammars [31, 78] and the lack of support for grammar design is still seen as one of the major drawbacks of grammatical design [20].

In Chapter 4, GRAM is presented to support the development of grammars for CDS. GRAM supports the analysis of single rules, however, extensive information on how the rules change individual designs is lacking as well as detailed information on how sequences of rule applications explore the design space. Providing more support to a) the development of the grammar rules due to a better understanding of how they explore the design space, and b) the application of sequences of rules, major improvements in the quality of solutions produced from design grammars are expected. Such analyses can help designers understand new and existing grammars in depth and allow them to make more informed decisions on reusing or developing engineering design grammar rules. The research in this chapter follows the direction of supporting grammar development with a novel approach of combining concepts developed in areas such as compiler design with grammars used in CDS. The presented research analyzes the potential of using concepts from compiler design to support grammar development and application through giving feedback on how a developed grammar explores the design space. There are two aspects to this which both are addressed. First, the grammar designer is given feedback on the rules, e.g. if there are redundant or do-undo rules and how a rule's application location influences the generated designs. Second, the application of the grammar can be analyzed in more detail to identify preferable rule application sequences or patterns in rule sequences that should be preferred or avoided. With this information, engineering designers are given a means to more efficiently develop and apply grammar rules for CDS. Several approaches to identify preferable search strategies and to learn meaningful sequences of rules have been developed, e.g. Vale and Shea [84] developed a machine learning (ML) based method to accelerate the CDS process through knowledge on rule sequences. What is unique about the research presented in this chapter is the way it supports multiple phases of the CDS process, namely the representation and the guidance step, in one method and that unlike most existing methods, it collects knowledge about the problem through rule analysis before the actual CDS search process is started. The rule development is supported through an increased understanding of how each individual rule transforms a given design and the synthesis process is supported through prior

knowledge about rules that are sensitive to rule application locations and about meaningful rule sequences to solve problems.

5.2 Method

Two concepts from compiler design, namely finite automata (FA) and transition graphs (see also Section 2.3.2) are used for the Network-based Rule Analysis Method. Similar to an automaton that accepts a given sequence of input symbols, a transition graph, constructed for an engineering design grammar, can accept or reject a sequence of grammar rules applied to the initial design. Each design that results from a modification, i.e. a rule application, then represents a state and each grammar rule represents a symbol a transforming state s to state t . Analyzing such a transition graph using techniques from data flow analysis, the influences of each rule application can be understood in detail.

The Network-based Rule Analysis Method is based on analyzing the designs that are generated during CDS and the rules that are used to do so. Figure 5-1 illustrates the three main steps. In the first step, designs are generated by searching through a generative tree. Starting from an initial design, i.e. the root of the generative tree, designs are generated through successive rule applications and each generated design is added to the generative tree as a child node of the previous design. Using tree-based search methods, such as Depth-First Search (DFS), i.e. expand first one rule sequence, or Breadth-First Search (BFS), i.e. from one design apply multiple different rules in parallel before moving to the next level, the design space can be explored. Other search methods are also possible to use in this step and the goal in this step is not to find an optimal design, but to explore a portion of the design space that can be analyzed in the following steps. Figure 5-1 (left) shows example representations of generative trees that are explored when using tree-based search methods. Each node in the tree represents one design and each edge between two nodes represents the rule that was applied to transform one design into another one.

In most algorithms, the same designs can be generated repeatedly and tree-based representations often represent these designs as multiple different nodes in the tree, each resulting from a different rule sequence. Many search algorithms store a list of already explored designs to avoid expanding on the same design more than once. The author proposes that representing these repeated designs as one unique design instead of multiple times in the generative tree can help gain useful insights and decrease the search space size. To do so, in the second step of the method, all generated designs are analyzed to identify unique and repeated designs. Uniqueness is a property that is problem dependent. The designer developing the grammar defines what uniqueness means for the given problem. The tree-based representations are traversed gradually and every time a previously undiscovered design is found, it is given a new, unique ID. Every time an already discovered design is found, its node in the tree-based representation is deleted and the edges, i.e. the rule with which the design was generated and the rules that were applied next, are connected to the already generated (unique) design. Doing this, the tree-based representations merge to one or more networks of design transitions. In these networks

(transition graphs), each node represents a unique design, or state, and each edge represents a transition from a source design to a target design.

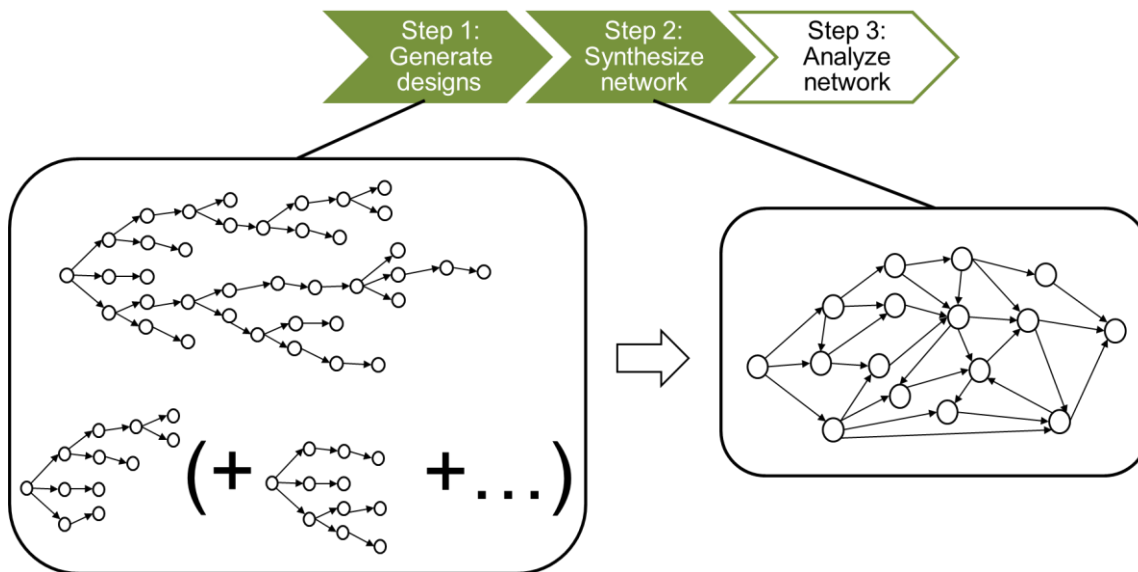


Figure 5-1 Steps 1 and 2: Network generation through generation of designs (left) and synthesis of designs to one network representation (right).

5

The transition graph can be analyzed to gain knowledge about a) the rules themselves, and b) rule application sequences. In the third step of the method (see Figure 5-2), different graph analyses are performed. The designer is given two methods to access the gained information. First, the network is represented visually for manual exploration of the generated designs and their relations. Second, the network is analyzed automatically and the following results are presented (compare with numbers in Figure 5-2):

1. Do-undo rule pairs are identified, i.e. pairs of rules where one rule un-does what the other did. A simple example of such a rule pair is two rules of which one adds a component and the other one deletes it. For the selection of an appropriate search algorithm, this information can be important because a repeated application of do-undo rules might get the synthesis process stuck in generating the same designs over and over again. The designer can use this information on do-undo rules to either change the grammar or select a search algorithm accordingly.
2. Loops in the transition graph can be identified. Similar to do-undo rules, a loop of rules describes a sequence of rules that, when applied to a given design, generate exactly the starting design. Avoiding such loops in the synthesis process, either by reformulating the grammar or through using this information in a search algorithm, can allow for a faster design space exploration.
3. Alternative rule sequences are identified, i.e. sequences of rules that transform a given design s to a design t via different paths in the transition graph. If such alternative paths exist, the designer can reason about the alternative paths and consider, e.g. if the rule set can be reduced by deleting or combining rules.

Additionally, interactive tools to study the transition graph help to answer the following questions:

4. Is it possible to reach design t from design s ? This enables, e.g., to analyze if design t can be synthesized starting the synthesis process from design s .
5. Which rules have to be applied to transform design s to design t ? Understanding the application of rules in sequences depending on the design s on which the sequence is applied can help to reason about improving the grammar rules and to learn meaningful sequences.
6. What is the shortest rule sequence to transform a design s into a design t ? Learning shortest rule sequences to transform one design into another can help to speed up the synthesis process.
7. How does the rule application location influence the generated results? The effect of a rule's application location can be analyzed by exploring the designs that are generated when applying a rule to the same design but at different locations.

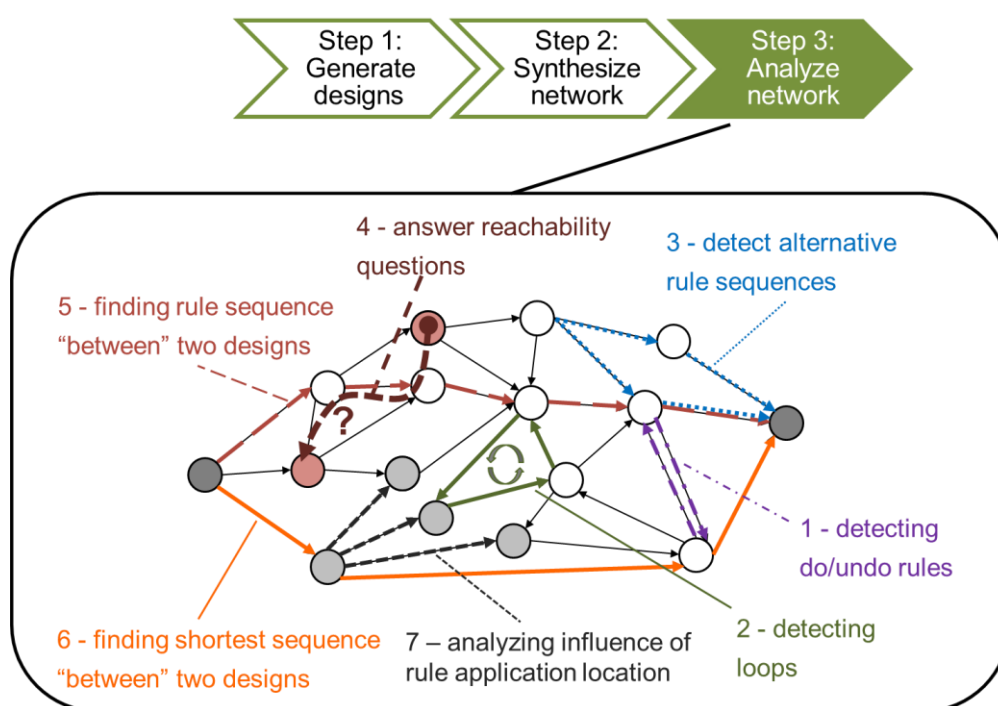


Figure 5-2 Step 3: Analyzing the transition graph to understand grammar rules and their application.

The transition graph is generated using GrGen, an open source graph rewriting tool [50]. It is visualized using OrganicVIZ [124], a graph visualization tool capable of representing large graphs and supporting graph analyses as well as providing several filtering options. Using OrganicVIZ, the user can manually study the transition graph to understand the search space. Nodes and edges can be changed in size and color to provide an overview and emphasize certain designs in the search space. Highly connected designs, i.e. designs with several edges connecting them to other designs can, e.g., be represented with larger nodes. Additional information, e.g. on each design's attributes, can be displayed to the user for each graph node and edge. For the automated graph analysis, graph grammar rules implemented in GrGen are used to detect do-undo rules as well as loops and alternative sequences. This information is stored in separate files for further consideration by the human designer. To find shortest-paths between any two designs in the transition graph, a BFS with backtracking

is implemented in c#. A console application is developed such that the designer can interactively search shortest paths between designs and determine the reachability between designs. More details on the implementation are given in Chapter 8.

In the remainder of this section the method is applied to two case studies. In Section 5.3, the gearbox synthesis task is used to analyze LHSs of rules in detail and understand to what extent the location at which a rule is applied can influence the synthesis process. In Section 5.4, the method is used to learn rule application strategies for a sliding tile puzzle. The sequences are learnt from the transition graph of a small scale problem and are used to solve a larger scale problem.

5.3 Case Study 1 (Gearbox Synthesis): Analyzing LHSs of Rules

The gearbox synthesis case study is used to demonstrate how the generation and analysis of transition graphs can be used to analyze grammar rules and their application in detail. The focus is on analyzing the LHSs of rules, i.e. their application conditions. When a rule can be applied on various locations in a design, also the selection of where to apply it influences the outcome. This is analyzed as well.

5

5.3.1 Generation of Designs

The five topologic rules of the gearbox grammar described in Section 4.3.1 are used to generate gearbox designs. The pseudo-code of the algorithm used to explore the space is given in Figure 5-3. The algorithm requires an initial design (*initialDesign*), the rule set (*ruleSet*) and a maximum number of rule applications (*maximumSequenceLength*) as input. Outputs are a list of explored designs (*graphs*) and a list (*transformations*) storing each applied rule, the design the rule is applied to, and the resulting design. Starting from an *initial design*, all rules in the rule set (*ruleSet*) are explored exhaustively until a given number of rule applications (*maximumSequenceLength*) from the *initial design*. The algorithm is implemented as a BFS with isomorphism check to not explore already found topologies repeatedly. It explores the design space in levels (*level*) until the maximum number of rule applications (line 2). For each *level*, the algorithm iterates through all rules in the rule set (line 3). The selected rule *r* is applied to all graphs from the previous level (*previous*, line 4). For each graph *g*, all matches *m* of the current rule *r* are identified (line 6) and for each match *m* a new graph (*newGraph*) is generated by applying rule *r* on graph *g* at match *m* (line 9). The generated graph (*newGraph*) is evaluated (line 10) and if it constitutes a topology that is not found yet in one of the previous levels, (*newTopology* returns *true* in line 11), then the design is stored in the list of designs to be expanded in the next level (*next*, line 12). Each generated graph (*newGraph*) is stored in the list *graphs* (line 14) and for each rule application the rule and the previous and the resulting graph are stored in *transformations* (line 15). Once all rules in the *ruleSet* are applied for the current *level*, the graphs from list *previous* are replaced with those from list *next* (line 19), list *next* is cleared (line 20) and the algorithm continues with the next *level* (line 3).

Algorithm: Exhaustive Generation**Input:** *maximumSequenceLength, ruleSet, initialDesign***Output:** *graphs, transformations*

```

1:  previous ← initialDesign;
2:  for level = 1 to maximumSequenceLength do
3:      for all rule r in ruleSet do
4:          for all graph g in previous do
5:              load(g);
6:              identify matches of r on g;
7:              for all match m in matches do
8:                  load(g);
9:                  newGraph ← apply(r,g,m);
10:                 evaluate(newGraph);
11:                 if newTopology(newGraph) then
12:                     next ← add(newGraph);
13:                 end if
14:                 graphs ← add(newGraph);
15:                 transformations ← add(g,r,newGraph);
16:             end for
17:         end for
18:     end for
19:     previous ← next;
20:     clear(next);
21: end for

```

Figure 5-3 Pseudo-code for exhaustive generation.

5.3.2 Results

Figure 5-4 shows the transition graph when the five topologic rules are applied exhaustively to an initial design. The initial design consists of input shaft, output shaft and a gear pair connecting the two. The sequence length is set to two rules to generate the transition graph in Figure 5-4. The transition graph is generated automatically and is visualized using OrganicViz. Same colors of the graph nodes indicate that these nodes have the same number of forward and reverse speeds. The pictorial images are generated using yComp [125] and added manually. yComp is a graph visualization tool based on yFiles [126]. It provides several graph layout algorithms and is integrated in GrGen. Node labels (UID) indicate the unique ID of each generated gearbox topology. This number is used to link the nodes in the OrganicViz visualization to the gearbox representations that are stored during the exhaustive generation and can be visualized, e.g. using yComp. The edge labels in Figure 5-4 indicate which rule is applied to transform a design into another one. Analyzing all outgoing edges from one node, the human designer can explore the effect of LHS matches when the same rule is applied at different locations on a design. In Figure 5-4, rule 5 (*replace a gear pair*) is, e.g., is applied at three different locations on the design with UID 2. Of the three rule applications two result in the same design (UID 17), while the third generates a different topology (UID 9).

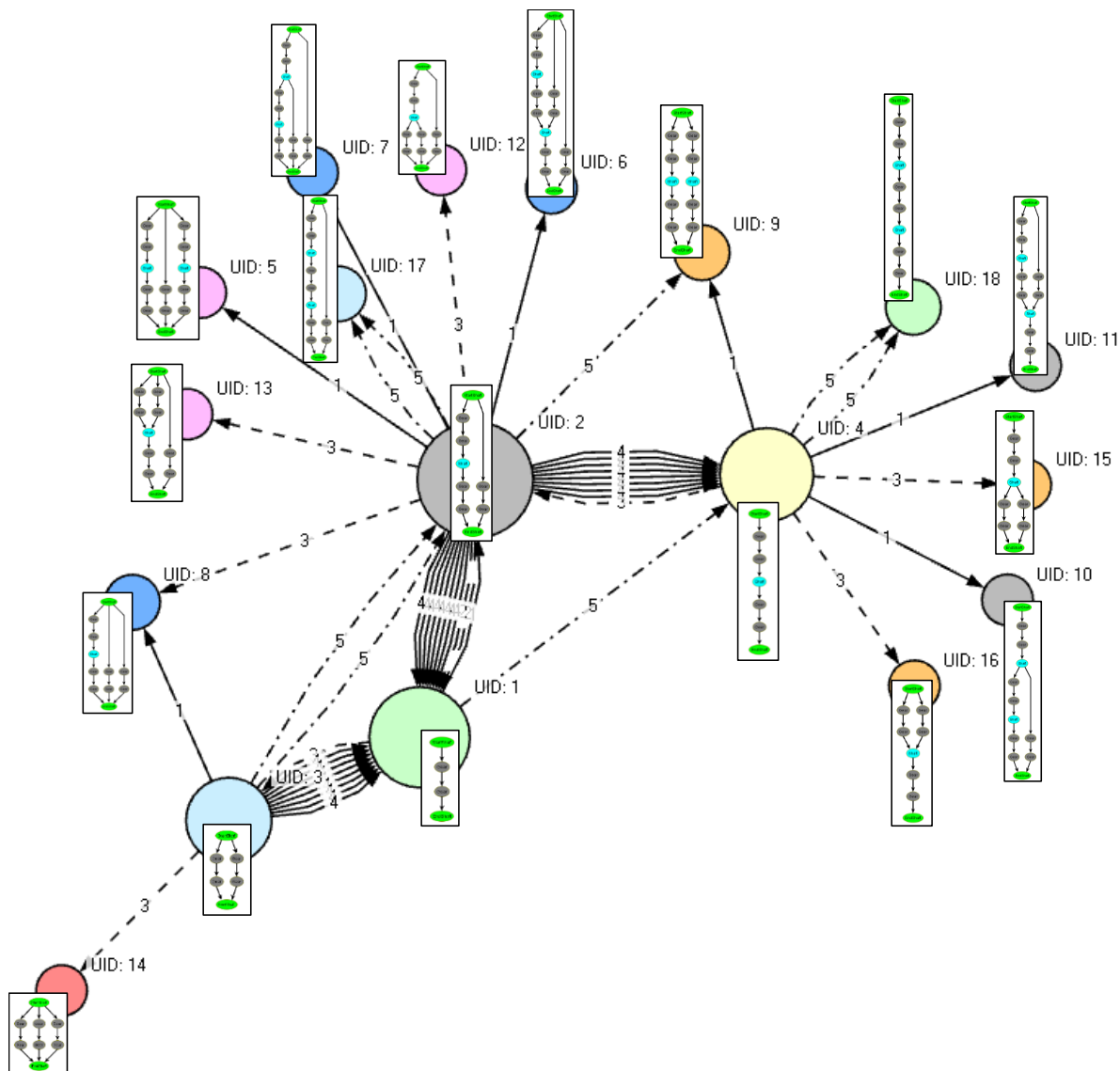


Figure 5-4 Transition graph for the gearbox case study. Rule sequences with two rules are explored exhaustively.

Between the designs with UID 1, UID 2, UID 3 and UID 4, rule 2 (*delete a shaft*) and rule 4 (*delete a gear pair*) are applied at various matches but result in the same designs. A zoomed in view on these designs is given in Figure 5-5. The human designer might find this interesting because he or she would expect a different behavior. When rule 2 (*delete a shaft*) is applied to the design with UID 2, the human designer might expect that this rule is only matched once, since only one shaft can be removed because input and output shaft must remain in the design. This comparison between expected and actual matches of rules facilitates the identification of rules whose LHSs might be formulated too ambiguously.

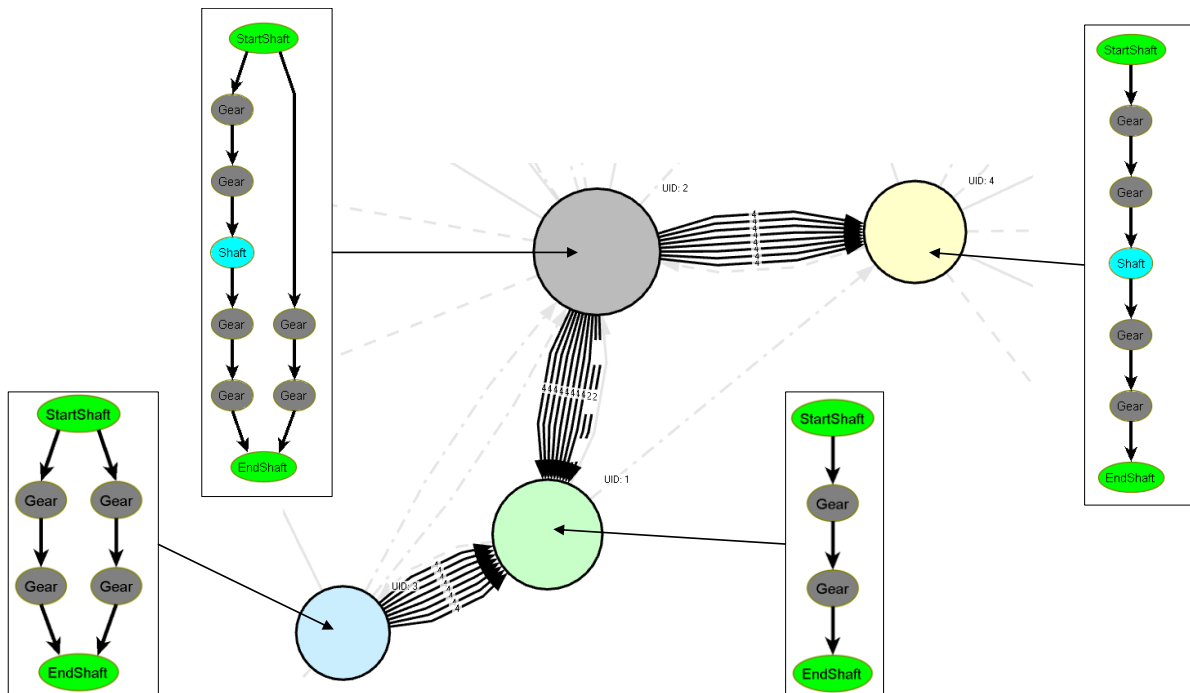


Figure 5-5 Zoomed in view at topologies with UID 1 - 4.

The transitions between the design with UID 2 and that with UID 1 are analyzed in more detail now to understand how the formulation of the LHS leads to several matches. In Figure 5-6, all matches of rule 2 (*delete a shaft*) on the design with UID 2 are shown (left). The rule formulation in GrGen is provided on the right of Figure 5-6. LHS and RHS of the rule are highlighted. The LHS of the rule consists of a node of type *shaft* ($S0$) and a node of type *gear* ($G0$). Additionally, a negative application condition (NAC) is defined using the *negative* statement. NACs are used to define patterns that, when present in the graph, prohibit the rule's application. In case of rule 2 this is when the *gear* $G0$ is reachable from *shaft* $S0$, or when *shaft* $S0$ can be reached from *gear* $G0$. The rationale behind this condition is that when no *gear* $G0$ exists that is neither reachable by *shaft* $S0$ nor can it be reached from $S0$, then there exist only power flow paths in the gearbox that involve $S0$. The rule is then not applied since removing $S0$ would result in an invalid design. The functions *reachableOutgoing()* and *reachableIncoming()* are used to identify the sets of nodes that can be reached or are reachable from *shaft* $S0$. The RHS of the rule is defined within the *modify* statement. First, the *shaft* $S0$ is deleted. Then, two rules are applied that remove dangling nodes in the graph. The first is called *deleteUnusedGears* and removes all gears that were connected to *shaft* $S0$, i.e. those gears that are mounted on *shaft* $S0$. The second is called *deleteUnusedGearPairsAndShafts*. It searches through the remaining graph to identify gear pairs and shafts that are not required any more since they are not connected to both input and output shaft of the gearbox to form a valid power flow path.

On the left side of Figure 5-6, the two matches of rule 2 on the graph with UID 2 are shown. The *shaft* $S0$ is matched to the same node, since it is the only node of type *shaft* in the graph. The two gears on the right power flow path through the gearbox can be matched to *gear* $G0$

such that the NAC is not fulfilled and the rule can be applied to remove the *shaft* *S0* and the then dangling gear nodes to transform the design into that with UID 1.

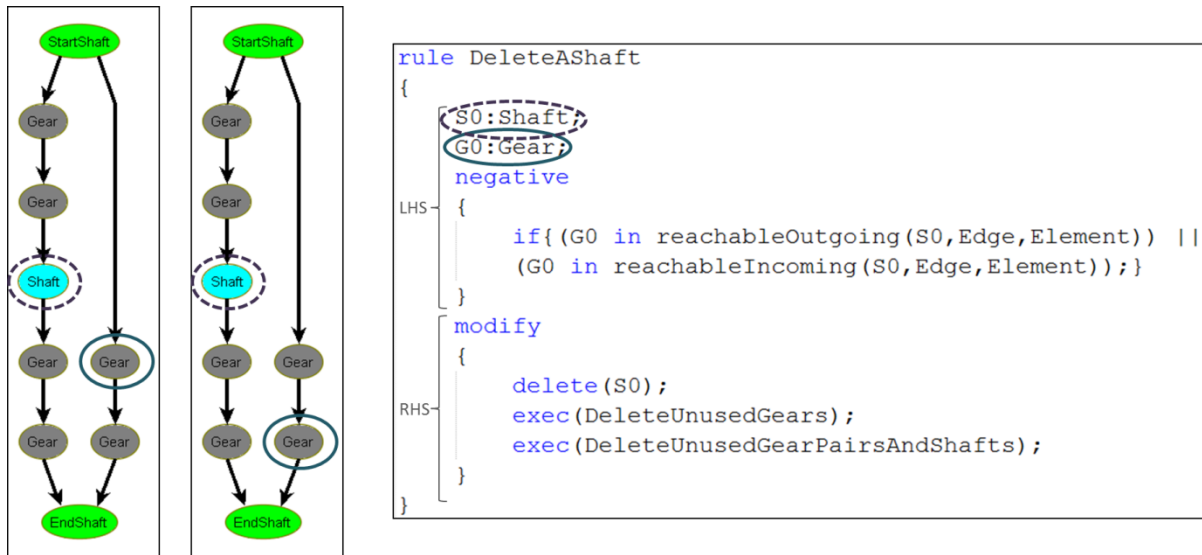


Figure 5-6 Rule 2 (*delete a shaft*) formulated using a negative application condition.

5

Comparing the graphs with UID 2 and UID 1, it seems reasonable, the rule designer might expect that two matches of rule 4 are found to delete either the first or the second gear pair of the left power flow path. The transition graph, however, shows that rule 4 is applied to eight different matches on the graph with UID 2 to transform it to the graph with UID 1. Figure 5-7 shows the rule formulation in GrGen for rule 4 (*delete a gear pair*). The expression “*G0:Gear ?--? G1:Gear*” indicates that two nodes of type *gear* have to exist and have to be connected by an edge. “*?--?*” indicates that the direction of the edge connecting the two nodes is not relevant. The remainder of rule 4 is structured similarly to rule 2 and therefore not described here in detail. On the left of Figure 5-7, the eight matches of rule 4 on the design with UID 2 are shown that transform it to the design with UID 1.

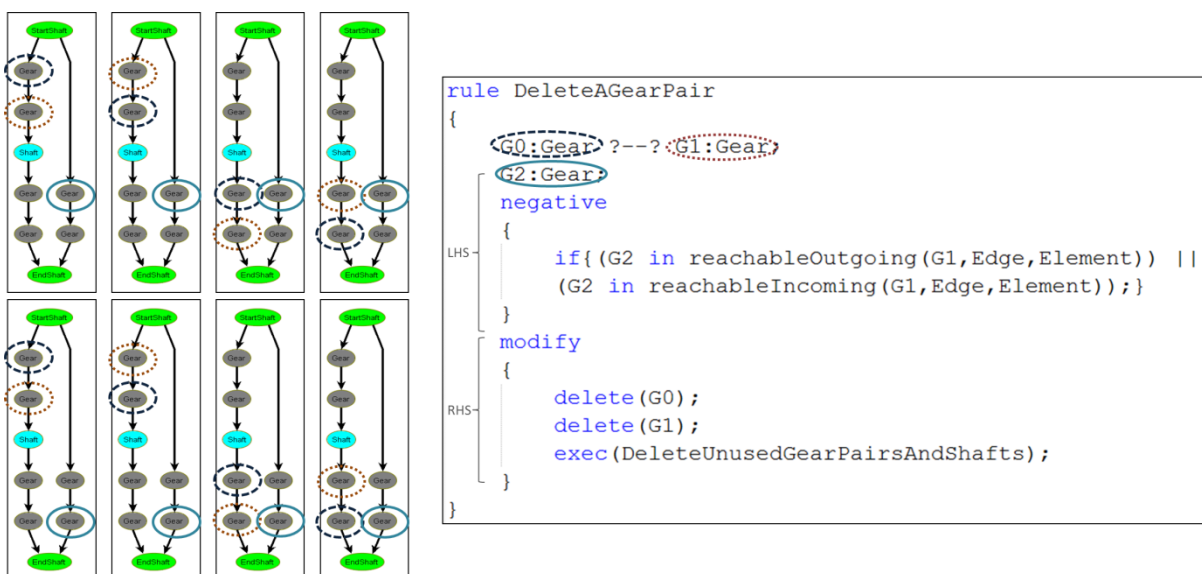


Figure 5-7 Rule 4 (*delete a gear pair*) formulated using a negative application condition.

When the human designer understands that rules are not matched as intended, e.g. more often than necessary, the LHSs of the rules can be formulated more carefully. Figure 5-8 and Figure 5-9 show how this can be done for rule 2 and rule 4, respectively. The LHSs are reduced to only contain the nodes that actually have to be removed. Instead of NACs, positive application conditions (PAC) are used (*independent* statement). The shaft (rule 2) or gear pair (rule 4) are removed from the graph when there is another gear that is neither reachable from the matched node(s), nor can it reach the matched node(s). Additionally, the direction of the edge between gear $G0$ and gear $G1$ in rule 4 is specified to be directed from $G0$ to $G1$ ($G0 \dashrightarrow G1$).

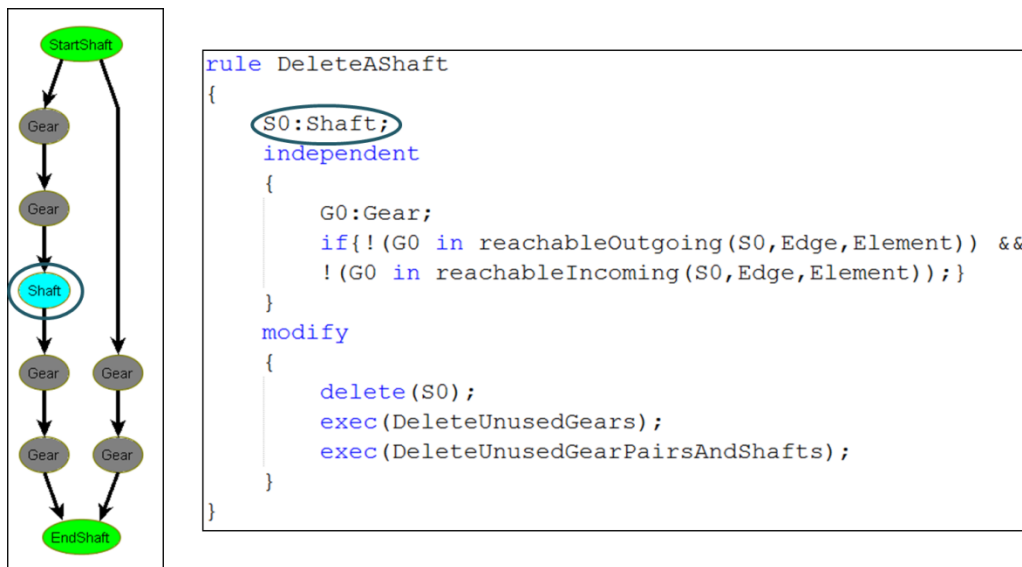


Figure 5-8 Rule 2 (*delete a shaft*) formulated using a positive application condition.

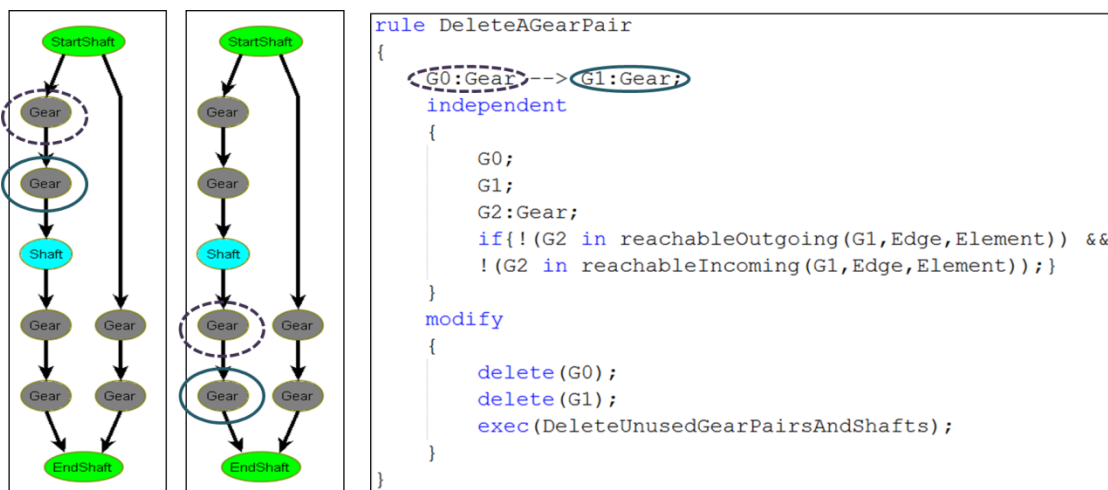


Figure 5-9 Rule 4 (*delete a gear pair*) formulated using a positive application condition.

Changing the formulation of the LHSs of rule 2 and rule 4 reduces the number of matches as intended. Figure 5-10 shows the transition graphs for rule sequences with two rules when NACs are used (left) and when PACs are used (right). In both cases 18 topologically different designs are generated, but the number of rule matches, i.e. edges in the transition graph varies. It is reduced from 50 to 30 when the LHSs are changed to using PACs. Regions where the number of edges changes are highlighted in Figure 5-10. This reduction of LHS matches

does not reduce the number of designs that are generated. The speed of exploring the search space is, however, changed significantly when PACs are used. For a sequence length of four rules, e.g., the designs are explored exhaustively within 25 minutes when using PACs. When using NACs, the process is stopped after approximately one hour due to lack of memory on the laptop that is used (MacBook Pro, Intel Core i7 CPU with 2.80 GHz, 8 GB RAM).

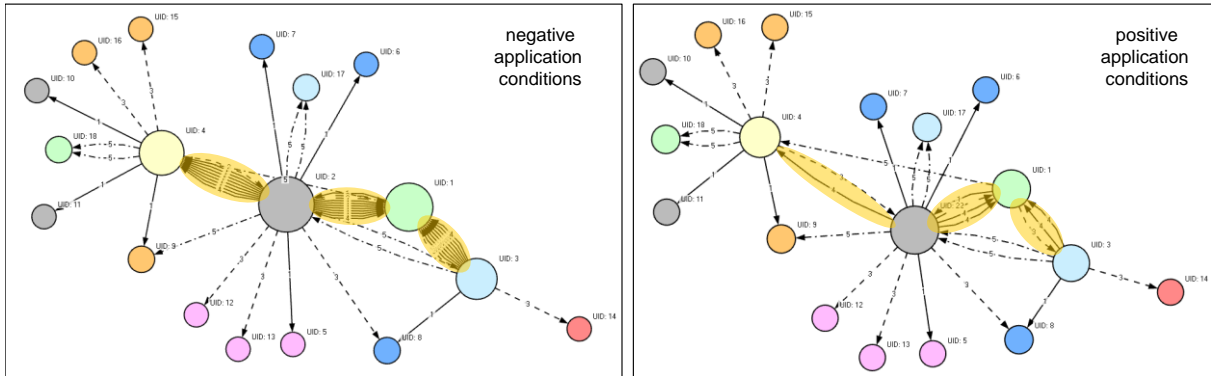


Figure 5-10 Transition graphs when using negative (left) or positive (right) application conditions for rules 2 and 4.

5

Figure 5-11 visualizes transition graphs for sequences with 1, 2, 3 and 4 rules. For the given rule set the number of topologically different designs increases drastically and it can be observed that designs are highly connected to each other. This means the same designs can be generated via numerous different rule sequences.

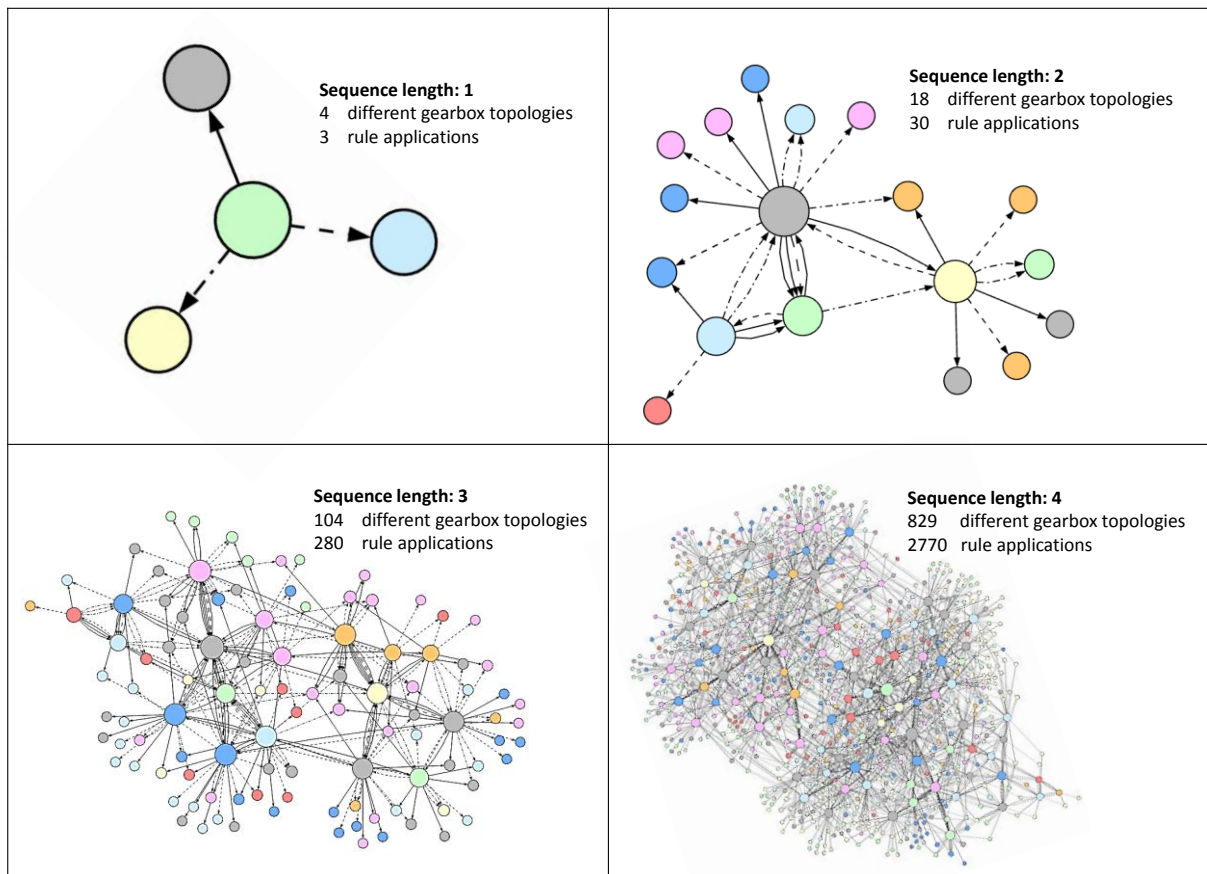


Figure 5-11 Transition graphs for rule sequences from one (top left) to four (bottom right) rule applications.

Besides analyzing the LHSs of the rules, also loops can be detected automatically when analyzing the transition graph. Examples for do-undo rules are, e.g., rule 1 (*create a new shaft*) \leftrightarrow rule 4 (*delete a shaft*), rule 3 (*create a gear pair*) \leftrightarrow rule 4 (*delete a gear pair*) and rule 1 (*create a new shaft*) \leftrightarrow rule 4 (*delete a gear pair*). Many loops are found which is expected for highly connected transition graphs. It is further found, that rule 5 (*replace a gear pair*) can be replaced by applying a sequence of rule 4 (*delete a gear pair*) and rule 1 (*create a new shaft*). The Network-based Rule Analysis Method is used to investigate whether rule 5 can be removed from the rule set. Transition graphs are generated for a sequence length of three rule applications. Two rule sets are used. In the first rule set, rule 5 is kept but rules 1 and 4 are removed. In the second rule set, rule 5 is removed. Transition graphs for both rule sets are shown in Figure 5-12. The transition graph for the complete rule set, i.e. rules 1 to 5, is shown in the lower left corner of Figure 5-11. Removing rules 1 and 4 from the rule set results in a reduced transition graph. Not even a quarter of the topologies are explored when compared to the complete rule set. Removing rules 1 and 4 is therefore considered inappropriate. When rule 5 is removed from the rule sets about half of the topologies are explored compared to the complete rule set. Besides the number of generated topologies, also the connectivity of designs varies depending on the used rule set. Using rule 5 results in a more closely linked transition graph, i.e. designs can be changed to others more quickly. Further, more designs are explored when rule 5 is included in the rule set when the exploration is limited to a certain number of rule applications.

For a sequence length of 4 rules the effect of having rule 5 in the rule set is even stronger. Only 331 gearbox topologies are explored requiring 981 rule applications when rule 5 is deleted from the rule set compared to 829 topologies and 2,770 rule applications when rule 5 is used.

Using the Network-based Rule Analysis Method, different aspects on the gearbox synthesis task and rule set are learnt. First, the LHSs of rules are analyzed and inefficient implementations of the LHS matches are discovered. Second, the highly interconnected nature of the search space and the exponential growth of the number of gearbox topologies when applying longer sequences of rules are understood from the transition graph visualizations. Third, do-undo rule pairs are identified and loops in the transition graph are analyzed automatically. This confirms the gained knowledge, that the gearbox designs are highly connected to each other. Fourth, it is discovered, that rule 5 can be replaced by the sequence rule 4 \rightarrow rule 1. Using the Network-based Rule Analysis Method, it is found, however, that it is useful to have rule 5 in the rule set. This is in accordance with the decision by Lin et al. [38] to develop this rule for gearbox synthesis.

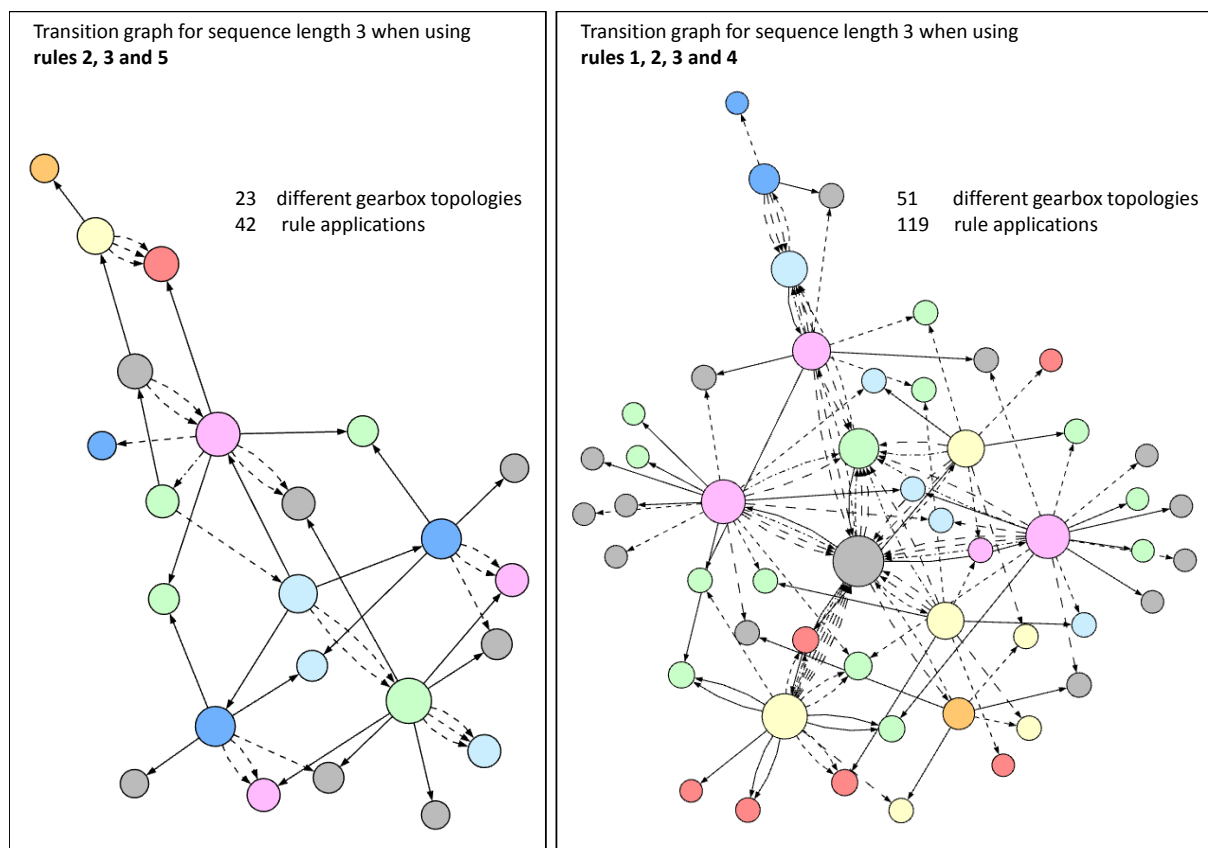


Figure 5-12 Transition graphs for rule sequences of 3 rule applications for different rule sets. Rules 1 and 4 are replaced by rule 5 (left) and rule 5 is replaced by rules 1 and 4 (right).

5.4 Case Study 2 (Tile Puzzle): Learning and Reusing Rule Sequences

The sliding tile puzzle is used as a second case study to demonstrate how transitions graphs can be used to learn and reuse rule sequences. Beneficial rule sequences are learnt on a sub-problem and then applied on a larger scale problem. Problem decomposition is a commonly used problem solving technique in engineering design [14]. Various methods exist to decompose problems into sub-problems, e.g. target cascading for parametric problems [127], or agent based approaches in optimization [128], where sub-problems are solved by different agents and then recombined. The sliding tile puzzle is decomposable into smaller sub-problems and, therefore, suits as a case study to demonstrate the use of learning rule sequences.

The sliding tile puzzle has been used frequently since its invention in 1879 [129] and it is still used in artificial intelligence research. In this puzzle, a number of tiles are arranged in a rectangle with one tile missing. By sliding tiles one after another into the missing spot, the tiles have to be ordered in a defined way. Even though the principle is easy to understand, for larger puzzles numerous possible states exist, leading to a vast number of designs that have to be explored when trying to solve the problem. In 1879, Johnson proved that for each n -tile puzzle where n is the number of tiles, there exist $(n+1)!$ states with only half of them

being solvable [130]. For the classical 8-tile puzzle (also called 3x3 puzzle), this leads, e.g., to 181,440 feasible states.

The sliding tile puzzle is challenging not only to humans trying to solve it, but also for optimization algorithms due to the vast number of possible arrangements. When trying to understand the problem, human experts sometimes learn sophisticated relations between certain states and apply sequences to switch between known states quickly. As, in analogy to this, the goal of the method in this section is to make human designers capable of understanding relations between different designs and grammar rules that transform those designs, the sliding tile puzzle is a well-suited case study. It demonstrates, that through analyzing transition graphs, human designers can gain deeper knowledge about designs and identify useful rule application sequences for design transformations. The method is applied on a small scale problem to gain knowledge in a first step. Then the learnt knowledge is applied on a larger scale problem.

5.4.1 Understanding the Small Scale Problem

A small version of the puzzle is used in this case study consisting of five tiles, numbered one to five, arranged on 2x3 tile grid. A grammar with four rules is developed to modify any given puzzle. The four rules (*Up (U)*, *Down (D)*, *Right (R)* and *Left (L)*) are visualized in Figure 5-13. The LHS of the rule shows an example design on which the rule can be matched and the RHS shows the design after the rule application. Involved tile positions are highlighted in grey. In the last column of the table in Figure 5-13, the possible matches for each rule are given for the 5-tile puzzle, indicated by the positions on which the empty tile can be positioned. All possible states of the 5-tile puzzle are explored exhaustively. The transition graph is generated consisting of all unique designs where each unique design represents a possible tile configuration as a vector, e.g. (1,2,3,4,5,0) represents the target design. The results from the manual and the computer-supported analysis of the transition graph are presented in Section 5.4.3.

Rule	LHS (example) (active positions highlighted)	RHS (example) (active positions highlighted)	Description	Matching condition of LHS (positions of empty tile)
Up			Tile below the empty position is slid UP.	
Down			Tile above the empty position is slid DOWN.	
Right			Tile left of the empty position is slid RIGHT.	
Left			Tile right of the empty position is slid LEFT.	

Figure 5-13 Rule set for the sliding tile puzzle.

5.4.2 Applying Knowledge on the Large Scale Problem

A 8-tile puzzle is used as a large scale problem to analyze whether rule sequences identified on a reduced problem, can help to solve larger scale problems. An example puzzle is given in Figure 5-14.

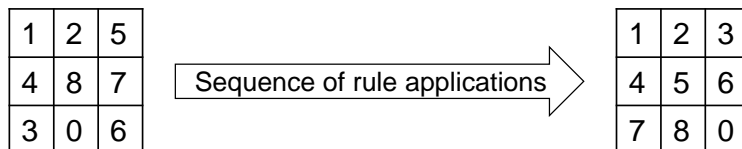


Figure 5-14 Example for the 8-tile puzzle.

The task is to find a sequence of rule applications transforming the given puzzle (left) to the desired puzzle (right) using the knowledge learnt on the 5-tile puzzle. While the 5-tile puzzle with its $6!/2 = 360$ solvable states can be explored exhaustively, the 8-tile puzzle has a significantly larger search space with $9!/2 = 181,440$ solvable states making it harder and for larger puzzles impossible to explore exhaustively. To apply the learnt knowledge, the smaller 5-tile puzzle is mapped into the 8-tile puzzle and only tiles within the smaller 2×3 regions are changed. The remaining tiles stay untouched. Changing between 2×3 regions, e.g. first looking at the upper region, then at the lower region in the 3×3 puzzle, tiles can move through the whole 3×3 puzzle.

Dividing the problem into smaller sub-problems, humans can more easily define heuristics or strategies to solve problems like the sliding tile puzzle. Figure 5-15 presents such heuristics for the larger scale puzzle. The rule set defined for the 5-tile puzzle can be kept. To define regions for sub-problems, the human can analyze the large-scale puzzle and select the region to modify in the next step, e.g. based on the position of the empty tile. In Figure 5-15 (left) the grey regions define the region in which the next modifications are performed. When the empty tile (indicated with "0" in Figure 5-15) is in the top or bottom row, the top or bottom region are selected. When it is in the middle row, further heuristics can be applied or one region can be selected randomly. Given the initial puzzle, the lower region is selected, however, it is still not clear how to change the tiles in the region.

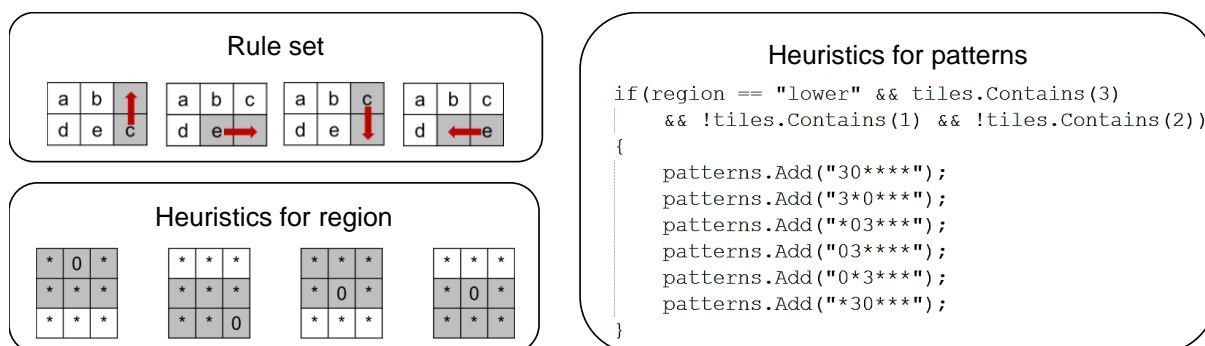


Figure 5-15 Rule set (top left) and heuristics for defining regions of sub-problems (bottom left) and search patterns (right) defined by the human designer.

General heuristics, e.g. trying to finish the puzzle from the top, can be implemented by defining search patterns. Following this general strategy, one can define search patterns as

given in Figure 5-15 (right). The patterns are described as six character strings and define desirable tile positions giving the tile numbers and "*" as a wildcard symbol. In Figure 5-15 (right) one example is given to identify all designs with tile "3" and the empty tile in the upper row of the selected region. This represents one part of the strategy commonly used by humans to move tiles from the upper row (tiles "1", "2", "3") into the middle row and then, in a next step, into the upper row, e.g. to replace the "5" in the initial design by a "3". The rule set, heuristics for the region, and heuristics for patterns depending on the region and the current tile configuration are implemented.

The process to solve the 8-tile puzzle is presented in Figure 5-16. In a first step, exhaustive knowledge, i.e. the transition graph, is generated for the small scale puzzle and stored in a 5-tile library. In the second step, partial knowledge about the 8-tile puzzle is generated using a BFS approach. Implemented using a queue, in each iteration the first element in the queue is expanded. The 8-tile puzzle is subdivided using the heuristic for the region. The considered region is highlighted in grey in Figure 5-16. The tiles are mapped from the 8-tile space to the 5-tile space. In this transformation, the tile numbers are mapped from the numbers (0,1,2,3,4,5,6,7,8) to the number (0,1,2,3,4,5). The empty tile (number 0) remains in both spaces, the remaining numbers have to be mapped such that the mapped design constitutes a solvable design in the 5-tile space. The mapping for the iteration shown in Figure 5-16 is, e.g., (5-tile space - 8-tile space): (1-4), (2-8), (3-3), (4-7), (5-6), (0-0). The same mapping is applied to the patterns defined in the pattern heuristic and for each pattern all designs that match it are identified in the 5-tile library. In Figure 5-16, five example designs matching the heuristics for patterns (30****, *03***, *30***, 03****, 3*0****) are shown, but many more are possible. The shortest paths, i.e. the rule sequences with the least rule applications, to these designs are identified using the 5-tile library. To avoid exploring an unnecessary large space of the 8-tile puzzle search space, for each pattern heuristic only the one with the shortest rule sequence is considered further. The rule sequences and their respective generated designs (transformed back into 8-tile space) are added to the queue and stored in the 8-tile library. This means that the 8-tile library stores a transition graph of the 8-tile space with the nodes representing puzzles and the edges representing rule sequences. This process is continued until the final design is found. As soon as it is found, the complete sequence to solve the 8-tile puzzle is generated (Step 3) by analyzing the 8-tile library in which the generated designs in 8-tile space and the sequences of rule applications to transform them are stored. As in Step 1, it is found by searching the shortest path using a BFS with backtracking with the only difference that instead of single rules, the transformations are sequences of rules.

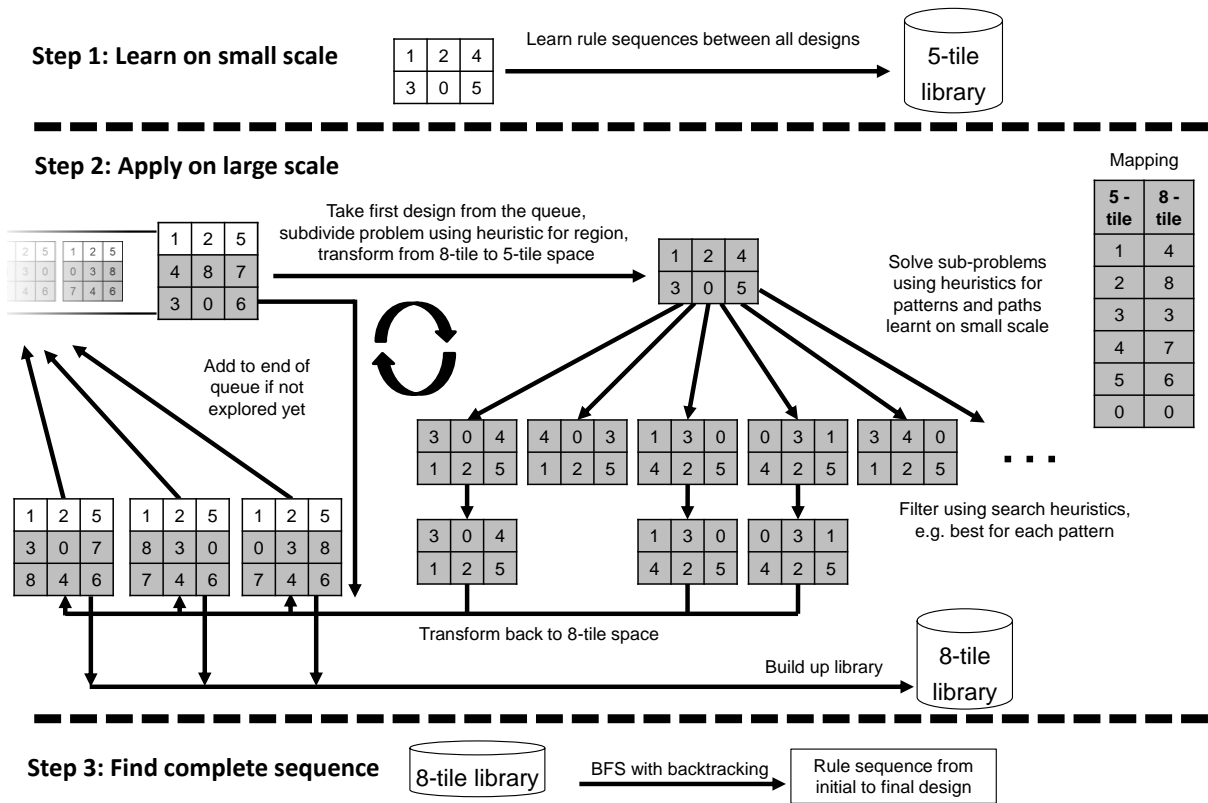


Figure 5-16 Process of gaining knowledge on small scale (Step 1) and applying it to explore (Step 2) and solve (Step 3) a larger scale problem.

5.4.3 Results

As for the tile puzzle, half of the states are not solvable, two transition graphs are generated for all possible permutations of the tiles with numbers (0,1,2,3,4,5) with "0" denoting the empty tile. Both graphs are visualized in Figure 5-17 representing exactly what has been demonstrated mathematically in 1879, namely, that for a given puzzle, half of the states are solvable, whereas the other half are not, and that no solvable puzzle can be transformed into an unsolvable one and vice versa.

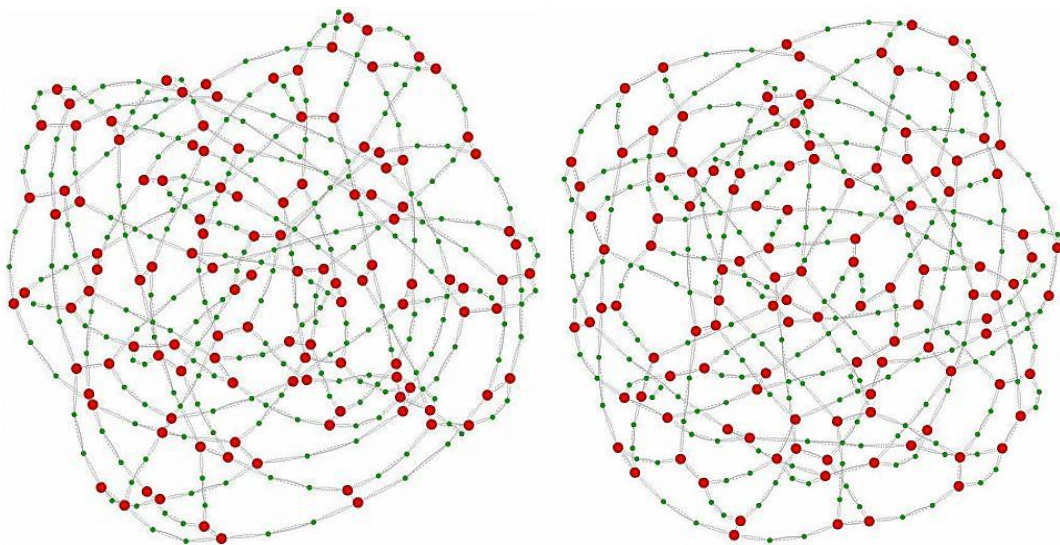


Figure 5-17 Transition graphs for solvable and unsolvable puzzles.

For any design s , the question of whether or not it is solvable can be answered by analyzing the reachability of the final state, design t , from the given design s . Any design for which no path to the final state t can be found is unsolvable. This means that to distinguish between the solvable and unsolvable transition graph in Figure 5-17, one has to identify in which graph the final state is present. Having a closer look at an excerpt of the transition graph (Figure 5-18) we can see relations between designs (states) expressed through rules (transitions) and identify designs that are reached via more rule applications than others. This is highlighted using different colors and node sizes. Do-undo rules can easily be found, as for any rule that transforms one design s into another design t in this case study there exists an undo-rule to transform t to s . This lies in the nature of the problem, that for each tile that is slid into an empty position, the previous position of the tile becomes empty, allowing it to be slid back.

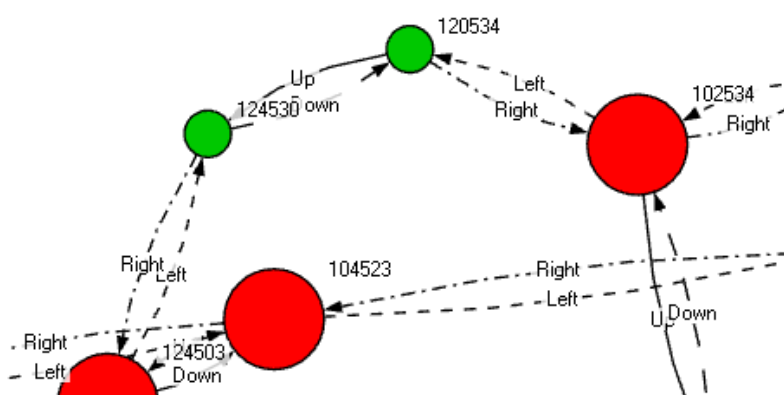


Figure 5-18 A zoomed in view of the transition graph showing designs and their transformations through rules.

Alternative paths, as well as loops can be detected (see Figure 5-19) representing what humans trying to solve the tile puzzle also easily experience, e.g. that sliding tiles in a squared region repeatedly will transform the puzzle to its initial state after a certain number of moves. Figure 5-19 shows an example of a loop that rotates four slides in counter clockwise direction. Understanding such loops can help to avoid the application of a sequence that describes such a loop (as then the whole sequence is negligible). It can also visually be recognized that there might be a shorter and a longer sequence of rule applications to transform a design from one state to another. Besides the manual interpretation of the transition graphs that represent the human's understanding of this simple problem well, automated analysis is also tested. As expected, two pairs of do-undo rules were identified, namely the pairs $Up \leftrightarrow Down$ and $Left \leftrightarrow Right$. Numerous alternative paths between designs are found and questions of reachability between any two designs are answered successfully. It has further been shown that the shortest paths can be found between any two designs. In the second part of the case study, the learnt knowledge from the 5-tile puzzle is successfully applied to the 8-tile puzzle. For any solvable puzzle in the 8-tile space, a sequence of rule applications is found to transform the given puzzle into the desired one. For the initial design presented in Figure 5-14, for example, 21 moves are identified to transform the initial puzzle (1,2,5,4,8,7,3,0,6) into the final configuration (1,2,3,4,5,6,7,8,0): (1,2,5,4,8,7,3,0,6) \rightarrow (Sequence: D, R, U, L, D) \rightarrow (1,2,5,3,0,7,8,4,6) \rightarrow

(Sequence: R, D, L, U, L, D, R, R, U) \rightarrow (1,2,3,0,7,5,8,4,6) \rightarrow (Sequence: L, U, R, D, L, L, U) \rightarrow (1,2,3,4,5,6,7,8,0).

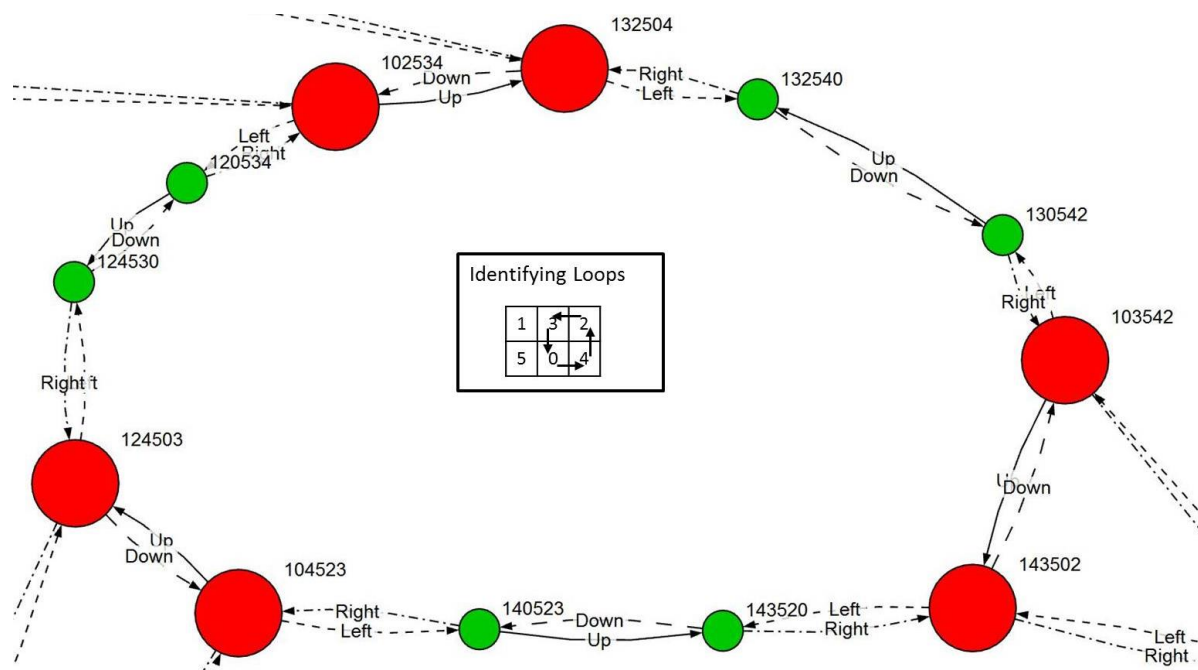


Figure 5-19 Loops are detected to reason about the rules and to avoid them during synthesis.

5.5 Discussion

The two case studies demonstrate different aspects of the Network-based Rule Analysis Method using transition graphs.

The gearbox case study shows how the transition graph can be used to analyze rule applications where the location of the rule application in the design influences the synthesis results. Results show how rules can be identified that are matched more often than intended, i.e. their LHSs are formulated too vaguely. Modifying the LHS formulation of the rules makes it possible to reduce the number of indifferent (from an engineering point of view) matches of these rules. While the generated results remain the same, these changes can have a strong influence on the run time and required working memory when the rules are used in the CDS process. Therefore these detailed analyses of the LHSs can not only help the human designer to understand rule matches, but also to speed up the synthesis process when LHSs are formulated more specific to the synthesis task.

In the sliding tile case study, the transition graph for a small scale problem, the 5-tile puzzle, is generated and analyzed giving insights into the grammar as well as the problem itself. Rule sequences with the least rule applications between any two designs in the transition graph can be identified by a shortest-path search algorithm. For the tile puzzle only one final state exists, but the presented method can likewise be used for problems with several final states, e.g. the gearbox case study. It would then present the rule sequence to the final state that is found first. In the tile puzzle case study, the design space is explored exhaustively for the small scale problem. The 8-tile puzzle is used to demonstrate that rule sequences learnt for small scale problems can be used to help human designers to solve large scale problems. For

the given puzzle, a rule sequence of 21 rule applications is found, while more sophisticated algorithms might find a shorter sequence of rule applications. The difference can be explained with the sequential subdivision of the 8-tile puzzle in 5-tile regions that restrict rule applications to smaller regions. The aim of this case study, however, is not to search for computer-competitive strategies, but to enable human designers to reason about the search space in a more systematic way.

For both case studies, the automated analysis of the synthesized transition graphs enables the designer to identify loops and alternative sequences of rules. The case studies demonstrate the potential of using visualizations and analysis of transition graphs to strengthen the human designers' understanding of developed grammar rules and the relations between designs and rule sequences. Depending on the problem at hand, it might be possible to explore the search space exhaustively for a smaller sub-problem, as in the tile puzzle case study, or for a limited number of rules in each applied rule sequence, as in the gearbox case study. For larger problems it is also possible to explore portions of the design space using stochastic search algorithms. Then, it is recommended to start the generation process repeatedly and from different initial designs to collect sufficient data and not let the choice of the initial design or the randomness of the algorithm bias the results. The different designs are then combined to one or more transition graphs depending on whether or not the same designs are generated from different initial designs.

A modified sliding tile puzzle is used in research by Vale and Shea [84] that is also directed towards finding beneficial rule sequences. Similar to the research in this chapter do-undo rules are identified. The method proposed in [84] learns performances of rule sequences from previous rule applications during the CDS process and uses ML techniques to decide on future rule applications. The focus is on accelerating the search process and learning during the CDS process. In this thesis, by contrast, the focus is on supporting the human designer. Future research could, however, combine the presented approach with methods like the one presented in [84]. Beneficial and counterproductive rule sequences could be learnt using the Network-based Rule Analysis Methods and provided to the ML-based search process as initial knowledge. Similarly, the ML-based search could provide visualizations of the learnt knowledge to the human designer to further increase their understanding.

5.6 Summary

The Network-based Rule Analysis Method is presented in this chapter. The method can support designers in developing and applying grammar rules. It has been shown that through both manual analysis of the transition graph or computationally through graph search algorithms, loops in rule applications can be identified. LHS matches of rules can be analyzed in detail when transition graphs are generated for exhaustively explored design spaces. Additionally, efficient rule application sequences can be identified through shortest path searches in the transition graph. Exhaustively searching small portions of the search space to collect data and generate transition graphs to visualize relations between different designs and sequences of rule applications can give human designers useful feedback about

the grammar they developed. The Network-based Rule Analysis Method is, therefore, an answer to research questions 5.1 and 5.2:

Research question 5.1: *How can grammar rules in CDS be systematically analyzed to identify the effect of a rule's application location on the generated designs?*

Research question 5.2: *How can grammar rules in CDS be systematically analyzed to identify sequences of rules that are beneficial or counterproductive?*

Through the generation of data, the synthesis of a transition graph and its analysis, grammar rules can be analyzed systematically. Designs where the same rule can lead to different designs depending on the rule's application location can be identified and based on the resulting designs, the effect of the application location can be analyzed. This information can, e.g. be used to improve the LHS of a rule to increase or decrease the number of LHS matches. In the gearbox case studies, the analysis of the LHS of two rules allowed to decrease the number of LHS matches to enable a more efficient search process. The synthesized network can also be used to identify sequences of rules that are beneficial or counterproductive. This has been shown in the sliding tile case study, where beneficial rule application sequences are identified by searching shortest paths in the transition graph.

The Network-based Rule Analysis Method is targeted at supporting grammar development and application. The following research contributions are achieved:

- **Contribution 1:** The Network-based Rule Analysis Method represents a novel approach to analyze grammar rules by combining a graph representation of generated designs and rule applications with network analysis algorithms and interactive visualization.

The method is based on a network where synthesized designs are represented as nodes while rule applications, i.e. transitions between the designs, are represented as edges. Individual rules and sequences of rule applications can be understood when analyzing this network through automated analysis or through visualization.

- **Contribution 2:** The presented method supports confirmatory and exploratory analysis of rules, rule sequences and the designs they generate.

The human designer is given a visual representation of the transition graph and he or she can interactively explore the generated designs and the transitions, i.e. rule applications, between designs. Visualization of the generated transition graph, automated network analysis and the search for paths between designs in the transition graph, either manually or automated through a console application, give the human designer the possibility to accept or reject hypotheses (confirmatory analysis), or to interact with the visualization to explore implicit information about the generated designs (exploratory analysis).

Figure 5-20 shows the positioning of the Network-based Rule Analysis Method with respect to the goals for this thesis. The method supports the rule development process (sub-goal G1) and allows for an improved search (sub-goal G3).

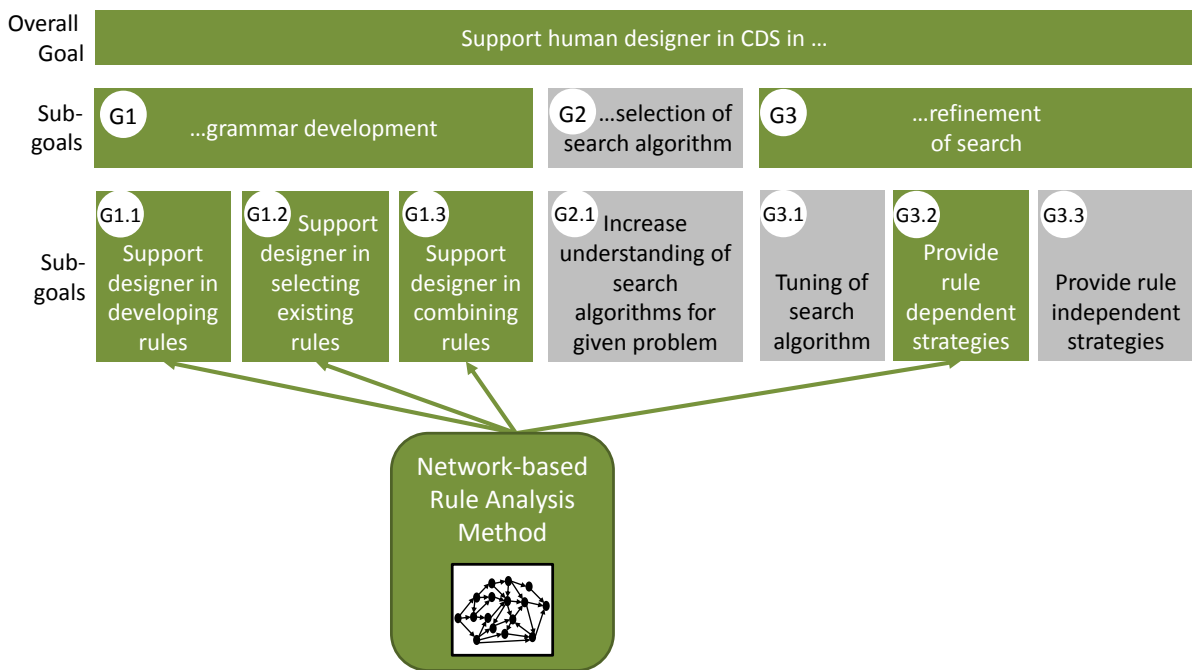


Figure 5-20 Positioning of the Network-based Rule Analysis Method with respect to the goals of this thesis.

The following contributions provide support for human designers:

- **The Network-based Rule Analysis Method allows the analysis of differences between locations where rules are applied and their impact on the synthesized designs.** This analysis enables designers to identify rules for which location matters for the defined scope. This can lead to an increased understanding of the individual rules and their ability to apply at various locations of a design. It can further be used to identify inefficiently formulated LHSs of rules. Modifying such rules can speed up the CDS process.
- **The method enables the identification of rules that can be combined or split.** The analysis of the transition graph and favorable designs allows the human designer to gain insights into the grammar rules. When certain rules always have to be applied in a sequence to generate meaningful results, the human designer can consider combining them into one rule. When, by contrast, one rule modifies designs too drastically, the human designer can recognize this and split the rule into several stepwise modifications, if possible, such that intermediate designs are generated. The Network-based Rule Analysis Method, thus, allows designers to analyze and create balance between knowledge-intensive and simple rules.
- **The presented method increases the understanding of how sequences of rules modify designs.** Analyzing the paths in the transition graph facilitates the understanding of how sequences of rules modify designs. It also gives the human designer insights into the different design alternatives generated by the same rule sequences, due to different locations at which the rules are applied. This information can be used to further develop the grammar, e.g. to further restrict rule application conditions (LHS) when rules are applied too frequently.

- **The Network-based Rule Analysis Method enables the definition of preferable sequences of rule application.** Preferable sequences of rule application, i.e. rules that when combined generate meaningful designs, can be identified. These rules can either be combined to form a more specific rule, or the knowledge about the meaningful sequence can be used to refine the search process.
- **The method enables the human designer to identify sequences of rules that should be avoided.** Avoidable sequences of rules, e.g. loops in the transition graph, can be identified. Avoiding those during rule application can speed up the synthesis process, because their application would not have any effect on the design.
- **The human designer can identify preferable rule sequences on reduced sub-problems.** When problems are decomposable into similar, smaller sub-problems, the knowledge learnt on a reduced sub-problem can be used to solve the larger scale problem, as shown for the sliding tile puzzle.
- **With the developed software prototype grammar rules can be analyzed and the presented method can be evaluated.** The prototype generates data by conducting one or several synthesis runs, synthesizes the transition graph and provides visualizations to the human designer.

6 Relation Visualization Method

Managing the different aspects of CDS requires not only a detailed understanding of each individual part, i.e. representation, evaluation and the search process but also of the interdependencies between them. The method presented in this chapter is aimed at supporting the development and application of grammars and at supporting the selection of an appropriate search algorithm. It aims to answer the following research question:

Research question 6.1: *How can search processes in CDS be visualized to better understand how grammar rules and a search algorithm explore the design space?*

The research methodology used to answer this question is to develop a method that allows the visualization of design space exploration and to implement it in a software prototype. The practical applicability of the method is then validated through two case studies. The structure of the chapter is as follows. Section 6.1 motivates the use of visualizations in CDS to visualize relations between grammar rules, performance objectives and design space exploration when using CDS approaches with search algorithms. In Section 6.2, the method is presented that visualizes the progression of the search algorithm in CDS. The method is then validated using two engineering design problems. The synthesis of bicycle frames is described in Section 6.3, the synthesis of gearboxes in Section 6.4. Results are presented (Section 6.3.4 and Section 6.4.1) and discussed (Section 6.5), including a discussion of the generality of the method. Section 6.6 summarizes the chapter and revisits the research question and the expected contributions of the presented method with respect to the overall goal of the thesis.

6.1 Motivation for Visualizations in CDS

In CDS, search and optimization algorithms are frequently used to guide the synthesis process. Selecting an appropriate algorithm is challenging. Knowing characteristics of the design space can support the selection of search algorithms [25], however, these characteristics are often not known. When comparing different algorithms on a problem, or when tuning algorithm settings, usually only the final designs, e.g. the number of valid designs generated, or metrics to evaluate the synthesis process, e.g. the convergence time of the algorithm, are considered. Designs generated in earlier stages of the synthesis process or ones that are rejected by the synthesis algorithm are commonly not considered.

The goal of the research in this chapter is to support engineering designers using CDS methods through visualizing how their selected search algorithm explores the design space. The focus is on making the search process explicit. Visualization and animation techniques are used for that purpose. These techniques are successfully applied in the field of software visualization and algorithm animation, e.g., to support communication between experts and to debug and optimize programs. The main focus of algorithm animation is, however, to support education in computer science [131]. A new method is presented that uses

animation techniques to visualize the progression of the search algorithm. The explored design space is represented in the context of design objectives and design characteristics, e.g., the structure of a design. The order in which designs are generated and the grammar rules causing design transformations are visualized. Being presented how a search algorithm and rules explore the space, the human designer can more easily understand how to steer the search into different directions by changing the search algorithm or the grammar rules. The method can, thus, help novices to gain a deeper understanding of the interplay between grammar rules and guidance of the synthesis process. Experts can use the method to analyze and further improve their CDS application, e.g., by improving parameter settings of the used search algorithm.

6.2 Method

In this research, a new approach to visualize the progression of search algorithms during CDS is presented. Changes considering the objectives as well as the topologies are represented to make the engineer understand not only the algorithm, but also the grammar that is used to generate design alternatives. The scope of the presented method are CDS methods that use grammars to generate new designs and that use search and optimization algorithms to guide the synthesis process. The method supports multi-objective algorithms. It is described in the following for the use of graph grammars. Its generality to other grammars, e.g. shape grammars, is discussed in Section 6.5.

6

Figure 2-3 presents a framework for CDS methods. Problem-specific knowledge is formalized, i.e. problem description, objectives and constraints are defined by the human designer. Appropriate simulation models or heuristics to evaluate generated designs are made available and a search algorithm is selected before the CDS process is started. The synthesis method itself is then often seen as a black box converting a given initial design into meaningful final designs. When the human designer is not content with the generated designs, the CDS process is usually restarted using, e.g., different problem formulations, initial designs, or different algorithm settings. Figure 6-1 presents the Relation Visualization Method and its integration in the CDS framework defined in [25] (see also Figure 2-3). Data acquisition is performed throughout the search process. The collected data is analyzed and visualizations are generated and presented to the human designer. This makes the CDS method more transparent as it visualizes the whole process. This includes, e.g., not only successful designs, but also those that are rejected by the algorithm.

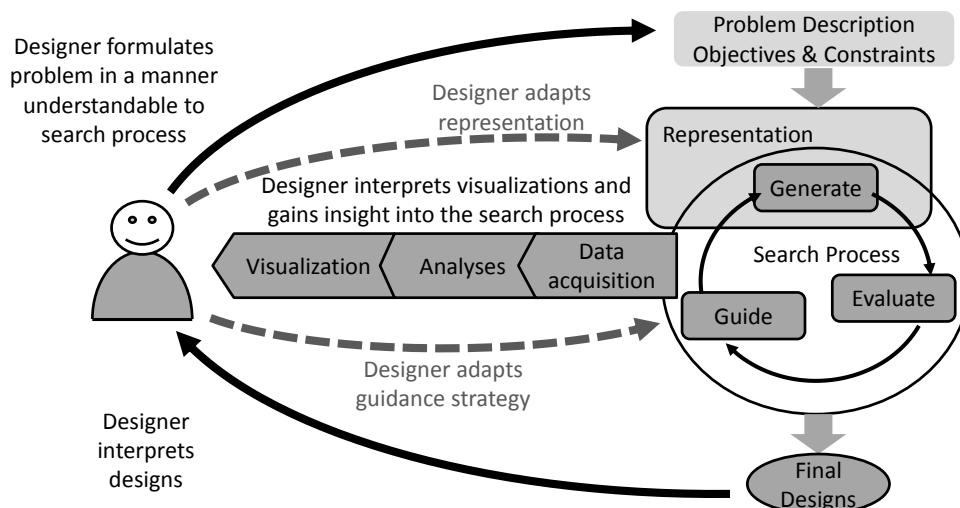


Figure 6-1 Suggested generic CDS process (adapted from [25]).

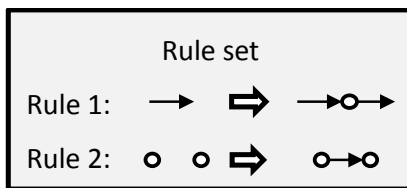
Similar to the visualization pipeline described in [108], the presented method consists of three steps (see also Figure 6-1): 1) data acquisition during CDS, 2) analyses, i.e. post-processing the data, and 3) presenting it to the human designer using static visualizations and animations. Data acquisition is conducted during the whole CDS process. Data consists of design related and process related data. Design related data includes the topology of each generated design, i.e. its graph representation, as well as the objective and constraint values for each design. Process related data is the iteration number (= index of the design), the index of the previous design and the applied rule. They are stored for each iteration of the CDS process.

In the second step, the collected datasets are analyzed. Unique designs are identified where uniqueness is a property that is problem dependent and is to be predefined by the user. It can, e.g., be unique topologies in a structural design problem or designs with different parameter values for parametric problems. For the example data (Figure 6-2) in this method description, uniqueness refers to designs with a unique topology. Thus, to identify unique designs, the topologies of all designs are compared to each other and each design is assigned a topology ID referring to a unique topology. For all objective function values and constraint violations, the change in the value is calculated for every iteration. Note that to calculate the change in objective and constraint values at iteration i the values of the design generated in iteration i have to be compared with those from the design on which the change in iteration i is applied. This does not necessarily have to be the design generated in iteration $(i-1)$ but can also be a design generated earlier in the process, e.g. when the design generated in iteration $(i-1)$ was rejected.

6.2.1 Example to Demonstrate the Method

To demonstrate the method, example data with datasets for six iterations is given in Figure 6-2 along with the corresponding rule set. The last column of the table shows the unique topology IDs identified in the post-processing step. The labels “#” for the index of each

generated design and “§” for the unique topologies are used to clarify the difference between the two.





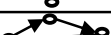




Process related data			Design related data			Unique topology ID
Iteration number (=index)	Index of previous design	Applied rule	Graph representation	Objective 1	Objective 2	
#0	-	-		8	10	§0
#1	#0	1		7	8	§1
#2	#1	2		3	7	§2
#3	#2	1		4	12	§3
#4	#2	1		10	3	§4
#5	#1	1		2	6	§5
#6	#5	2		1	2	§4

Figure 6-2 Example data to visualize the Relation Visualization Method. At the top an example rule set is shown consisting of two rules. The table (bottom) shows datasets for six iterations. The last column of the table shows the unique topology IDs identified in the post-processing step.

Based on the datasets, three visualizations are generated: a) the *unique topology space*, and b) the *performance space*, and c) *rule analysis plots* (Figure 6-3).

The **unique topology space (UTS)** (Figure 6-3, left) presents the unique topologies identified in the graph isomorphism check. All generated designs and their relations by means of rule transformations are visualized in the UTS. Each unique topology is represented by a node and the transformation between the topologies is represented by an edge, labeled with the number of the rule that transformed the topology. Each node is labeled with unique topology ID (§) of the topology it represents.

For each iteration step one the following three options applies:

1. The applied rule changed the topology (§A) and a design with a previously undiscovered topology (§B) is generated. Then a new node (§B) is added to the UTS and the current node (§A) is connected to this node (§B) with an edge labeled with the rule number that created this transformation.
2. The applied rule changed the topology (§A) and a design with an already explored topology (§B) is generated. Then the node representing the current design (§A) is connected to the node representing the resulting topology (§B) and an edge labeled with the rule number is added that connects the two nodes. Additionally the size of the revisited node (§B) is increased to show that it is visited repeatedly.

- The applied rule does not change the topology ($\$A$). Then only the size of the current node ($\$A$) is increased to show that it is visited repeatedly.

The graphs inside the nodes in Figure 6-3 are presented for illustrative purposes only. For a larger number of designs, only the nodes are displayed.

In the **performance space (PS)** (Figure 6-3, top right) the performances of the generated designs, i.e. their objective function values, are plotted in a multidimensional space defined by the objectives. This means that each design generated during the CDS process is represented by a data point in the PS. For the given datasets (Figure 6-2), a two-dimensional plot is sufficient representing the two objectives. The rules transforming one design into another design are depicted as arrows between the data points in the plot, representing the change of the performance in vector form (Figure 6-3, right). The labels of the data points (#) here refer to the iteration in which the design is generated, i.e. the index of the design.

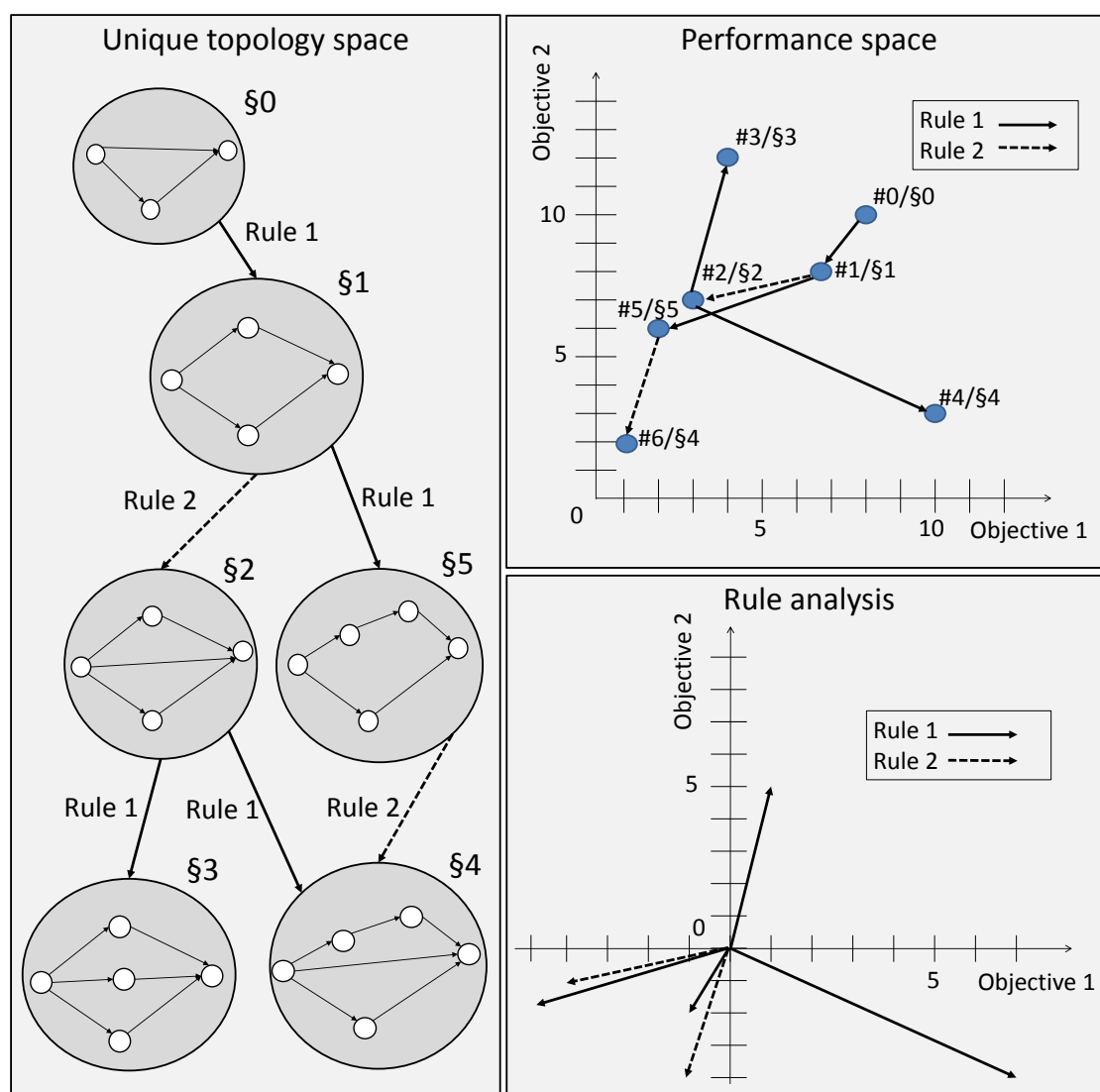


Figure 6-3 Unique topology space (left) with nodes representing unique topologies ($\$$) and edges representing rule applications, performance space (top right) with nodes representing positions of designs (#) in the performance space and vectors representing changes of objective function values due to rule applications, and rule analysis plot (bottom right) representing changes by rule applications as vectors.

Rule analysis plots (Figure 6-3, bottom right) help to increase the understanding of how each individual rule influences the objective function values. All vectors of the PS are visualized in a separate plot with their starting points set to the origin. Using this visualization, it can, e.g. be easily seen that for the given example rule 1 can increase and decrease both objectives, while rule 2 always decreases objective 1 and objective 2.

The visualizations in Figure 6-3 show the progression of the algorithm from iteration 1 to 6. They enable interpretations to increase the understanding of the problem:

- For each rule, it can be analyzed whether it changes the topology or not and which objectives it influences and to what degree.
- For each design, it can be analyzed which are its predecessor and successor, considering both topology and objective function values. It is interesting to see in the example above, that in iteration 6, when rule 2 is applied to the design from iteration 5, the same graph as in iteration 4 is generated (UTS). Considering the performances (PS), however, there is a difference between the design generated in iteration 4 and the one generated in iteration 6.

6.2.2 Implementation Details

The method is integrated in the generic framework that is presented in more detail in Chapter 8. It is implemented in c# and provides an interface to Matlab in an evaluation module. Different search algorithms are implemented in a guidance module. The UTS is visualized using OrganicVIZ [132], a graph visualization tool capable of representing large graphs and supporting graph analyses as well as providing several filtering options. OrganicVIZ was developed to support visually augmented analysis of complex socio-technical networks for engineering systems design. It provides a slider for visualizing the process stepwise. The synthesis process can be replayed with this slider. The node representing the topology of the currently changed design is highlighted. The PS and rule analysis representations are visualized using an implemented Matlab application presenting the data and providing a slider and some selection options for the user to navigate through the data. When sliding the slider from the first to the last iteration, the progression of the whole synthesis process can be observed. Visualizing the progression in time facilitates the understanding of the algorithm and the relationship between rules, rule sequences and resulting designs in the design space.

6.3 Case Study 1: Bicycle Frame Synthesis

A bicycle frame synthesis task is taken as a first case study to demonstrate the Relation Visualization Method. The bicycle frame design case study is used in Chapters 6 and 7 in this thesis to validate the presented methods. It is introduced in detail in the following.

6.3.1 Introduction to the Bicycle Frame Synthesis Case Study

Existing research on bicycle frame synthesis focuses on multi-objective optimization using simulated annealing [109] or shape annealing [133]. Bicycle frames are commonly known

products that are available with different topologies, shapes and in different sizes. Research on synthesizing bicycle frames using CDS methods has generated promising results in the past, which makes them a reasonable case study.

6.3.1.1 Problem Formulation

The task is adapted from [133] and is to synthesize a bicycle frame according to the following three objectives:

- 1) minimize the mass of the frame (decreases material cost)
- 2) minimize the number of joints (decreases manufacturing costs)
- 3) minimize the mean deflection of the frame (maximizes stiffness of the frame)

The following constraints are imposed:

- 1) the maximum stress in each bar ≤ 750 MPa
- 2) the maximum deflection in x- and y direction of each bar ≤ 5 mm

The synthesis process is started from a given frame geometry (see Figure 6-4). The positions for the rear wheel, crank, saddle, handle bar and head tube are fixed in this case study, i.e. their positions are not modified during the synthesis process.

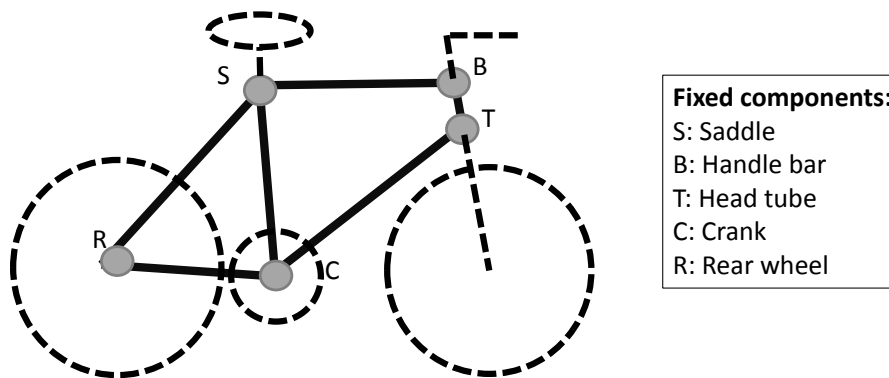


Figure 6-4 Initial design of the bicycle frame.

6.3.1.2 Representation

A graph grammar is used in this case study and is implemented using GrGen. Two node types are used to represent the frame as a graph, namely joints and bars. Joint nodes possess attributes for x- and y- positions, whereas bar nodes possess attributes for inner and outer diameter. Joint and bar nodes are connected via undirected edges to show the connectivity of nodes. Using this representation, the length of each bar can be easily calculated from the x- and y-positions of the joints it is connected to. Figure 6-5 gives an example graph representation of the bicycle design shown in Figure 6-4.

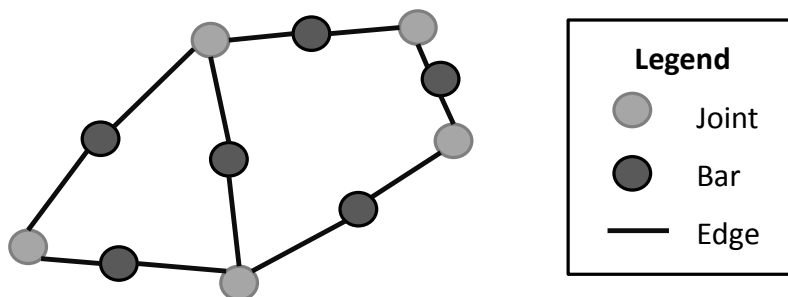


Figure 6-5 Graph representation of the initial design.

A rule set of nine rules is defined with four topologic and five parametric rules. All rules are applied with equal probability in the synthesis process. Topologic rules change the graph topology by adding or deleting nodes and edges, e.g. adding a bar between two joints. When a bar is added, its diameter and thickness are selected randomly within defined ranges. Parametric rules change attribute values of nodes, e.g. moving the position of a joint. An overview of the rules is given in Figure 6-6.

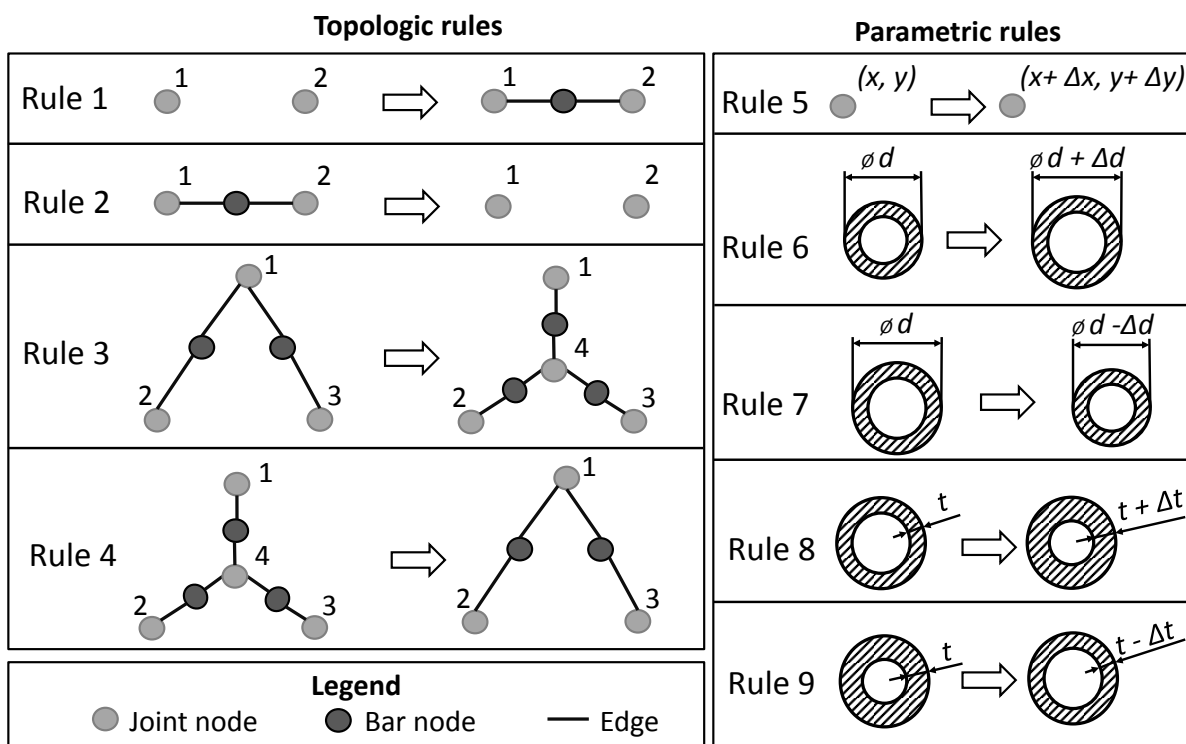


Figure 6-6 Schematic representation of the rule set.

Rule 1: Two previously unconnected joints are connected with a bar.

Rule 2: A bar is removed; this rule only applies when the remaining joints still form a valid frame geometry, i.e. the fixed components are connected to form a frame.

Rule 3: A v-type connection of three joints consisting of two bars is replaced by a star-type connection consisting of three bars and one additional node.

Rule 4: A star-type connection consisting of three bars is replaced by a v-type connection consisting of two bars; this rule only applies when the joint to be removed (joint 4) is not a fixed component.

Rule 5: The position of a joint is moved resulting in changes of the lengths of the bars connected to it; this rule is not applicable to joints representing a fixed component.

Rule 6: The diameter of a bar is increased, the thickness remains the same as before.

Rule 7: The diameter of a bar is decreased, the thickness remains the same as before.

Rule 8: The thickness of a bar is increased, the outer diameter stays the same.

Rule 9: The thickness of a bar is decreased, the outer diameter stays the same.

6.3.1.3 Evaluation

In each iteration of the CDS process, the generated design is evaluated. The graph representation is transformed into a vector and matrix format and the Finite Element Method (FEM) is used for structural analysis to evaluate the objective function and constraint values of the design. Beam elements are used to model the bicycle frame as suggested in [134]. The FEM is implemented in Matlab and uses one-dimensional beam elements. Although different load cases are required to fully analyze a bicycle frame (see, e.g., [109, 133, 134] for details on different load scenarios) only one load case is used in this study. A full analysis of the bike is not required to validate the method in this section but a sufficiently accurate assessment of frame designs, which is approximated with the following load case. The frame is fixed at the rear wheel mounting position and a load of 1,000 N is applied to the saddle.

6.3.2 Search Algorithms

Two different algorithms are used to guide the synthesis process, the Burst algorithm and the Simulated Annealing (SA) algorithm. Strengths of the Burst algorithm are its modular structure and easy adaptability to the design task. It is not tuned to any engineering problem, which makes it a suitable algorithm for the generic framework. It is developed for a problem independent modification-based framework similar to the one used in the presented research and was successfully applied to different design problems in the past [135, 136]. As a second algorithm, an implementation of the SA algorithm is used. Both algorithms are presented in more detail in the following.

6.3.2.1 Burst Algorithm

A schematic overview of the Burst algorithm is given in Figure 6-7. During the algorithm execution “Bursts” of rule applications are applied. The length of each Burst is randomly selected between one and a maximum Burst length defined by the user. In each iteration of a Burst, one rule is applied and the changed design is selected for the next iteration, independent of its performance. Once the Burst is finished the length for the next Burst is defined and a design is selected randomly from the archive to start the next Burst with. The synthesis process starts from an initial design and stops after a predefined number of iterations. For storing promising designs during algorithm execution, a Pareto archive is used. The archive is initialized with the design in the first iteration, i.e. the initial design. In each

iteration, the generated design is compared to the designs stored in the archive to see if it is a new Pareto-optimal design. To qualify for the archive, a design has to meet all constraints and be Pareto-optimal according to the objectives, e.g. mass, number of joints and mean deflections for the bicycle frame case study. If yes, it is stored in the archive and the archive is updated. Archive updates are performed using a global Pareto filtering algorithm as described in [137] to eliminate dominated designs. When the archive size exceeds the predefined maximum size of the archive, it is reduced. Archive reduction is done stepwise. First, the archive is analyzed and for each design, the distances to all other designs in the archive are calculated according to the following formula:

$$Distance(Design\ i) = \sum_{j=1}^n \left\{ \left[1 - \frac{Objective1(Design\ i)}{Objective1(Design\ j)} \right]^2 + \left[1 - \frac{Objective2(Design\ i)}{Objective2(Design\ j)} \right]^2 + \left[1 - \frac{Objective3(Design\ i)}{Objective3(Design\ j)} \right]^2 \right\}^{\frac{1}{2}} \quad (2)$$

The design with the smallest distance to all other designs is deleted, thus removing a design from the most explored region of the search space.

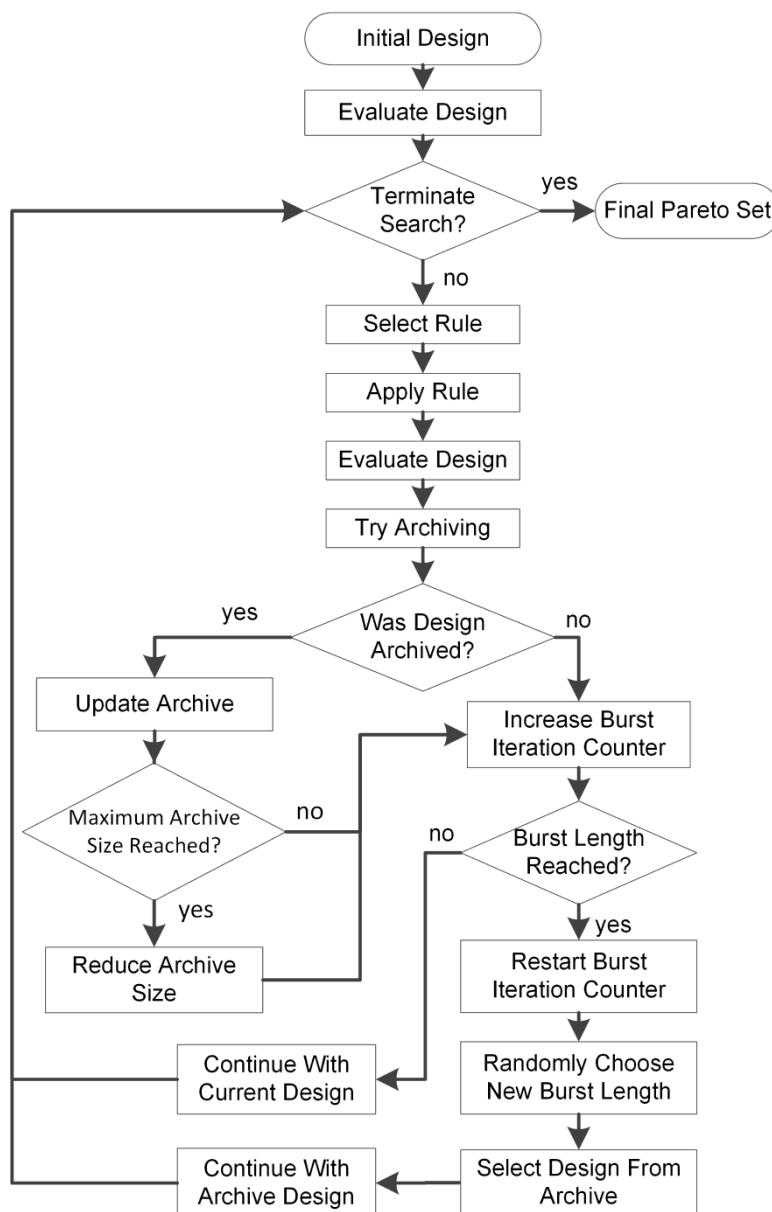


Figure 6-7 Schematic overview of the Burst algorithm.

6.3.2.2 Simulated Annealing Algorithm

An overview of the simulated annealing (SA) algorithm used in this case study is given in Figure 6-8.

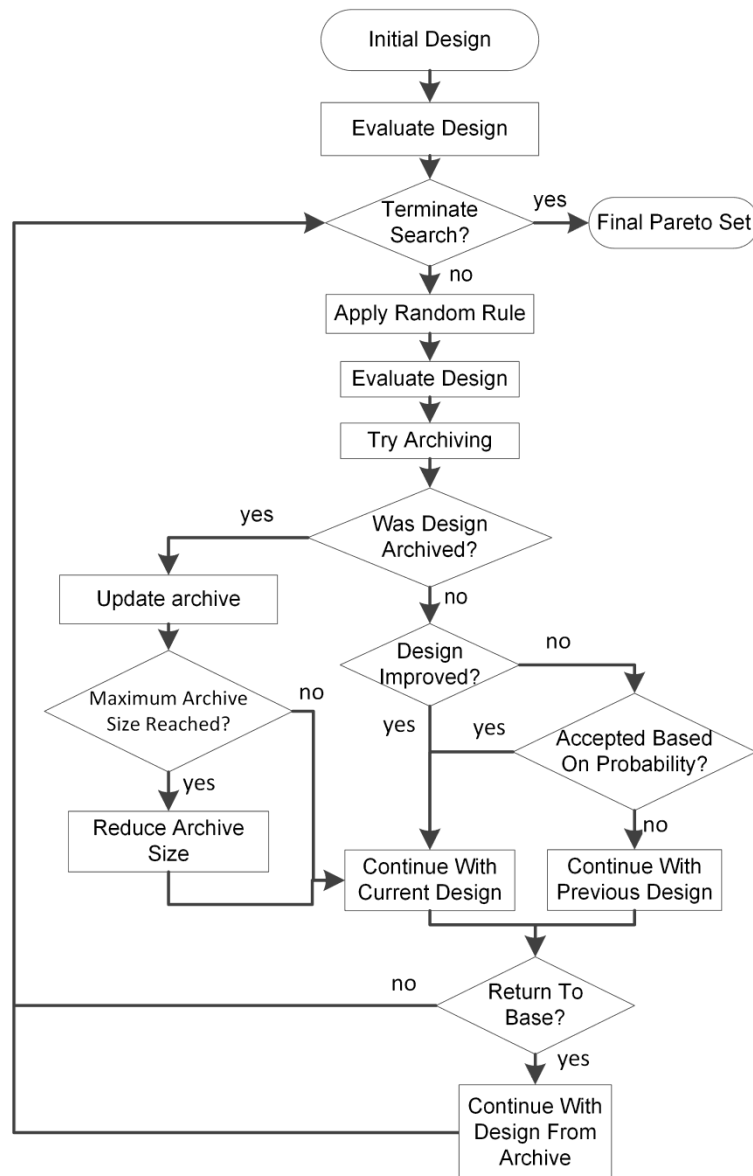


Figure 6-8 Overview of the SA algorithm.

Similar to the Burst algorithm, a Pareto archive is used to store non-dominated designs. In each iteration, the generated design is compared to the design in the previous iteration. When the design improved considering the objectives, i.e. when at least one objective function value decreased, it is accepted and taken as the starting design for the next iteration. When the design did not improve it is still accepted with a certain probability. The probability $P(i)$ to accept an inferior design is calculated based on the iteration according to formula (3).

$$P(i) = e^{-\Delta f(i)/T(i)} \quad (3)$$

$\Delta f(i)$ is the change in the objective function value resulting from the rule application in iteration i . A weighted sum is used to calculate the objective function value $f(i)$ for the design generated in iteration i :

$$f(i) = 10 \cdot \text{mass}(i) + 10000 \cdot \text{meanDeflection}(i) + \text{numberOfJoints}(i) \quad (4)$$

The weights are defined to yield the individual objectives to the same order of magnitude. $T(i)$ is the temperature as a function of the current iteration i and is calculated according to formula (5).

$$T(i) = 100 \cdot 0.9^i \quad (5)$$

This means that in the beginning of the synthesis process, inferior designs are accepted with a higher probability to support the exploration of the design space. Towards the end of the process, inferior designs are accepted with a lower probability to exploit already explored designs.

For multi-objective SA, often a return to base (RTB) option is implemented, where the process is restarted periodically from a design found earlier in the process, e.g. a design stored in the archive.

6.3.3 Scenarios

Results are generated for the following scenarios:

- Burst algorithm with maximum Burst length set to 1, 5, and 20
- SA without RTB
- SA with RTB every 100 iterations

For each scenario, ten runs with 1,000 iterations each are conducted. The archive size for all scenarios is limited to ten designs.

6.3.4 Results

Designs for all scenarios are generated and the corresponding UTS, PS and rule analysis plots are visualized. Figure 6-9 visualizes how a UTS plot can be animated to gain insights about the algorithm. The Burst algorithm is used to generate the data for this visualization with the maximum Burst length set to five. The UTS is presented for four different stages of the synthesis process: after 5, 20, 100 and 1,000 iterations. Subfigure a) shows the initial design and three of its successors after five iterations. It can be seen that one of the two direct successors of the initial design (the right one) is generated but not further explored. The other one is explored further. This can be seen by the increased node size and the succession of another design. When the size of a node increases this means that either a parametric rule is applied to a design with this topology or that the application of a topologic rule on another design resulted in this topology. Subfigure b) shows that the synthesis process is continued from the initial design and a new topology is generated and often revisited (upper right node). Subfigure c) visualizes how the synthesis process explores more topologies based on this recently explored design until iteration 100, and subfigure d) demonstrates the

remaining designs generated until iteration 1,000. It can be seen that after iteration 100, two more topologies are strongly exploited (large node size) and many more topologies are explored based on those designs.

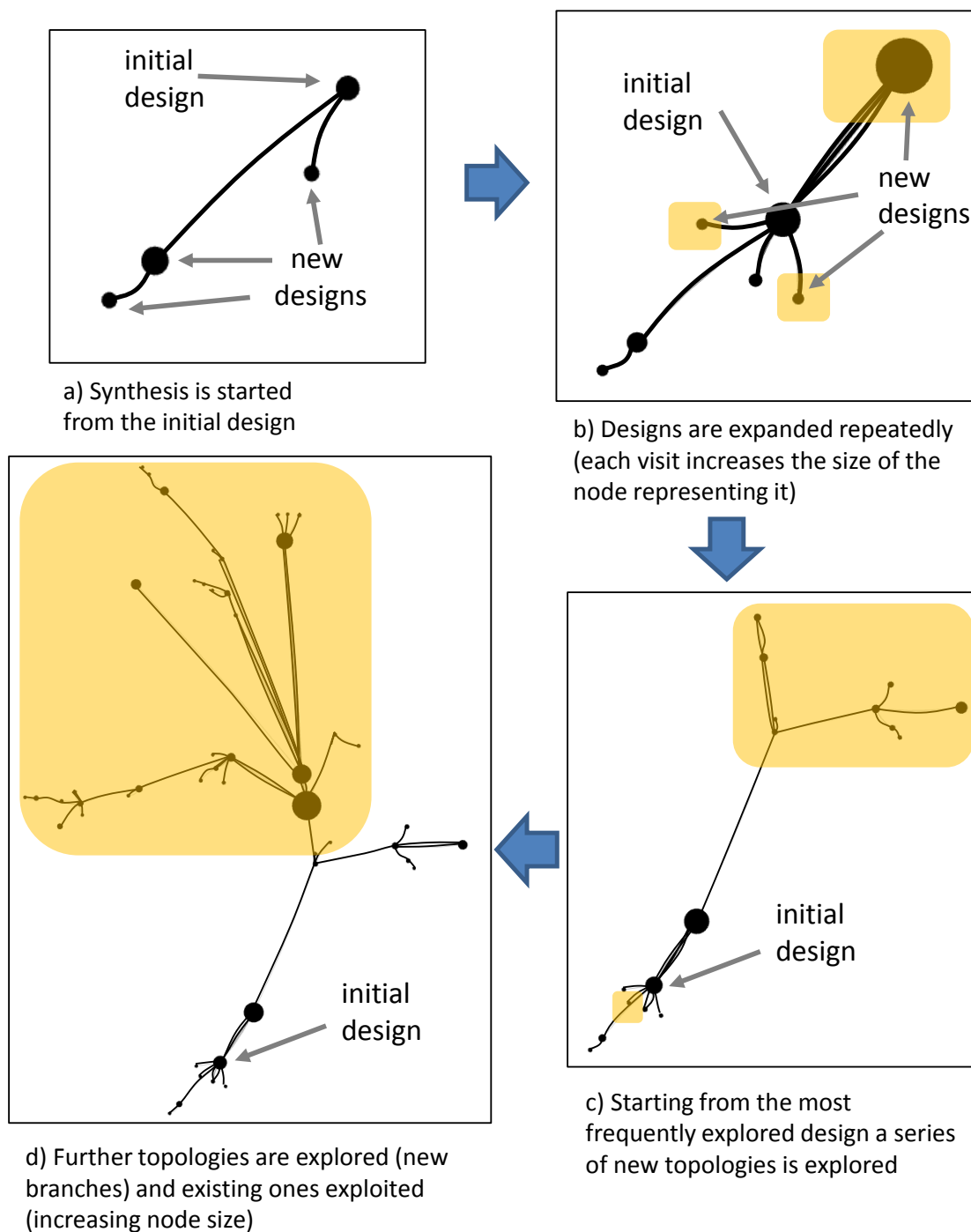


Figure 6-9 Visualization of the progress of the algorithm in the unique topology space (UTS). a) shows the start and the following subfigures (b), c), d)) show how new topologies are explored (new nodes) and existing ones are exploited (increasing node size). Designs that are explored from one subfigure to the next are highlighted.

Figure 6-10 shows the UTSs for different versions of the Burst algorithm. The visualizations show all unique designs generated after 1,000 iterations. The Burst algorithm with a Burst length of one is run first and the UTS is visualized (Figure 6-10, top). From the UTSs it can be

observed that only few topologies are exploited extensively. Considering the generated designs, it is found that only one or two topologically different designs remain in the final Pareto set in each run. Increasing the maximum Burst length results in the generation of more unique topologies.

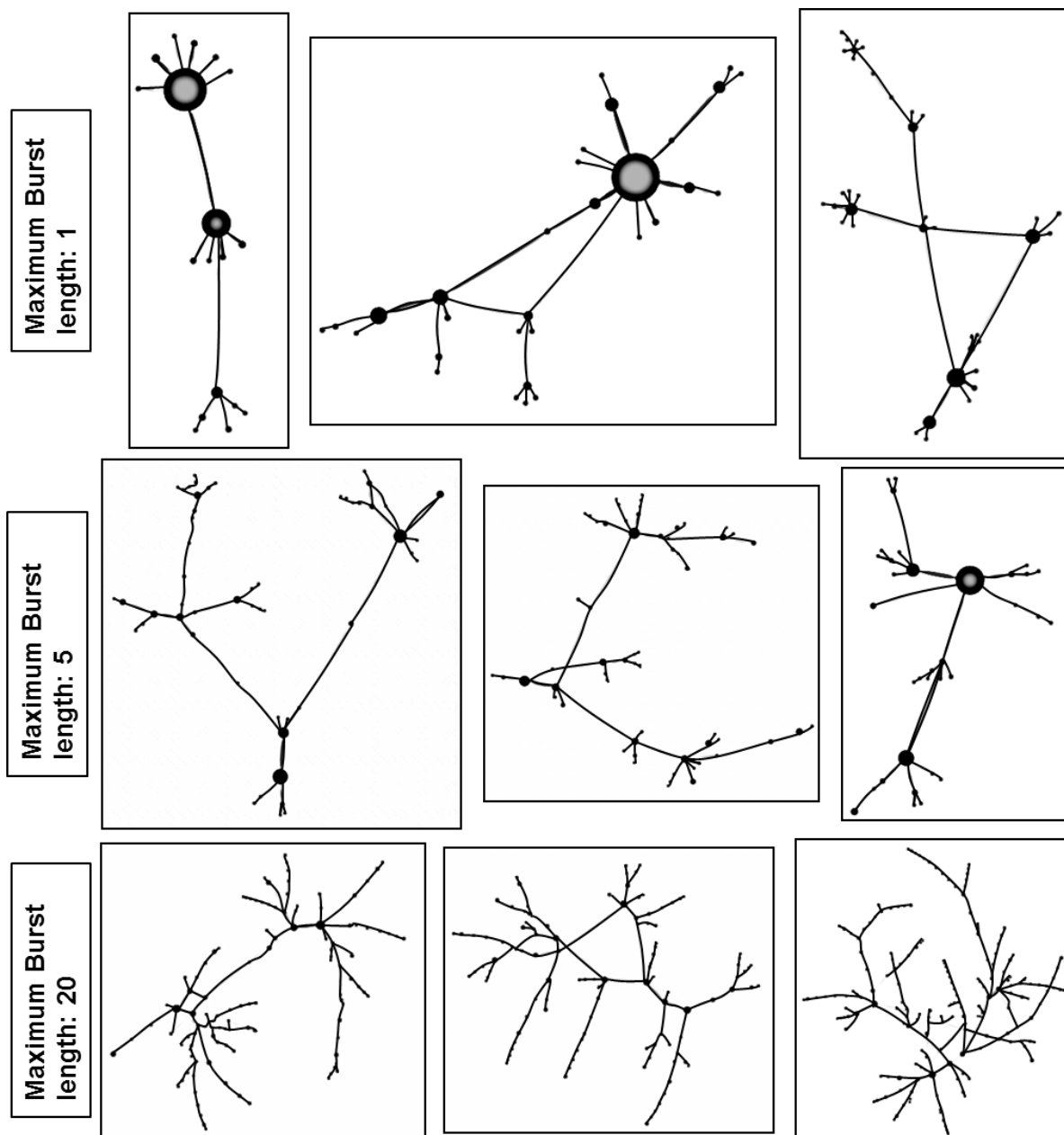


Figure 6-10 Unique Topology Spaces (UTSs) for three example runs with 1,000 iterations each, using the Burst algorithm with the maximum Burst length set to 1 (top), 5 (middle) and 20 (bottom).

Visualizing the UTSs for maximum Burst lengths of five and 20 (Figure 6-10, middle and bottom), it can be observed that more different topologies are explored. Comparing the UTSs for the three different Burst lengths it can be seen that the selection of the Burst length has an influence on how the algorithm explores the design space. When the Burst length is set to a low value only few unique topologies are generated and some of them are exploited. The extreme case with a Burst length of one, which results in a greedy search, shows that the search often focuses on only one or two topologies. With larger Burst lengths, a larger

number of different topologies are generated. This means that the algorithm explores the space to a greater extent with respect to different topologies, however, for each topology less parametric changes are explored. The designer can analyze the effect of exploring more topologies in the UTs while the effects of exploiting already generated topologies by searching for optimized parameters can be observed in the PS plots, which are explained later.

UTS plots for both versions of the SA algorithm are shown in Figure 6-11. The plot on the left shows how the SA algorithm continues to apply rules to the previous design unless it is rejected. This can be observed by the chain-like representation in the UTS plot. Note that the curvy representation is due to the visualization settings in OrganicViz aiming to represent graphs in organic forms. The zoomed in view (Figure 6-11, bottom) shows topologies that are generated but not further explored (only one incoming and no outgoing edge) as well as some that are generated, but in a following rule application the previous topology is recreated. On the right in Figure 6-11, the UTS for the SA algorithm with RTB every 100 iterations is shown. It can be observed, that the chain-like structure of the SA algorithm still exists, however, for some of the restarts new chains are followed from previously generated topologies.

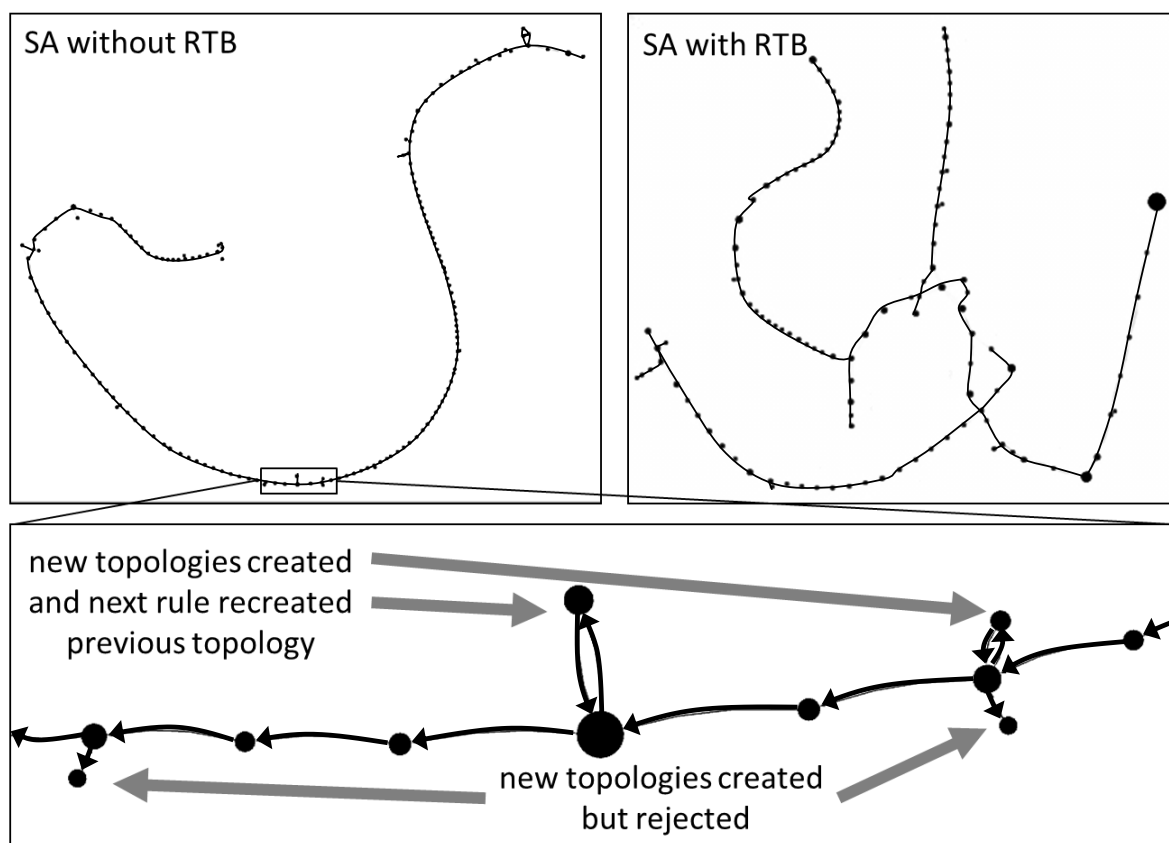


Figure 6-11 Unique Topology Spaces (UTSs) of example runs with 1,000 iterations each; left: SA without RTB with zoomed in view (bottom); right: SA with RTB every 100 iterations.

Figure 6-12 shows an example PS representation for one synthesis run of the SA algorithm without RTB. Even though it is hard to visualize a 3D space (Figure 6-12 a)) on paper, the reader can observe which areas of the PS are explored during the synthesis process.

Additionally, it is possible to analyze which rules change which objectives. Figure 6-12 b) represents a projection of the PS representation on the “*mass-numberOfJoints*”-plane. Figure 6-12 c) and d) show the projected plot split up into topologic (Figure 6-12 c)) and parametric (Figure 6-12 d)) rules. It can be observed that the topologic rules (black, solid lines) can change the number of joints, whereas the parametric rules (grey, dashed lines) cannot but influence the mass.

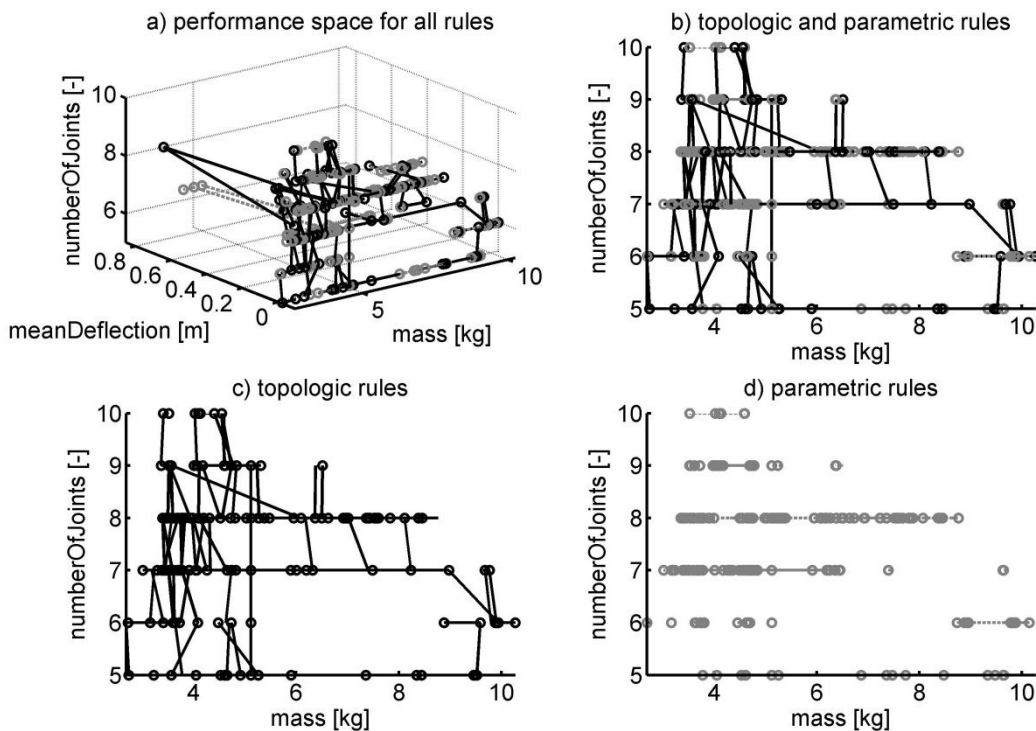


Figure 6-12 Example performance space (PS) plots. Topologic rules are presented with black, solid lines, parametric rules with grey, dashed lines.

In Figure 6-13 the rule analysis plots are presented for the parametric rules 8 and 9 (Figure 6-13, left) and the topologic rules 3 and 4 (Figure 6-13, right). It can be seen that increasing the thickness of a bar adds mass to the frame but for most cases decreases the mean deflections, while decreasing the thickness results in the opposite. Similarly, it is visualized that rule 3 always adds a bar, while rule 4 removes one and both rules either add or remove mass from the frame, depending on where the rule is applied. Comparing the designs generated in all scenarios (defined in Section 6.3.3), there is no noticeable difference regarding performance or number of different topologies generated in the final Pareto archive. The only exception is the Burst algorithm with the maximum Burst length set to one, for which only one or two different topologies remain in the final Pareto archive.

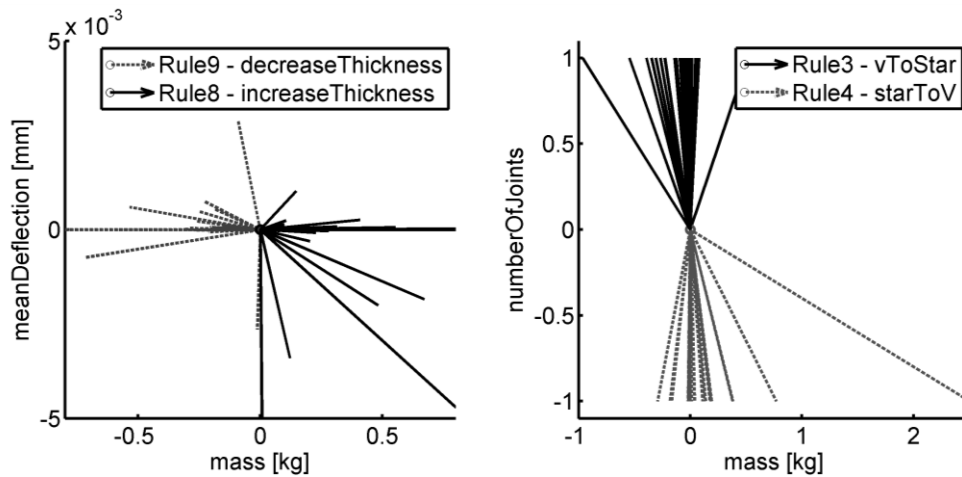


Figure 6-13 Example rule analysis plots for parametric (left) and topologic (right) rules.

Six example designs generated with the Burst algorithm with a maximum Burst length of 20 are presented in Figure 6-14.

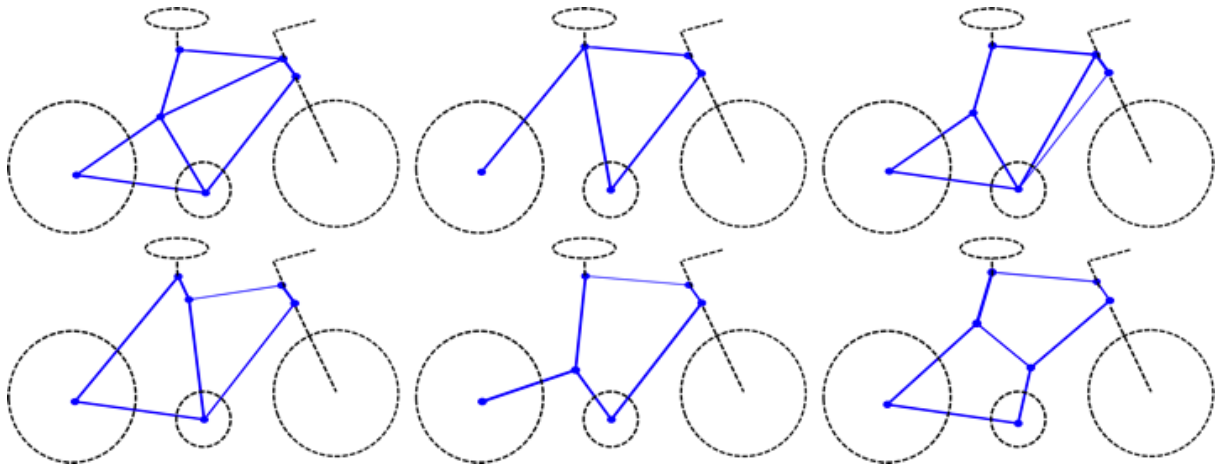


Figure 6-14 Example bicycle frames generated with the Burst algorithm with a maximum Burst length of 20.

6.4 Case Study 2: Gearbox Synthesis

The synthesis of gearbox designs is taken as a second case study. A detailed problem formulation for the gearbox synthesis task is given in Section 4.3.1. Ten experiments are conducted for each scenario described in Section 6.3.3. The Burst and SA algorithm are used as described in Section 6.3.2. The resulting visualizations are presented in the following. Special attention is given to aspects where visualizations are different from those in the bicycle frame case study.

6.4.1 Results

Figure 6-15 visualizes the progression of the synthesis process through four UTS representation for an experiment using the Burst algorithm with a maximum Burst length of 20. The UTS is presented after 5, 20, 100 and 1,000 iterations respectively. Each node represents a unique gearbox topology. The nodes are colored whereby each color represents a combination of design characteristics. The number of forward and reverse speeds are defined as design characteristics in this case study, i.e. nodes with the same color represent

gearbox designs with different topologies but with the same number of forward and reverse speeds.

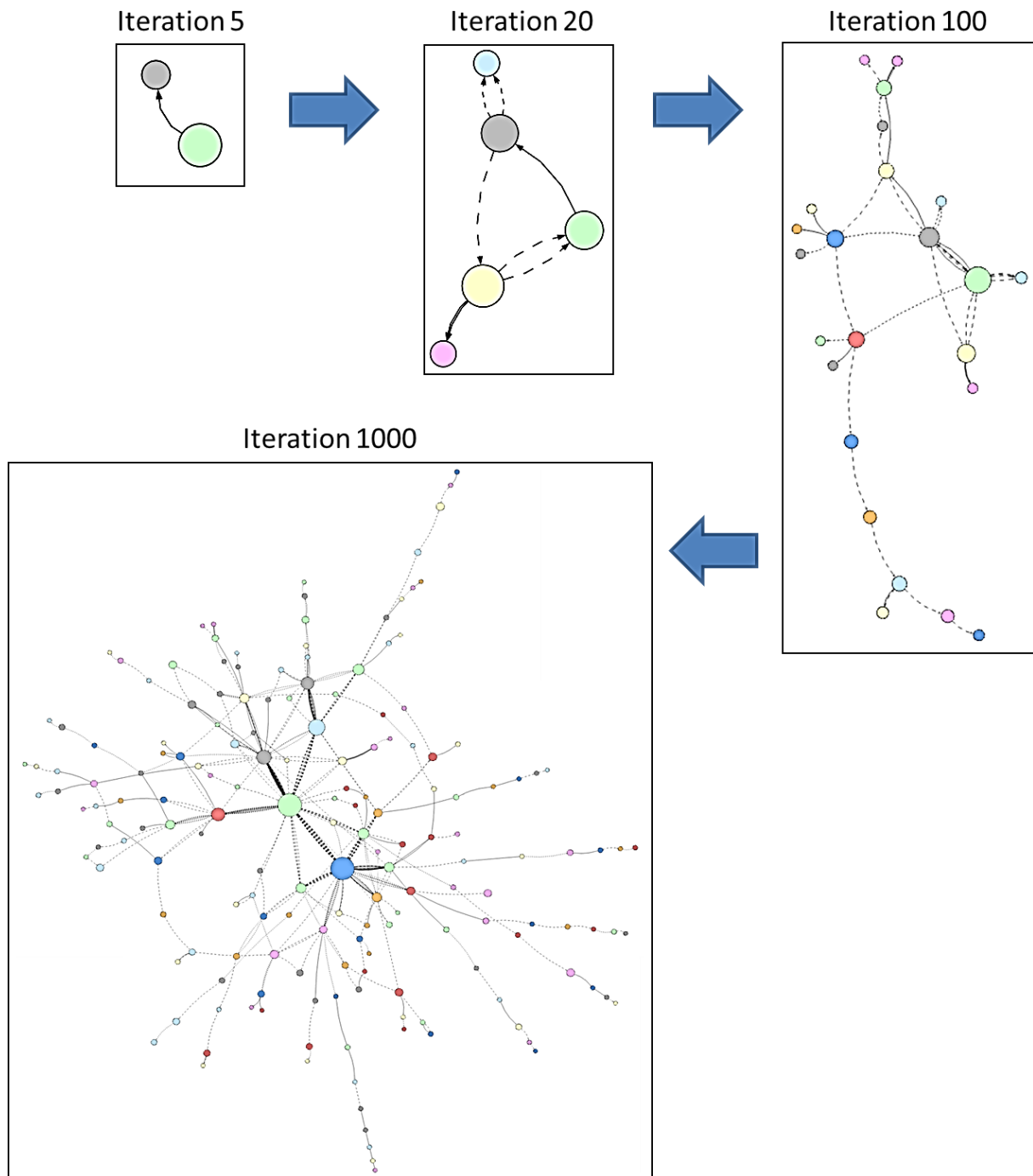


Figure 6-15 Visualization of the progress of the search space exploration in the unique topology space (UTS) for the gearbox case study. The Burst algorithm with a maximum Burst length of 20 is used.

In Figure 6-16, the UTs for three experiments with maximum Burst lengths of 1, 5 and 20 are visualized. The effect of changing the search algorithm parameter “maximum Burst length” on the explored design space is obvious when comparing these visualizations. When the maximum Burst length is set to one, the same four topologies are discovered for all experiments.

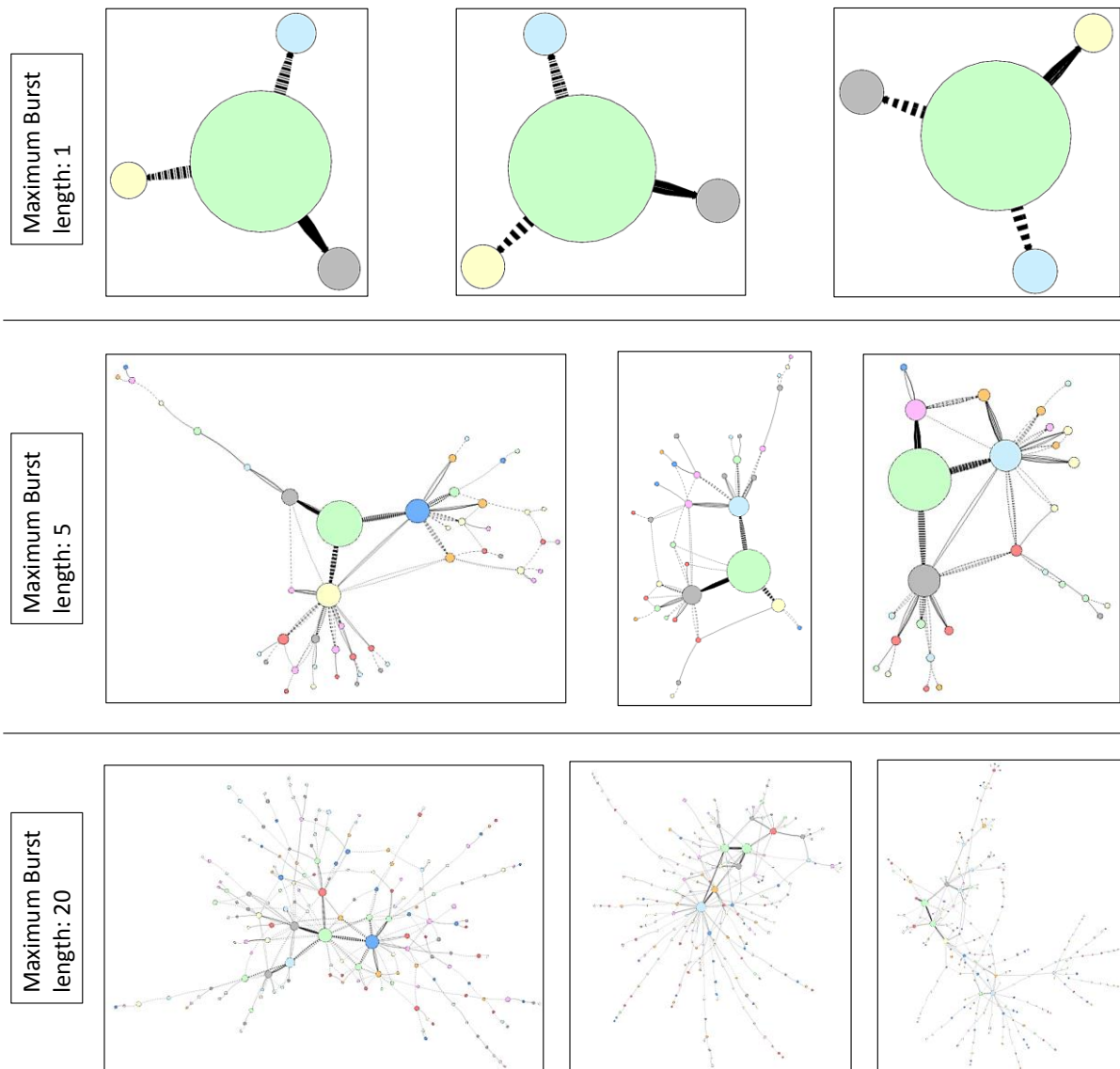


Figure 6-16 Unique Topology Spaces (UTSs) for three example runs with 1,000 iterations each, using the Burst algorithm with the maximum Burst length set to 1 (top), 5 (middle) and 20 (bottom).

Figure 6-17 represents the UTS explored when the maximum Burst length is set to one and visualizes pictograms of the gearboxes that are represented by these four nodes. Only three topologic rules (*add a shaft*, *add a gear pair*, *replace a gear pair*) find a LHS match on the initial design. Applying each of these three rules to the initial design generates the three shown topologies. None of these topologies fulfills the criteria (*number of speeds == desired number of speeds*, i.e. five forward and one reverse speed) to be stored in the archive. This means the next Burst restarts from the initial design, i.e. only the presented topologies and their parametric variations are explored with this search algorithm setting.

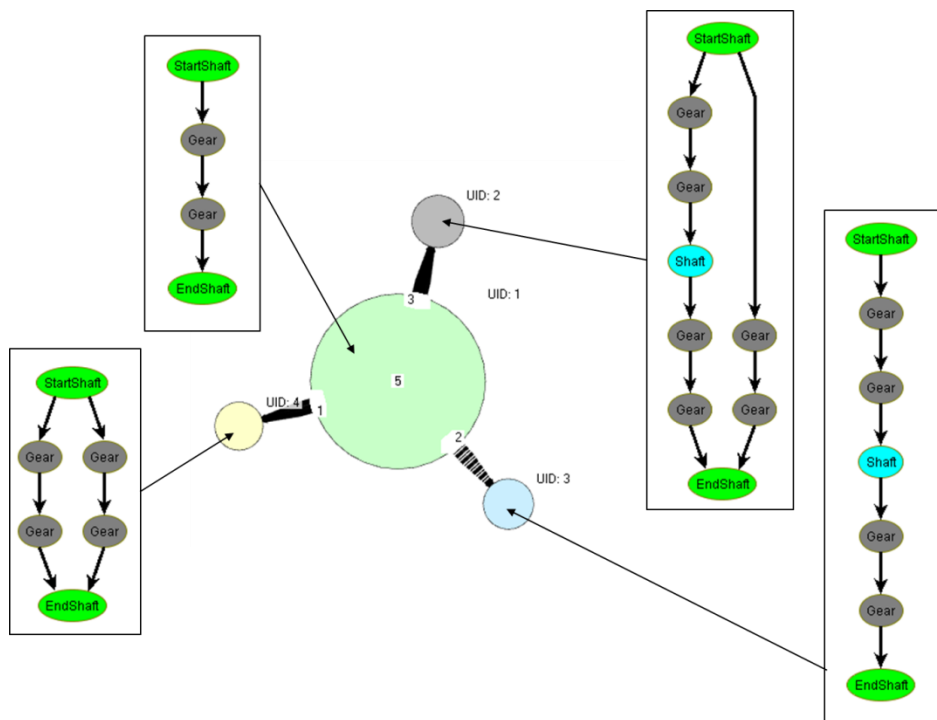


Figure 6-17 UTS and pictorial representations for the topologies generated with the Burst algorithm (maximum Burst length = 1).

The UTS visualizations for two experiments with 1,000 iterations each using the SA algorithm are shown in Figure 6-18. For the visualization on the left, no RTB is performed, for the visualization on the right a RTB is performed every 100 iterations. The progression of the SA algorithm can be observed when the UTS visualization is explored interactively. In general, the visualization of the UTS for the SA algorithm can be considered as exploring the designs along a chain. From this chain, single nodes or smaller groups of designs are branching that are, e.g., explored but rejected and the process is then continued from the previously explored design (compare also Figure 6-11). In the gearbox case study, the same topologies are explored repeatedly during the synthesis process also when the SA algorithm without RTB is used. In the UTS, these sequences of rule applications that explore new topologies and then, after several rule applications, generate a topology found earlier are visualized as loops. Considering only topologies, this effect of generating the same topologies repeatedly is similar to the effect of implementing a RTB option which is why the general structure of the UTS for the two experiments in Figure 6-18 looks similar. The effect of the RTB option can, however, provide benefits for the search process through restarting the synthesis process from a topology with optimized parameters.

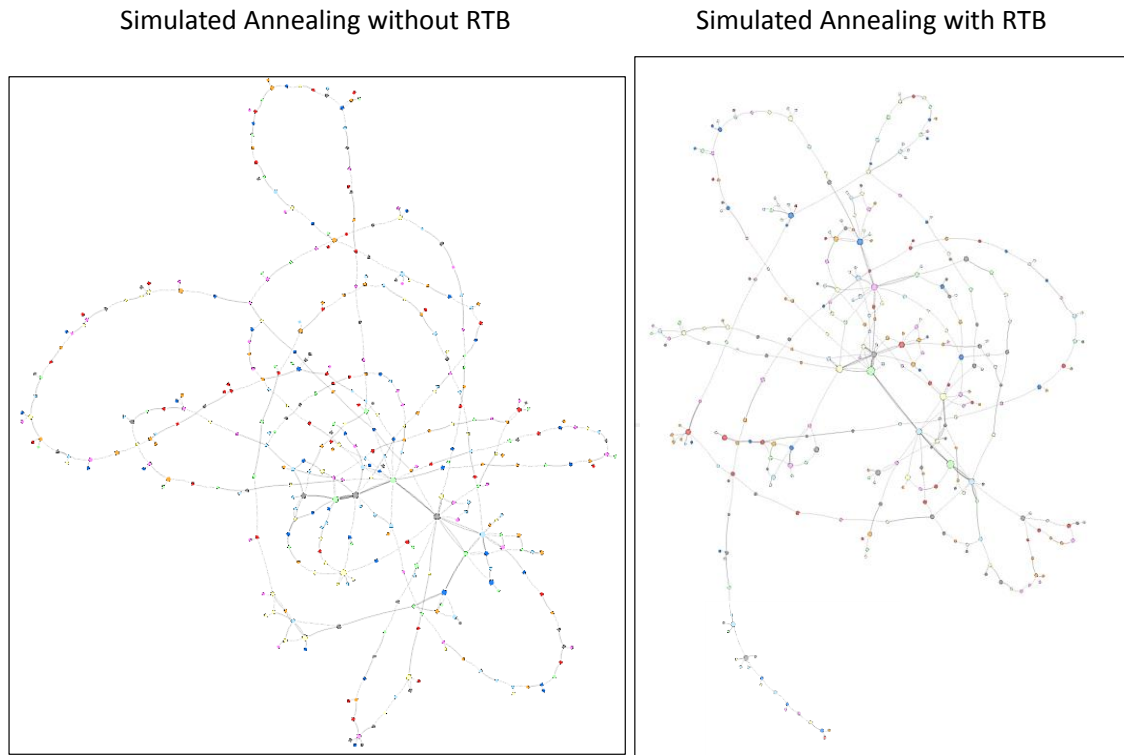


Figure 6-18 Unique Topology Spaces (UTSs) for two experiments with 1,000 iterations each using the SA algorithm without RTB (left) and with RTB every 100 iterations (right).

Figure 6-19 shows a GUI that can be used to visualize the PS progression. The data is generated during 1,000 iterations using the SA algorithm with RTB every 100 iterations. The six subfigures show the explored regions of the performance space after 10, 50, 100, 500, 700 and 1,000 iterations, respectively. Note that the scale on both axes changes during this series of subfigures. The PS is projected to two dimensions (“collisions–ratio error”-plane) in Figure 6-19 to be easier readable. The tool, however, allows arbitrary rotations and projections to enable the user to better understand the explored space. The progression of the SA algorithm can be observed when replaying the synthesis process using the slider. In the beginning of the process the SA algorithm accepts more designs with inferior performance, i.e. higher objective values. This can be observed on the first four subfigures in Figure 6-19, where designs with high objective function values are explored and the search is continued from these designs. Towards the end of the process, inferior designs are accepted with decreasing probability. In the last two subfigures of Figure 6-19 this can be observed. The space is explored around designs that are found previously and are stored in the Pareto archive. This can also be seen in the interactive tool when looking at the restarts every 100 iterations. The search is then restarted from a Pareto optimal design.

Rule analysis representations are given in Figure 6-20 (top). The human designer can interpret these representations to understand each rule’s influence on the objectives and on how the changes of objective values are linked. Rule 8 (*shorten a shaft*) always reduces the mass of a gearbox and often resolves collisions. When no collisions between the shaft and other gearbox elements exist it influences the mass only. These links between changes in objectives can be interpreted when analyzing the rule analysis plots.

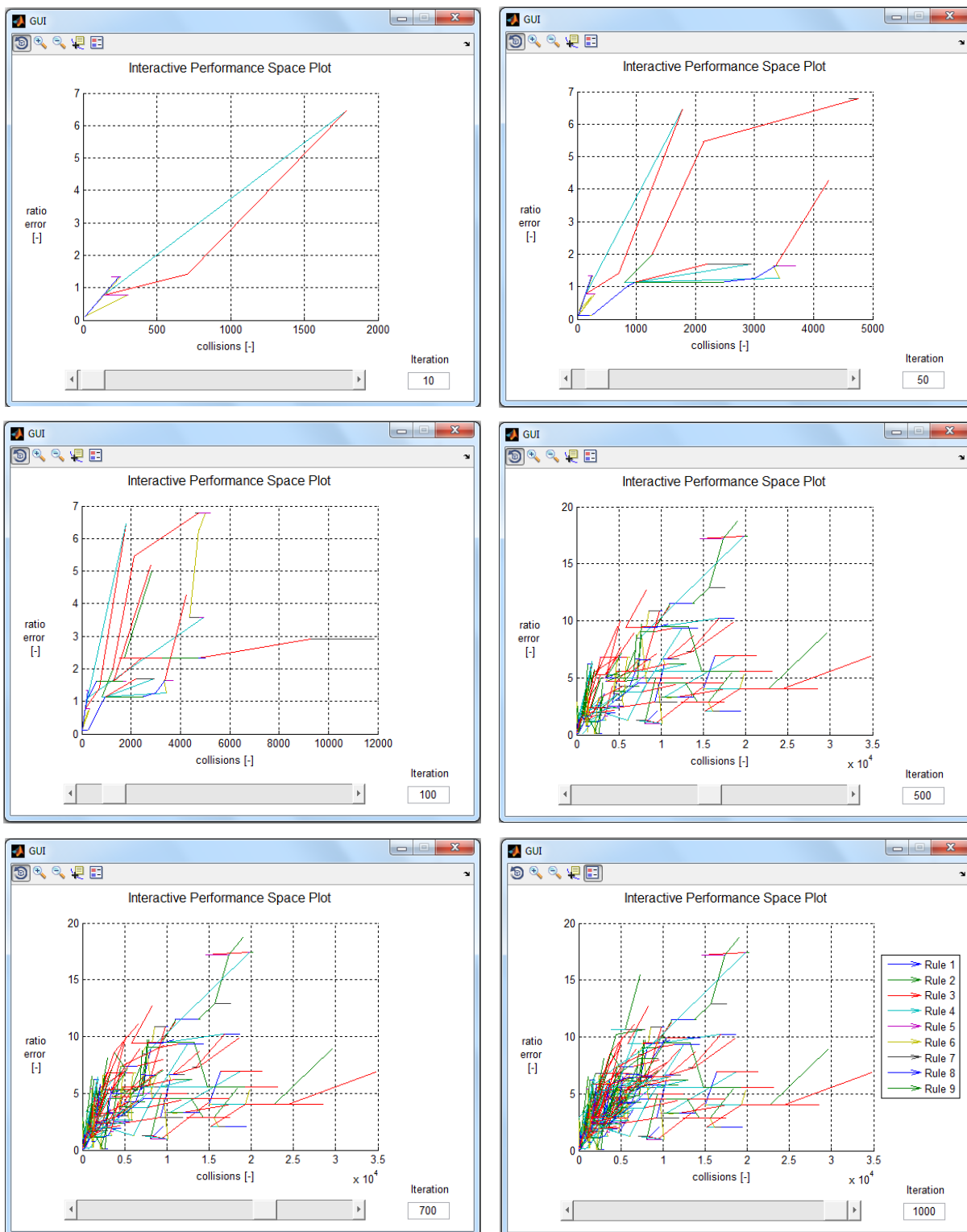


Figure 6-19 Exploration of the performance space (PS) during SA with RTB.

6.5 Discussion

In the following section, the Relation Visualization Method and the presented results are discussed. First, the different representations (UTS, PS and rule analysis plots) are reviewed based on the results of the case studies. The method is then compared to GRAM (Chapter 4)

and the Network-based Rule Analysis Method (Chapter 5) and its generality and limitations are discussed.

Two algorithms with different settings are applied to the same design synthesis task for both case studies. Using the presented method, UTSs are generated to visualize interactively how the individual algorithms address the problems. Visualizing the UTS for a Burst length of one (Figure 6-10 (top) for the bicycle frame and Figure 6-16 (top) for the gearbox), the human designer understands the strong focus of this algorithm parameter setting on exploiting few topologies. In case of the gearbox case study, the exploration of the design space is even restricted to four different topologies. Runs with longer Burst lengths lead to more discovered topologies. The visualization of the UTS allows the human designer to achieve better synthesis results because the increased understanding of the algorithm progression is used to adapt algorithm parameters which leads to better results. Comparing, e.g., the representations for the different maximum Burst lengths (Figure 6-10 and Figure 6-16), the increasing maximum Burst length is reflected in the amount of exploration around the archive designs.

Besides the influence of algorithm parameter settings, also the general concepts of algorithms can be seen. For the bicycle frame case study, conceptual differences between the Burst and the SA algorithm are easily recognized by the structure of the UTS representations. The frequent restarts of the Burst algorithm are reflected in the star-like structure, whereas the SA algorithm without RTB is reflected by a chain-like structure. The SA with RTB combines these two aspects. The exploration of the search space is not only dependent on the general concepts of the search algorithm used. Also the specific design problem and the grammar rule set influence how the space is explored. This can be seen, e.g. in Figure 6-18, where no difference between using the SA algorithm with or without RTB is visible due to the way the gearbox rules are developed. When rule 2 (*delete a shaft*) or rule 4 (*delete a gear pair*) are applied and numerous dangling nodes are deleted, a sequences of rules that add gears and shafts (rule 1, 3 and 5) can recover the pervious gearbox design (compare also to detected loops in transition graphs in Chapter 5). Exactly this ability of the Relation Visualization Method to visualize the design space exploration for the combination of a specific rule set and a specific algorithm accounts for its superiority compared to learning about and selecting search algorithms only based on a generic description of their principles.

In addition to the UTS, the PS visualizes how the algorithm progresses with respect to the objective function values. It also visualizes the parametric aspects of the synthesis process, i.e. how a parametric rule changes a design with respect to the objectives. This supports the human designer to understand in which phase of the synthesis process the algorithms explore designs far away from the Pareto front and in which phases the search is narrowed to exploit Pareto-optimal designs. Together with the UTS plot this gives insights on how the search algorithm explores and exploits the design space. This enables the designer to reason about the CDS method at hand and, e.g., adapt parameters of the algorithm to focus more on exploring or exploiting the space.

To summarize, the presented method visualizes the algorithm progression given the actual engineering design problem and grammar rules. Visualizing the search algorithm progression on the concrete problem gives the human designer more insights into the search algorithm used in the CDS method, than can be understood by just learning about the search algorithm's behavior in a text book. This is because it is visualized in the context of the actual problem that includes the grammar rules and the problem-specific evaluation.

While the UTS and the PS focus on the algorithm, the rule analysis plot predominantly supports reasoning about the grammar rules. It gives the human designer a clear picture how each individual rule influences the objective function values. This information is not only helpful for debugging grammar rules, but also to identify favorable as well as non-influential rules and to reason about their use.

Differences between the Relation Visualization Method, the Grammar Rule Analysis Method (Chapter 4) and the Network-based Rule Analysis Method (Chapter 5) are discussed in the following.

The presented method is an extension to and complements GRAM (Chapter 4). GRAM supports the development of grammar rules before they are used in a CDS method. Rule analysis is performed without considering the search algorithm that is used in the CDS process. The Relation Visualization Method, by contrast, focuses on the interconnection between grammar rules and search algorithm during CDS. Besides visualizing the algorithmic aspects of the CDS search process, the presented method is also an extension to GRAM's rule analysis. While GRAM analyzes the influence of a rule on each objective individually, the method presented in this chapter shows interdependencies between these changes in objectives by using a multidimensional vector, i.e. showing how changes in objectives are linked for each rule. Figure 6-20 visualizes this by comparing the rule analysis plot and the boxplots as used in GRAM. The colored stars highlight example rule applications in the rule analysis plots at the top of Figure 6-20. At the bottom of Figure 6-20 the same rule applications are highlighted in the box plots. The link between the changes in the two objectives is missing in GRAM's visualization. For some rule applications, e.g. the one highlighted in purple on the left of Figure 6-20, the change of one objective is not shown in GRAM's visualization because it constitutes an outlier when considering the objectives separately. The rule analysis plots, therefore, give additional information to the human designer, while GRAM's boxplots, especially with the color-coding to indicate changes in desired and undesired directions, allow a quick overview.

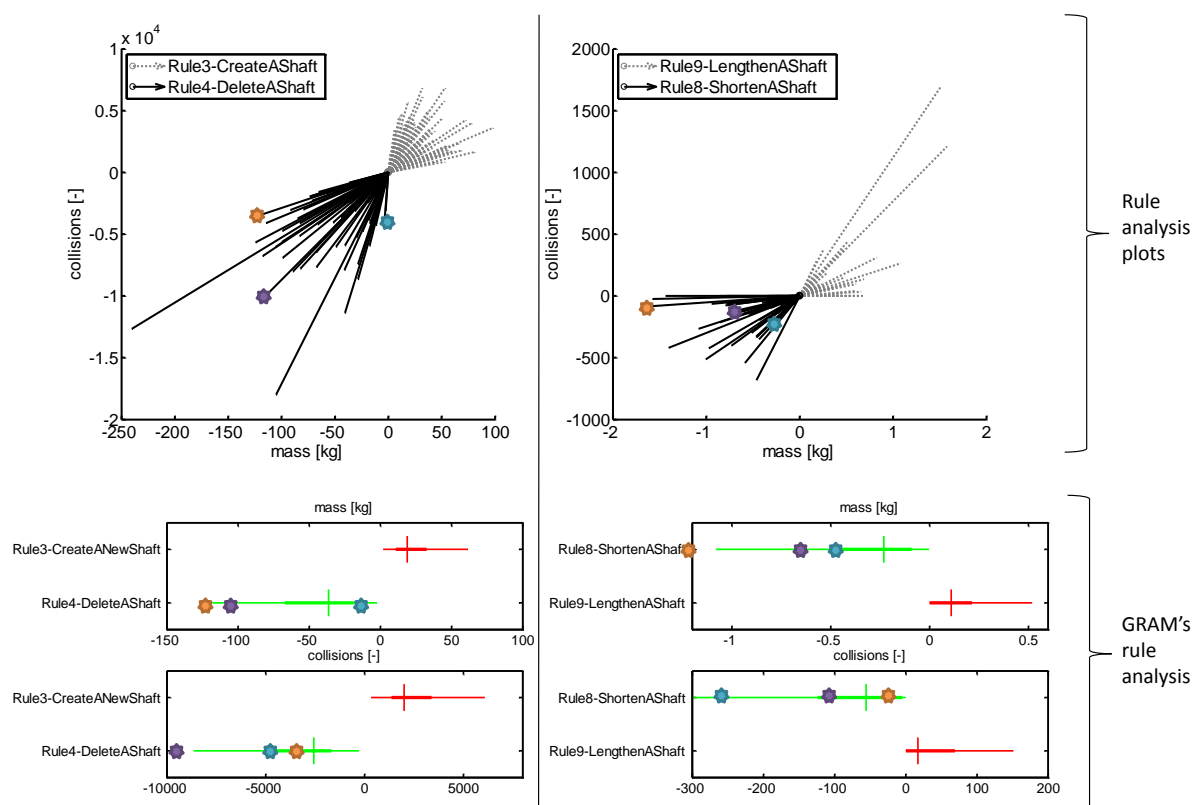


Figure 6-20 Example rule analysis plots for topologic (left) and parametric (right) rules are shown at the top. GRAM's boxplots for rule analysis are shown at the bottom for comparison.

Further, with the presented method not only different topologies are represented (as with GRAM), but also the transformations between them are visualized explicitly through the UTS representation. The Relation Visualization Method supports the understanding of the search algorithm and facilitates an analysis of the rule set in the context of the algorithm and the design task at hand, which is superior for analyzing and understanding the CDS process in depth.

In Chapter 5, the Network-based Rule Analysis Method is presented that uses network analysis techniques to find superior rule application sequences using the same representation of generated designs and their relations as in this chapter. The differences are that in Chapter 5 the representation of unique designs and transformations is generated by exploring a small portion of the search space exhaustively, and that this representation is used to analyze rules and their application to learn meaningful sequences. Using the Relation Visualization Method, the designs are generated during a CDS process and are used to better understand the search algorithm and rules in the conducted CDS process.

To summarize, both, GRAM and the Network-based Rule Analysis Method are developed to support the rule development process and find reasonable rule sequences before they are applied in a CDS search process. The method presented in this chapter, by contrast, supports the understanding of the search algorithm in relation to both the rules and defined objectives, thus extending the set of methods for CDS method development. Future research could include validating the presented method with different user groups, i.e. CDS novices and experts, and considering their feedback for further refining the method.

To apply the method, only minor implementation effort is required. One routine has to be included in the CDS process to store data during the execution of the CDS method. A second routine has to be implemented for post-processing the collected data.

Limitations to the method are given when more than three objectives have to be considered. Then, the visualization of the PS has to be presented using different visualization techniques. The presented method is implemented as a post-process visualization of the CDS process. With an increased computational effort, it could, however, also be integrated in the CDS process as a live visualization method. Graph isomorphism checks are then done after every iteration and UTS, PS and rule analysis plots are presented. Doing so, the method could be used for interactive search algorithms and would allow the user to take more informed decisions during the CDS process.

In the presented case studies, the method is applied to graph grammars. In general, any type of grammar can be used for which unique designs can be identified and for which intermediate designs can be evaluated.

6.6 Summary

In this chapter, a new visualization approach is presented to systematically represent the search process when using CDS methods with grammars and search algorithms. It answers research question 6.1:

Research question 6.1: *How can search processes in CDS be visualized to better understand how grammar rules and a search algorithm explore the design space?*

The Relation Visualization Method analyzes the designs generated during the synthesis process and visualizes how the design space is explored with respect to design characteristics and objectives. The selected algorithm as well as the grammar rules can be analyzed with this approach to support the human designer in understanding and applying a CDS method. The results of the case studies demonstrate how the method provides information on the different components in CDS. A major contribution of the presented method is that it systematically visualizes the relations between grammar rules, performance objectives and search algorithm progression for grammar-based CDS methods. The progression of the CDS process can be animated and relations between the generated designs are visualized. These relations can, e.g., be the rules that transform one design into another one, or the distance between two designs in the performance space. Another contribution is that by visualizing the CDS process, the presented method makes properties of the search algorithm visible, e.g. its tendency to explore and exploit the design space. This knowledge can be important for selecting and tuning search algorithms. To the knowledge of the author, the method is unique in its ability to visualize search algorithm progression and design space exploration in CDS. It can further be used for debugging search algorithms as well as grammar rules and for teaching algorithms in a CDS context. The presented research can be useful for both novices to CDS to help them gain a deeper understanding of the

interplay between grammar rules and guidance of the synthesis process, as well as for experts aiming to further improve their CDS application by improving parameter settings of the selected search algorithms, or by further refining their design grammar. Additionally, the presented method constitutes a novel approach to interactively visualize design space exploration considering not only design objectives, but also the characteristics and interdependencies of different designs.

The research contributions of the Relation Visualization Method can be summarized as follows:

- **Contribution 1:** The Relation Visualization Method is a method to systematically analyze grammar rules.

Similar to GRAM and the Network-based Rule Analysis Method, the Relation Visualization Method contributes to analyzing grammar rules. In contrast to the aforementioned methods, the search algorithm that is used to generate the analyzed designs is also subject to analysis in the Relation Visualization Method.

- **Contribution 2:** The Relation Visualization Method visualizes how search algorithms explore the search space and permits exploratory and confirmatory analysis.

The interactive performance and topology space visualizations allow the human designer to retrace the CDS process in detail and reason about the changes invoked by the rules, as well as the decisions taken by the search algorithm. These imply decisions about whether to continue applying rules to the current design or revisiting an already explored design. Further, the balancing between topologic and parametric changes of the generated designs can be presented in these visualizations. These decisions, taken by the search algorithm, can be retraced by the human designer to see how the search algorithm explores and exploits the search space. The method, therefore, enables exploratory and confirmatory analysis of the relations between search algorithm, grammar rules, generated designs and performance objectives.

Figure 6-21 shows the positioning of the Relation Visualization Method with respect to the goals of this thesis. The Relation Visualization Method supports the rule development process (sub-goal G1) and the selection (sub-goal G2) of a search algorithm that is suitable for the design task at hand. The rule development process is supported through additional information on each rule's performance (sub-goal G1.1), when compared to GRAM. Comparing different implementations of rules, the human designer can also decide among different rules and, e.g., reuse already existing implementations where appropriate (sub-goal G1.2). Given a set of rules and a search algorithm, the search process can further be refined (sub-goal G3) by tuning the search algorithm (sub-goal G3.1) through appropriate parameter settings.

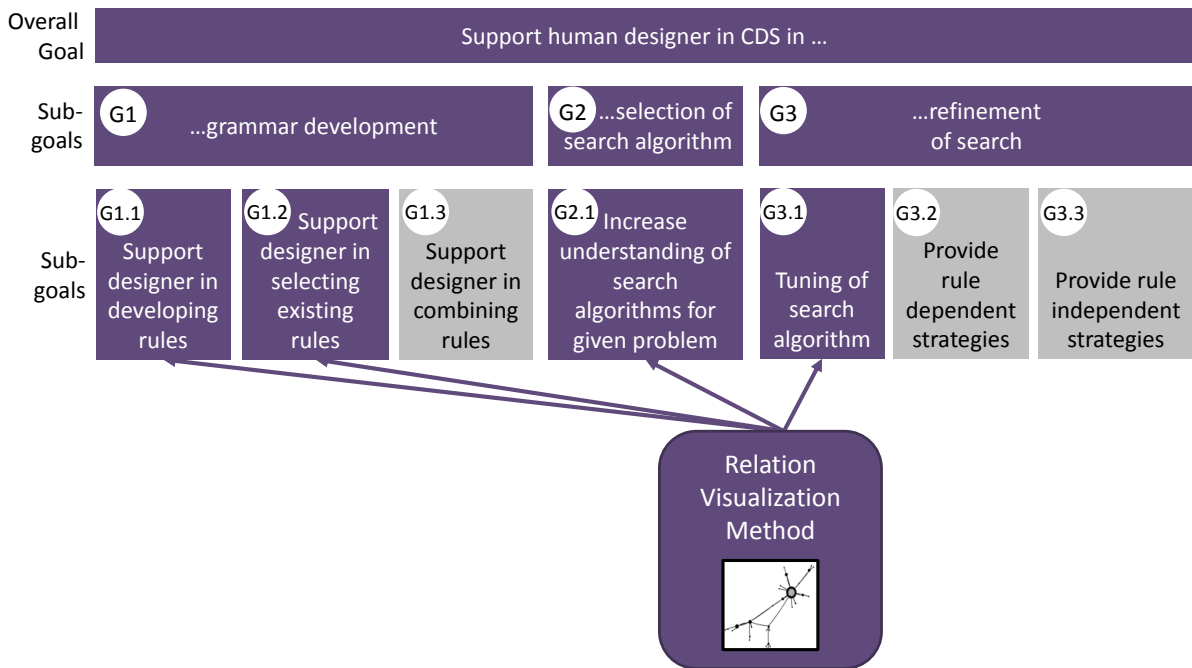


Figure 6-21 Positioning of the Relation Visualization Method with respect to the goals of this thesis.

The following contributions of the research presented in this chapter support the human designer in CDS:

- The Relation Visualization Method supports the analysis of grammar rules.** It uses static rule analysis plots to represent how rules change objectives, but also how these changes are linked for multiple objectives. Additionally, the interactive performance space visualization and the unique topology space visualize how the rules explore and exploit designs in the search space when combined with a search algorithm. This allows designers to analyze each grammar rule and the changes it applies to designs in detail.
- The Relation Visualization Method enables a more informed decision on a search algorithm for a given problem.** Understanding how search algorithms explore and exploit the search space, the human designer can reason about different algorithms. Besides visualizing the general principles of search algorithms, the Relation Visualization Method presents details on the behavior of the search algorithm for the design task at hand. Comparing different search algorithms for the given problem and rule set facilitates an informed decision on the search algorithm to use.
- The Relation Visualization Method supports to analyze the effect of search algorithm parameters.** The possibility to link the search algorithm to the design task at hand allows the human designer to take informed decisions on setting the parameters of the search algorithm. When a decision is taken on which search algorithm to use for the search, the human designer can tune its algorithm parameters for the given problem by comparing the performance and topology space visualizations for different parameter settings.

7 Search Strategy Comparison Method

The success of CDS methods depends on several decisions made by the designer, e.g. on defining an appropriate representation of the design problem, meaningful design evaluations, a suitable algorithm to guide the process, as well as reasonable strategies to select rules to generate design alternatives. This chapter focuses on rule selection strategies, which is part of the guidance step. For most engineering tasks two different kinds of rules are required to change the topology and parameters of a design. Most real world engineering tasks require both topologic and parametric rules. Topology describes the structure, or configuration, of a design whereas parameters describe its geometry and spatial arrangement. Common approaches to this challenge include a) to split the synthesis process in phases, e.g. to decouple topologic and parametric changes [38, 121], and b) to allow the change of topology and parameters at the same time but add more intelligence to the search on how rules are selected, e.g. by using trajectories that pre-define rule application probabilities and dynamic rule qualities to steer the amount of topologic and parametric variation throughout the synthesis process [111]. Other approaches aim to learn meaningful sequences of rules [84, 138], however, these methods focus mainly on topologic rules. A good strategy to apply topologic and parametric rules is an important prerequisite to successfully use grammars for CDS. Researchers in CDS usually have preferred approaches to address design synthesis tasks that require topologic and parametric rules and variables. The challenge of handling both at the same time is a known issue. However, no research is known to the author that compares different strategies to better understand their use in the automated design synthesis process on one problem that would permit their fair comparison. A common strategy is to use topologic rules until a valid design is found and then switch to applying parametric rules to find optimally directed designs. Another strategy is to apply both topologic and parametric rules throughout the synthesis process. Researchers also often make assumptions of the superiority of their chosen way with respect to the problem. The research presented in this chapter compares different strategies to combine topologic and parametric rules during CDS to target the following research question:

Research question 7.1: *How can different strategies for selecting topologic and parametric rules in CDS methods using grammars be compared to each other?*

To investigate this research question, four different strategies are presented and applied to two case studies. Two versions of an algorithm with different levels of complexity are used to analyze the influence of the algorithm on the synthesis results for all strategies. To evaluate the generated designs, different metrics for the quantity and quality of the generated designs are defined. The strategies are then compared according to these metrics.

The remainder of this chapter is structured as follows. Section 7.1 describes the method used to compare different strategies, i.e. a) the algorithms used in the synthesis process (Section 7.1.1), b) the strategies for the selection of the rule type (Section 7.1.2), and c) the

metrics with which these strategies are compared (Section 7.1.3). The synthesis of gearboxes is used as a first case study and described in Section 7.2. The synthesis of bicycle frames is used as a second case study and described in Section 7.3. Results for both case studies are discussed in Section 7.4. In Section 7.5, the method is summarized and the expected contributions are revisited to link the analysis of search strategies to the overall goal of this thesis.

7.1 Method

In this chapter, four different strategies for the selection of topologic and parametric rules during the search process are compared. For this purpose, two different case studies are chosen. The first is a gearbox synthesis task using a graph grammar with five topologic and five parametric rules. The synthesis task is to generate a gearbox that has a desired number of speeds and correct gear ratios. Further, the gearbox must fit into a bounding box, contain no interferences between parts, called collisions, and the mass is minimized. More details are given in Section 7.2. The second case study is a bicycle frame synthesis task using a graph grammar with four topologic and five parametric rules. The task is to design a bicycle frame that withholds mechanical loads, is lightweight and has a minimal number of welding positions between frame members. The two design tasks are chosen since there is a strong coupling between generating a valid topology, i.e. one with the correct number of speeds for the gearbox or a valid frame connecting all bike components in the bicycle frame synthesis, and satisfying and minimizing the geometric constraints and objectives. The synthesis tasks are formulated as multi-objective search problems and the Burst algorithm is used to find design solutions. The strategies are compared based on the performance of the search process and the generated designs. Figure 7-1 gives an overview of the elements used to compare different strategies in this chapter. Two different versions of the Burst Algorithm (see Section 7.1.1) are used for the search and each of them is combined with each of the four strategies (see Section 7.1.2) to generate designs resulting in eight different combinations of algorithm and strategy. The generated designs for all eight combinations are then analyzed using five metrics (see Section 7.1.3) for comparing the generated designs and the search process. Note that even though the strategies are implemented as elements of the search algorithm, the aim of this research is not to improve the algorithm itself, but to compare different strategies for selecting topologic and parametric rules during the search process.

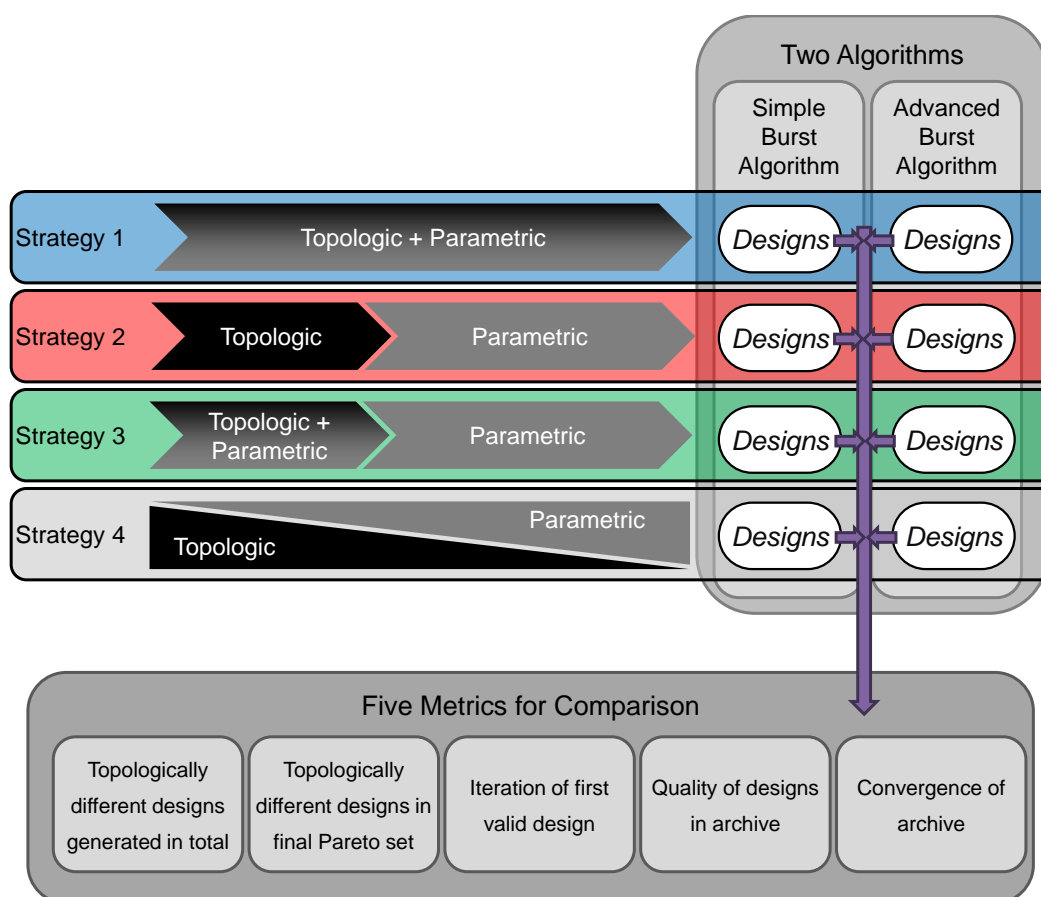


Figure 7-1 Overview of the method used to compare different search strategies.

7.1.1 Burst Algorithm

For the generation of design solutions, two versions of the Burst algorithm are used. The Burst algorithm is generic and not tuned to any engineering problem. This is desirable for comparing different strategies as neither the algorithm, nor the strategies should incorporate any information on the specific task in order to allow for a fair comparison. The Burst algorithm is introduced in Section 6.3.2.1. The implementation that is used for the research in this chapter, is shown in Figure 7-2. Differences to the Burst algorithm described in Section 6.3.2.1 are highlighted.

Hard constraints can be defined for designs to qualify for the archive. As long as there is no design in the archive, the algorithm conducts a greedy search, i.e. in each iteration the generated design is compared to the previous one. When the design improved, the current design is kept for the next iteration, when it did not improve, the previous one is used again. To avoid that the search gets trapped in local minima of infeasible designs, a restart option is implemented that starts the synthesis from the initial design in case there is no feasible design found within a defined number of iterations. When there is at least one design in the archive, the algorithm switches to the Burst mode. The selection of the rule for the next rule application is done in two steps. The set of allowed rules (all rules, only topologic rules, or only parametric rules) is defined by the strategy. Then, the exact rule within the set of allowed rules is selected according to the algorithm.

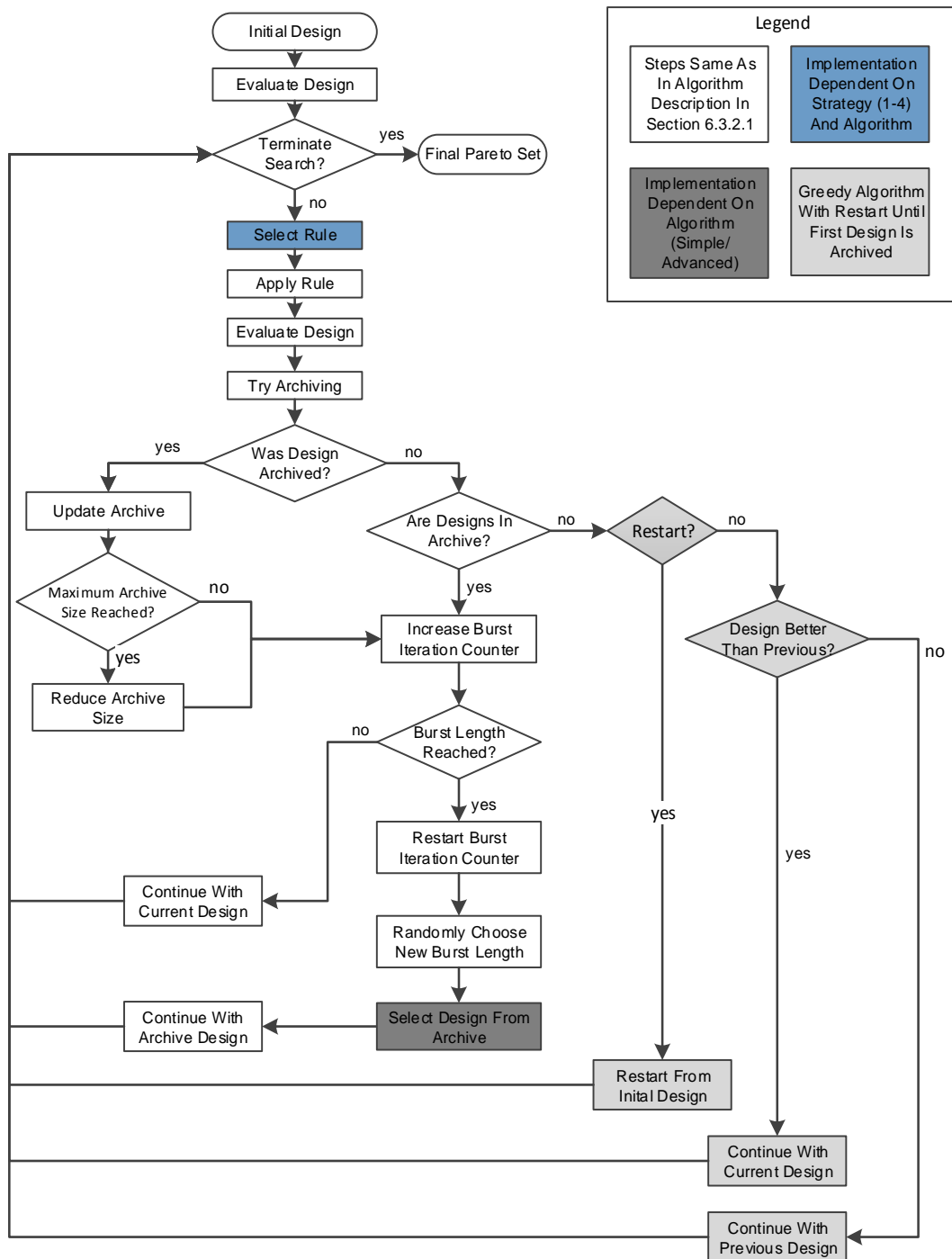


Figure 7-2 Overview of the Burst algorithm used in this chapter.

To show influences of the algorithm on the design synthesis process, two different versions of the Burst algorithm are used. A summary of their differences is given in Figure 7-3. First, the simple Burst algorithm uses random selection of designs from the archive and random selection of rules from the defined rule set. Second, an advanced version of the weighted Burst algorithm by Vale [135], is used where the selection of the exact rule is based on weights. Different from the work by Vale, here the weights are not predefined by the user but rather are adaptive and regularly updated during the search process based on rule performance, similar to the work by Shea [111]. This algorithm is called the advanced Burst algorithm in the remainder of this chapter. In the advanced Burst algorithm, the probability

to choose a certain rule is based on the rule's previous performance. This means that for all rule applications the quality of the rule is calculated according to formula (6).

$$Quality(Rule_i) = \frac{\sum |\Delta f_{accepted}(Objective1, Objective2, Objective3)|}{number\ of\ rule\ applications} \quad (6)$$

The rule quality is defined here as the sum of absolute changes in the objective function achieved by accepted (i.e. the objective function improved) rule applications divided by the number of rule applications. Tracking the rule quality and using it for rule probability calculations is a common approach to feed previous performance of rules back to the search algorithm. The assumption is that rules that have performed well in previous applications are preferred. The probability for a rule is calculated based on its quality and a minimum probability ($Prob_{min}$) that is divided among all active rules n in the rule set to enable the occasional application of all rules for statistical purposes.

$$Probability(Rule_i) = \frac{Quality(Rule_i)}{\sum_{j=1}^n Quality(Rule_j)} * (1 - Prob_{min}) + \frac{Prob_{min}}{n} \quad (7)$$

Based on these rule probabilities, the rules are selected randomly using the roulette wheel selection technique. Rule probabilities are updated regularly to update changes in the performance of single rules.

In the advanced Burst algorithm, also the selection of designs from the archive can be more advanced than in the simple Burst algorithm. First, a pool of potential candidates for the next Burst is selected from the archive using hierarchical sorting of the designs by the soft constraints, i.e. constraints added as additional objectives using penalty functions. Using two soft constraints, the worst designs from the archive with respect to the first constraint are rejected. From the remaining designs, the worst designs with respect to the second constraint are rejected. The designs in the archive with the fewest constraint violations are thus candidates for the next iteration. The design for the next Burst is then selected randomly from this pool of candidate designs.

	Select Design From Archive	Select Rule
Simple Burst Algorithm	randomly	randomly (from subset defined by strategy)
Advanced Burst Algorithm	various options, e.g. <ul style="list-style-type: none"> hierarchical sorting randomly 	selection (from subset defined by strategy) based on rule quality calculated from previous rule performance

Figure 7-3 Comparison of simple and advanced Burst algorithm.

7.1.2 Strategies for Rule Type Selection

Four different strategies are compared for selecting the rule type for the next modification. Depending on the strategy, different subsets of the complete rule set are defined. These subsets are the basis for the selection of the exact rule to apply. Figure 7-4 gives an overview of Strategies 1 – 4.

In **Strategy 1**, the rule subset is equivalent to the complete rule set, i.e. the rule type is selected randomly. This means that topologic and parametric rules can be applied at any stage of the synthesis process, i.e. building designs using topologic rules and perturbing them using parametric rules is carried out in one phase. It aims not to accept the first topologically valid design, but to improve topology and parameters at the same time continuously throughout the process to potentially benefit from their interdependencies.

Strategy 2 has two phases in the synthesis process. Until a design is found that is stored in the archive, the rule subset is the complete rule set, i.e. topologic and parametric rules are applied. When at least one design in the archive exists, the strategy switches to only applying parametric rules. This is the strategy used effectively in [4] in combination with simulated annealing search using a single, weighted objective function. The strategy is chosen in previous work due to a common occurrence of losing valid topologies during the search process.

Strategy 3 is similar to strategy 2 but allows only topologic rules until the first design in the archive exists, then only parametric rules. Strategy 2 and Strategy 3 are approaches with two phases, where phase 1 can be seen as the building phase, i.e. building a valid topology, while phase 2 perturbs the created design trying to optimize parameter values. The idea behind these strategies is that once a topologically valid design is found, its topology has to be protected so as not to lose it in the process.

Strategy 4 decides the rule type based on a probability for each rule type. It can be seen as a middle way between Strategy 1 and Strategies 2 and 3. Strategy 1 allows topologic changes at any time, which enables big changes in the design not only of parameters but also of topology even at later stages of the synthesis process. Strategies 2 and 3 do not allow any topologic changes once a valid topology has been found, which can be seen as narrowing the optimization process of the topology. To tradeoff possible disadvantages of these strategies, trajectories are used in Strategy 4 that predefine the probability with which topologic and parametric rules are applied at any state of the synthesis process.

A linear trajectory is used to change the probability for the rule types during the synthesis process. In the implementation used in this chapter, the probability for applying a topologic rule ($probTop$) is calculated based on the current iteration (it) in the synthesis process and the maximum number of iterations (it_max):

$$probTop = 0.8 - \left(\frac{it}{it_max} \right) \cdot 0.6 \quad (8)$$

This means that the probability to apply topologic rules changes during the synthesis process from 80% in the beginning to 20% at the end of the process.

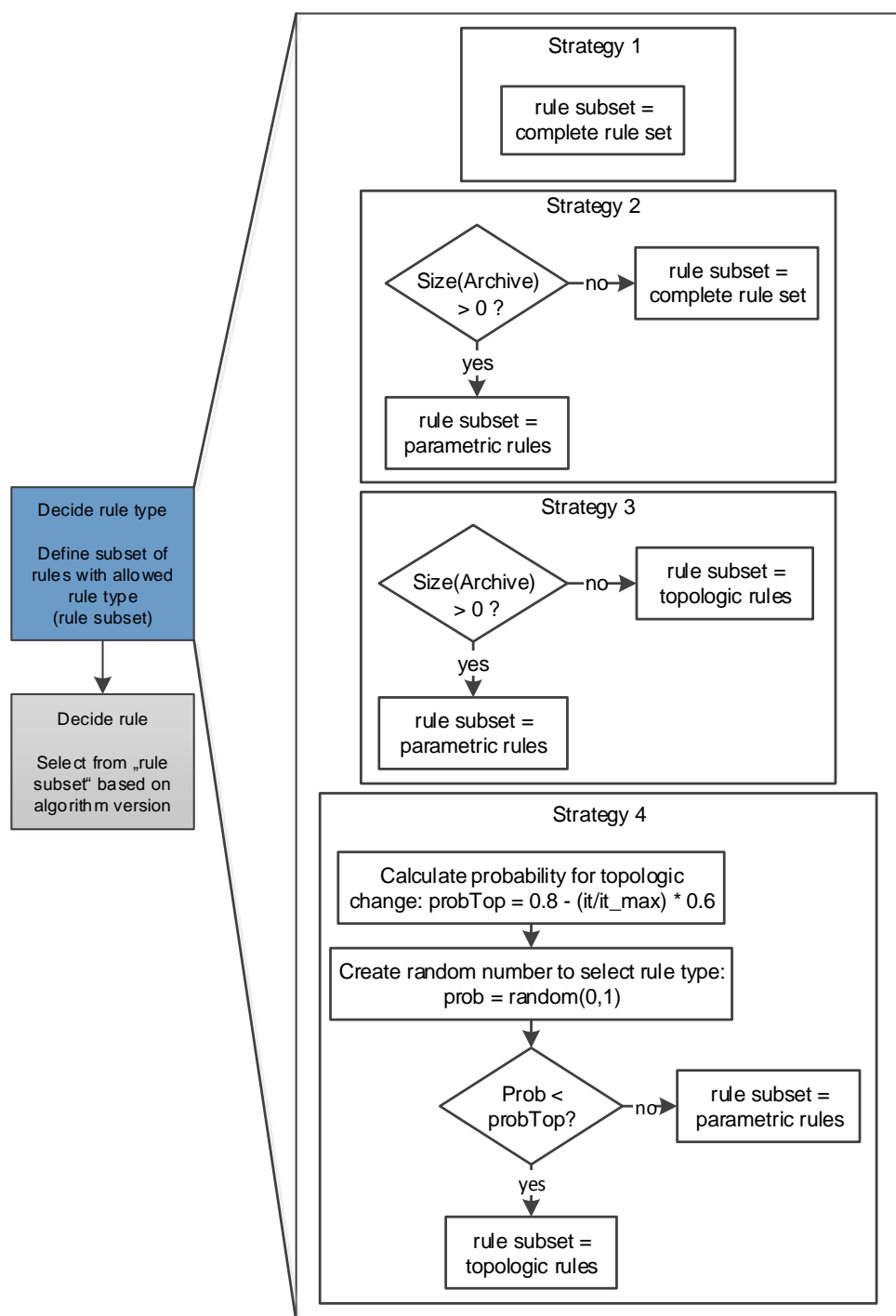


Figure 7-4 Strategies for topologic and parametric rule applications.

7.1.3 Metrics for Comparing the Strategies

For a fair comparison among the four strategies, different metrics are used. As an indicator for exploring the search space for topologically different designs, a) the number of topologically different designs found during the synthesis process and b) the number of topologically different designs in the Pareto archive are evaluated as well as c) the iteration in which the first valid topology is generated. Additionally, d) the quality of the designs in the Pareto archive and e) the convergence ratio of the archive during the synthesis process are considered.

To define the **number of topologically different designs** within a set of designs, the graphs of all designs are compared. With each graph describing the topology of a design, topologically different designs are identified by checking for non-isomorphic graphs, i.e. graphs that do not have the same structure.

The **quality of the designs** is defined by how often designs from one strategy dominate designs from another strategy. The Pareto fronts of all runs for one strategy are compared pairwise to all runs from the other strategies. In each comparison, the designs of two Pareto fronts are combined and filtered using a global Pareto filter to generate a common Pareto front. The strategy that contributes more designs to the common Pareto front is said to be superior and wins the comparison, similar to the fraction of covered sets in [139]. The more comparisons a strategy wins, the better is the quality of its generated designs according to this metric.

To visualize the **convergence of the archive**, the volume (for three objectives as in the case studies of the following sections) of the explored search space is calculated regularly and the ratio of the currently covered volume divided by the maximally covered volume, called convergence ratio in this section, is plotted over the iterations. This metric is similar to the metric to calculate dominated areas in [139] but extended to dominated volumes for the three-objective case in this section. The volume is estimated using a Monte Carlo algorithm. The rationale behind this metric is that as the archive progresses towards better designs, i.e. minimizing the objective values, the designs move towards the origin, i.e. the volume approaches the maximal achievable volume. The maximal covered volume is the volume described by a cuboid spanning from a reference point to the origin. This represents the maximal achievable improvement as a point in the origin represents the minimum value for all three objectives.

7

7.2 Case Study 1: Gearbox Synthesis

The synthesis of gearboxes is used as a first case study in this section as it is an established CDS problem [27, 34, 38, 119-122]. A graph grammar is used to represent gearbox designs and transformations formally. The task is to develop a gearbox that fits into a defined bounding box and has a defined number of forward and reverse speeds. The grammar is formulated and implemented as a graph grammar consisting of a metamodel and a rule set in GrGen, an open source graph rewriting tool [50]. A more detailed description of the gearbox case study is given in Section 4.3.1. The results of the different strategies and algorithms are analyzed in Matlab.

7.2.1 Generation

The rule set consists of five topologic and five parametric rules. A more detailed description of the different rules is given in Section 4.3.1.

7.2.2 Evaluation

The synthesis task is formulated as an optimization problem as follows:

$$\begin{aligned}
 & \text{Minimize mass} \\
 & \text{s.t.} \quad \text{number of speeds} = \text{desired number of speeds} \\
 & \quad \text{collisions} = 0 \\
 & \quad \text{ratioErrors} < 0.5
 \end{aligned} \tag{9}$$

Mass, number of speeds, collisions and ratio error are analyzed as described in Section 4.3.1.3.

7.2.3 Guidance

In previous work by Lin et al. [38], the SA algorithm and a single weighted objective function are used, formulating the constraints for the number of speeds as soft constraints, i.e. adding them as objectives using penalty functions. Using this problem formulation and the simulated annealing algorithm, designs with the desired number of speeds are often lost during the search process. For this research, therefore, the constraint to have the desired number of speeds is implemented as a hard constraint with respect to the Pareto archive. This means that only designs with the desired number of speeds are considered for storage in the archive. This decision seems advantageous as a) every Burst is started from a topologically valid solution and b) the number of objectives in the archive is reduced (three instead of five). The other constraints, i.e. collisions and ratio errors, are formulated as soft constraints, i.e. adding two additional objectives to the synthesis task.

A weighted sum (adapted from [38]) is used for calculating a single objective function value for the first phase in the Burst algorithm, until at least one design is added to the archive and for calculating the rule quality in the advanced Burst algorithm:

$$\begin{aligned}
 f(\text{design}) = & \\
 & | \text{number of forward speeds} - \text{desired number of forward speeds} | * 2 \\
 & + | \text{number of reverse speeds} - \text{desired number of reverse speeds} | * 0.4 \\
 & + \text{collisions} * 0.001 \\
 & + \text{ratio error} / \log(16) * 0.2
 \end{aligned} \tag{10}$$

For the second phase of the Burst algorithm, when there is at least one design in the archive, the three objectives (minimize *mass*, minimize *collisions*, minimize *ratioErrors*) are considered individually. The following parameter settings for the algorithms are used. The maximum Burst length is set to 20, the maximum number of iterations is 10,000 and the archive is sampled every 10 iterations to calculate the archive convergence. Rule probabilities are updated every 20 iterations when the advanced Burst algorithm is used and the pool for the selection of the next candidate in the advanced Burst algorithm is created by taking the best one-third of the designs with respect to the soft constraint for collisions, i.e.

priority is given to minimizing the collisions since they must not exist in the final design whereas ratio errors can be negotiable [38]. The maximum archive size is set to 30.

7.2.4 Results

To compare the presented strategies, 20 experiments with 10,000 iterations are run for each strategy and each algorithm.

Table 7-1 shows the number of topologically different designs generated during the synthesis process and remaining in the final archive. As expected, Strategies 2 and 3 generate only one topology due to the two phase approach, which prohibits further changes of the topology once a design with the desired number of speeds is found. Strategies 1 and 4 allow the application of topologic rules throughout the whole synthesis process enabling the exploration of several topologically different designs within one synthesis process. On average, between four and six topologically different designs are explored with both of these strategies during the synthesis of which on average 1.2 and 1.5, respectively, remain in the final Pareto set, the others are dominated. The number of topologically valid designs explored during the synthesis process are similar for both algorithms. The number of topologically different solutions among all 20 runs for each strategy and algorithm are presented in the last column of Table 7-1. The simple Burst algorithm explores more different topologies than the advanced Burst algorithm. Among the strategies, however, no statement can be made about individual strategies. Comparing the number of designs in the Pareto set for the different strategies, one can see that all strategies have archives with close to 30 designs which is the predefined maximum archive size.

Algorithm	Strategy	Average number of designs in Pareto set	Average number of top. diff. designs in final Pareto set	Average number of top. diff. designs during synthesis	Number of top. diff. designs in 20 runs
Simple	1	28.1	1.5	5.8	23.0
	2	28.0	1.0	1.0	19.0
	3	28.2	1.0	1.0	19.0
	4	27.8	1.2	4.3	14.0
Advanced	1	28.5	1.0	3.9	15.0
	2	27.9	1.0	1.0	15.0
	3	28.4	1.0	1.0	16.0
	4	27.5	1.1	5.2	18.0

Table 7-1 Comparison of topologically different designs in Pareto set for both algorithms and all four strategies.

Figure 7-5 shows the final archive of one run for Strategy 1 using the simple Burst algorithm. Three different topologies (labelled 1-3 and represented by the respective graphs) exist with each topology exploring a specific area of the parametric design space. Topology 1 consists of the most components (22 gears and 7 shafts) while topologies 2 and 3 have fewer components (20 gears and 6 shafts). Even though topology 2 and topology 3 have the same number of components, they cover different regions of the design space. All designs with topology 2 have a lower mass than designs with topologies 1 and 3. Designs with topology 3

have, on average, less collisions and a smaller ratio error than designs with topology 2. The design with topology 1 has a mass comparable to that of designs with topology 3, collisions and ratio errors are between those with topologies 2 and 3. During the synthesis process that resulted in the designs shown in Figure 7-5, ten different topologies are generated, however, only the three shown are non-dominated.

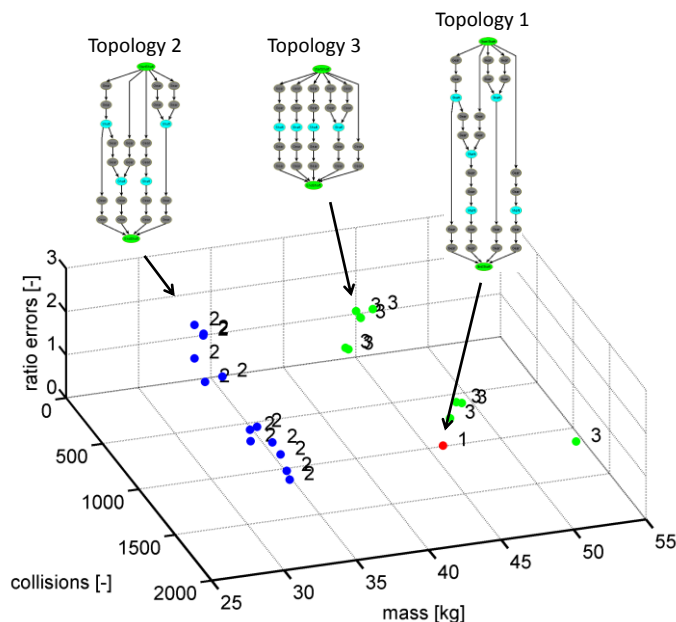


Figure 7-5 Final Pareto Set of one experiment in Strategy 1 using the simple Burst algorithm.

Boxplots for the iteration number at which the first design is stored in the archive, i.e. when the first design with the desired speed is found, for each strategy are shown in Figure 7-6. The boxplots show the results summarized for all 20 experiments. For both algorithms, Strategies 1 and 2 find the first design for the archive later than Strategies 3 and 4, which relates to the probability with which topologic rules are applied. Further, it can be see that the advanced Burst algorithm speeds up the process of finding the first archived design, especially for Strategies 1 and 2.

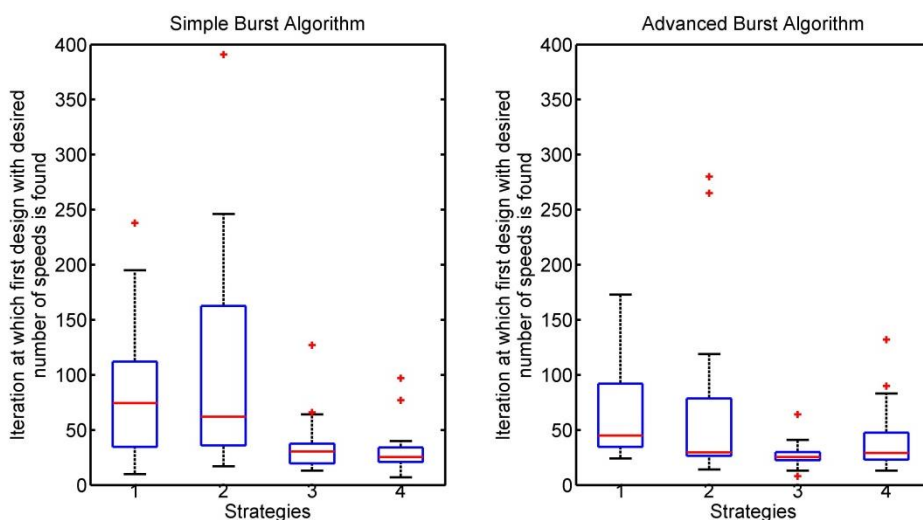


Figure 7-6 Boxplots of iterations when the first design with the desired speed numbers is found for simple (left) and advanced (right) Burst algorithm.

The convergence ratios of the Pareto sets using the volume estimation described in Section 7.1.3 are shown in Figure 7-7 and Figure 7-8 and represent mean values of 20 experiments. For the gearbox case study, the reference point is calculated by taking the first designs in the archives of all strategies and all runs and taking the mean value for each individual objective. This allows the definition of a reference point that is the same for all strategies and runs and thus enables a more fair comparison than when taking individual reference points for each strategy. For the simple Burst algorithm (Figure 7-7), all four strategies show comparable convergence ratios after 8,200 iterations. Strategy 4 starts to explore the space more slowly, however achieves convergence ratios similar to Strategy 1 after 2,600 iterations, similar to Strategy 2 after 6,000 iterations and similar to Strategy 3 after 8,200 iterations. Before iteration 400 and after iteration 8,200 no significant difference between the four strategies could be identified at the $p \leq 0.05$ level. Between strategies 1, 2 and 3 no significant difference exists during the whole synthesis process. For the advanced Burst algorithm (Figure 7-8) there exists a statistically significant ($p \leq 0.05$) difference between Strategy 4 and the other strategies for the synthesis process between iterations 50 and 8,800. The convergence is slower for Strategy 4 during this phase of the synthesis process. Between strategies 1, 2 and 3 no significant difference exists.

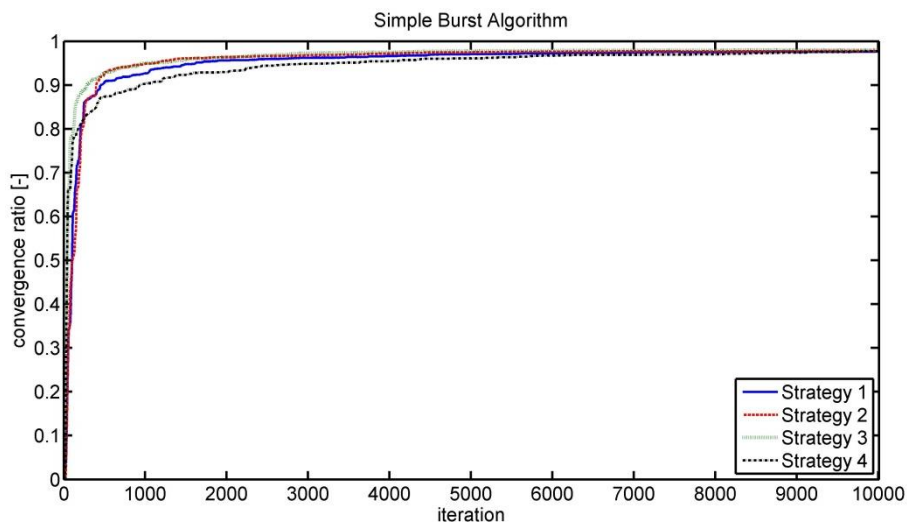


Figure 7-7 Convergence of the Pareto sets using the simple Burst algorithm (10,000 iterations).

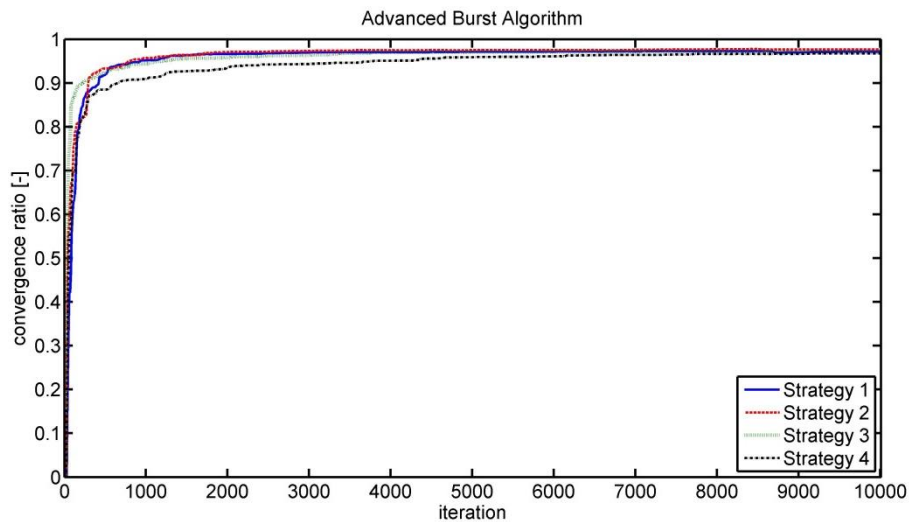


Figure 7-8 Convergence of the Pareto sets using the advanced Burst algorithm (10,000 iterations).

The best designs generated with each strategy are shown in the objective space in Figure 7-9 and Figure 7-10. The designs represent the Pareto-optimal designs for each strategy among the 20 experiments. The Pareto sets for all 20 runs are combined and filtered to identify the non-dominated designs. Note that the axes are scaled differently in these figures as the advanced Burst algorithm generates designs with better results, i.e. lighter gearboxes with less collisions and ratio errors. Gearboxes without collisions are generated for three out of the four strategies, which is a significant improvement compared to work in [38], where collisions have to be removed manually, e.g. by shortening shafts or moving components.

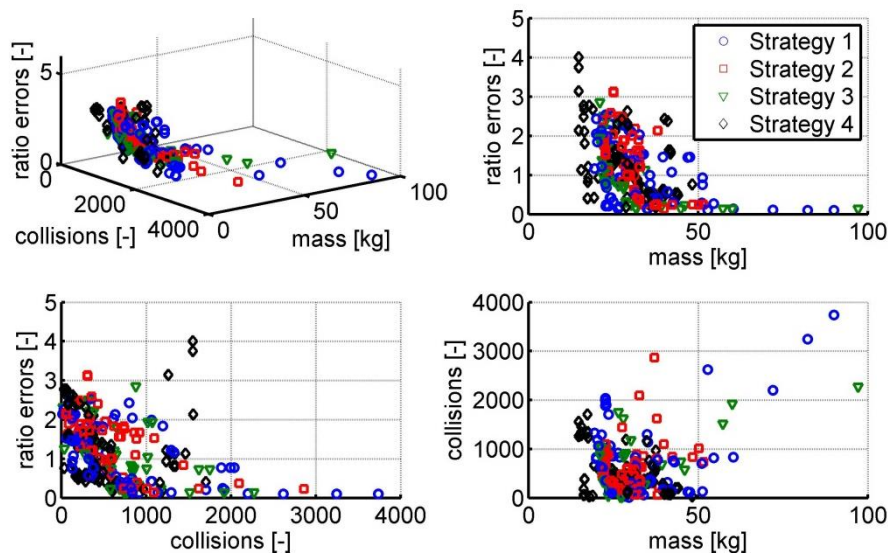


Figure 7-9 Combined Pareto archives for Strategies 1 - 4 using the simple Burst algorithm (10,000 iterations).

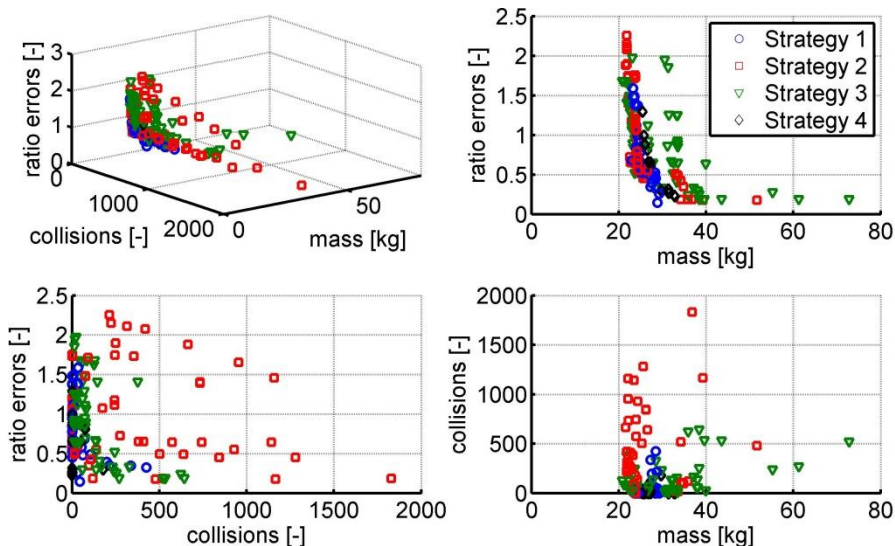


Figure 7-10 Combined Pareto archives for Strategies 1 - 4 using the advanced Burst algorithm (10,000 iterations).

To compare the quality of the designs among the different strategies, pairwise comparisons are conducted as described in Section 7.1.3 and results are shown in Figure 7-11. The figure shows how often each strategy wins against every other strategy in a matrix format. Each cell (except those on the diagonal) shows a pie chart which indicates the percentage that the strategy in the row wins against the strategy in the column. The colors in the pie chart mark the strategies. For the simple Burst algorithm, Strategy 3 wins more often than all other strategies. For the advanced Burst algorithm, the strategies are more similar. Strategy 2 wins the comparisons against all other strategies most often, however the difference is smaller than for the simple Burst algorithm.

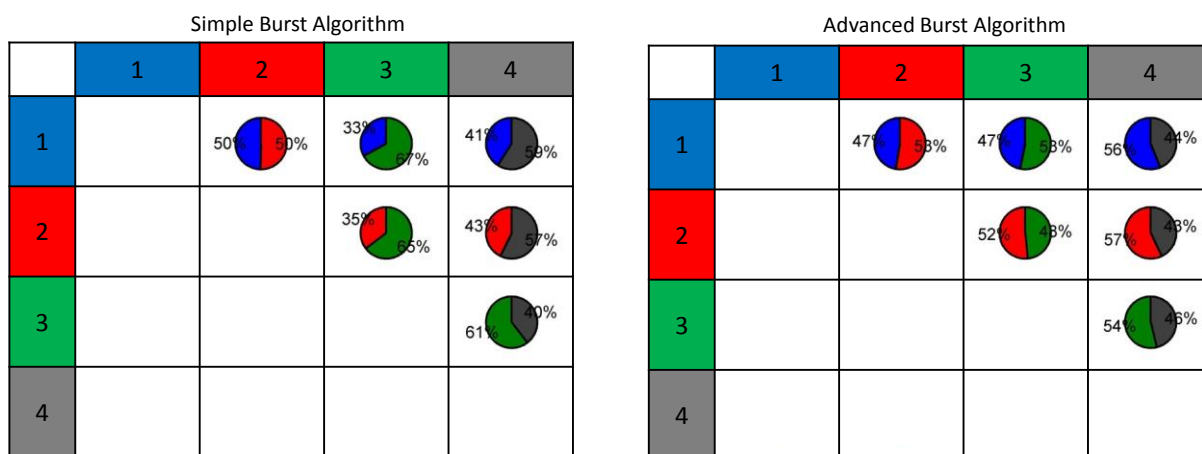


Figure 7-11 Visualizing dominant strategies in pairwise comparisons.

7.3 Case Study 2: Bicycle Frame Synthesis

The synthesis of bicycle frames is considered in the second case study. The task is to synthesize bicycle frames with minimized mass, deflection and number of joints between bars while considering constraints on deflection and stress in the frame members.

7.3.1 Generation

Bicycle frames are represented as graphs consisting of bar nodes, to describe frame members, and joint nodes, to describe connection points between frame members. A graph grammar is used for this problem consisting of four topologic and five parametric rules. A detailed description of the meta model and graph grammar rules is given in Section 6.3.1.

Two initial designs are considered. First, the synthesis process is started from a diamond frame (Diamond Bicycle Frame) as described in Section 6.3.1. This means the grammar rules are used to perturb an initial bicycle frame. Second, the synthesis process is started from scratch (Void Bicycle Frame), i.e. only the positions of the fixed components are defined, but no initial frame is predefined. In this case a valid frame has to be generated first before it can be perturbed. The two different initial designs are considered to mimic different approaches to CDS problems, namely perturbing existing designs versus building up designs from scratch. Both initial designs are visualized in Figure 7-12.

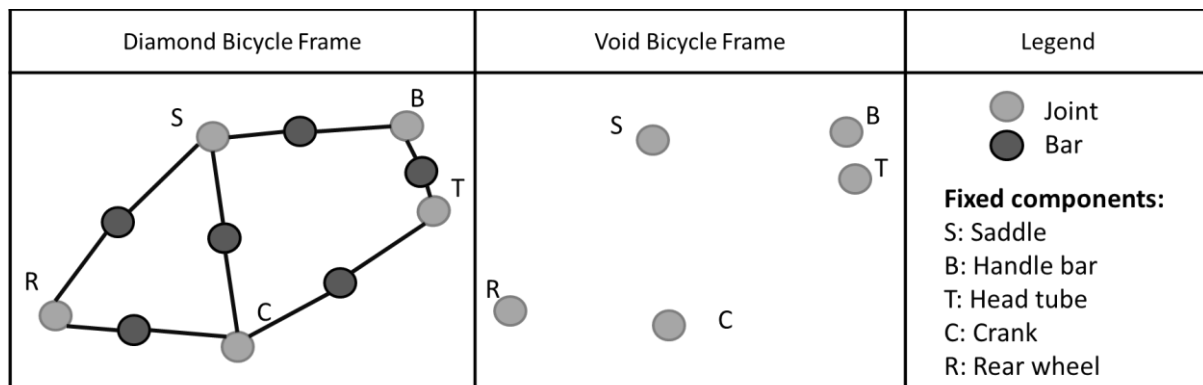


Figure 7-12 Initial designs for bicycle frame synthesis case study.

20 runs are conducted for each of the two initial designs for both Burst algorithm versions (simple and advanced Burst) and for each strategy (Strategy 1 - 4).

7.3.2 Evaluation

The synthesis task is formulated as an optimization problem as follows:

$$\begin{aligned}
 & \text{minimize} && \text{mass, meanDeflection, numberOfJoints} \\
 & \text{s.t.} && \text{maximumStress} \leq 750 \text{ MPa} \\
 & && \text{maximumDeflection} \leq 5 \text{ mm}
 \end{aligned} \tag{11}$$

Mass, number of joints, deflection and stress are analyzed as described in Section 6.3.1.

7.3.3 Guidance

A bicycle frame design has to fulfill several constraints to qualify for the Pareto archive. First, it has to constitute a valid frame, i.e. all fixed components of the bicycle (see Figure 7-12) have to be connected to form a frame. A graph rule is used to check if a design represents a valid frame geometry. Second, the constraints on maximum stress and mean deflection have to be fulfilled as described in the problem formulation in (11). Third, additional constraints

are defined to decrease the objective function values and mimic market requirements. The mass has to be less than 10 kg, mean deflections have to be less than 5 mm and the frame is allowed to have a maximum of 20 joints.

A weighted sum is used for calculating a single objective function value for the first phase in the Burst algorithm, until at least one design is added to the archive and for calculating the rule quality in the advanced Burst algorithm:

$$f(\text{design}) = 10 * \text{mass} + 10000 * \text{meanDeflection} + 1 * \text{numberOfJoints} \quad (12)$$

Common values for mass and number of joints of synthesized bicycle frames are 1-10 [kg] and 5-20 [-], respectively. Mean deflections are commonly between 10^{-4} and 10^{-2} [m]. The weights for the objectives *mass*, *meanDeflection* and *numberOfJoints* in formula (12) are chosen such that the three objectives contribute to the objective function value ($f(\text{design})$) with approximately the same order of magnitude. Between *mass* and *numberOfJoints*, more emphasis is put on minimizing *mass*.

For the second phase of the Burst algorithm, when there is at least one design in the archive, the three objectives (minimize *mass*, minimize *numberOfJoints*, minimize *meanDeflection*) are considered individually. The same parameter settings for the algorithms are used as for the gearbox case study (see Section 7.2). The maximum Burst length (*BL_max*) is set to 20, the maximum number of iterations (*it_max*) is 10,000 and the archive is sampled every 10 iterations to calculate the archive convergence. Rule probabilities are updated every 20 iterations when the advanced Burst algorithm is used. The maximum archive size (*A_max*) is set to 30. In contrast to the gearbox case study (see Section 7.2) no hierarchical sorting is used when selecting designs from the Pareto archive in the advanced Burst algorithm. Instead, the designs are selected randomly from the Pareto archive so as not to give priority to any of the three objectives.

7.3.4 Results

20 experiments with 10,000 iterations are run for each strategy (1 - 4), each algorithm (simple and advanced Burst) and for each of the two initial designs (diamond and void frame). Results for the synthesis process started from the diamond frame are presented first, followed by those when the synthesis process is started from the void frame. Then the results are compared to identify the effect of the initial design on the synthesis results.

7.3.4.1 Results for Diamond Frame as Initial Design

Table 7-1 gives an overview of how many designs are generated for each strategy and algorithm when the synthesis process is started from the diamond frame design. The values in Table 7-1 represent the mean values over 20 experiments. Strategies 1 and 4 apply topologic rules also after a first valid topology, i.e. a frame design connecting all fixed components, is found. This can be seen by the high number of topologically different designs in the final Pareto sets generated with these strategies. Strategies 2 and 3 only generate one

valid topology in each experiment. The average number of designs in the final Pareto set is also higher for Strategies 1 and 4, than for Strategies 2 and 3.

Algorithm	Strategy	Average number of designs in final Pareto set	Average number of topologically different designs in final Pareto set	Iteration in which first valid design is found
Simple	1	22	9	18
	2	12	1	13
	3	11	1	14
	4	23	9	11
Advanced	1	24	11	8
	2	13	1	15
	3	14	1	9
	4	24	9	11

Table 7-2 Comparison of topologically different designs in Pareto set for both algorithms and all four strategies (Initial design: diamond frame).

When comparing the iterations in which the first valid design is found (last column) it can be observed that there is no significant difference between the simple and the advanced Burst algorithm. This is because the first valid designs is found very early in the synthesis process when starting the process from the diamond frame. The advanced Burst algorithm adapts the probabilities to select a rule according to its previous performance. Rule probabilities are updated every 20 iterations. When starting the synthesis process from the diamond frame, the first valid design is most often found before the rule qualities are updated for the first time.

The convergence ratios of the Pareto sets are presented in Figure 7-13 for the simple Burst algorithm and in Figure 7-14 for the advanced Burst algorithm. A fixed reference point is used in the bicycle frame synthesis case study to calculate the volume of the discovered search space for the Pareto sets sampled every ten iterations during the synthesis process. The reference point is defined at (10 kg, 0.005 m, 20) for the objectives (*mass*, *meanDeflection*, *numberOfJoints*) and represents the upper boundary of the three objectives for designs stored in the Pareto archive. The volume estimation is performed as described in Section 7.1.3 and each curve represents the mean value of 20 experiments. The noise in the curves is due to the Monte Carlo Simulation used to estimate the covered volumes. It is higher than in the gearbox case study due to the larger search space in the bicycle frame case study compared to the gearbox case study. Large steps in the curves (see, e.g., the curve for Strategy 2 around iteration 2300 in Figure 7-18) mostly result from single experiments that find a first valid design late in the synthesis process. Before a valid design is found, the convergence ratio is zero resulting in a low mean value for the strategy. Once a valid design is found for the given experiment, the convergence jumps to a higher value. Other reasons for rapid changes in the convergence ratio can be explained as follows. When a new design is generated with a smaller number of joints than the designs in the Pareto archive and this new design is Pareto optimal, suddenly a large portion of the maximum discoverable design space is discovered through this design resulting in a sudden raise of the convergence ratio.

The curves in Figure 7-13 and Figure 7-14 confirm the observations about the iteration in which the first valid design is found. All strategies find archivable designs early in the synthesis process resulting in a rapid raise of the convergence ratio.

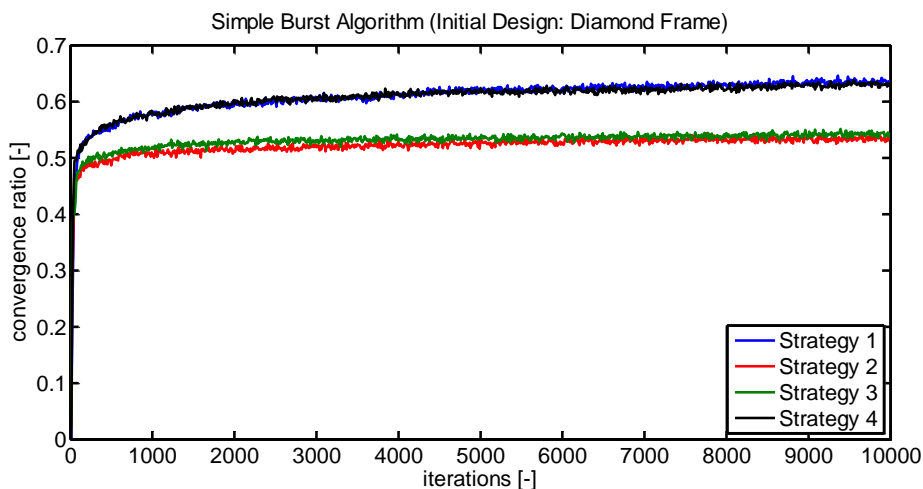


Figure 7-13 Convergence of the Pareto archive for the simple Burst algorithm (Initial design: diamond frame).

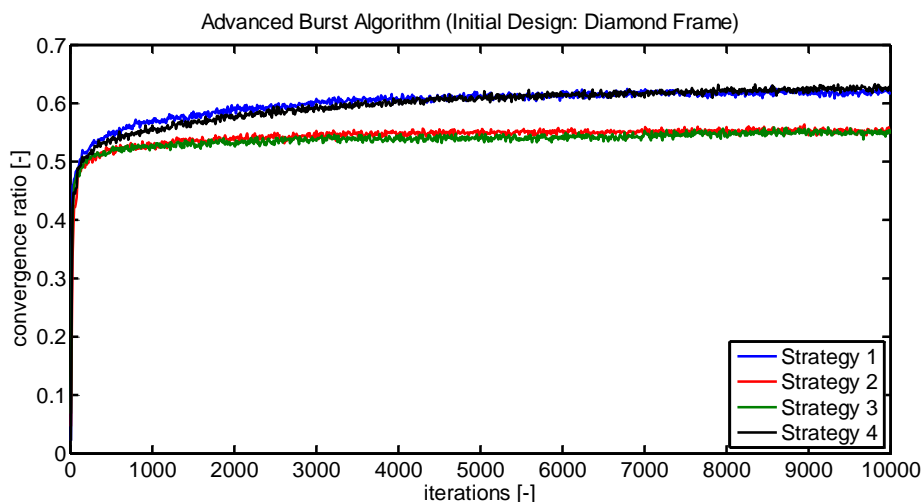


Figure 7-14 Convergence of the Pareto archive for the advanced Burst algorithm (Initial design: diamond frame).

The best designs generated with each strategy are shown in the objective space in Figure 7-15 for the simple Burst algorithm and in Figure 7-16 for the advanced Burst algorithm. Each point in the objective space represents one design. The Pareto-optimal designs for each strategy are identified by combining the Pareto sets for all 20 experiments and filtering the resulting set of designs to identify the non-dominated designs. For both algorithms, designs with five to eight and nine bars, respectively, exist in the combined Pareto set. Among the strategies there is no clear tendency which strategy generates superior designs. This is why the pairwise comparison is conducted as described in Section 7.1.3.



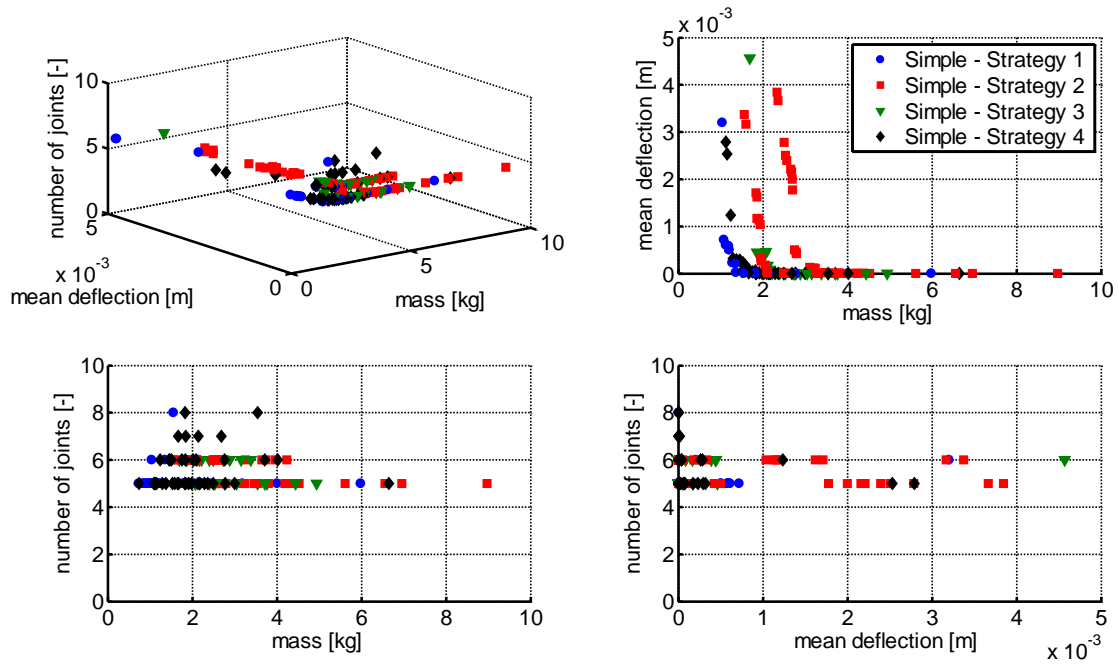


Figure 7-15 Combined Pareto archives for all strategies when using the simple Burst algorithm (Initial design: diamond frame).

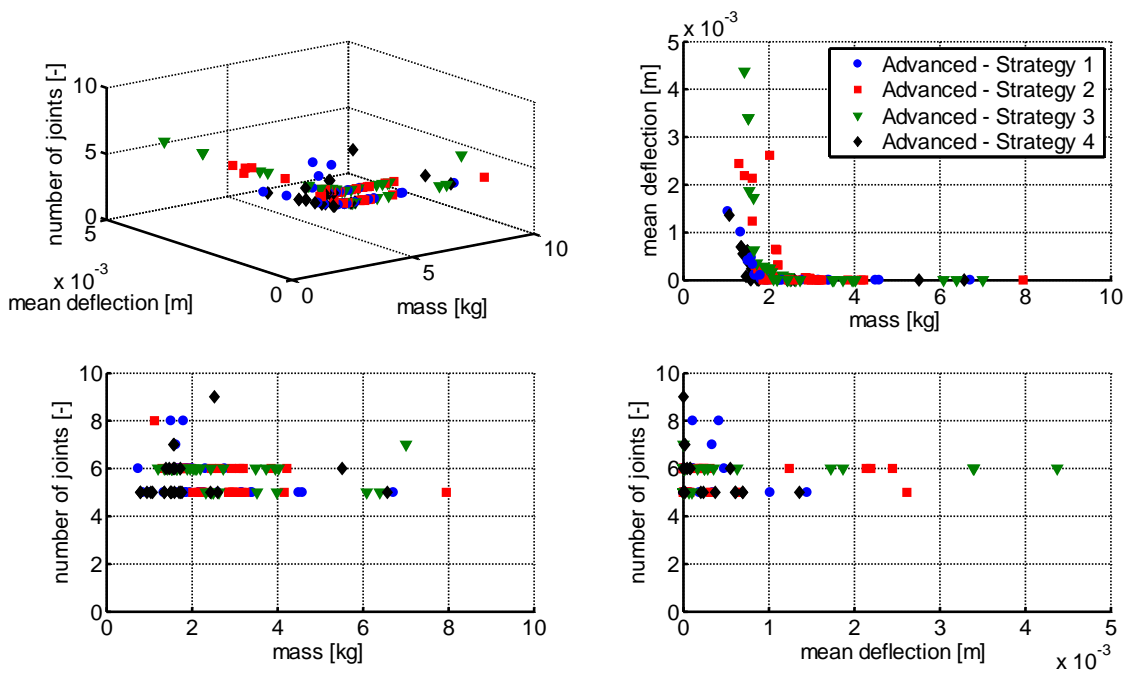


Figure 7-16 Combined Pareto archives for all strategies when using the advanced Burst algorithm (Initial design: diamond frame).

The pairwise comparison of all final Pareto designs of two strategies is conducted and results are shown in Figure 7-17. Strategy 1 outperforms all other strategies for both algorithms, followed by Strategy 4. Of the remaining strategies, Strategy 3 outperforms Strategy 2 for both algorithms.

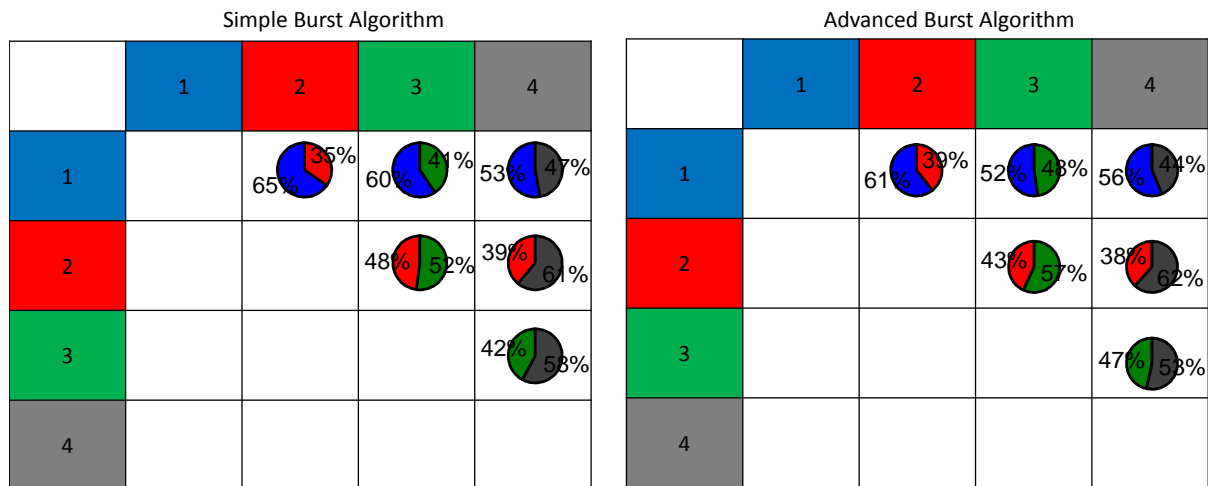


Figure 7-17 Pairwise comparison of strategies (Initial design: diamond frame).

7.3.4.2 Results for Void Frame as Initial Design

The results are different when the synthesis process is started from a void frame design. Average numbers of designs and topologically different designs as well as the iteration in which the first valid design is found are shown in Table 7-3. Again, Strategies 2 and 3 generate only one topologically valid design. Strategies 1 and 4 generate more different topologies in the final Pareto set.

The first valid designs are found later in the synthesis process when compared to the synthesis starting from the diamond frame. Strategy 3, in which only topologic rules are applied until a valid design is found, is faster than the other strategies in finding the first valid design. Also Strategy 4, which applies topologic rules with a high probability in the early stages of the synthesis process, finds valid designs faster than Strategies 1 and 2. Those have no preference on either topologic or parametric rules. When using the advanced Burst algorithm, Strategies 1, 2 and 4 require less iterations to find the first topologically valid design. Strategy 3 requires slightly more iterations on average than when using the simple Burst algorithm. This can be explained by two experiments in which the first valid design is found around iteration 700. These two experiments strongly increase the mean value for Strategy 3.

Algorithm	Strategy	Average number of designs in final Pareto set	Average number of topologically different designs in final Pareto set	Iteration in which first valid design is found
Simple	1	22	9	621
	2	11	1	732
	3	15	1	147
	4	22	8	360
Advanced	1	22	10	544
	2	14	1	702
	3	18	1	208
	4	22	7	275

Table 7-3 Comparison of topologically different designs in Pareto set for both algorithms and all four strategies (Initial design: void frame).

The convergence ratios are presented in Figure 7-18 for the simple Burst algorithm and in Figure 7-19 for the advanced Burst algorithm. In Figure 7-18, the fact that Strategies 3 and 4 apply topologic rules with a high probability in the beginning of the synthesis process and find a valid frame earlier than the other strategies can be retraced. The same applies for the fact that valid design are found earlier in the process when using the advanced Burst algorithm. With the simple Burst algorithm, for all strategies there are experiments that find a first valid design late in the synthesis process, resulting in a stepwise rise of the convergence ratio curves. For the advanced Burst algorithm, by contrast, the experiments find valid designs earlier in the synthesis process resulting in a more steady curve. This can be seen in Figure 7-19.

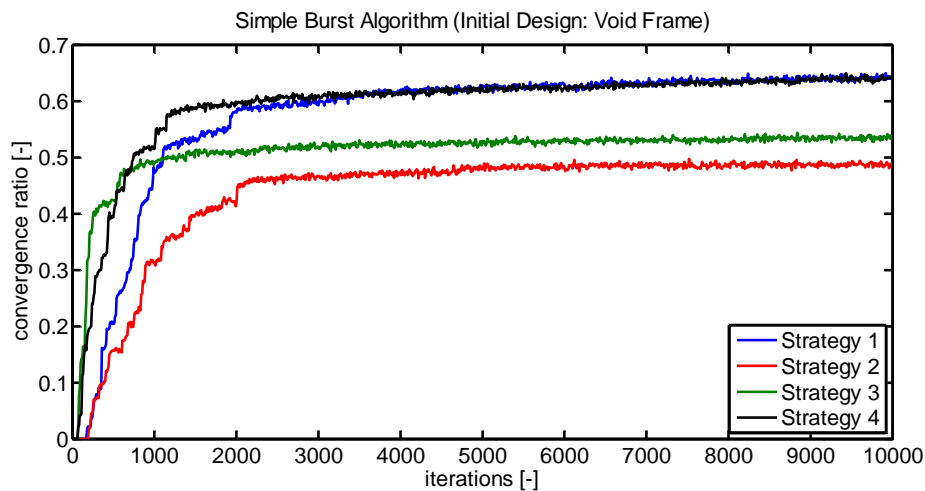


Figure 7-18 Convergence of the Pareto archive for the simple Burst algorithm (Initial design: void frame).

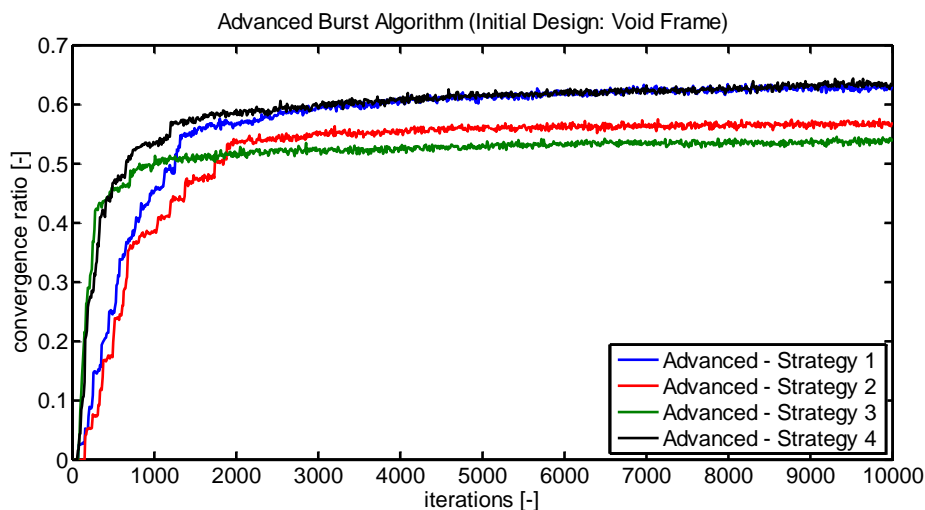


Figure 7-19 Convergence of the Pareto archive for the advanced Burst algorithm (Initial design: void frame).

The combined and filtered Pareto archives after 10,000 iterations are presented in Figure 7-20 for the simple Burst algorithm and in Figure 7-21 for the advanced Burst algorithm. The results are similar to those for the frame synthesis starting from the diamond frame. The designs consist of five to eight bars and there is no clear trend which strategy generates superior designs.

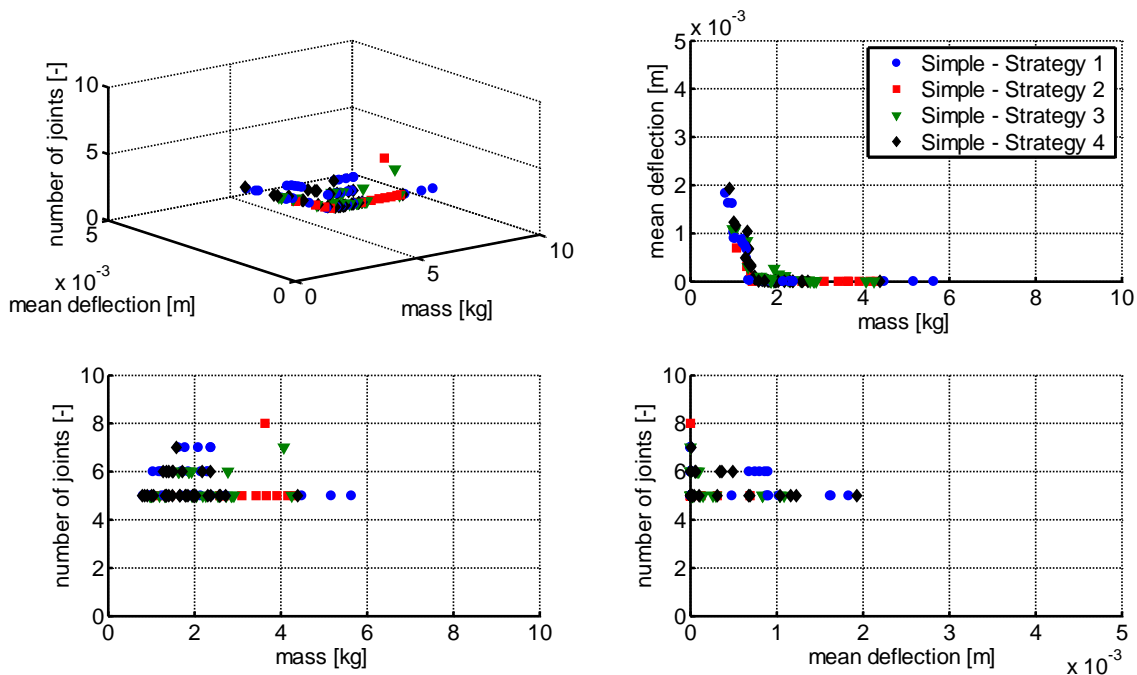


Figure 7-20 Combined Pareto archives for all strategies when using the simple Burst algorithm (Initial design: void frame).

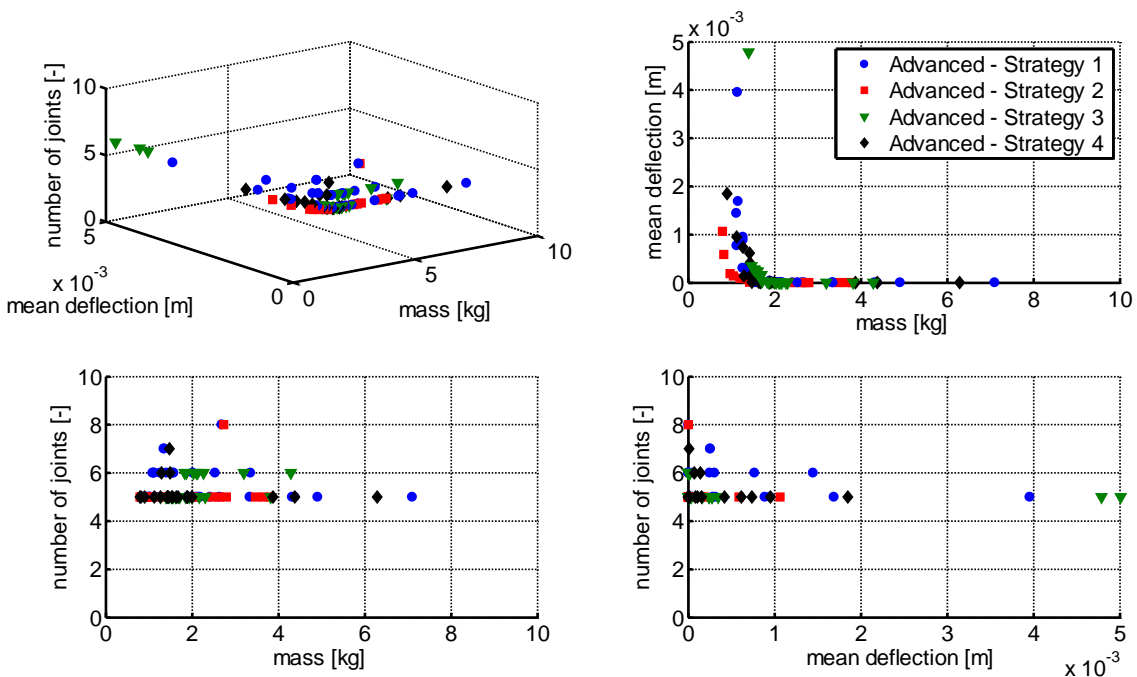


Figure 7-21 Combined Pareto archives for all strategies when using the advanced Burst algorithm (Initial design: void frame).

Results from a pairwise comparison of the final Pareto sets of all strategies are presented in Figure 7-22. For both algorithms, Strategy 1 wins most comparisons, followed by Strategy 4. Strategy 3 is slightly superior to Strategy 2 when the simple Burst algorithm is used. When using the advanced Burst algorithm it is the other way around.

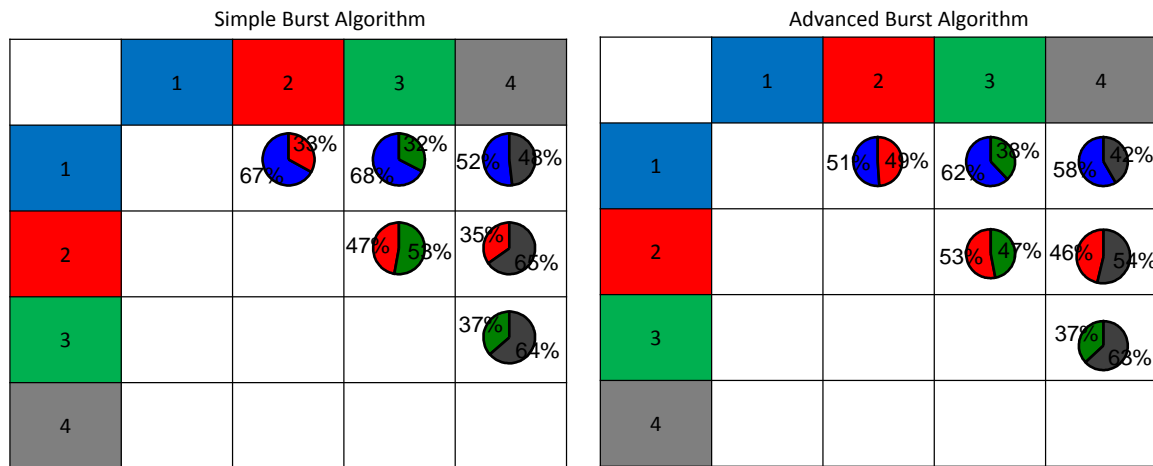


Figure 7-22 Pairwise comparison of strategies (Initial design: void frame).

7.3.4.3 Comparing Results for Different Initial Designs

The effect of the initial design on the generated results is further analyzed. Figure 7-23 shows the results from pairwise comparing the designs generated when the synthesis is started from the diamond frame and the void frame. Colored portions in the pie charts represent the comparisons won by designs when the synthesis is started from the diamond frame. There is almost no difference in the performance of the designs. When using the advanced Burst algorithm, Strategy 2 shows superior results when the synthesis is started from the void frame as a result of the higher number of different topologies that are generated when the synthesis process is started from the void frame.

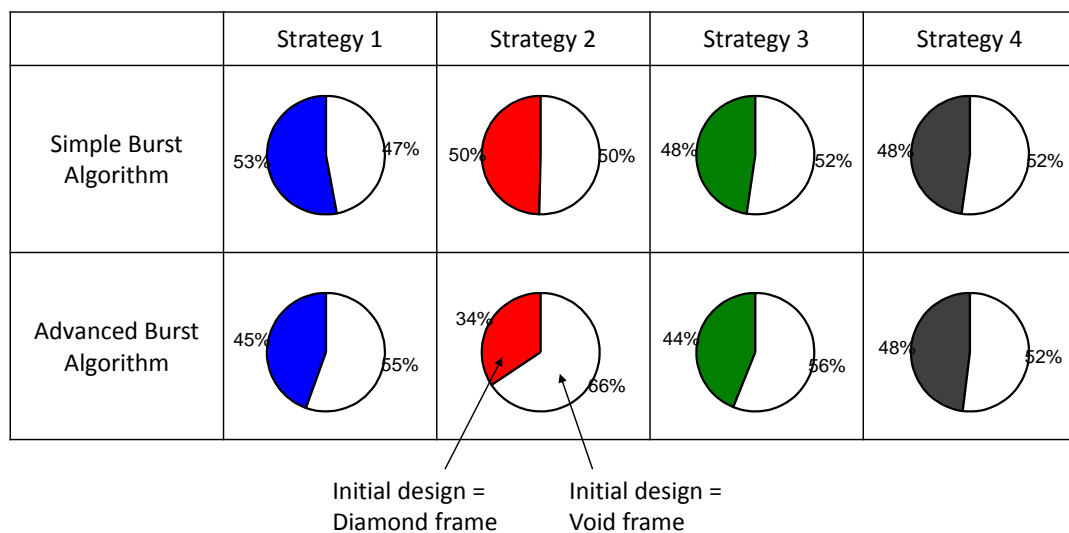


Figure 7-23 Pairwise comparison of designs generated when the synthesis is started from the diamond frame (colored portion of the pie chart) and the void frame (white portion of the pie chart).

Analyzing the designs in more detail confirms that more diverse topologies are generated when the design process is started from the void frame than when starting from the diamond frame. Figure 7-24 (left) shows two example designs that are generated frequently when starting the synthesis from the diamond frame. These designs are generated by adding one bar to the initial design. When starting the synthesis from the void frame, by contrast,

more diverse frame geometries are found. Examples are shown in Figure 7-24 (right). Considering that the designs show similar (simple Burst algorithm) or even slightly superior (advanced Burst algorithm) performance when the synthesis is started from the void frame, it is advantageous to start the synthesis from the void frame for two reasons. First, it is not required to define an initial design, i.e. less effort is needed from the human designer and the initial design is not biased. Second, as a result of not giving bias to the initial design, more diverse designs are generated that means the solution space is explored to a greater extent.

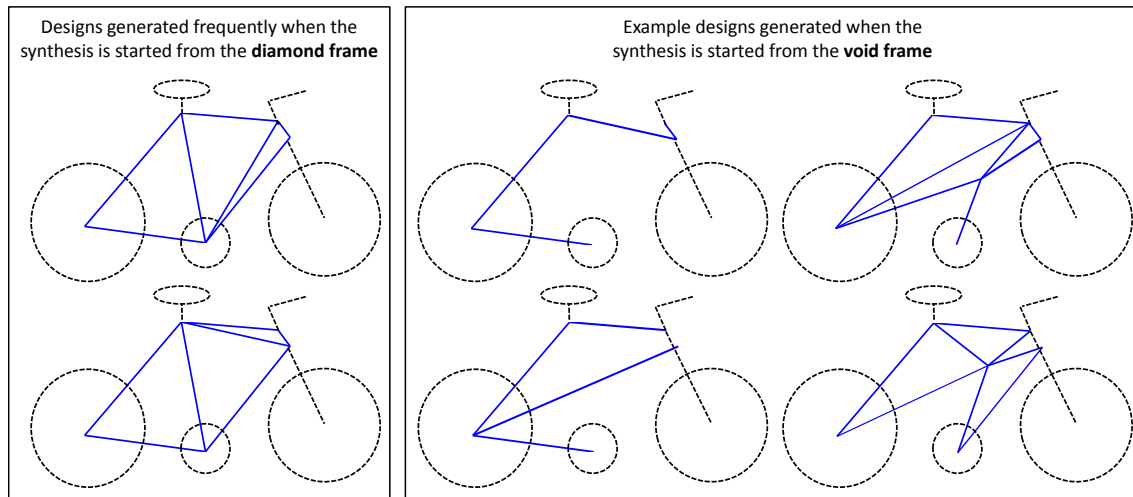


Figure 7-24 Example bicycle frames generated when the synthesis process is started from the diamond frame (left) and the void frame (right).

7.4 Discussion

For both case studies the four strategies generate different designs and explore the design space. There is no significant difference in the computing times among the strategies, however, differences in the number as well as quality of the generated designs exist when comparing the strategies.

For the gearbox case study, both strategies with two phases (Strategies 2 and 3) explore the design space early in the synthesis process through extensive parametric changes. Especially Strategy 3 finds designs with the correct number of speeds early, which reflects that only topologic rules are applied in the beginning. Strategy 4 shows a different effect in the beginning. Archivable designs, i.e. meeting the hard constraint, are found early in the process as well due to high probabilities of applying topologic rules. As the trajectories set high probabilities to apply topologic rules in the beginning of the synthesis process, these designs are topologically modified often and valid designs are transformed to invalid ones. When the number of topologic changes decreases, this strategy explores the space similar to the other strategies. Not all archives generated with Strategies 1 or 4 contain more than one topology. In general, the experiments with Strategies 1 and 4 discover more topologies but dominated designs are discarded, which shows that it is worthwhile investing in topologic changes also after a first valid design is found since a design with optimized parameters might be dominated by a topologically different design. Strategies 2 and 3 mostly exploit the

search space near the first archived solution, whereas Strategies 1 and 4 can explore the space in different regions due to designs with different topologies in the archive, which is desirable for CDS. It is, however, found that through restarting the process, also Strategies 2 and 3 generate topologically different designs. For the given design task, for example, almost every restart of the process results in a new topology. One way to overcome the drawback of Strategies 2 and 3 to generate only one topology is, therefore, to restart the process repeatedly to generate different topologies for which then parametrically optimized versions are searched. Comparing the two algorithms, it can be seen that the differences in the strategies are more obvious in the simple Burst algorithm than in the advanced Burst algorithm.

For the bicycle frame synthesis task, there is a clear tendency that a larger region of the design space is explored when using strategies that allow topologic and parametric rules throughout the whole synthesis process, i.e. Strategies 1 and 4.

When the synthesis process is started from a diamond frame, these strategies show superior to the two phase strategies (Strategies 2 and 3). With Strategies 1 and 4, not only are several topologically different designs explored in one experiment, but the final Pareto set also contains more designs. The final Pareto sets covers a greater portion of the search space (see convergence ratio plots) and when comparing the final Pareto sets pairwise to those of Strategies 2 and 3, those generated with Strategies 1 and 4 win more comparisons. With these observations it can be stated that the quality as well as the quantity of the designs generated with Strategies 1 and 4 are superior. Both algorithms find the first valid design early in the synthesis process. The diamond frame already constitutes a topologically valid design, thus requiring only parameter changes to generate a valid design.

When the synthesis process is started from a void frame, most strategies find the first valid design faster when the advanced Burst algorithm is used than when the simple Burst algorithm is used. This can be explained by the rule probabilities on which the rule selection is based in the advanced Burst algorithm. The positive effect of applying topologic rules is learnt from the rule quality calculation and drives the process to valid designs more rapidly than when this information on rule performances is not used.

The effect of the initial design on the synthesis process can be seen in the early stage of the synthesis process. When the synthesis is started from the void frame, more iterations are required before a first valid design is found than when the synthesis is started from the diamond frame. Starting from a void design can, however, enable the generation of more diverse topologies than when starting the synthesis from a valid topology.

Comparing the gearbox case study and the bicycle frame case studies, it can be observed that the difference between the individual strategies is also dependent on the problem. For the gearbox case study, the difference between the strategies is not as obvious as for the bicycle frame case study. In the gearbox case study, Strategies 1 and 4 show promise to discover more topologically different solutions and explore even distant or disconnected areas of the search space. Strategies 2 and 3, however show similar performance when the

process is restarted repeatedly from the initial design. For the bicycle frame case study, Strategies 1 and 4, i.e. the strategies allowing parametric and topologic rule applications throughout the whole synthesis process, are superior. They lead to a higher convergence ratio and more as well as more different designs.

The results depend on the way the synthesis problem is formulated. This includes the formulation of constraints and upper and lower bounds for parameters, e.g. for gear and shaft sizes in the gearbox case study. Similarly, the calculation of the rule qualities for the advanced Burst algorithm is based on a weighted sum. Changing the formulation of the problem or the weights can change the results.

To summarize, for the given case studies the commonly used strategy of finding a suitable topology first and then searching for optimized parameters did not show significantly better results. When using an approach where only one valid topology is generated, restarts of the synthesis process help to overcome the drawback of limiting the search to one topology. It is further shown that algorithms that permit returns to topologically valid designs stored in a Pareto archive, like the Burst algorithm, are well suited for mixed topologic and parametric problems and more advanced algorithms that, for example, learn from previous rule applications, like the advanced Burst algorithm, can make the exact strategy for rule selection less important to obtain good solutions, as in the gearbox synthesis problem. They can also speed up the process of finding a first valid design, as in the bicycle frame synthesis.

7.5 Summary

The research presented in this section shows that strategies for selecting parametric and topologic rules in a CDS process influence the performance of the search. This effect is, however, dependent on aspects of the algorithm, e.g. the use of dynamic rule qualities, aspects of the problem as well as the grammar rules, e.g. the amount of topologic versus parametric rules, and on the initial design. Results show that commonly used strategies to generate a valid design first, and then search for optimized parameters are applicable when the synthesis process is restarted repeatedly from the initial design to overcome the drawback of restricting the search to one topology. These strategies did, however, not find better designs for the given case studies than strategies allowing topologic changes throughout the whole synthesis process. This means that the widespread assumption that generating a topology first and then applying rules for parametric changes yields better results cannot be concluded for the given case studies. Strategies that allow topologic changes throughout the whole synthesis process showed their potential for generating different topologies, exploring their parametric search spaces and eliminating dominated topologies in one synthesis run. For mixed topologic and parametric design synthesis, an algorithm should be selected that allows returns to topologically valid designs from a Pareto archive when a one phase strategy is used in which topologic and parametric changes are conducted throughout the synthesis process. For two phase strategies, the synthesis process should be restarted repeatedly from the initial design. To conclude, various strategies can be applied successfully to the same problem and using an appropriate algorithm and more

advanced rule selection strategies that, e.g. learn from previous rule applications, can make the exact strategy for topologic and parametric rule selection less important to obtain good solutions.

Research question 7.1 is therefore answered in this chapter.

Research question 7.1: *How can different strategies for selecting topologic and parametric rules in CDS methods using grammars be compared to each other?*

Metrics are defined for comparing different strategies and a systematic method is presented to analyze designs generated with each strategy. Besides the metrics, a systematic process to generate data, analyze it and compare different search strategies is developed. The applicability of the method is shown in the case studies. Considering the different metrics, no strategy shows clearly superior for the gearbox or the bicycle frame task. For the gearbox synthesis task, the strategies are comparable. For the bicycle frame task, the difference between strategies can, however, be observed, e.g., Strategies 1 and 4 generate more different topologies than Strategies 2 and 3. This suggests that even though no unique strategy is found to be superior for the given design synthesis tasks, it is worthwhile comparing different strategies. A trend can be found when comparing different strategies and some might show better than others. The search algorithm or the initial design of a CDS process have an influence on which strategy is superior for a given design task. When starting the synthesis process from a diamond frame design, Strategy 4 wins most pairwise comparisons of the final Pareto sets when using the simple Burst algorithm. When the same algorithm is used but the synthesis process is started from a void frame, Strategy 3 wins most pairwise comparisons. Similarly, also the choice of the algorithm has an influence. Using the advanced Burst algorithm in a synthesis process started from the void frame shows that Strategy 1 wins most pairwise comparisons. To conclude, for a given synthesis task no single strategy exists that is superior for applying topologic and parametric rules. Differences between strategies are, however, existing and can be influenced by other aspects such as the algorithm chosen for the synthesis process and the initial design from which the synthesis process is started.

The following research contributions are achieved:

- **Contribution 1:** Definition of metrics to compare different strategies.

The described metrics enable a comparison of rule selection strategies in a defined way considering different aspects. Besides the number of topologically different designs or the quality of the designs in terms of objective function values, also the convergence of the search process can be compared. The human designer can, thus, not only reason about the differences with respect to the final results, but also considering the speed of the process with respect to finding first valid designs or converging to a final Pareto front.

- **Contribution 2:** Systematic method to support human designers in comparing strategies for selecting topologic and parametric rules for given problems.

The method described in this chapter is independent of the synthesis task, the exact strategies for topologic and parametric rule application and the search algorithm. It can therefore be used as a general method to compare different strategies for rule applications. Test runs for each considered strategy have to be performed. With the presented method, the human designer is then given an objective means to reason about the different strategies and select the most desirable for a given design problem. When comparing different strategies in combination with different algorithm implementations, the human designer can also reason about different combinations of search algorithm and rule selection strategy. If there is no significant difference between different strategies considering one aspect, e.g. the convergence of the Pareto archive for longer run times, as in the gearbox case study, designers can compare other aspects, like the number of topologically different valid designs generated, to determine a preferred strategy, or they can select a random strategy, if none of the tested ones appears to be superior.

Figure 7-25 shows the positioning of the method to compare different strategies for topologic and parametric rule application with respect to the goals of this thesis. The decision of when to select topologic and parametric rules during the CDS process is challenging. The analysis of different strategies allows the human designer to understand how the selection of a strategy influences the search process and the generated designs. Rule selection strategies are dependent on the rule type but not on the exact rules. The presented metrics help to reason about differences in search strategies. If a superior strategy exists it is observed by the human designer (sub-goal G3.3). This helps the human designer to refine the search process (sub-goal G3).

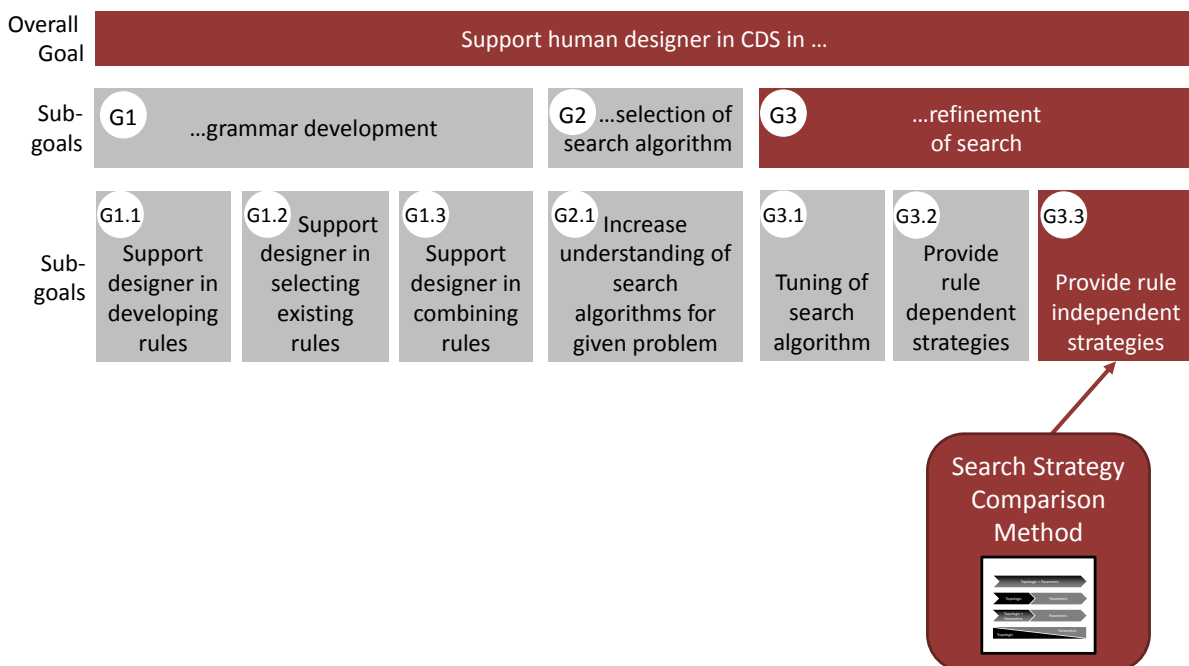


Figure 7-25 Positioning of Search Strategy Comparison Method with respect to the goals of the thesis.

Further contributions of the research in this chapter are as follows:

- **A grammar rule set for gearbox synthesis is developed enabling the generation of gearboxes without collisions.** Case study 1 shows that the developed gearbox rule set enables the generation of gearboxes without collisions. The addition of two rules to shorten and lengthen shafts eases the process of finding gearbox designs without collisions between components. This is a significant improvement compared to previous research, where collisions have to be removed manually.
- **Different strategies for topologic and parametric rule applications are presented.** Four strategies are introduced and discussed in detail for both case studies. Comparing the methods on a given problem and interpreting the analysis results, the human designer can decide which strategy to use.
- **The effect of the initial design on the generated designs can be explored.** As conducted for the bicycle frame synthesis case study, the effect of starting the synthesis process from different initial designs can be analyzed using the proposed metrics.
- **The developed software prototype enables automatic analysis of the generated designs.** The synthesized designs during a CDS process are analyzed and visualizations are generated automatically, when the human designer uses the Search Strategy Comparison Method.

8 Implementation

A general framework for CDS is developed in the thesis. A schematic overview of this framework and its modules is given in Figure 8-1. The *CDS process module*, *post-processing module* and the *output and visualization software* reflect the three step process of data generation, analysis and visualization used throughout this thesis.

Some implementation aspects are already described in Chapters 4, 5, 6 and 7 where required to understand the respective methods. In this chapter, a general overview of the framework and the interfaces to external software is given (Section 8.1). Then, the different implemented options the human designer can select from for data generation (Section 8.2), analysis (Section 8.3) and visualization (Section 8.4) are presented. Section 8.5 describes the usage of the framework. Its generality and possible extensions are discussed in Section 8.6.

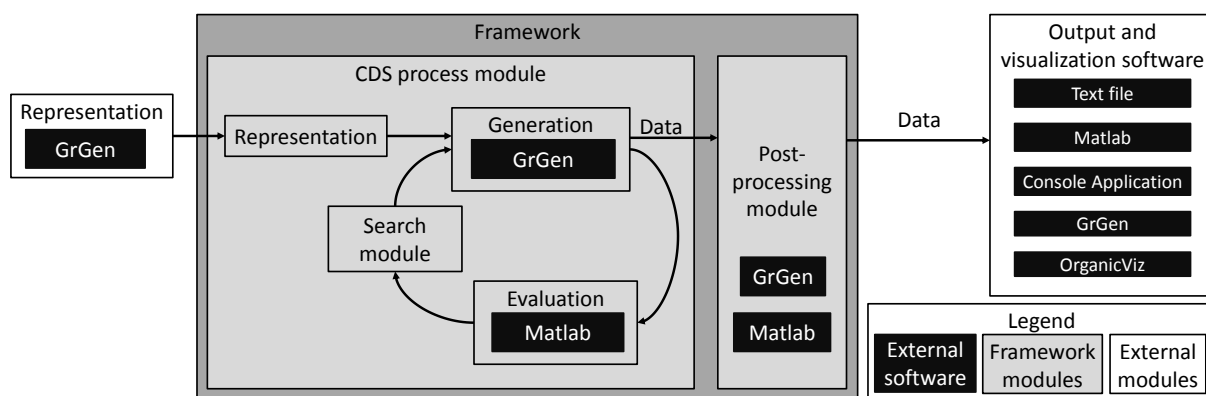


Figure 8-1 Overview of the generic CDS framework implemented in this thesis.

8.1 Generic Framework for CDS

Figure 8-1 presents the generic framework for CDS. It is implemented in c# and interfaces GrGen via DLLs and Matlab via the Matlab COM Automation Server. The software architecture facilitates the integration of other grammar systems and simulation tools. In the current implementation only GrGen is linked to the framework. The following sections, therefore, assume that GrGen is used for graph rewriting. The synthesis process is performed within the *CDS process module*. The *post processing module* performs analyses and the different *visualizations* present the analysis results to the human designer.

The **CDS process module** reflects the four steps of CDS as presented in Figure 2-3. The framework is implemented problem independently. It provides abstract classes that have to be instantiated and implemented for the specific problem at hand. These problem-specific elements are a) the model and grammar rules in the representation step, and b) a means to evaluate the generated designs in the evaluation step.

The CDS process module expects a problem-specific **representation**. This representation consists of a meta model that defines available edges and nodes for the graph representing the design and a set of grammar rules. Rule designers can develop grammar rules

independently from the generic framework. The meta model and rules are compiled and integrated in the generic framework as DLLs. In the **generation** step, the precompiled rules are then applied using GrGen.

Various options exist to **evaluate** the generated designs and their choice depends on the problem at hand. Two evaluation approaches are implemented in this thesis. The gearbox designs (see Section 4.3.1.3) are evaluated using grammar rules implemented in GrGen. Those rules are directly applied to the graph representation of the gearbox design and evaluation results are stored as attributes in temporary nodes that are added to the graph. These evaluation results are then read by the evaluation module of the framework and the temporary nodes are deleted. The bicycle frame designs (see Section 6.3.1.3) are evaluated using the FEM for structural analysis implemented in Matlab. A mapping is implemented that transforms the graph structure of a design into matrices that describe the components and their connections in Matlab.

The Burst and SA algorithm are implemented in different versions in the **search module**. They are implemented problem independently and for multi-objective search and optimization with up to three objectives.

In the **post-processing module**, the generated designs are analyzed. These analyses are done either by functions implemented in the framework, through rules using GrGen or using Matlab.

8.2 Data Generation Options

Figure 8-2 gives an overview of the different options for data generation in the generic framework.

Options \ Algorithms	A Selection of design from Pareto archive				B Strategies for rule type selection					C Selection of rule		
	None	Random	Maximum distance	Filtering by objectives	Only topology	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Random	Advanced (rule quality)	Predetermined
Random generation	■					■				■		
Burst algorithm		●		▲		●▲	▲	▲	▲	●▲	▲	
Simulated Annealing	●	●		▲		●▲	▲	▲	▲	●▲	▲	
Exhaustive without location	●				●							●
Exhaustive considering location	●				●							●

Legend	
Option selected in Chapter 4 (GRAM)	■
Option selected in Chapter 5 (Network-based Rule Analysis)	●
Option selected in Chapter 6 (Relation Visualization)	●
Option selected in Chapter 7 (Search Strategy Comparison)	▲
Option is available	
Option not available	

Figure 8-2 Overview of the data generation options.

Five different algorithms are available to generate data. They are shown in the rows in Figure 8-2.

- **Random generation** generates designs by starting from an initial design and applying rules that are selected randomly in a sequence without returning to previous designs.

- The **Burst algorithm** executes Bursts of rule applications, of a given length defined by a parameter, and returns to previous designs frequently.
- The **simulated annealing** algorithm is available in two versions with and without returns to base and is implemented using the “vanilla” schedule [140].
- Two versions of **exhaustive generation** of designs are available. Both start the generation from an initial design and execute a BFS until a given depth of the search tree. The first version (exhaustive generation without considering location) applies each rule once on each design in the search tree. The second version (exhaustive generation considering location) applies each rule as often as possible on each design in the search tree, i.e. it enables applying the same rule at different locations in one design.

The Burst and SA algorithm allow returns to previously discovered designs. Three methods are available to select a design from the Pareto archive for further exploration. They are shown in Figure 8-2 in the columns of category “A - Selection of design from Pareto archive”:

- **Random** selection means that one design is selected randomly from all designs in the Pareto archive.
- In the **maximum distance** selection method, the distances to all other designs are calculated and summed for each design. The design with the maximum total distance is then selected for further exploration assuming that it constitutes a design in a less explored area of the Pareto archive.
- When focus should be given to certain objectives, a **filtering by objectives** reduces the designs in the Pareto archive to those superior with respect to these objectives. A design is then chosen randomly from the filtered set of designs.

There are five different strategies for selecting the type of rule, i.e. topologic or parametric, for the next iteration of the search process. They are shown in Figure 8-2 in the columns of category “B - Strategies for rule type selection”. The first option means that only topologic rules are applied during the synthesis process, as for the research in Chapter 5 on rule sequences. The other strategies for rule type selection are introduced in detail in Section 7.1.2.

For selecting the exact rule that is applied in the next iteration, there are three options. They are shown in Figure 8-2 in the columns of category “C - Selection of rule”. Either the rule is selected randomly or it is selected based on rule qualities calculated from statistics on the rule’s previous performance. The third option is to have the rule numbers predefined, which applies for the exhaustive generation methods.

The colored pictographs in Figure 8-2 indicate which options for data generation are selected for the research presented in Chapters 4, 5, 6 and 7. The cells highlighted in grey indicate which options (column) can be selected for which algorithm (row). Some options, e.g. using the maximum distance between designs for selecting a design from the Pareto archive, are not used for data generation within this thesis. This option is, however, used for reducing the archive when the archive size exceeds the maximum.

8.3 Data Analysis Options

Different data is stored during the data generation depending on the analyses to be conducted. This means the options for data analysis have to be selected before the data generation is started. The generated data is then processed in the *post-processing module*. Figure 8-3 summarizes the different data analyses. The different data analyses are presented in the columns and are labeled A – E. Row I indicates how each analysis is implemented, i.e. in Matlab or using GrGen rules. In the rows of category II, the grey cells indicate for which data the analysis can be performed. The colored pictographs indicate for which data the analyses are performed in the chapters of the thesis.

I - Implementation / II - Available data		Analyses	A	B	C	D	E
			Analyzing changes in objective / constraint values	Monte Carlo simulation to estimate volume covered by Pareto set	Pairwise comparison of final Pareto fronts	Comparison of topologies (graph isomorphism check)	Generation and analysis of transition graph
I		Implementation of analysis using ...	Matlab	Matlab	Matlab	GrGen	GrGen
II Available data	Graph representation of Pareto designs generated during data generation				▲	
		... all designs generated during data generation				■ ● ● ● ● ● ● ●	● ● ● ● ● ● ● ●
	Objective and constraint values of Pareto designs generated during data generation	▲	▲	▲		
		... all designs generated during data generation	■ ● ●				

Legend	
Analysis performed in Chapter 4 (GRAM)	■
Analysis performed in Chapter 5 (Network-based Rule Analysis)	●
Analysis performed in Chapter 6 (Relation Visualization)	●
Analysis performed in Chapter 7 (Search Strategy Comparison)	▲
Data for which analysis can be conducted	
Data for which analysis cannot be conducted	X

Figure 8-3 Overview of the data analysis options.

8.4 Data Visualizations

Different **output files** are generated to visualize the analysis results and present them to the human designer. Figure 8-4 gives an overview of the available visualizations (columns), the software used to visualize the analysis results (row I) and the analyses required (rows of category II). The colored pictographs indicate which visualizations are used in Chapters 4, 5, 6 and 7 and which analyses are required. Besides these visualizations, text files are generated that can directly be read by the human designer, e.g. for presenting the detected loops in the transition graph (see Chapter 5). Additionally, the graphs of the final designs and one topology of each topologically different design are stored and can be visualized using yComp [125], a graph visualization tool.

Method/ Visualization		Chapter 4 (Grammar Rule Analysis Method)			Chapter 5 (Network-based Rule Analysis Method)		Chapter 6 (Relation Visualization Method)			Chapter 7 (Search Strategy Comparison Method)		
		Boxplots	Design space plot	Diversity ratio	Console application to analyze transition graph	Unique topology plot / transition graph	Performance space plot	Rule analysis plot	Unique topology plot	Convergence plot	Strategy comparison	Pareto set plots
I - Implementation/ II - Analysis												
I Implementation of visualization in ...		Matlab	Matlab	Matlab	Console using GrGen	OrganicViz	Matlab GUI	Matlab	Matlab	Matlab	Matlab	Matlab
II - Analysis	A Analyzing changes in objective / constraint values	■	■				●	●				▲
	B Monte Carlo simulation to estimate volume covered by Pareto set									▲		
	C Pairwise comparison of final Pareto fronts										▲	
	D Comparison of topologies (graph isomorphism check)		■	■	●	●						
	E Generation and analysis of transition graph				●	●	●					

Figure 8-4 Overview of the visualizations used in this thesis and the respective analyses.

8.5 Usage of the Framework

Figure 8-5 visualizes the modifications that are required to adapt the generic framework to a specific design problem.

- Model and grammar rules for the problem are developed and compiled. When GrGen is used, the model is defined in a *gm file*, while the rules are defined in a *grg file*. The generated *DLLs* are added to the framework project. The user provides rule number, rule name and its type, i.e. topologic or parametric, to the GUI of the framework.
- Name and desired direction of change are defined for all objectives. For constraints the name, and upper and lower bound are defined.
- A simulation or adequate evaluation for generated designs is provided and mappings from the graph representation to the representation in the simulation are implemented. The simulation is linked to the framework. For Matlab this interface is available and only the path to the simulation file is to be specified. One function (*Simulation.simulate*) has to be overwritten to call the model transformation, start the simulation and return the evaluation results to the framework.
- The path to a folder in which the output files are stored is to be provided.

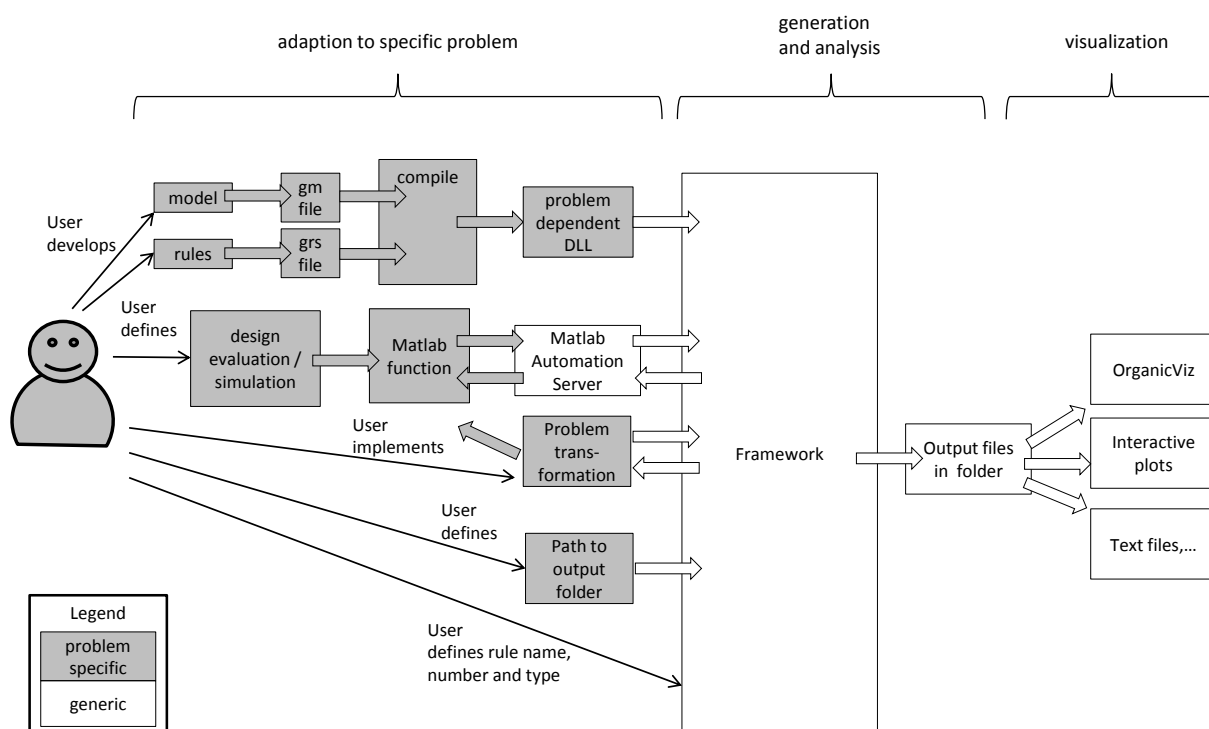


Figure 8-5 Adaption of the framework to a specific problem.

Once the problem-specific adaptations are performed, the human designer can select the options presented in Sections 8.2 and 8.3 and start the synthesis process to generate and analyze the data.

Figure 8-6 shows the GUI of the framework. The orange numbers indicate different areas of the GUI that are described in the following. The user interface consists of three regions for selecting data generation options (1) and data analysis options (2) and for presenting synthesis results (3). In the data generation options, the user can select and deselect each of the grammar rules (1a). The number of iterations for one synthesis run and the number of runs can be defined in the search options (1b). The user can further select the search algorithm (1c), specify if and how often a RTB is conducted (1d), select a maximum size for the Pareto archive and define how designs are selected from the archive when the RTB option is active (1e), and select the search strategy for applying topologic and parametric rules (1f). Links between the analysis options and the methods developed in this thesis (2) ease the selection of appropriate analyses. The data generation and analysis process is started by the “Start Synthesis”-Button (4). The result output area (3) gives an overview of the designs in the final Pareto archive. The user can select designs from the final design list (3a) and an image of the design (3b) and key evaluation results (3c) are presented. For detailed information and the visualizations, the output files can be found in the output folder for which the path is given (5).

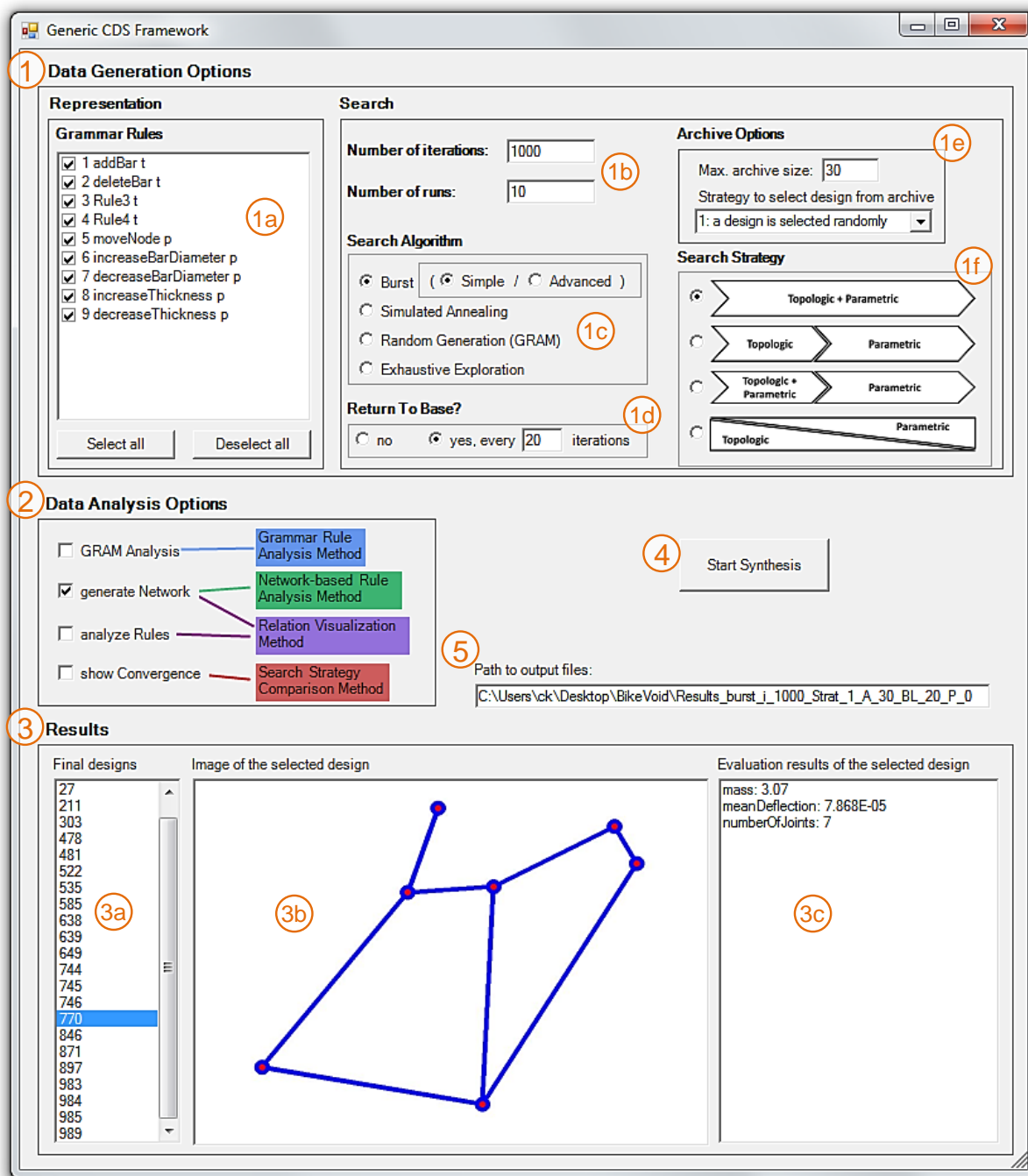


Figure 8-6 Overview of the GUI of the generic framework showing options for data generation and analysis.

8.6 Discussion

The implemented generic framework and the analysis methods enable human designers to use the presented methods (Chapters 4, 5, 6 and 7) for grammar design, analysis and application. Through its object-oriented and modular implementation, the adaption to different design tasks can be conducted with few modifications.

In its current implementation, the framework supports the use of GrGen for grammar development and application. The use of other grammars, e.g. shape grammars, is possible. The implementation effort is, however, higher since also parts of the representation and generation module have to be modified.

The implementation is done using the .NET Framework that allows for straight forward extension of the framework, e.g. to use other software for the simulation of generated designs. The decoupling of data generation and analysis from the visualization through defined output files permits the use of different visualization tools. When the human designer wishes to use different external software to visualize analysis results, only the code generating the output files has to be modified.

The implementation of the analysis methods within one software prototype enables various combinations of data generation and analysis, even more than what is presented in this thesis. It is, e.g., possible to analyze how the UTS is explored using the same algorithm but different search strategies for topologic and parametric rule selection, thus combining the Relation Visualization Method (Chapter 5) and Search Strategy Comparison Method (Chapter 7). Figure 8-4 can be used to identify which analyses and software are required to generate a specific visualization and Figure 8-3 indicates which data is required for the analysis. The user, however, has to trade information gained in the analyses for required memory and runtime. The more analyses are preformed, the more data has to be stored and processed requiring time and memory. This is in particular important, when analyses are required for CDS processes with many iterations and when many different graph representations have to be stored and checked for isomorphisms.

Another advantage of the implemented framework is that it can be used not only for the analyses presented, but also for the actual CDS process once the human designer has defined grammar rules and search algorithm.

9 Discussion and Future Work

In this chapter, the developed methodology is discussed in a broader sense. The specific contributions of the methods presented in Chapters 4, 5, 6 and 7 are discussed in detail in the summary sections of the respective chapters. Commonalities and differences between the methods in the developed methodology are discussed in Section 9.1. The adapted process for grammar development and application when using the developed methodology is presented in Section 9.2. The generality of the methods is discussed in Section 9.3 and recommendations on how to use the methods are given for two example scenarios. Section 9.4 addresses how the methods contribute to supporting human designers in CDS. Section 9.5 discusses limitations of the methodology and proposes future research.

9.1 Comparison of the Methods

In Figure 9-1, an overview of the presented methods is given. The methods are positioned qualitatively along two axes to describe a) the level of information the human designer receives about the performance of the rules, i.e. how the rules change the designs regarding objectives and constraints, and b) the level of information the human designer receives about the search process when using the respective method. This representation is meant to visualize the differences between the presented methods and ease the selection of an appropriate method. Reasons for the relative positioning of the methods in Figure 9-1 are presented in the following paragraphs. Overlaps between the methods exist and are discussed throughout the thesis. They are, however, not shown in Figure 9-1 to improve readability.

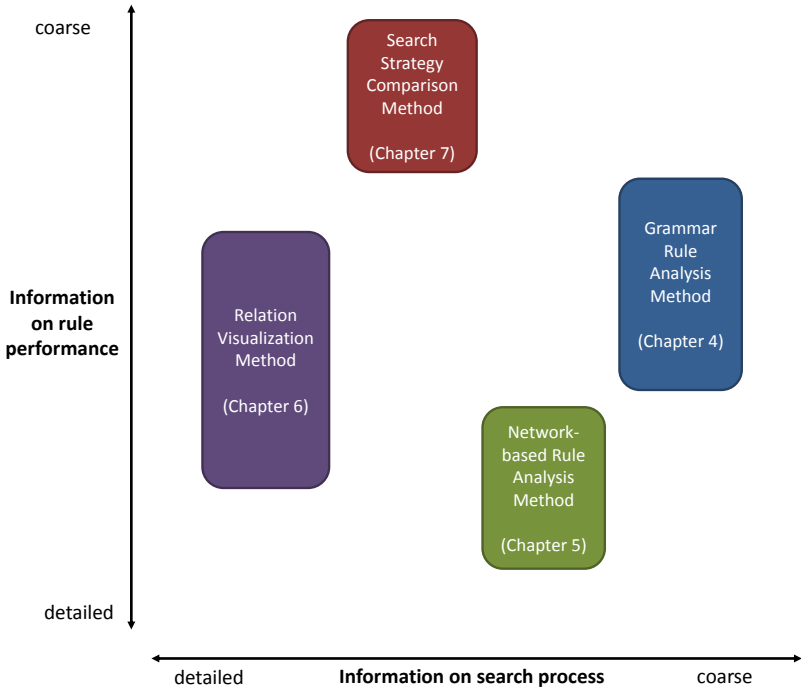


Figure 9-1 Qualitative classification of methods based on the information provided by the methods.

GRAM provides information on the rules at various levels of detail. For example, detailed bar charts represent the influence of each individual rule on the objectives, whereas validity and diversity ratio give high level information on the rule set. The search process is not analyzed explicitly in GRAM.

The **Network-based Rule Analysis Method** provides more detailed information considering the search process than GRAM because the data it uses can be generated using different search algorithms. The influence of the search algorithm on the explored designs is visible in the synthesized transition graph. The information on the rules is more detailed than when GRAM is used, since single rules can be analyzed in more detail comparing different LHS matches and also sequences of rules are analyzed.

The **Relation Visualization Method** provides detailed information on how a search algorithm explores the design space during CDS and, therefore, delivers more information on the algorithm than the other methods in this thesis. The level of detail regarding information on rules is between that of GRAM and that of the Network-based Rule Analysis. It provides information on the links between changes in objectives, which GRAM does not, however, only single rule applications and no sequences of rules are considered.

The **Search Strategy Comparison Method** gives more information on the search algorithm than the Network-based Rule Analysis, but less than the Relation Visualization Method since only the progression of the Pareto archive is presented to the user. In the Relation Visualization Method all generated designs, i.e. also the discarded ones, are visualized. There is little information about the rules because only the difference between different strategies for selecting types of rules are analyzed.

9.2 Modified Process for Grammar Development and Application

Figure 9-2 presents the modified process for grammar development and application that can be applied when using the presented methodology. It is based on scenario 5 of the user control scenarios defined by Chase [59] (see also Figure 2-4) since this scenario is used in this thesis. Other scenarios can similarly be modified by adding the proposed step to the grammar development phase.

In a first step, the object representation, control mechanisms and grammar rules are developed by a human designer or developer. In the next step, which is proposed in this thesis, the generated grammar rules are analyzed. The dashed arcs between the first step and the proposed methods indicate that it is not required to use all methods. Depending on which information the human designer intends to gain, they can select the appropriate methods. GRAM, the Network-based Rule Analysis Method and the Relation Visualization Method support rule development. This is indicated by the colored arcs leading back to the first step. The human designer can use the information gained in the analysis to further improve the grammar rules in an iterative process. Once the developed grammar rules are as intended, the grammar is applied. Information gained in the Network-based Rule Analysis Method, the Relation Visualization Method and the Search Strategy Comparison Method

influences the control in the grammar application phase. In Figure 9-2 this is presented by the colored arcs heading to the step where the rule is determined that is applied next. There are often interdependencies among the three steps “determination of rule”, “determination of object” and “determination of matching condition” [59]. In this thesis, matching conditions are defined within the grammar rules and LHSs of the rules narrow the number of objects on which a rule can be applied. The rule to apply in the next iteration is determined by the search algorithm. Using the proposed methodology, the human designer can take informed decisions on how the computer control selects the next rule. The information gained in the Network-based Rule Analysis Method can be used to determine beneficial sequences of rule applications. Knowledge gained on analyzing the LHS of rules enables the human designer to formulate advanced control mechanisms for determining the object, i.e. the location where rules are applied. Information on how search algorithms explore the design space can be gained through the Relation Visualization Method and used to decide on an appropriate search algorithm for controlling the rule application process. The Search Strategy Comparison Method can be used to determine which search strategy to use for applying topologic and parametric rules.

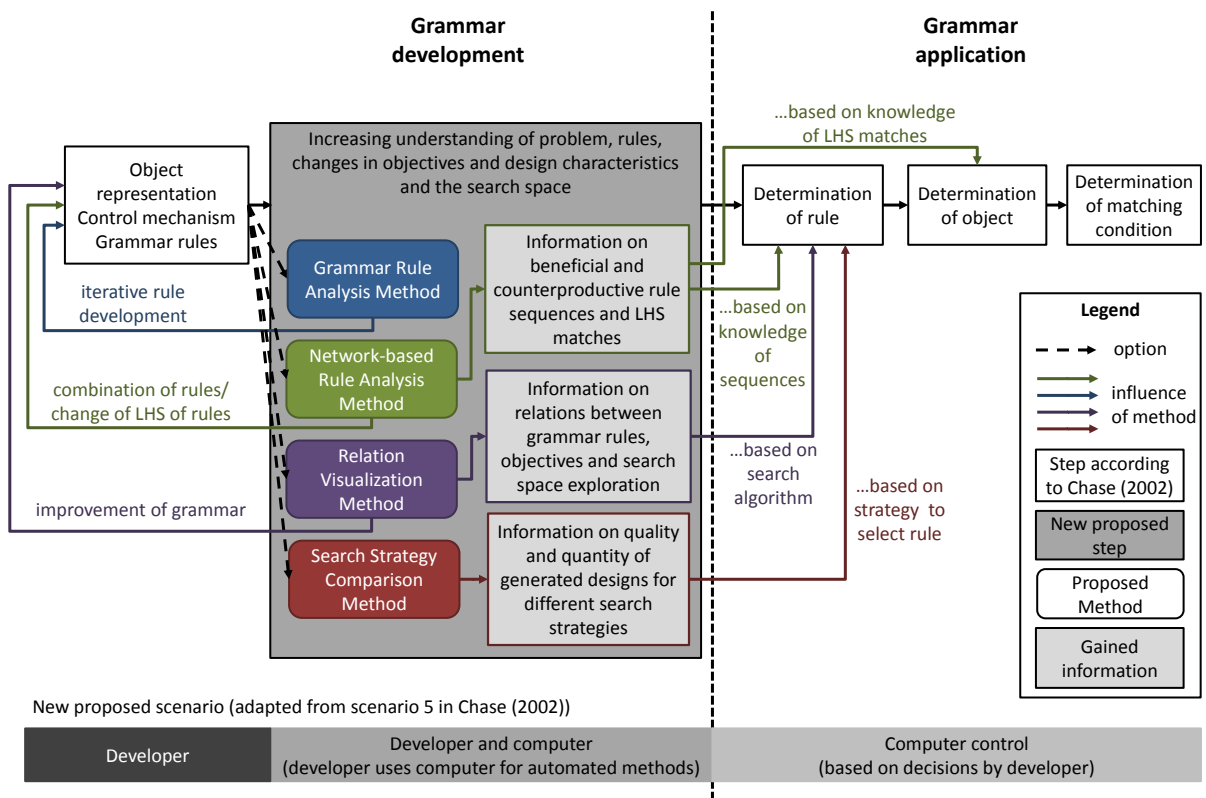


Figure 9-2 Modified process for grammar development and application when using the proposed methodology.

Using the modified process for grammar development and application, the human designer and the computer collaborate in an additional step during grammar development. The computer performs automated analyses of generated data and provides the designer with visualizations. The designer interprets these to gain information on the design task, the rules, how rules change objectives and design characteristics, as well as how search algorithms

explore the search space. This information can then be used to support both grammar development and grammar application.

The case studies show how grammars and search algorithms are analyzed systematically using the presented methods and how visualization of these analyses provide information for the human designer. For the gearbox synthesis case study, e.g., different grammars are analyzed and compared using GRAM. An unnecessary rule is deleted from the rule set based on this analysis and additional rules to shorten and lengthen shafts are added based on knowledge of the design task from previous research. Sequences of rule applications are then analyzed systematically. While one rule seems redundant to a sequence of two other rules at first glance, its superiority for exploring diverse topologies is understood through the Network-based Rule Analysis Method and it is decided to keep this rule in the rule set. The Relation Visualization Method shows that the unique topology space for gearbox designs is highly connected, i.e. there are various sequences of rule applications that transform an initial design into a desired one. Different strategies for applying topologic and parametric rules are then compared using the Search Strategy Comparison Method to better understand the issue of applying topologic and parametric rules together or separately. Results show that the search strategies differ mostly in the early iterations of the synthesis process. It is found that strategies that allow topologic and parametric changes throughout the synthesis process explore different regions of the search space in one synthesis run. This is desirable for CDS and is due to designs with different topologies in the archive. Having strategies with two phases, i.e. only parametric rules are applied once a valid design is found, different topologies can be generated when the process is started repeatedly. As an additional result, gearbox designs without collisions are found. This is a significant improvement compared to previous research by Lin et al. [38] and can be explained by the additional rules that are added.

9.3 Generality of the Methodology

Figure 9-2 presents the modified grammar development and application process proposed in this thesis where the presented methods are optional and can be selected by the human designer to support grammar development and application. Which specific methods to use and how to interpret the visualizations depends on the specific design scenario. In the following, two example design scenarios are presented and recommendations are given on how to apply the presented methodology. The first scenario considers the design of a new product from scratch. The second scenario describes a re-design situation. Finally, the generality of the methods is discussed with respect to the types of grammars that can be supported.

In the first scenario a **new design** is to be generated from scratch using a grammar-based CDS method and no grammar exists. Additionally, the aim is to generate a broad variety of designs. In this case it is recommended to use all presented methods of the methodology. In a first step, an initial grammar is developed. Then, GRAM is used to support the grammar development process by giving feedback on each rule's performance. The grammar rules are

improved in an iterative process until they perform as intended by the designer. Using the Network-based Rule Analysis Method, the rule set can further be improved by identifying whether rules can be combined. Inefficient implementations of rules can be detected and the analysis of LHS matches of rules allows to analyze in detail whether the grammar rules apply as intended. Since the aim in this scenario is to generate diverse designs, the designer should test whether the grammar rules sufficiently explore the design space or whether they recreate the same designs repeatedly. This can be done by visually analyzing the transition graph to see how strongly the designs are connected through rule applications. In case the space is explored to a small extent only, the grammar rule set should be extended by rules that perturb existing designs to a greater degree to allow for a faster exploration of the space. Using the Relation Visualization Method, an appropriate algorithm can then be identified. To explore the search space, e.g. the Burst algorithm with a large Burst length can be used and the archive size should be set to a sufficiently high number to allow for the storage of diverse designs. Testing several settings for the maximum Burst length and archive size allows to tune the algorithm. Finally, an appropriate search strategy should be selected to determine which rule type to apply in which phase of the synthesis process. The decision can be based on the information gained when using the Network-based Rule Analysis Method and the Relation Visualization Method. When the designs are highly connected to each other this means that the same designs are generated repeatedly. Results in Section 7 show that in this case a search strategy where topologic and parametric rules are applied throughout the synthesis process allows for the generation of diverse designs. In case the designs are only sparsely linked to each other and valid designs are destroyed frequently, a strategy with two phases for topologic and parametric rule applications should be used and the synthesis process should be restarted repeatedly to generate ample diverse designs. If the rules generate valid designs within a reasonable number of iterations, the synthesis process can be started from a void design to not bias the results and foster the generation of different designs.

The second design scenario is a **re-design** situation in which an existing design is to be re-designed, e.g. to meet further requirements. The aim is to modify the existing design such that the additional requirements are met without radically changing the product. It is assumed that the existing product was generated using a grammar-based CDS method, i.e. a design grammar exists. In this scenario, it can be sufficient to exploit the search space around existing designs, i.e. starting the synthesis process from the existing design. The Burst algorithm with a small Burst length or the SA algorithm can be used. The Relation Visualization Method should be applied to test which algorithm parameter settings to use to exploit the search space near existing designs. Having selected the search algorithm and specified the parameter settings, the designer can synthesize new products. Only if no new designs are found that meet the additional requirements, the human designer has to analyze the grammar rules using GRAM or the Network-based Rule Analysis Method to further develop the grammar and adapt the rule set to the new situation.

Although all case studies in this thesis use graph grammars, the presented methods are similarly applicable to other **types of grammars**. In general, any type of grammar can be used as long as unique designs can be identified and intermediate designs can be evaluated. This means, the methods are also applicable, e.g., to shape grammars when unique shapes can be determined computationally.

9.4 Research Contributions

Supporting the human designer when using grammar-based CDS methods is the goal of this thesis. This goal is achieved through the development of a methodology that can be integrated in the existing grammar design and application process (see Section 9.2). Sub-goals are to support grammar development (sub-goal G1), search algorithm selection (sub-goal G2) and refinement of the search process (sub-goal G3). The following sections discuss how these sub-goals are achieved. The development of a software prototype that embodies the developed methods is an additional contribution.

9.4.1 Methodology for Supporting Design Grammar Development and Application

The key contribution of this thesis is the development of a methodology for supporting grammar development and application. Processes for generating data, analyzing it and producing visualizations are defined. Additionally, criteria for grammar rules and for comparing search strategies are developed that serve as measures for assessing grammar rules and search strategies, respectively. The presented methods can be seen as information visualization and visual analytics methods. Confirmatory analysis, i.e. analyses that test hypotheses, are possible. An example is to analyze if a sequence of rule applications describes a counterproductive loop in the transition graph. Exploratory analysis is similarly supported and enables the human designer to discover implicit information. The methods developed in this thesis, can therefore be seen as a new approach to introduce systematic and visual analysis of design grammars in CDS. The methodology is integrated into the grammar development and application process presented in [59]. The human designer benefits from a modified process emphasizing on systematic analysis of grammar rules and search algorithms and explicitly showing potential influences of the analysis results on grammar development and application (see Figure 9-2).

9.4.2 Supporting Human Designers During Grammar Development

Three methods are presented to overcome the lack of support during grammar development [31]. GRAM (Chapter 4) allows an analysis of grammar rules independent a the search algorithm. This method can be applied to get feedback on each rule's performance and the search space the rules explore under random application. Additionally, GRAM supports debugging grammar rules which is favorable during rule development. The network-based rule analysis (Chapter 5) supports grammar development through analysis results on LHS matches of rules and beneficial or counterproductive sequences. The human designer can

use the visualizations to understand the consequences that, e.g., the combination of two rules has on the designs that can be generated by the grammar. These relationships between available rules and generated designs are also visible when using the Relation Visualization Method (Chapter 6). The transition graph can be filtered manually and rules can be disabled. This enables human designers to visualize the transition graph that results when only a subset of the available rules is used. Doing so, they can visualize the potential of each rule on exploring the search space and identify beneficial and counterproductive rules. Using these methods supports the human designer to gain important information on the rule set during development. The different methods can be used at different stages of the grammar development process, e.g. GRAM for early phases, the Network-based Rule Analysis Method once the individual rules are developed to reason about combining certain rules or changing their application conditions, and the Relation Visualization Method to reason about combining the rules with an algorithm and exploring how changes in design objectives are linked through the rules. This is, however, only a suggestion and the methods can be used individually or combined in many ways. Using the implemented software framework, even additional combinations of data generation, analysis and visualization are possible to support rule development.

9.4.3 Supporting the Selection of the Search Algorithm

When setting up a CDS method using grammars, there are dependencies between the grammar developed to describe design transformations and the optimization and search algorithm selected to guide the synthesis process. Knowing the search space that is described by a grammar and the properties of a grammar, e.g. its tendency to explore and exploit designs, allows the human designer to find an appropriate search algorithm that can successfully be combined with the grammar to generate unexpected or novel designs. While GRAM and the Network-based Rule Analysis Method focus more on analyzing and visualizing information about the grammar, the Relation Visualization Method addresses exactly this combination between grammar rules and search algorithm. It is therefore well-suited for informing the human designer which algorithms can best be applied to a search problem. For less experienced designers, it can be helpful to use the method to visualize how different search algorithms explore the search space. This gives an impression of the characteristics of the different search algorithms. Through increasing the human designer's understanding of the search algorithm's behavior when combined with the grammar rules, the designer is capable of taking a more informed decision on which search algorithm to select for a given CDS task.

9.4.4 Supporting the Refinement of the Search Process

Once the human designer has developed a set of grammar rules and selected an appropriate search algorithm, the CDS process can be started to generate design alternatives. Often the synthesis process is then tuned to increase its performance. In common approaches often advantageous algorithm parameters are identified using a trial and error approach. Using

the presented methods enables a more methodological approach to refining the search process. The unique topology and performance space plots in the Relation Visualization Method can be used to find appropriate algorithm parameter settings, e.g. a Burst length that balances between exploring new topologies and exploiting the parameter space of already explored designs. Besides finding appropriate search algorithm settings, predefined sequences of rules can be used, e.g. to speed up the synthesis process for the given problem. Those sequences can be identified using the Network-based Rule Analysis Method. This is different to approaches where information about rule sequences is gathered during the CDS process. Using the sequence analysis method, information is gained about beneficial sequences already during rule development. Search algorithms with learning can additionally be applied and the knowledge on preferable sequences gained during rule sequence analysis can be fed to the learning algorithms as initial knowledge. Finally, the human designer has to make a decision about the overarching strategy of exploring topologies and parameters of designs during the CDS process. Different strategies exist and four exemplary ones are presented in Section 7.1.2. The human designer can select an appropriate strategy when enough information exists on how the design space is explored by the search algorithm. This can be, e.g., a result of the Relation Visualization Method, where the relations between changing topologies and parameters of designs are visualized in the UTS. When the human designer is unsure about the search strategy to use for applying topologic and parametric rules, different ones can be tested and compared using the metrics presented in Section 7.1.3. To summarize, all three methods in Sections 5, 6 and 7 have aspects to support the human designer in refining the search process and can be used either individually or in combination. Advantages of the methods in Sections 5 and 6 are that the knowledge already exists when the respective methods are used to support grammar development and search algorithm selection.

9.4.5 Providing a Software Prototype to Support CDS

The software prototype, described in Chapter 8, is an additional contribution. It is used to validate the methods presented in this thesis. Besides implementing the automated analyses of the developed methods, the software prototype constitutes a generic framework for CDS using grammars implementing the generation, evaluation and guidance step of CDS. Arbitrary initial designs as graphs and grammar rules implemented in GrGen can be integrated to represent the design task. Interfaces to external simulation software enable the evaluation of synthesized designs and default search algorithms and strategies for selecting topologic and parametric rules are available to guide the search process. Due to its generic implementation, the framework can easily be extended. The software prototype can therefore be used a) as a tool to apply the developed methods, b) as a generic CDS framework once the grammar rules and search algorithm are defined, or c) as a platform to conduct further research on analyzing grammars and search algorithms.

Figure 9-3. gives an overview of how the presented methods contribute to achieve the sub-goals (see also Figure 1-1) and the overall goal of this thesis to support the human designer in CDS. The different colors and the arrows indicate which method fulfills which goals.

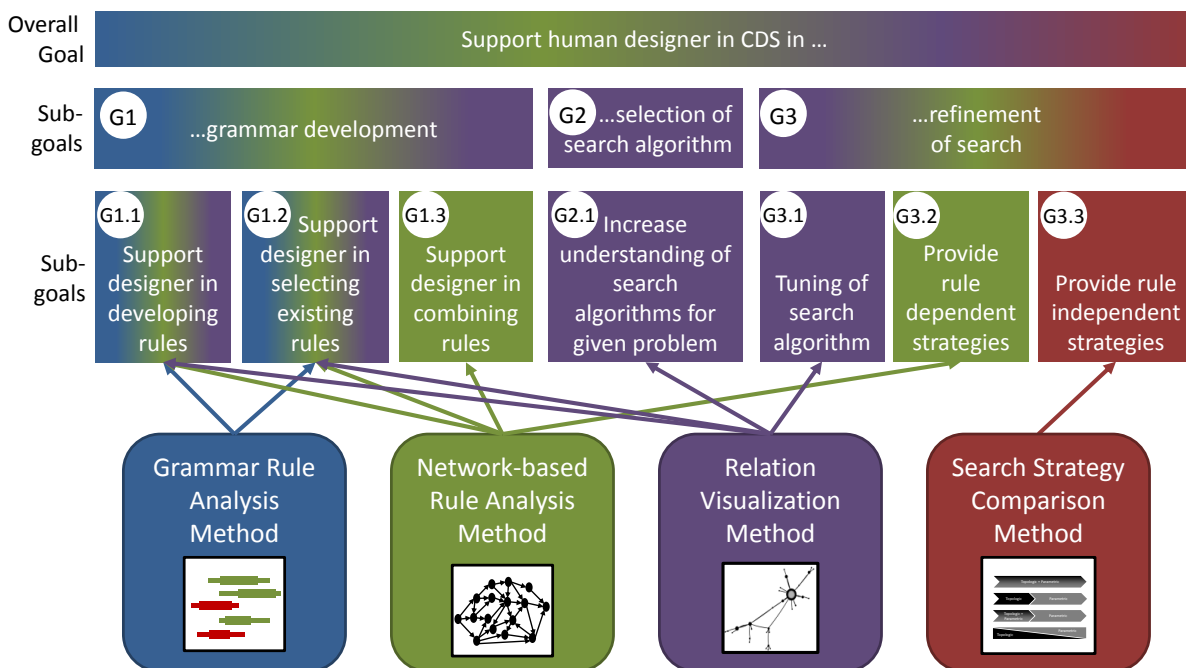


Figure 9-3 Overview of the methods and their contributions in supporting the human designer.

9.5 Limitations and Future Work

The presented methodology is a first approach to systematically analyze and visualize different aspects of grammar development and application. There are limitations to the presented methods and the presented software prototype. These along with different future directions are discussed in the following paragraphs.

An extensive **user study with human subjects** and diverse design tasks is necessary to further evaluate the presented methodology. Such a study can be conducted with different user groups where at least one group contains experts having developed grammars and applied search algorithms for CDS tasks in the past. Another group should contain novices or less experienced designers who are introduced to the concepts of CDS, grammar development and search algorithms. Ideally the different groups are compared to control groups solving the same tasks without the presented methods. Recordings of the study and interviews of the participants are expected to be helpful for further improving the methodology and the methods.

Different **criteria for evaluating grammar rules and for comparing search strategies** are presented in this thesis. Examples are the diversity and validity ratio for grammar rule sets (GRAM) or the convergence ratio and the pairwise comparisons of final Pareto archives to compare the quality of generated designs (Search Strategy Comparison Method). The criteria are developed in the style of published research in optimization, based on experience or through discussions with researchers and anonymous reviewers. They can be considered as

a starting point for further research into defining commonly accepted criteria for evaluating grammar rules as well as for comparing generated designs in an objective way. Further defining these criteria enables more objective comparisons of existing and future research on CDS methods.

The methods in this thesis present information to the human designer. Some of the information is highlighted to direct the designer to interesting or unexpected aspects of the presented information, e.g. with the color coded bar charts in GRAM. To analyze the presented information, the designer is required to have a general understanding of the CDS process and how its components, i.e. grammar rules, search algorithm, are linked. Providing the human designer with **guidelines on how to interpret the analysis results** and how changes of one aspect influence other aspects of the CDS process could further support the human designer.

Visualizing information for design tasks with more than three objectives becomes complex. Examples are the design space representations for GRAM or the performance space representations for the Relation Visualization Method. For more objectives, **advanced data visualization methods** have to be employed that allow the representation of information in many dimensions while still being straightforward to interpret by the human designer. This is a challenging task and an active area of research in information visualization.

Further improvements are possible considering the developed software prototype. In its current implementation, it supports different external software for visualization. Because of this design decision, different software can be used, but the visualizations are not linked to one another. When, for example, the human designer uses the Relation Visualization Method and animates the unique topology space in OrganicViz it would be preferable that also the performance space representation in Matlab is animated accordingly. Due to the software architecture in the current implementation, the user input is, however, only received by the software the user is operating with. Linking the external software or providing a **specific visualization tool** for the presented methods could improve the human designer's experience when interacting with the visualizations. Requirements for such a visualization tool should be derived from the user study described above.

The methods are presented as post-processing methods. This permits a generic implementation and many different combinations of data generation, analysis and visualization. Another scenario is to analyze and visualize grammar rules and their application using **live visualizations**. This requires an integration of the analysis and visualization tools in the CDS process. When live visualizations are combined with interactive search algorithms, additional benefits may arise from the collaboration of human designer and computational methods. The ability of the human designer to recognize patterns and pitfalls of the search algorithm when interpreting the live visualizations and the interactivity with the algorithm could allow for improved rule applications and design generation compared to current interactive search algorithms.

10 Conclusions

Grammatical approaches to Computational Design Synthesis (CDS) have successfully been applied to solve various engineering design tasks. These approaches support the human designer during conceptual and embodiment design by synthesizing numerous designs. While the approaches are capable of generating known but also new designs, their application is not yet widely spread. One reason for the hesitation of industry, but also researchers, is that there is only limited support of human designers when developing a CDS method using design grammars. Successfully applying a CDS method is challenging for several reasons. Problem-specific engineering knowledge has to be formulated in a model describing a design formally and grammar rules have to be developed describing possible design transformations. For a fully automated CDS process, an evaluation routine has to be provided to allow for the comparison of generated designs. Different search algorithms and search strategies can be used to steer the synthesis process. Selecting a suitable search algorithm for a given design task and grammar rules is another challenge.

The motivation for this thesis is thus to support human designers to overcome some of the challenges in setting up CDS approaches with grammars to make these approaches more easily accessible. Being aware of possible changes to grammar rules and search algorithms and their strong coupling enables the development of advanced computational methods that automatically generate meaningful or even innovative concepts for engineering design problems. A long term goal is then to enable also designers who are not familiar with CDS approaches to benefit from the advantages of these methods.

Two areas for improvement are identified from a literature review and discussions with researchers in CDS and are addressed in the thesis. First, more methodological support should be provided for the development of engineering design grammars. The lack of support in this phase is seen as one of the major drawbacks of CDS. Second, support should be provided for the application of grammar rules. This includes supporting the human designer in selecting an appropriate search algorithm and refining the search process to synthesize promising designs.

A new methodology is developed to address these issues. It incorporates aspects of formal network analysis and research on information visualization and visual analytics. The developed methodology extends the rule development and application process by an additional step in which the human designer analyzes various aspects of the CDS process. These include individual grammar rules, sequences of rule applications, search algorithms, and search strategies. According to the three step process of information visualization, data is generated, analyzed and presented to the human designer. Automated analysis and manual interaction between the human designer and the generated visualizations enable to gain information about the design task, grammar rules, the search process and dependencies between these. The methodology consists of four methods that can be applied individually or combined.

In Chapter 4, the Grammar Rule Analysis Method (GRAM) is presented that systematically analyzes developed rules and the influences they have on synthesized designs considering their objectives and characteristics. GRAM does not require the selection of a search algorithm allowing its application to support the rule development process. With GRAM the human designer is given a means to systematically analyze and reason about developed grammar rules. Criteria are defined to assess the performance of grammar rules. These enable a comparison of grammar rules and, thereby, GRAM does not only provide a more scientific approach to rule development but also facilitate the reuse of existing grammar rules since they can be assessed and compared based on defined criteria.

The Network-based Rule Analysis Method is presented in Chapter 5. This method combines the concepts of transition graphs, network analysis algorithms and interactive visualizations to support grammar rule analysis. It provides automated analyses and interactive visualizations. When the human designers interacts with these visualizations, they can manually explore information about the effect of rule applications on generated designs and identify beneficial or counterproductive rule sequences. This knowledge can then be used for grammar development or to identify advanced rule application strategies.

A method to visualize relations between grammar rules, design topologies and objectives is presented in Chapter 6. The Relation Visualization Method is inspired by research on algorithm visualization and post-processes and visualizes data generated during synthesis. The progression of the CDS process can be animated using interactive visualizations that enable an analysis of both the grammar itself and how search algorithms explore and exploit the search space. The gained information supports grammar development, the selection of an appropriate search algorithm and the refinement of the search process through suitable parameter settings of the selected search algorithm. This method is unique in its ability to visualize search algorithm progression and design space exploration in CDS and can further be used for teaching algorithms in a CDS context.

Chapter 7 presents different strategies for the topologic and parametric exploration of the search space and provides metrics to systematically compare these strategies for a given design task. The Search Strategy Comparison Method supports human designers through a defined process and metrics for evaluating the quantity and quality of synthesized designs. The human designer can reason about which search strategy to apply for a given design task. The developed metrics can further be used to compare synthesis results in a broader scope.

All presented methods are successfully validated using case studies from the mechanical engineering domain, i.e. gearbox synthesis and bicycle frame synthesis, or based on a mathematical example, a sliding tile puzzle. The case studies demonstrate how grammars and search algorithms are analyzed systematically using the presented methods and how visualizations of these analyses provide information for the human designer.

A modular software prototype for CDS is implemented to evaluate the presented methods and allows the use of any graph grammar implemented in GrGen. It provides interfaces to simulation software for design evaluation and includes several search algorithms to guide

the synthesis process. The developed software prototype can, therefore, not only be used to demonstrate the potential of the developed methodology for supporting human designers in CDS, but also for the application and search phase once the human designer has defined grammar rules and search algorithm.

Through the developed methodology, the goals of the thesis are achieved. Since research on a methodology for systematically supporting human designers in developing and applying design grammars for CDS is still in its infancy, there is, however, ample room for further research. An extensive user study analyzing how novices and CDS experts address synthesis tasks with and without the support by the presented methodology can deliver important insights for further adapting the methods and visualizations to specific requirements of designers. Considering the presented methods and the software prototype, improvements can be made by computationally linking the visualizations generated in different software systems. With a long term perspective, research should also address issues to ease the reuse and adaptation of existing grammars. Through systematic grammar rule analysis for given problems and metrics with which rules can be compared, the methods presented in this thesis are a step in this direction.

References

- [1] Königseder, C. and Shea, K. (2015). "Comparing Strategies for Topologic and Parametric Rule Application in Automated Computational Design Synthesis", *Journal of Mechanical Design*, **138**(1), pp. 011102-1 - 011102-12.
- [2] Königseder, C. and Shea, K. (2015). "A Method for Visualizing the Relations Between Grammar Rules, Performance Objectives and Search Space Exploration in Grammar-Based Computational Design Synthesis", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), Boston, MA, USA.
- [3] Königseder, C., Stanković, T., and Shea, K. (2015). "Improving Generative Grammar Development and Application Through Network Analysis Techniques", International Conference on Engineering Design (ICED), Milano, Italy.
- [4] Königseder, C. and Shea, K. (2014). "Systematic Rule Analysis of Generative Design Grammars", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **28**(3), pp. 227-238.
- [5] Königseder, C. and Shea, K. (2014). "Analyzing Generative Design Grammars", in *Design Computing and Cognition '14*, J. S. Gero and S. Hanna, eds., Springer International Publishing, pp. 363-381.
- [6] Königseder, C. and Shea, K. (2014). "Strategies for Topologic and Parametric Rule Application in Automated Design Synthesis using Graph Grammars", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), Buffalo, NY, USA.
- [7] Königseder, C. and Shea, K. (2014). "The Making of Generative Design Grammars", Workshop on Computational Making, International Conference on Design Computing and Cognition (DCC), London, UK.
- [8] Königseder, C., Shea, K., and Campbell, M. I. (2012). "Comparing a Graph-Grammar Approach to Genetic Algorithms for Computational Synthesis of PV Arrays", in *CIRP Design 2012* A. Chakrabarti, ed., Springer London, Bangalore, India, pp. 105-114.
- [9] Kumar, M., Campbell, M. I., Königseder, C., and Shea, K. (2012). "Rule Based Stochastic Tree Search", in *Design Computing and Cognition '12*, J. S. Gero, ed., Springer Netherlands, pp. 471-587.
- [10] Königseder, C. and Shea, K. (2015). "Visualizing Relations Between Grammar Rules, Objectives and Search Space Exploration in Grammar-based Computational Design Synthesis (currently under review)", *Journal of Mechanical Design*.
- [11] Königseder, C., Stankovic, T., and Shea, K. (2015). "Improving Design Grammar Development and Application Using Transition Graphs (currently under review)", *Design Science Journal*.
- [12] Simon, H. A. (1969). *The sciences of the artificial*, MIT Press.
- [13] Newell, A. and Simon, H. A. (1956). "The logic theory machine--A complex information processing system", *IRE Transactions on Information Theory*, **2**(3), pp. 61-79.

- [14] Pahl, G., Beitz, W., Feldhusen, J., and Grote, K.-H. (2007). *Engineering Design: A Systematic Approach*, Springer London.
- [15] Purcell, A. T. and Gero, J. S. (1996). "Design and other types of fixation", *Design Studies*, **17**(4), pp. 363-383.
- [16] Kurtoglu, T. and Campbell, M. I. (2009). "Automated synthesis of electromechanical design configurations from empirical analysis of function to form mapping", *Journal of Engineering Design*, **20**(1), pp. 83-104.
- [17] Schmidt, L. C. and Cagan, J. (1997). "GGREADA: A graph grammar-based machine design algorithm", *Research in Engineering Design*, **9**(4), pp. 195-213.
- [18] Wyatt, D. F., Wynn, D. C., Jarrett, J. P., and Clarkson, P. J. (2012). "Supporting product architecture design using computational design synthesis with network structure constraints", *Research in Engineering Design*, **23**(1), pp. 17-52.
- [19] Blessing, L. T. and Chakrabarti, A. (2009). *DRM, a Design Research Methodology*, Springer London.
- [20] Chakrabarti, A., Shea, K., Stone, R., Cagan, J., Campbell, M., Hernandez, N. V., and Wood, K. L. (2011). "Computer-Based Design Synthesis Research: An Overview", *Journal of Computing and Information Science in Engineering*, **11**(2), pp. 021003-1 - 021003-10.
- [21] Münzer, C., Helms, B., and Shea, K. (2013). "Automatically Transforming Object-Oriented Graph-Based Representations Into Boolean Satisfiability Problems for Computational Design Synthesis", *Journal of Mechanical Design*, **135**(10), pp. 101001-1 - 101001-13.
- [22] Antonsson, E. K. and Cagan, J. (2001). *Formal Engineering Design Synthesis*, Cambridge University Press.
- [23] Chakrabarti, A. (2002). *Engineering Design Synthesis: Understanding, Approaches and Tools*, Springer London.
- [24] Shea, K. and Starling, A. C. "From discrete structures to mechanical systems: a framework for creating performance-based parametric synthesis tools", in *Proceedings of the AAAI 2003 Symposium on Computational Synthesis: From Basic Building Blocks to High Level Functionality*, pp. 210-217.
- [25] Cagan, J., Campbell, M. I., Finger, S., and Tomiyama, T. (2005). "A framework for computational design synthesis: Model and applications", *Journal of Computing and Information Science in Engineering*, **5**(3), pp. 171-181.
- [26] Gips, J. and Stiny, G. (1980). "Production Systems and Grammars - a Uniform Characterization", *Environment and Planning B: Planning and Design*, **7**(4), pp. 399-408.
- [27] Starling, A. C. and Shea, K. (2005). "A parallel grammar for simulation-driven mechanical design synthesis", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), Long Beach, CA, USA, pp. 427-436.

- [28] Gmeiner, T. and Shea, K. (2013). "A Spatial Grammar for the Computational Design Synthesis of Vise Jaws", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), Portland, OR, USA, p. 11.
- [29] Hoisl, F. and Shea, K. (2013). "Three-Dimensional Labels: A Unified Approach to Labels for a General Spatial Grammar Interpreter", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **27**(4), pp. 359-375.
- [30] Hoisl, F. R. (2012). "Visual, Interactive 3D Spatial Grammars in CAD for Computational Design Synthesis", Dissertation, Technische Universität München, Germany.
- [31] McKay, A., Chase, S., Shea, K., and Chau, H. H. (2012). "Spatial grammar implementation: From theory to useable software", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **26**(2), pp. 143-159.
- [32] Fu, Z., Depennington, A., and Saia, A. (1993). "A Graph Grammar Approach to Feature Representation and Transformation", *International Journal of Computer Integrated Manufacturing*, **6**(1-2), pp. 137-151.
- [33] Helms, B., Eben, K., Shea, K., and Lindemann, U. (2009). "Graph Grammars - a Formal Method for Dynamic Structure Transformation", 11th International DSM Conference, Greenville, SC, USA, pp. 93-96.
- [34] Schmidt, L. C., Shetty, H., and Chase, S. C. (2000). "A Graph Grammar Approach for Structure Synthesis of Mechanisms", *Journal of Mechanical Design*, **122**(4), pp. 371-376.
- [35] Mullins, S. and Rinderle, J. (1991). "Grammatical approaches to engineering design, part I: An introduction and commentary", *Research in Engineering Design*, **2**(3), pp. 121-135.
- [36] Rinderle, J. (1991). "Grammatical approaches to engineering design, part II: Melding configuration and parametric design using attribute grammars", *Research in Engineering Design*, **2**(3), pp. 137-146.
- [37] Li, X., Schmidt, L. C., He, W., Li, L., and Qian, Y. (2004). "Transformation of an EGT Grammar: New Grammar, New Designs", *Journal of Mechanical Design*, **126**(4), pp. 753-756.
- [38] Lin, Y. S., Shea, K., Johnson, A., Coultate, J., and Pears, J. (2010). "A Method and Software Tool for Automated Gearbox Synthesis", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), San Diego, CA, USA, pp. 111-121.
- [39] Patel, J. and I., C. M. (2010). "An Approach to Automate and Optimize Concept Generation of Sheet Metal Parts by Topological and Parametric Decoupling", *Journal of Mechanical Design*, **132**(5), pp. 051001-1 - 051001-11.
- [40] Helms, B., Shea, K., and Hoisl, F. (2009). "A Framework for Computational Design Synthesis Based on Graph-Grammars and Function-Behavior-Structure", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), San Diego, CA, USA, pp. 841-851.

- [41] Kurtoglu, T. and Campbell, M. I. (2006). "A Graph Grammar Based Framework for Automated Concept Generation", International Design Conference (DESIGN), Dubrovnik, Croatia, pp. 61-68.
- [42] Stanković, T., Štorga, M., Shea, K., and Marjanovic, D. (2013). "Formal Modelling of Technical Processes and Technical Process Synthesis", *Journal of Engineering Design*, **24**(3), pp. 211-238.
- [43] Oberhauser, M., Sartorius, S., Gmeiner, T., and Shea, K. (2014). "Computational Design Synthesis of Aircraft Configurations with Shape Grammars", in *Design Computing and Cognition '14*, J. S. Gero and S. Hanna, eds., Springer International Publishing, pp. 21-39.
- [44] Stöckli, F. R. and Shea, K. (2015). "A Simulation-driven Graph Grammar Method for the Automated Synthesis of Passive Dynamic Brachiating Robots", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), Boston, MA, USA.
- [45] Pugliese, M. and Cagan, J. (2002). "Capturing a rebel: modeling the Harley-Davidson brand through a motorcycle shape grammar", *Research in Engineering Design*, **13**(3), pp. 139-156.
- [46] McCormack, J. P., Cagan, J., and Vogel, C. M. (2004). "Speaking the Buick language: capturing, understanding, and exploring brand identity with-shape grammars", *Design Studies*, **25**(1), pp. 1-29.
- [47] Sumbul, A. and Chase, S. (2006). "Grammar Representations to Facilitate Style Innovation - An Example From Mobile Phone Design", eCAADe Conference, Volos, Greece.
- [48] Siddique, Z. and Rosen, D. W. (1999). "Product platform design: a graph grammar approach", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), Las Vegas, NV, USA.
- [49] Agarwal, M. and Cagan, J. (1998). "A blend of different tastes: the language of coffeemakers", *Environment and Planning B: Planning and Design*, **25**(2), pp. 205-226.
- [50] Geiß, R., Batz, G., Grund, D., Hack, S., and Szalkowski, A. (2006). "GrGen: A Fast SPO-Based Graph Rewriting Tool", in *Graph Transformations*, A. Corradini, et al., eds., Springer Berlin Heidelberg, pp. 383-397.
- [51] Jakumeit, E., Buchwald, S., and Kroll, M. (2010). "GrGen.NET", *International Journal on Software Tools for Technology Transfer*, **12**(3-4), pp. 263-271.
- [52] Helms, B. and Shea, K. (2012). "Computational Synthesis of Product Architectures Based on Object-Oriented Graph Grammars", *Journal of Mechanical Design*, **134**(2), pp. 021008-1 - 021008-14.
- [53] Helms, B. (2013). "Object-Oriented Graph Grammars for Computational Design Synthesis", Dissertation, Technische Universität München, Germany.
- [54] Alber, R. and Rudolph, S. (2003). "'43'—A Generic Approach for Engineering Design Grammars", AAAI Spring Symposium 'Computational Synthesis', Stanford, CA, USA.

- [55] Hoisl, F. and Shea, K. (2011). "An interactive, visual approach to developing and applying parametric three-dimensional spatial grammars", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **25**(4), pp. 333-356.
- [56] Chau, H., Chen, X., McKay, A., and de Pennington, A. (2004). "Evaluation of a 3D Shape Grammar Implementation", in *Design Computing and Cognition '04*, J. Gero, ed., Springer Netherlands, pp. 357-376.
- [57] Grasl, T. and Economou, A. (2011). "GRAPE: using graph grammars to implement shape grammars", Symposium on Simulation for Architecture and Urban Design, Society for Computer Simulation International, Boston, MA, USA, pp. 21-28.
- [58] Grasl, T. and Economou, A. (2013). "From topologies to shapes: parametric shape grammars implemented by graphs", *Environment and Planning B: Planning and Design*, **40**(5), pp. 905-922.
- [59] Chase, S. C. (2002). "A model for user interaction in grammar-based design systems", *Automation in Construction*, **11**(2), pp. 161-172.
- [60] Klint, P., Lämmel, R., and Verhoef, C. (2005). "Toward an engineering discipline for grammarware", *ACM Transactions on Software Engineering Methodology*, **14**(3), pp. 331-380.
- [61] Zheng, L. and Chen, H. (2009). "A Systematic Framework for Grammar Testing", International Conference on Computer and Information Science, Shanghai, China, pp. 1013-1019.
- [62] Knight, T. W. (1998). "Designing a shape grammar", in *Artificial Intelligence in Design '98*, J. S. Gero and F. Sudweeks, eds., Springer Netherlands, pp. 499-516.
- [63] Brown, K. (1997). "Grammatical design", *Intelligent Systems and their Applications*, **12**(2), pp. 27-33.
- [64] Knight, T. and Stiny, G. (2001). "Classical and non-classical computation", *arq: Architectural Research Quarterly*, **5**(4), pp. 355-372.
- [65] Ibrahim, M. S., Bridges, A., Chase, S. C., Bayoumi, S. H., and Taha, D. S. (2012). "Design grammars as evaluation tools in the first year studio", *Journal of Information Technology in Construction*, **17**(Special Issue CAAD and innovation), pp. 319-332.
- [66] Rudolph, S. (2006). "Semantic validation scheme for graph grammars", in *Design Computing and Cognition '06*, J. Gero, ed., Springer Netherlands, pp. 541-560.
- [67] Cagan, J. (2001). "Engineering Shape Grammars", in *Formal Engineering Design Synthesis*, E. K. Antonsson and J. Cagan, eds., Cambridge University Press, pp. 65-92.
- [68] Shea, K. and Cagan, J. (1999). "Languages and semantics of grammatical discrete structures", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **13**(04), pp. 241-251.
- [69] Orsborn, S., Cagan, J., and Boatwright, P. (2008). "Automating the Creation of Shape Grammar Rules", in *Design Computing and Cognition '08*, J. S. Gero and A. K. Goel, eds., Springer Netherlands, pp. 3-22.
- [70] Orsborn, S., Cagan, J., and Boatwright, P. (2008). "A methodology for creating a statistically derived shape grammar composed of non-obvious shape chunks", *Research in Engineering Design*, **18**(4), pp. 181-196.

- [71] Klein, D. and Manning, C. D. (2002). "A generative constituent-context model for improved grammar induction", Annual Meeting on Association for Computational Linguistics, Association for Computational Linguistics, Philadelphia, PA, USA, pp. 128-135.
- [72] Shea, K. (2004). "Explorations in using an Aperiodic Spatial Tiling as a Design Generator", in *Design Computing and Cognition '04*, J. Gero, ed., Springer Netherlands, pp. 137-156.
- [73] Chase, S. and Liew, P. (2001). "A framework for redesign using FBS models and grammar adaptation", in *Computer Aided Architectural Design Futures 2001*, B. de Vries, et al., eds., Springer Netherlands, pp. 467-477.
- [74] Oster, A. and McCormack, J. (2011). "A Methodology for Creating Shape Rules During Product Design", *Journal of Mechanical Design*, **133**(6), pp. 061007-1 - 061007-12.
- [75] Knight, T. W. (1994). *Transformations in Design: A Formal Approach to Stylistic Change and Innovation in the Visual Arts*, Cambridge University Press.
- [76] Chase, S. C. and Ahmad, S. (2005). "Grammar transformations: using composite grammars to understand hybridity in design", CAAD futures, Vienna, Austria.
- [77] Al-kazzaz, D. A. and Bridges, A. H. (2012). "A framework for adaptation in shape grammars", *Design Studies*, **33**(4), pp. 342-356.
- [78] Gips, J. (1999). "Computer Implementation of Shape Grammars", NFS/MIT Workshop on Shape Computation.
- [79] Shea, K., Cagan, J., and Fenves, S. J. (1997). "A shape annealing approach to optimal truss design with dynamic grouping of members", *Journal of Mechanical Design*, **119**(3), pp. 388-394.
- [80] Campbell, M. I., Cagan, J., and Kotovsky, K. (2003). "The A-Design approach to managing automated design synthesis", *Research in Engineering Design*, **14**(1), pp. 12-24.
- [81] Campbell, M. I., Cagan, J., and Kotovsky, K. (1999). "A-design: An agent-based approach to conceptual design in a dynamic environment", *Research in Engineering Design*, **11**(3), pp. 172-192.
- [82] Stanković, T., Shea, K., Štorga, M., and Marjanović, D. (2009). "Grammatical evolution of technical processes", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), American Society of Mechanical Engineers, San Diego, CA, USA, pp. 895-904.
- [83] Landry, L. H. and Cagan, J. (2011). "Protocol-Based Multi-Agent Systems: Examining the Effect of Diversity, Dynamism, and Cooperation in Heuristic Optimization Approaches", *Journal of Mechanical Design*, **133**(2), pp. 021001-1 - 021001-11.
- [84] Vale, C. A. W. and Shea, K. (2003). "A Machine Learning-Based Approach To Accelerating Computational Design Synthesis", International Conference on Engineering Design (ICED), Stockholm, Sweden.
- [85] Bolognini, F., Shea, K., Vale, C. A. W., and Seshia, A. A. (2006). "A Multicriteria System-Based Method for Simulation-Driven Design Synthesis", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), Philadelphia, PA, USA, pp. 651-661.

-
- [86] Schotborgh, W. O. (2009). "Knowledge engineering for design automation", Dissertation, University of Twente, Netherlands.
- [87] Poppa, K. R., Stone, R. B., and Orsborn, S. (2010). "Exploring Automated Concept Generator Output Through Principal Component Analysis", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), Montreal, Canada, pp. 185-192.
- [88] Deb, K. (2014). "Multi-objective Optimization", in *Search Methodologies*, E. K. Burke and G. Kendall, eds., Springer US, pp. 403-449.
- [89] Marler, R. T. and Arora, J. S. (2004). "Survey of multi-objective optimization methods for engineering", *Structural and Multidisciplinary Optimization*, **26**(6), pp. 369-395.
- [90] Grune, D., van Reeuwijk, K., Bal, H. E., Jacobs, C. J., and Langendoen, K. (2012). *Modern compiler design*, Springer New York.
- [91] Aho, A. V., Lam, M. S., Sethi, R., and Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools*, Addison-Wesley Longman Publishing Co., Inc.
- [92] Hopcroft, J. E. and Ullman, J. D. (1969). *Formal languages and their relation to automata*, Addison-Wesley Longman Publishing Co., Inc.
- [93] Fekete, J.-D., van Wijk, J., Stasko, J., and North, C. (2008). "The Value of Information Visualization", in *Information Visualization*, A. Kerren, et al., eds., Springer Berlin Heidelberg, pp. 1-18.
- [94] Anscombe, F. J. (1973). "Graphs in statistical analysis", *The American Statistician*, **27**(1), pp. 17-21.
- [95] Card, S. K., Mackinlay, J. D., and Shneiderman, B. (1999). *Readings in information visualization: using vision to think*, Morgan Kaufmann Publishers Inc.
- [96] Keim, D., Mansmann, F., Schneidewind, J., Thomas, J., and Ziegler, H. (2008). "Visual Analytics: Scope and Challenges", in *Visual Data Mining*, S. Simoff, et al., eds., Springer Berlin Heidelberg, pp. 76-90.
- [97] Nagel, H. (2006). "Scientific visualization versus information visualization", Workshop on state-of-the-art in scientific and parallel computing, Sweden.
- [98] Keim, D., Mansmann, F., Stoffel, A., and Ziegler, H. (2009). "Visual Analytics", in *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, eds., Springer US, pp. 3341-3346.
- [99] Thomas, J. and Cook, K. (2005). "Illuminating the path: the research and development agenda for visual analytics", report by Pacific Northwest National Laboratory (PNNL), Richland, WA, USA.
- [100] Keim, D., Andrienko, G., Fekete, J.-D., Görg, C., Kohlhammer, J., and Melançon, G. (2008). "Visual Analytics: Definition, Process, and Challenges", in *Information Visualization*, A. Kerren, et al., eds., Springer Berlin Heidelberg, pp. 154-175.
- [101] "Bell telephone laboratories low-level linked list language" (1966). Film by: Knowlton, K., (16-minute black and white film).
- [102] "Sorting out sorting" (1981). Film by: Baecker, R., Media Centre Production, University of Toronto (30 minutes).

- [103] Brown, M. H. and Sedgewick, R. (1985). "Techniques for Algorithm Animation", *IEEE Software*, **2**(1), pp. 28-39.
- [104] Price, B. A., Baecker, R. M., and Small, I. S. (1993). "A Principled Taxonomy of Software Visualization", *Journal of Visual Languages and Computing*, **4**(3), pp. 211-266.
- [105] Kerren, A. and Stasko, J. T. (2002). "Algorithm animation", in *Software Visualization*, Springer Berlin-Heidelberg, pp. 1-15.
- [106] Hundhausen, C. D., Douglas, S. A., and Stasko, J. T. (2002). "A meta-study of algorithm visualization effectiveness", *Journal of Visual Languages and Computing*, **13**(3), pp. 259-290.
- [107] Messac, A. and Chen, X. (2000). "Visualizing the optimization process in real-time using physical programming", *Engineering Optimization*, **32**(6), pp. 721-747.
- [108] Diehl, S. (2007). *Software visualization: visualizing the structure, behaviour, and evolution of software*, Springer Berlin Heidelberg.
- [109] Suppakitnarm, A., Seffen, K., Parks, G., Connor, A., and Clarkson, P. (1999). "Multiobjective optimisation of bicycle frames using simulated annealing", Conference on Engineering Design Optimization, Ilkley, UK, pp. 357 - 364.
- [110] Campbell, M. I., Rai, R., and Kurtoglu, T. (2012). "A Stochastic Tree-Search Algorithm for Generative Grammars", *Journal of Computing and Information Science in Engineering*, **12**(3), pp. 031006-1 - 031006-11.
- [111] Shea, K. (1997). "Essays of Discrete Structures: Purposeful Design of Grammatical Structures by Directed Stochastic Search", Dissertation, Carnegie Mellon University, USA.
- [112] Wyatt, D. F., Wynn, D. C., and Clarkson, P. J. (2014). "A Scheme for Numerical Representation of Graph Structures in Engineering Design", *Journal of Mechanical Design*, **136**(1), pp. 011010-1 - 011010-13.
- [113] Plaisant, C. (2004). "The challenge of Information Visualization Evaluation", Working Conference on Advanced Visual Interfaces, ACM, Gallipoli, Italy, pp. 109-116.
- [114] Seriai, A., Benomar, O., Cerat, B., and Sahraoui, H. (2014). "Validation of Software Visualization Tools: A Systematic Mapping Study", IEEE Working Conference on Software Visualization, pp. 60-69.
- [115] McConathy, D. A. (1993). "Evaluation methods in visualization: combating the Emperor's new clothes phenomenon", *ACM SIGBIO Newsletter*, **13**(1), pp. 2-8.
- [116] Sensalire, M., Ogao, P., and Telea, A. (2009). "Evaluation of Software Visualization Tools: Lessons Learned", *IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pp. 19-26.
- [117] Bracewell, R. H., Shea, K., Langdon, L. T. M., Blessing, L. T. M., and Clarkson, P. J. (2001). "A Methodology For Computational Design Tool Research", International Conference on Engineering Design (ICED), Glasgow, UK.
- [118] Chomsky, N. (1956). "Three models for the description of language", *IRE Transactions on Information Theory*, **2**(3), pp. 113-124.

- [119] Li, X. and Schmidt, L. (2004). "Grammar-Based Designer Assistance Tool for Epicyclic Gear Trains", *Journal of Mechanical Design*, **126**(5), pp. 895-902.
- [120] Starling, A. C. (2004). "Performance-based computational synthesis of parametric mechanical systems", Dissertation, University of Cambridge, UK.
- [121] Swantner, A. and Campbell, M. I. (2012). "Topological and parametric optimization of gear trains", *Engineering Optimization*, **44**(11), pp. 1351-1368.
- [122] Pomrehn, L. P. and Papalambros, P. Y. (1995). "Discrete optimal design formulations with-application to gear train design", *Journal of Mechanical Design*, **117**(3), pp. 419-424.
- [123] Lohse, G. L., Min, D. W., and Olson, J. R. (1995). "Cognitive Evaluation of System Representation Diagrams", *Information and Management*, **29**(2), pp. 79-94.
- [124] Stanković, T., Štorga, M., Stojić, I., and Savšek, T. (2012). "Tracability Visualization Toolkit", International Design Conference (DESIGN), Dubrovnik, Croatia.
- [125] "yComp", <http://www.info.uni-karlsruhe.de/software.php/id=6>, Kroll, M., Beck, M., Geiß, R., Hack, S., and Leiß, P. (last accessed: July 17, 2015).
- [126] "yWorks", <http://www.yworks.com> (last accessed: July 17, 2015).
- [127] Kim, H. M., Michelena, N. F., Papalambros, P. Y., and Jiang, T. (2003). "Target cascading in optimal system design", *Journal of Mechanical Design*, **125**(3), pp. 474-480.
- [128] Barbati, M., Bruno, G., and Genovese, A. (2012). "Applications of agent-based models for optimization problems: A literature review", *Expert Systems with Applications*, **39**(5), pp. 6020-6028.
- [129] Slocum, J. and Sonneveld, D. (2006). *The 15 Puzzle: How It Drove the World Crazy. The Puzzle that Started the Craze of 1880. How America's Greatest Puzzle Designer, Sam Loyd, Fooled Everyone for 115 Years*, Slocum Puzzle Foundation.
- [130] Johnson, W. W. and Story, W. E. (1879). "Notes on the '15' Puzzle", *American Journal of Mathematics*, **2**(4), pp. 397-404.
- [131] Karavirta, V., Korhonen, A., Malmi, L., and Naps, T. (2010). "A comprehensive taxonomy of algorithm animation languages", *Journal of Visual Languages and Computing*, **21**(1), pp. 1-22.
- [132] Cash, P., Stanković, T., and Štorga, M. (2014). "Using visual information analysis to explore complex patterns in the activity of designers", *Design Studies*, **35**(1), pp. 1-28.
- [133] Suppakitnarm, A., Parks, G. T., Shea, K., and Clarkson, P. J. (2004). "Conceptual design of bicycle frames by multiobjective shape annealing", *Engineering Optimization*, **36**(2), pp. 165-188.
- [134] Covill, D., Begg, S., Elton, E., Milne, M., Morris, R., and Katz, T. (2014). "Parametric Finite Element Analysis of Bicycle Frame Geometries", *Procedia Engineering*, **72**(0), pp. 441-446.
- [135] Vale, C. A. W. (2002). "Multiobjective Dynamic Synthesis via Machine Learning", Dissertation, University of Cambridge, UK.

- [136] Bolognini, F. (2008). "An Integrated Simulation-based Generative Design Method for Microelectromechanical Systems", Dissertation, University of Cambridge, UK.
- [137] Mattson, C. A., Mullur, A. A., and Messac, A. (2004). "Smart Pareto filter: obtaining a minimal representation of multiobjective design space", *Engineering Optimization*, **36**(6), pp. 721-740.
- [138] Campbell, M. I., Rai, R., and Kurtoglu, T. (2009). "A Stochastic Graph Grammar Algorithm for Interactive Search", International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (ASME IDETC), San Diego, CA, USA, pp. 829-840.
- [139] Zitzler, E., Deb, K., and Thiele, L. (2000). "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results", *Evolutionary computation*, **8**(2), pp. 173-195.
- [140] Raphael, B. and Smith, I. F. (2003). *Fundamentals of computer-aided engineering*, John Wiley & Sons.