Philipps-Universität Marburg
Reference Number: 1180

Department of Mathematics
and Computer Science

Model-Driven Optimization with a Focus on the Effectiveness and Efficiency of Evolutionary Algorithms

Dissertation

in attainment of the degree of Doctor of Natural Sciences

Master of Science
Stefan Wolfgang John
born in Burghausen

1. Reviewer: Prof. Dr. Gabriele Taentzer
2. Reviewer: Prof. Dr. Matthias Tichy

Submitted: 11 August 2023
Defended: 11 October 2023
Published: Marburg, 2023

Abstract

Optimization problems are ubiquitous in software engineering. They arise, for example, when searching for a modular software design or planning a cost-efficient development process. Search-based software engineering (SBSE) is concerned with solving optimization problems by applying search-based algorithms. Among the most popular are evolutionary algorithms, which are the focus of this thesis. Following the example of natural evolution, they use selection, mutation, and crossover operators to evolve existing solutions.

In the hope of enabling the use of SBSE without optimization expertise, model-driven optimization (MDO) relies on model-driven engineering (MDE); models and model transformations are used to specify optimization problems and solution algorithms. Two ways of representing solutions, the model-based approach (MB-MDO) and the rule-based approach, have established. However, the implications of choosing one over the other are not clear. Therefore, we compare both approaches qualitatively and quantitatively and pursue MB-MDO as the more promising approach in the rest of the thesis.

How to design efficient and effective evolutionary algorithms is a central question in MB-MDO. Moreover, how to perform crossover there is not yet known. We first present a framework that highlights and explains the core concepts of evolutionary algorithms in MB-MDO and formalizes them based on graph transformation theory. It not only contributes to the understanding of evolutionary algorithms in MB-MDO, but in particular facilitates their precise specification, analysis, and evaluation. The framework is used to define important properties of mutation operators and to evaluate their impact on the efficiency and effectiveness of evolutionary algorithms. Furthermore, a general, graph-based approach for the construction of crossover operators in MB-MDO is presented. The general approach is also concretized for the Eclipse Modeling Framework (EMF). Finally, an evaluation of a prototypical implementation shows the relevance of crossover operators for evolutionary algorithms in MB-MDO.

Zusammenfassung

Optimierungsprobleme sind in der Softwareentwicklung allgegenwärtig. Sie treten beispielsweise bei der Suche nach einem modularen Softwaredesign oder der Planung eines kosteneffizienten Entwicklungsprozesses auf. Die suchbasierte Softwaretechnik (SBSE, von engl. search-based software engineering) befasst sich mit der Lösung von Optimierungsproblemen durch Anwendung suchbasierter Optimierungsverfahren. Zu den beliebtesten gehören evolutionäre Algorithmen, die im Mittelpunkt dieser Arbeit stehen. Angelehnt an die natürliche Evolution werden bei diesen Selektions-, Mutations- und Kreuzungoperatoren verwendet, um bestehende Lösungen weiterzuentwickeln.

In der Hoffnung, den Einsatz von SBSE ohne tiefere Optimierungskenntnisse zu ermöglichen, setzt die modellgetriebene Optimierung (MDO, von engl. model-driven optimization) auf modellgetriebene Entwicklung (MDE, von engl. model-driven engineering); Modelle und Modelltransformationen werden zur Spezifikation von Optimierungsproblemen und Lösungsalgorithmen herangezogen. Dabei haben sich zwei Arten Lösungen zu repräsentieren etabliert, der modellbasierte (MB-MDO, von engl. model-based MDO) und der regelbasierte Ansatz. Es ist jedoch nicht klar, welche Auswirkungen die Entscheidung für den einen oder anderen Ansatz hat. Daher vergleichen wir beide Ansätze sowohl qualitativ als auch quantitativ und verfolgen im weiteren Verlauf der Arbeit MB-MDO als den vielversprechenderen Ansatz.

Wie sich effiziente und effektive evolutionäre Algorithmen entwickeln lassen, ist in MB-MDO eine zentrale Frage. Darüberhinaus ist dort noch kein Ansatz zur Umsetzung von Kreuzungsoperatoren bekannt. Wir stellen zunächst ein Framework vor, welches die Kernkonzepte evolutionärer Algorithmen in MB-MDO herausstellt, erklärt und diese auf Grundlage der Graphentransformationstheorie formalisiert. Es trägt damit nicht nur zum Verständnis evolutionärer Algorithmen in MB-MDO bei, sondern ermöglicht insbesondere deren präsize Spezifikation, Analyse und Evaluation. Mit Hilfe des Frameworks werden zwei wichtige Eigenschaften von Mutationsoperatoren definiert und deren Einfluss auf die Effizienz und Effektivität evolutionärer Algorithmen evaluiert. Desweiteren wird

ein genereller, graphbasierter Ansatz zur Konstruktion von Kreuzungsoperatoren in MB-MDO vorgestellt. Der generelle Ansatz wird zudem für das Eclipse Modeling Framework (EMF) konkretisiert. Die Evaluation einer prototypischen Implementierung zeigt abschließend die Relevanz von Kreuzungsoperatoren für evolutionäre Algorithmen in MB-MDO.

Contents

"Modeling in its broadest sense is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer, or cheaper than reality instead of reality for some purpose. A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger, and irreversibility of reality."

Rothenberg et al.: The Nature of Modeling (1989)

1 Introduction

As early as 1956 George A. Miller, an American psychologist, was among the first to investigate capacity limits in human information processing [Mil56]. He observed that even in simple classification and counting tasks our short-term memory can only handle a very small number of items before we start to make mistakes. It is surely debatable whether a single number can sufficiently describe the capacity of something as complex as our short-term memory [MHB14]. Still, considering the complexity of systems and technologies we have to deal with, the order of magnitude of what Miller found to be the limit for the number of items we can process correctly is highly relevant; it is 1!

Problems faced in software engineering as well as in other engineering disciplines are utterly complex nowadays. Typically, thousands of lines of code are scattered over myriads of artifacts distributed among various places. Understanding such systems solely on the source code level is a hopeless task. Thus, a key element to comprehension is abstraction [Kra07]. Looking at complex systems from different perspectives and at different levels of detail allows us to process information in chunks closer, with regard to their number of elements, to Miller's observed capacity limits. According to Rothenberg et al. [Rot+89], models are a natural means to handle abstraction and, consequently, play a vital role in today's software development.

Introduced over two decades ago, a paradigm called *model-driven engineering (MDE)* aims at leveraging the power of abstraction by treating *everything as a model* [Béz04]. It promotes a change in the role of models: from a sheer means of documentation to first-class entities participating (along with model transformations) in the configuration, production, and execution of systems. Practicing MDE by embracing models throughout the development process is expected to increase comprehension, productivity, and in consequence software quality. Recent research shows that MDE can live up to these expectations but its adoption can be challenging [Moh+13; HWR14; Lie+14; Buc+20].

1.1 Motivation

In the last years a new domain called *model-driven optimization (MDO)* has emerged, applying MDE to the tasks of specifying and solving optimization problems. Optimization problems lie at the heart of many software engineering tasks – architecture design, release planning, resource allocation, and refactoring [HMZ12], to name a few. Solving such problems with search-based algorithms is subject of the well established domain of *search-based software engineering (SBSE)* [HJ01]. Developing an effective and efficient search-based algorithm for a problem at hand is anything but trivial and requires expertise in the domain of interest as well as in SBSE. In particular, considerable effort has to be put into finding a suitable *representation* of solutions within the algorithm, as well as into developing fitting *search operators* that can effectively and efficiently be used to explore the search space of possible solutions.

Hierarchies, dependencies, constraints, and other relations between the objects of interest regularly need to be considered in software engineering problems. As discussed by Burton and Poulding on the example of planning the next release of a software product [BP13], expressing such complex structures in traditional representations (e.g., strings of integers or real numbers) can be cumbersome and lead to unwieldy representations which are hard to comprehend and maintain. Often the complexity needs to be hidden in implicit assumptions about the data structure chosen for the representation. Developing search operators respecting these assumptions becomes an error prone task. Furthermore, the implementation

of a representation and its associated search operators typically needs to take place on a technical level (e.g., a specific programming language) making this step even harder for domain experts not proficient in programming.

Advocating models and model transformations as the central components of search-based optimization, MDO seeks to remedy these problems. Among domain experts and software engineers alike, models are already an established means to discuss domain concepts. Consequently, founding the representation of optimization problems on models and model transformations can be considered a natural choice. SBSE becomes more accessible to both domain experts and software engineers who can now formulate optimization problems and develop search-based algorithms using the language and tools they are familiar with. Complex structural requirements can be made explicit facilitating their adherence and validation. Overall, a tighter integration of search-based optimization into software development could be achieved. Modeling artifacts naturally emerging from development processes, e.g., architecture or class diagrams, can be reused and potentially become subject to optimization. Considering projects applying MDE, where many parts of a software system are already captured by or generated from models and model transformations, a prospect of Harman et al. [Har+12] might become reality: SBSE as a central aspect of software development.

Because of their simplicity and flexibility [Fog97], evolutionary algorithms have become the most popular optimization technique in SBSE [HMZ12] and are currently in the focus of MDO. Inspired by nature, a search is typically performed on a population of solutions. Two representations of solutions are distinguished. Changes are performed on their *genotype* which represents solutions within an evolutionary algorithm. The *phenotype* constitutes their external representation and is used to evaluate their quality. To evolve a population, search operators (called *evolutionary operators* in this context) mimic the biological concepts of *mutation* and *crossover* (also known as *recombination*) to create offspring. Selection steps realize survival of the fittest and favor good solutions to evolve towards better populations. Evolution takes place iteratively by generating new populations until some desired termination condition (e.g., a solution with acceptable quality is found) is met. While mutation is meant to induce small changes in a single solution, crossover recombines the information of multiple solutions. The realization of these concepts strongly depends on the *encoding* of

an optimization problem, i.e., the mapping between its phenotypes and genotypes. For example, in traditional string-based encodings, mutation typically swaps the value at one or multiple positions in a solution string. Crossover cuts parent strings apart and recombines the created substrings. When solution strings capture complex structures (as discussed above), the application of evolutionary operators often produces offspring violating structural constraints. Intricate repair steps are needed to restore solutions meeting these constraints. In contrast, with models and model transformations structural constraints may already be considered explicitly by evolutionary operators.

In the last years, different realizations of MDO have been proposed and developed. In particular, two approaches on how to use models and model transformations to encode solutions have established. The *model-based approach (MB-MDO)* encodes solutions directly as models and model transformations are used to gather new solutions [Bur+12; BP13; KLW13; ZM16; Str17; BZS18; Hor+22]. In contrast, the *rule-based approach (RB-MDO)* encodes each solution as a sequence of rule calls, where a rule call constitutes a model transformation rule together with the parameters required for its application [Abd+14; FTW15; Bil+19]. For both approaches, how to design efficient and effective evolutionary algorithms and, in particular, their evolutionary operators is an open research topic.

1.2 Problem Statement

Both encoding approaches in MDO have gained similar attention in literature and are backed by a solid tool environment to facilitate further research. Up to now, however, both approaches have been developed and investigated independently from each other. No work has yet compared their features and limitations or compared them with regard to their optimization performance. As their advantages and drawbacks remain unclear, users need to decide for one approach or the other based on their intuition rather than making an informed decision. Also from a research perspective, this knowledge gap constitutes a problem as it becomes hard to estimate, which approach might be the more promising one to pursue. Consequently, our first contribution tries to close this gap by answering the following research questions:

RQ1 What are the conceptual advantages and drawbacks of each encoding approach in MDO?

RQ2 How do both encoding approaches in MDO compare with regard to their optimization effectiveness and efficiency?

The insights gathered from the comparison build the foundation for our subsequent focus on MB-MDO. Without a clear specification, many concepts involved in the application of evolutionary algorithms in MB-MDO remain implicit; ambiguities and misunderstandings can be the result. Furthermore, in the absence of a formal basis, the formulation and precise reasoning about properties of evolutionary operators and evolutionary algorithms becomes difficult; only limited research on such properties can be found so far [BZJ21]. Without a deeper understanding of the intricacies of MB-MDO, however, finding an effective and efficient algorithm for a specific problem resorts to trial and error. A formalization may provide the necessary means (e.g., by its underlying theory) for the development and analysis of new operator concepts. We argue that a formal framework is needed to systematically investigate MB-MDO and address the following questions:

RQ3 How can the key concepts of evolutionary algorithms in MB-MDO be specified in a precise yet general manner that captures the large variety of evolutionary algorithms found in practice?

RQ4 How can experiments in MB-MDO be specified in a precise and reproducible way?

RQ5 Which properties of evolutionary operators are important for the effectiveness and efficiency of evolutionary algorithms in MB-MDO?

While in RB-MDO, mutation and crossover operators developed for the traditional string encoding can be adopted seamlessly, we find the most obvious limitation of MB-MDO to be the lack of a general concept for how to perform crossover on models. In evolutionary algorithms crossover is meant to create new solutions to a problem by recombining already well optimized parts of existing solutions. It is expected to speed up the search process and help overcome local optima. Crossover has often been shown to be practically beneficial for specific problems (e.g., [HK18; APS20]). Recently, its benefit for general optimization problems has also been proven theoretically [DHK12; PD18]. As models can be structurally complex and typically underlie structural constraints,

it is yet unknown how a recombination of models can be done in MB-MDO. This shortcoming raises two questions:

RQ6 How can crossover be performed in MB-MDO?
RQ7 Can crossover be beneficial for the effectiveness and efficiency of evolutionary algorithms in MB-MDO?

Obviously, RQ7 depends on finding one or multiple solutions for RQ6. We also realize that the absence of a crossover operator in MB-MDO may be favored by the lack of a formal foundation for MB-MDO (and MDO in general). Thus, we are confident that tackeling RQ3 may also greatly contribute to the development of crossover operators.

1.3 Contributions

The following contributions are made by this thesis to answer the research questions, tackle the aforementioned shortcomings, and deepen the understanding of MDO. Hereby, underpinned by their dominant usage in SBSE and MDO, we focus on evolutionary algorithms as the search-based optimization technique. Both encoding approaches, RB-MDO and MB-MDO, are compared both qualitatively and quantitatively. Backed by the outcome of this comparison, the more promising model-based approach is focused in the rest of the thesis. We present a formal framework for MB-MDO based on graph transformation theory which relies on sets of graph transformation rules to perform mutation. In that context, *soundness* and *completeness* are defined as properties of such sets. An empirical study affirms the importance of both properties for the design of efficient and effective evolutionary algorithms in MB-MDO. Based on our framework we also extend MB-MDO by proposing a generic construction of crossover operators for graph-like structures. Finally, we provide an instantiation and prototypical implementation of this formal crossover approach for the Eclipse Modeling Framework (EMF) [Ste+08; emf] and empirically show its appropriateness.

The contributions of this work have been published as journal and conference papers. The contents of the publications are included in mostly their original form in the following chapters; mild adaptions have been made to consolidate

the work and to enhance readability. In particular, references to the publications were substituted by cross-references within the thesis. The appendices of the individual publications are included as a comprehensive appendix at the end of the thesis in the order of the chapters to which they belong. In the following, the contributing publications are summarized. For each publication the authors, the journal/conference in which it was published, the contributions of the thesis author to the respective work, and the targeted research questions are listed.

1.3.1 Searching for Optimal Models: Comparing Two Encoding Approaches [Joh+19a]

Publication details

Authors: Stefan John, Alexandru Burdusel, Robert Bill, Daniel Strüber, Gabriele Taentzer, Steffen Zschaler, Manuel Wimmer
Appeared in: Journal of Object Technology, Volume 18, Number 3 (2019)
Presented at: 12th International Conference on Model Transformations (ICMT)
Contributions: The thesis author acted as the lead of the comparison project. He largely contributed to the design of the comparison, performing the qualitative analysis, the design and preparation of the quantitative experiments, and the interpretation of the results. The paper was co-written by all authors, with the thesis author contributing major parts to the following sections: Optimization Problems, Model-driven Optimization, Qualitative Comparison, Quantitative Comparison, Discussion, Threats to Validity, and Conclusion.
Targeted research questions: RQ1, RQ2

In MDO the two approaches on how to encode solutions, RB-MDO and MB-MDO, have been developed and investigated independently from each other. So far, apart from being model-driven, i.e., based on models and model transformations, the relation between them remains unclear. Our contribution addresses this lack of knowledge and provides a systematic comparison of both approaches.

To get a deeper insight into their peculiarities, both approaches are first compared from a qualitative perspective. Their differences and similarities as well as the implications thereof are discussed. Most prominently, the analysis reveals that the representation of solutions as rule-call sequences (RB-MDO) grants more flexibility in the choice and configuration of evolutionary operators. However,

maintaining the validity of such sequences also comes with an overhead and can have negative effects on an optimization. Especially dependencies between rule calls can pose a problem which cannot easily be overcome.

Based on an evaluation, the approaches are also compared quantitatively. To that end, we considered three optimization problems from the software engineering domain: the class responsibility assignment problem (CRA) [BBL10], the next release problem (NRP) [BRW01], and a refactoring use case (REF) [LK13]. We observe that MB-MDO in most cases outperforms RB-MDO, especially for large models. Furthermore, the maybe most substantial feature of RB-MDO, the possibility to apply crossover, shows to be detrimental for the quality of resulting solutions. We discuss possible reasons for these results and relate our findings to the properties we identified in the qualitative analysis for both approaches.

From these results we consider MB-MDO the more promising approach to tackle optimization problems with MDO. All the more, as the future addition of a crossover operator might unfold untapped potential. Consequently, the comparative study constitutes the basis for our decision to focus our further research on MB-MDO.

1.3.2 A Graph-Based Framework for Model-Driven Optimization Facilitating Impact Analysis of Mutation Operator Properties [Joh+23a]

Publication details

Authors: Stefan John, Jens Kosiol, Leen Lambers, Gabriele Taentzer
Appeared in: International Journal on Software and Systems Modeling (2023)
Contributions: The thesis author was the main contributor to the design and execution of the experiments and the interpretation of the results. He also set up, documented, and published the artifacts of the evaluation [Joh+23b]. The development of the framework was a joint effort of all authors. The paper was co-written by all authors. The thesis author was the main contributor to the Running Example and Evaluation sections, and was instrumental in the detailed discussion of the use cases. He did not contribute to the formal proofs.
Targeted research questions: RQ3, RQ4, RQ5

So far, there is no formal basis consolidating the concepts of MB-MDO. To deepen the understanding of this domain, we propose a formal framework that defines the ingredients of evolutionary algorithms used in MB-MDO. As graphs and graph transformations are well-suited to represent models and model transformations, the framework is based on the theory of graph transformation. Since crossover has not yet been investigated for MB-MDO, regarding change operators the framework focuses on mutation. Specifically, it details the use of graph transformation rules as *element mutation operators*, i.e., as operators implementing a change of a single population element (being a model). The formalization, however, is done with the extension to further operators (e.g., crossover) in mind.

With the framework we also provide a comprehensive overview over the key aspects of evolutionary algorithms in the context of MB-MDO and discuss their characteristics. These contributions are meant to serve as a guidance for the rigorous design and specification of evolutionary algorithms in the context of MB-MDO; to illustrate this aspect, the instantiation of the framework is shown on NSGA-II [Deb+02], a state-of-the-art evolutionary algorithm, and the CRA case as the treated optimization problem. The formal setting of our framework also enables a precise reasoning about properties of such evolutionary algorithms and, in particular, their evolutionary operators. Drawing on this attainment we define two properties of (sets of) element mutation operators of evolutionary algorithms used in MB-MDO. Soundness captures if (sets of) element mutation operators can generate models that violate so-called feasibility constraints of an optimization problem. Feasibility constraints are considered constraints which might be violated by solutions created throughout the optimization process. To be considered meaningful, however, a solution needs to fulfill them, i.e., it needs to be feasible. Completeness reflects whether or not a given set of element mutation operators can be used to reach every possible feasible solution of an optimization problem. The definition of both properties on the level of element mutation operators allows for their static analysis. In the case of soundness, to some extent existing tooling can be used to support this analysis [Nas+18; Nas+20].

To investigate the impact of these properties on an optimization, we also propose and formally define various metrics to measure the performance of evolutionary algorithms. An empirical evaluation, based on three state-of-the-art evolution-

ary algorithms (NSGA-II, SPEA2 [ZLT01], and PESA-II [Cor+01]) and three optimization problems from the domain of software engineering (the CRA, SCRUM [BZJ21], and NRP cases), reveals the importance of soundness and completeness for the development of effective and efficient evolutionary algorithms for MB-MDO. It also showcases how our framework can contribute to conduct clearly defined and repeatable experiments.

1.3.3 A Generic Construction for Crossovers of Graph-like Structures [TJK22]

Publication details

Authors: Gabriele Taentzer, Stefan John, Jens Kosiol
Appeared in: Proceedings of the 15th International Conference on Graph Transformation (2022)
Presented at: 15th International Conference on Graph Transformation (ICGT)
Contributions: The development of the approach was a joint effort of all authors. The paper was co-written by all authors. The thesis author was the main contributor to the Running Example section as well as to the examples discussed for the crossover construction. He did not contribute to formal proofs.
Targeted research questions: RQ6

Traditionally, apart from well-designed mutation operators, crossover is a substantial part of any evolutionary algorithm. For various optimization problems its potential to exploit and recombine good parts of existing solutions has, both pratically and theoretically, shown to be beneficial regarding efficiency and effectiveness of an optimization [DHK12; PD18; HK18; APS20]. Thus, the absence of crossover in MB-MDO can be considered a major shortcoming.

Drawing on our proposed framework, we formally define a generic construction for crossover operators of graph-like structures. In particular, typed, attributed graphs are supported which are a typical means to represent models in MB-MDO. Therefore, the presented construction can be used to implement crossover operators for MB-MDO but might also become useful in other contexts. The construction is applicable regardless of the semantics of the underlying graph-like structure, i.e., it is problem independent. However, it provides several

configuration points which allow to adapt the resulting crossover operators to specific optimization problems to leverage domain specific knowledge.

We show that our construction is correct and complete in the sense that for any two solutions to an optimization problem two valid offspring solutions can be calculated. Furthermore, we discuss that our construction can be interpreted as a unifying framework covering various existing crossover approaches for graphs and even for string representations (e.g., the well-known k-point crossover [ES15]).

1.3.4 Towards a Configurable Crossover Operator for Model-Driven Optimization [JKT22a]

Publication details

Authors: Stefan John, Jens Kosiol, Gabriele Taentzer
Appeared in: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (2022)
Presented at: 5th Workshop on Artificial Intelligence and Model-driven Engineering 2022 (MDEIntelligence)
Contributions: The thesis author was the main contributor to the development and implementation of the approach, the design and execution of the experiments, and the interpretation of the results. He set up, documented, and published the open-source project MD◎ver [mdover] containing the implementation, as well as the artifacts of the evaluation [JKT22b]. The paper was co-written by all authors, with the thesis author being the main contributor to the following sections: Related Work, Running Example, Implementation, and Initial Evaluation.
Targeted research questions: RQ6, RQ7

From our generic approach to crossover we derive a more concrete crossover operator. It adheres to our formal definition of graph-like crossovers but is suited for the application to EMF-based models. EMF is one of the widely adopted frameworks for modeling and is the basis for various tools implementing MDO [Bil+19; Str17; BZS18; Hor+22]. However, EMF imposes structural constraints on models which can easily be violated if models are recombined carelessly. The proposed crossover operator is designed to respect these constraints while still being independent of the optimization problem at hand. At

the same time, if needed, it still offers several points of configuration to allow users to incorporate domain-specific knowledge. We also provide a prototypical, generic implementation designed for extension [mdover]. First experiments with the CRA case show that even in this prototypical form crossover can improve the effectiveness of evolutionary algorithms in MB-MDO.

1.4 Outline

Chapter 2 corresponds to the first contribution. Both of the encoding approaches are analyzed and compared qualitatively as well as quantitatively. Motivated by the outcome of this comparison, Chapter 3 presents a formal framework for MB-MDO. Based on this framework, we define and evaluate soundness and completeness as desirable properties of sets of model transformation rules used in mutation operators. A formal definition of a generic construction of crossover operators for MB-MDO is presented in Chapter 4 followed by an EMF specific concretization and an initial evaluation of the same. Chapter 5 summarizes the achievements of this work and discusses future challenges in MDO.

Chapters 2 to 4 are based on related but individual publications. The context is provided by the comprehensive introduction in Chapter 1 and the coherent conclusion in Chapter 5.

2

Comparison of Approaches to MDO

Preface: *This chapter corresponds to the publication* Searching for Optimal Models: Comparing Two Encoding Approaches [Joh+19a].

2.1 Introduction

Many software engineering problems give rise to a tremendous space of possible solutions that differ in various qualities, such as their performance, resource efficiency, and understandability. To efficiently find optimal solutions, *search-based software engineering (SBSE)* [HJ01] seeks to formulate the problem as an optimization problem over one or multiple fitness functions capturing the qualities of interest. By using metaheuristic search techniques, the available solution space can be explored efficiently. Due to their generality, a technique of particular relevance are genetic algorithms [HMZ12], which use the evolutionary operators of mutation, crossover, and selection to perform a guided search over the search space.

Model-driven engineering (MDE) is a paradigm that aims to raise the level of abstraction in a broad range of application domains by the use of models, which are continuously refined and transformed. Recently, research combining SBSE and MDE for a range of purposes has become increasingly popular. The term *search-based model-driven engineering* (SBMDE, [BSA17]) has been proposed as an umbrella term for these efforts. One particular line of research in SBMDE, which we call *model-driven optimization* (MDO), aims to reduce the level of

expertise required by users of SBSE techniques[1]. In MDO, models are used to specify optimization problems and transformation rules are used to explore the search space. Thus, rather than becoming involved in the intricacies of the used optimization technology, users interact with a domain-specific formulation of their problem. They can rely on the familiar modeling and model transformation tools to inspect the solutions and specify the change operations.

Recently, a variety of MDO frameworks has emerged [Bil+19; Abd+14; ZM16; Str17] and been applied successfully in numerous use-cases, including security-oriented software refactoring [Rul+18], model generation [SNV18], transformation modularization [Fle+17], and various more examples [BSA17]. A key distinction in MDO frameworks concerns the way in which solutions are encoded [ZM16]: The *model-based encoding* approach represents solutions as models. In the *rule-based encoding* approach, a solution is a sequence of rule calls in the context of a given input model. Both encoding approaches have distinct advantages: The model-based approach reduces the overhead of applying transformations before the solution is evaluated. It also removes the effort for tracking detail information of the rule calls. The rule-based approach, instead of only incrementally changing the solutions, allows to go back in time easily and deviate from changes made earlier, which may allow the search to move faster through the search space.

So far, there has been no systematic assessment of how the choice of encoding impacts the performance of the search. In existing use-cases, the approach was chosen in an ad-hoc manner, based on the availability of a specific MDO framework. Systematic evidence for the suitability of the chosen approach may help developers in selecting a solution that best fits their use-case and lower the acceptance threshold for MDO in practice.

In this chapter, we aim to compare the two main encoding approaches in MDO frameworks. We study the implementation of these approaches in two state-of-the-art MDO frameworks that differ in the encoding approach used, but otherwise share the same technological basis. This setup allows us to attribute any observed differences in performance to the used encoding. Specifically, we consider MOMoT [Bil+19; momot] and MDEOptimiser [BZS18; mdeo], which follow the rule-based and the model-based approach, respectively. Both are built atop of the

[1] As opposed to applying SBSE techniques to solve MDE problems.

EMF modeling platform [Ste+08; emf], the Henshin model transformation language [Are+10; Str+17], and the MOEA evolutionary search framework [moea]. Problems are specified by meta-models; model transformations are performed by Henshin transformation rules.

The main contributions of the chapter are as follows:

(1) A *qualitative comparison* between the model-based and the rule-based encoding in MDO frameworks, based on a systematic study of their features.
(2) A *quantitative comparison* of both encodings with their implementations in MOMoT and MDEOptimiser, based on their performance (regarding solution quality and execution time) in a set of three diverse use cases.
(3) *Insights into the applicability* of both encoding approaches; their strengths and weaknesses. We study whether the differences can be attributed to the different encoding approaches.

The chapter is structured as follows: Section 2.2 introduces the use-cases considered in this chapter. Section 2.3 describes MDO. Section 2.4 and 2.5 are devoted to the qualitative and quantitative comparative evaluation, respectively. Section 2.6 provides interpretations for the observed results, while Section 2.7 points out threats to their validity. Section 2.8 discusses related work. Section 2.9 concludes the chapter.

2.2 Optimization Problems

For our experimental analysis, we focus on three combinatorial optimization problems. Such problems can be described by providing [GJ79]: (i) a problem domain; (ii) problem instances, each with a finite set of candidate solutions; (iii) a function which maps each candidate solution to a rational number.

Combinatorial optimization problems are often encountered in fields such as engineering, software engineering and finance [CR04]. In contrast to other optimization categories (e.g. Integer Programming, Linear Programming), where sets of equations need to be solved, requirements of combinatorial problems are usually formulated by objects and their relationships. This makes them ideal candidates for Model-driven Optimization.

2.2.1 Problem Descriptions

In this section, we briefly introduce the case studies we will use for quantitative evaluation. We provide only a rough description of the key features of each case study; more detail can be obtained from the websites linked in the footnotes.

Class Responsibility Assignment (CRA). The CRA use case [BBL10] stems from the domain of software design and, initiated by the *Transformation Tool Contest 2016 (TTC'16)* [FTW16], has been addressed by several works in the last years [FSK17; BZ18; Str17]. A software system is defined by a set of features (methods and attributes) and dependencies. Dependencies can be functional, i.e., one method calling another one, or data-driven, i.e., an attribute is processed by a method. Classes can be added to encapsulate features and modularize the system. With the constraint of assigning all features to classes, the optimization goal is to reach a good modular software design. To assess the quality of such a class diagram, the *CRA-Index* is used, which aggregates the metrics of cohesion and coupling. To increase the maintainability and comprehension of a system, combinations of high cohesion and low coupling, reflected by higher CRA-Index values, are desirable [BDW98]. To facilitate easier comprehension of the model changes needed to reach a modular design, we define an extended CRA case (CRA ext.) with the additional objective to minimize the number of transformations performed by the optimization process.

Next Release Problem (NRP). The NRP is about planning which features to include in the next release of a software product. Features are requested by customers in terms of requirements and are implemented as software artifacts. The two optimization objectives are to minimize the development costs, by developing as few artifacts as possible, and to maximize customer satisfaction, by fulfilling requirements. Typical versions of the problem [Dur+11] do not allow nested requirements, partial fulfillment of requirements, or dependencies between software artifacts. Based on the work of Burton and Poulding [BP13], our problem specification[2] considers these additional facets.

[2] see `https://mde-optimiser.github.io/case-studies/nrp/` (visited on 06/2023)

	CRA	CRA ext.	NRP	Ref	Ref ext.
objective	single	multi	multi	single	multi
min. transf.	no	yes	no	no	yes
coarse obj.	no	no	yes	no	no
constrained	yes	yes	no	no	no
mut. changes	small	small	large	large	large
mut. complexity	low	low	medium	high	high
mutable types	yes	yes	no	yes	yes

Table 2.1 – Optimization characteristics covered by the considered use cases.

Refactoring (REF). The REF use case[3] has been taken from the TTC'13. Program refactoring is a common task in agile development typically performed manually. Developers strive to reduce the complexity and increase the comprehensibility of their programs without affecting their function. While there is a great variety of refactor operations possible, we restrict ourselves to attribute location changes to not require any control flow information about the program. The objective is to minimize the number of elements (classes and attributes), by moving duplicate attributes to existing or newly generated superclasses. In an extended version (REF ext.) we want the refactoring itself to be as simple as possible to ease manual review. Therefore, we additionally minimize the total number of refactoring operations.

2.2.2 Coverage of Selected Use Cases

With the selection of the above use cases, we seek to cover a wide range of optimization characteristics (Table 2.1) which might influence both encoding approaches differently. We selected single- as well as multi-objective problems. In particular, problems where the number of applied transformations needs to be minimized are, hypothetically, easier to address by a rule-based encoding. NRP includes an objective (maximize customer satisfaction) which guides the optimization only in a coarse manner, i.e., multiple transformations might be

[3] see `http://martin-fleck.github.io/momot/casestudy/class_restructuring/` (visited on 06/2023)

needed to change a solution's fitness with regard to that objective. The CRA case comprises an additional constraint. With regard to transformation rules, the use cases differ widely. The CRA case uses atomic rules which perform only small changes and contain basic rule elements. In the other cases, more complex rule structures (e.g., control flow elements) can be found. Their rules also allow larger parts of a problem instance to be restructured in one mutation. In terms of their structure, optimization problems differ in whether or not instances of types may be removed and created. While CRA and REF contain such *mutable types* (e.g. classes), NRP only allows to select or deselect existing type instances.

Covering these aspects of optimization problems mitigates the risk of introducing a bias into our comparison. A systematic analysis of the impact of each aspect, however, is not in the scope of this chapter and is left for future work.

Regarding problem size, we considered five models, ranging from 9 features and 14 dependencies (model A) to 160 features and 600 dependencies (model E) for CRA. For NRP, two models were used. Model A with 5 customers, 25 requirements and 63 software artifacts and model B with 25 customers, 50 requirements and 203 artifacts. Models A (19 classes, 18 attributes, 15 generalizations) and model B (6 classes, 68 attributes, 4 generalizations) were used for REF.

2.3 Model-driven Optimization

We now introduce the MDO approach on the example of the CRA problem and illustrate its concepts in the context of Genetic Algorithms (GAs), a specific type of evolutionary algorithms. We stick to GAs because they are prominently used in SBMDE literature [BSA17] and are supported by both of the compared frameworks.

2.3.1 Preliminaries

For a basic understanding of the optimization process, we revisit GAs. GAs [Gol89] work on a population of solutions. Solutions are represented externally by their

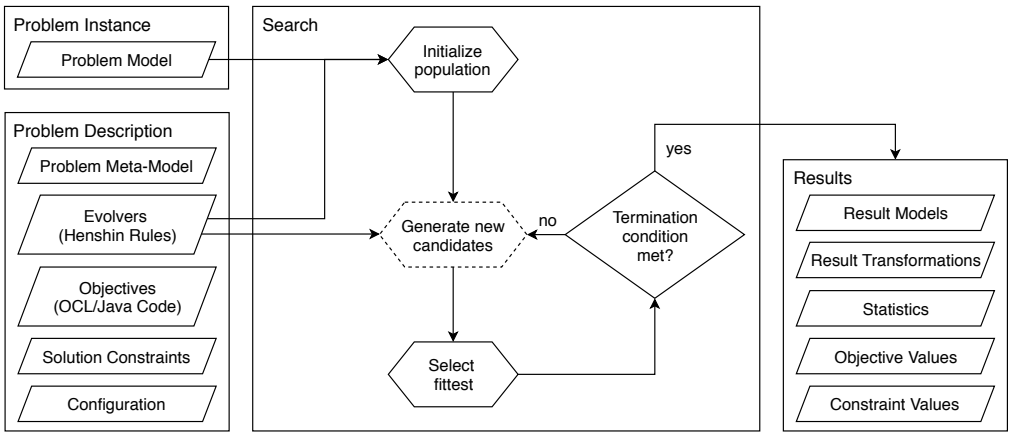
phenotype. An *encoding* maps phenotypes to an internal representation, the *genotype*. Each individual of the population has an assigned fitness, which represents the quality of a solution with regard to the desired optimization goals. A search is performed iteratively by evolving existing solutions. While the fitness is usually evaluated on the phenotype, changes to solutions are performed on their genotype. Inspired by nature, evolution commonly comprises *mutation* and *crossover*. Mutation induces changes in a single solution while crossover aims at recombining multiple solutions in the hope of generating fitter offspring. After an evolution, the population of the next iteration is selected, favoring fit solutions. This cycle continues until a predefined termination criterion (e.g. fitness threshold, number of evolutions) is met.

Both encoding implementations considered in this chapter use Henshin [Are+10; Str+17] to specify and perform model transformations. Henshin is a rule-based transformation language based on the paradigm of graph transformation. A transformation rule is visually represented as a graph in which nodes and edges are annotated with actions such as *delete*, *preserve*, *create*, and *forbid* (Fig. 2.3). Henshin provides an interpreter engine to apply rules to input models. Roughly, a rule is applied to an input model by finding a match of its *preserve* and *delete* elements, and performing the changes specified by the *delete* and *create* elements. The existence of *forbid* elements prevents a rule from being applied.

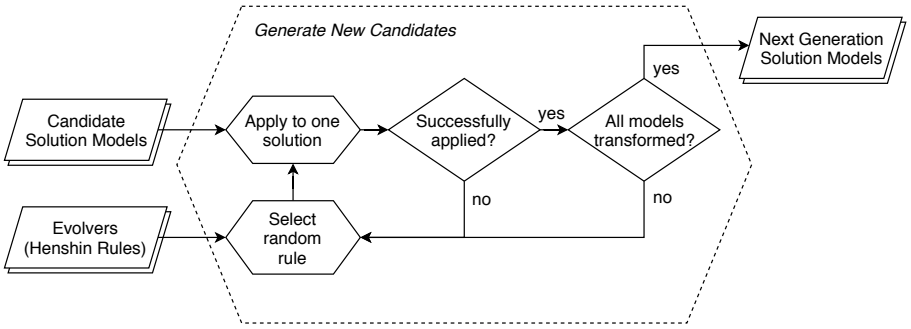
Three easily confused terms related to transformation rules need to be distinguished. First, a *rule* specifies how a transformation has to be performed and may contain formal parameters. Second, a *rule call* consists of a rule and the actual arguments used to execute the rule. And third, a rule call executed in the context of a specific model is called *transformation*.

2.3.2 Model-driven optimization

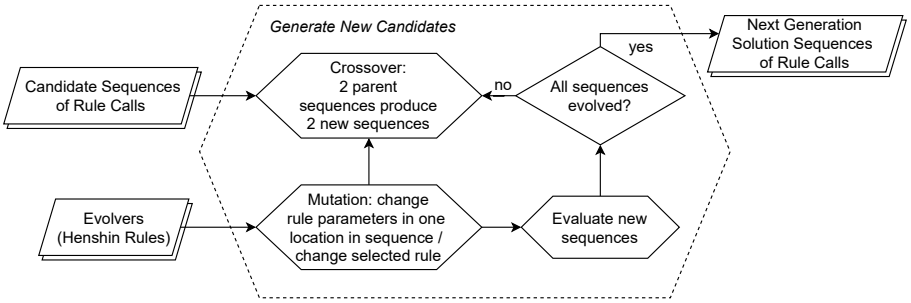
MDO relies on model-driven engineering concepts to specify search problems. In general, a search problem specification comprises (i) a search space description, defining all possible solutions to the problem on the genotype level; (ii) a method for encoding individual solutions; (iii) a set of evolutionary operators used to explore the search space; and (iv) a method for evaluating the fitness of individual solutions based on the optimization goals. In Fig. 2.1a we include a general



(a) Abstract architecture of both tools.



(b) Overview of solution candidates generation in MDEOptimiser.



(c) Overview of solution candidates generation in MOMoT.

Figure 2.1 – Architectural overview for MDEOptimiser and MOMoT.

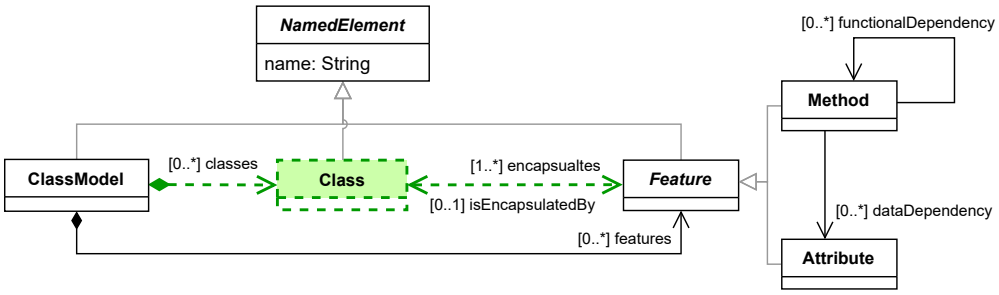


Figure 2.2 – Meta-model of the CRA case.

overview of the architecture used by tools implementing MDO. In the following sections, we describe the core specification components.

2.3.2.1 Encoding Approaches, Solution Space, Search Space

In MDO, a meta-model specifies the structure of a family of optimization problems. Model instances represent the phenotype of concrete problems and solutions. Regarding the *encoding* of solutions, i.e., their representation in terms of a genotype, the model-based and rule-based encodings differ. The model-based encoding aims to reduce the computational time spent for translating solutions between the geno- and the phenotype by using model instances for both. Thus, fitness can be calculated directly on newly found solutions. Fig. 2.1b shows an overview of the model-based approach. The rule-based approach uses sequences of rule calls as genotype. To evaluate the quality of a solution, its whole sequence needs to be applied to the input model before its fitness can be determined. Fig. 2.1c shows an overview of the rule-based approach.

Knowledge about immutable parts of a problem is needed to specify its *solution space*. We define the solution space to contain all solutions to a problem on the phenotype level, i.e, all solution models. Fig. 2.2 shows the meta-model for the CRA case. White solid elements are immutable, while colored dashed elements may be created or removed. The *search space* defines possible solutions on the genotype level. In the model-based approach it equals the solution space. In the rule-based approach it contains all rule call sequences that can be constructed with the chosen set of transformation rules.

2.3.2.2 Evolutionary Operators

In the model-based encoding, mutation is implemented in terms of applying one or multiple transformation rules to a solution model. To our knowledge, how to perform crossover to effectively recombine parts of several models remains an open research problem. For that reason, the model-based approach usually relies on mutation to explore the search space. Mutation in the rule-based approach alters the sequence of a solution. Rule calls are added, removed or changed. The approach also facilitates the use of traditional crossover operators specified for sequential encodings.

To perform model changes, in the CRA case we rely on four Henshin rules (Fig. 2.3), proposed by Burdusel and Zschaler [BZ16]. Unassigned features can be assigned to newly created classes by *addUnassignedFeatureToNewClass* and to existing classes by *addUnassignedFeatureToExClass*. *moveFeatureToExClass* relocates a feature; *deleteEmptyClass* decreases the number of available classes.

2.3.2.3 Fitness Functions

The quality of solution models is evaluated using fitness functions implemented in Java or OCL. Both encoding approaches allow to formulate objectives on the meta-model. In the rule-based approach, additionally, objectives on the sequence of rule calls are possible. Along with objectives, additional constraints (not covered by the meta-model) might be needed to distinguish between valid problem instances and valid solutions. It is up to the optimization algorithm used how violating solutions are treated. In the CRA case, Java classes implement the calculation of the CRA-Index, the retrieval of the number of applied rules, and the constraint of assigning all features to classes.

2.3.3 Implementing Tools

MOMoT as well as MDEOptimiser rely on EMF, Henshin, and the MOEA framework as their technological basis. Both offer a mature DSL to facilitate an easy configuration of optimization runs.

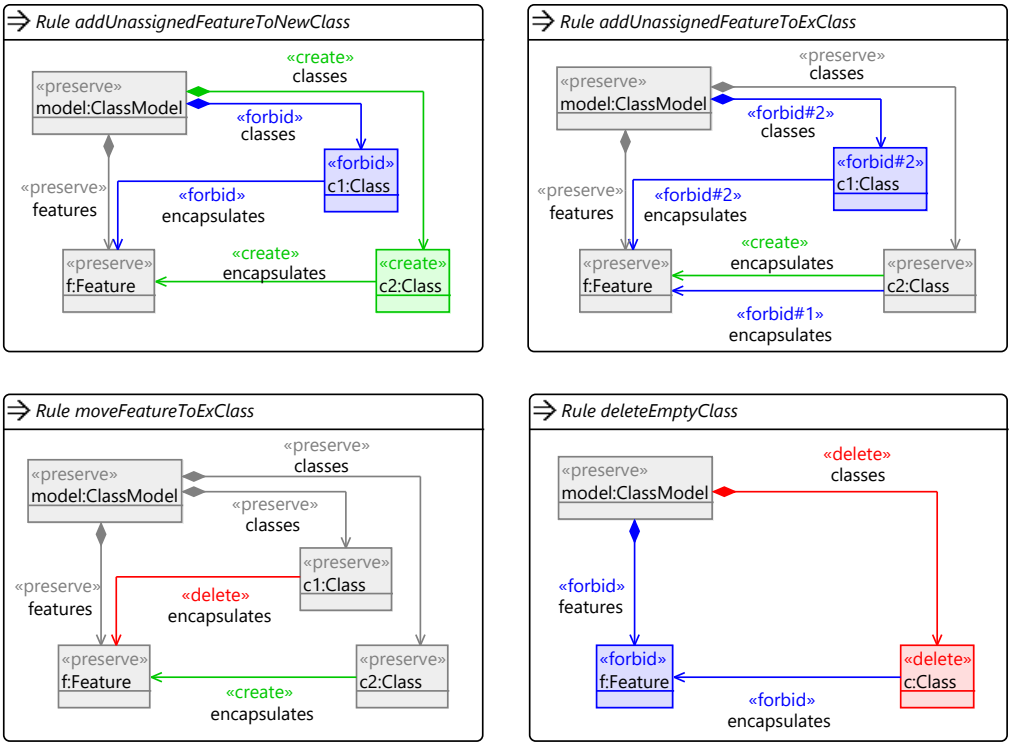


Figure 2.3 – Henshin transformation rules of the CRA use case.

Both allow the user to specify the transformation rules used as mutation operators. The responsibility for generating consistent models is left to the user, who must ensure that the provided rules do not generate invalid model instances when they are applied. As MOMoT relies on the rule-based approach, rules are additionally required to match uniquely when their parameters are set. Otherwise, applying a solution sequence may lead to non-deterministic results.

As an implementation detail, MOMoT works with a fixed solution length, which needs to be specified as an additional parameter. By default, each solution of the starting population is initialized by randomly assigning rule calls to each slot of its sequence. In MDEOptimiser, following the model-based approach, solutions of the initial population are generated by applying a single random mutation to the input model provided by the user. To achieve a fair comparison, we adapted MOMoT to randomly assign a rule call to just one slot of initial solutions.

	Model-based	Rule-based
problem description	initial model	initial model
solution representation	model	rule call sequence
mutation operator	rule call (sequence)	alteration of sequence slots
crossover operator	adhoc implementations	traditional variants for sequential encodings
transformation rules	both constructive and destructive rules needed	depending on use case: constructive and/or destructive rules needed
solution quality	fitness of solution model	fitness of transformation sequence and fitness of resulting model
runtime	time of applying mutation	time of applying evolutionary operators and repair plus applying sequence to initial model
consistency	meta-model conformance	meta-model conformance
completeness	depends on search rules	depends on search rules (and solution length)

Table 2.2 – Summary of similarities and differences of both encoding approaches.

2.4 Qualitative Comparison

In MDO the structure and immutable parts of optimization problems are defined by meta-models and instance models, respectively. They are independent of the chosen encoding. The representation of solutions, however, influences most other parts of the optimization process (Table 2.2). In the following, we consider the main ingredients of MDO and discuss them for our two encodings.

Regarding the *choice of evolutionary operators*, the rule-based approach is more flexible than the model-based approach. Representing solutions as sequences allows the use of traditional crossover operators. Crossover is generally expected to accelerate the optimization process by recombining (potentially good) parts

of already fit solutions. Additionally, the rule-based approach brings more flexibility in the choice of mutation. First, decisions of the past can be reconsidered by changing existing rule calls or entirely deleting them from a solution. Second, problems like CRA and NRP, in contrast to REF, are monotonic in the sense that their whole solution space can be explored by only adding elements to (or removing them from) the initial model. For the rule-based approach, purely constructive (or purely destructive) rules are sufficient in these cases. In case of the CRA, *addUnassignedFeatureToNewClass* and *addUnassignedFeatureToExClass* are such constructive rules (Fig. 2.3). In the model-based encoding, in contrast, additional rules are needed (e.g., *deleteEmptyClass* and *moveFeatureToExClass* in CRA). As a result, evolution steps are potentially spent on reverting prior transformations, slowing down the search. Additionally, more effort has to be put in rule creation.

The rule-based flexibility, however, comes at the cost of making additional repair steps necessary. Rule calls might be dependent on each other, e.g., a rule call in the CRA case creates a class to which a later rule call assigns a feature. As such, changing a single slot in a sequence of rule calls may invalidate subsequent rule calls, which needs to be addressed and may lead to a loss of evolutionary information. Compared to the model-based approach, repair steps as well as the necessity to apply a sequence to the input model to determine the fitness of the resulting model, cause a *runtime overhead* in each evolution step. Its magnitude depends on the size of the solution sequences, which is capped by the chosen solution length in MOMoT. This additional parameter also makes finding a good optimization configuration a bit harder. Note, however, that using sequences of a specific length is not a strict requirement of the rule-based approach but merely an implementation detail of MOMoT.

The rule-based encoding allows insight into how a solution has been found. It also allows more variety in formulating *objectives* as it facilitates to not only reason about the quality of a model but also of the rule call sequence generating it. We exploited that functionality in the extended CRA and REF cases, where the number of rule calls has to be minimized. In MDEOptimiser, as its encoding does not naturally support such objectives, we had to implement an additional rule call counter, effectively extending the information stored by the encoding.

Sequences of the rule-based approach do not, per se, fulfill specific *consistency*

constraints. However, applying a sequence produces a solution model *consistent* with the problem meta-model. This sort of consistency is naturally given for the model-based approach. *Completeness*, i.e., whether or not the entire solution space can be explored, in both approaches depends on the transformation rules provided by the user. In the rule-based approach, additional care has to be taken to choose a sufficiently large solution length when fixed size solutions are used, as solution length caps the number of changes which can be applied to an input model.

2.5 Quantitative Comparison

For the quantitative comparison, we are first interested in how both encoding approaches compare with regard to their optimization performance (runtime and solution quality) when the implementing tools are configured similarily. The qualitative comparison, however, shows that the rule-based approach offers configuration options for the evolutionary operators which are not present in the model-based approach. To analyze their influence on optimization performance, we also compare multiple MOMoT configurations.

We aim at answering the following research questions (subscripts are indicating the associated chapter):

- C_2Q1: How do MDEOptimiser and MOMoT compare, in terms of optimization performance, when their behavior is aligned as much as possible?
- C_2Q2: Can the optimization performance of the rule-based approach profit from the ability to apply a traditional crossover operator?
- C_2Q3: Can the optimization performance of the rule-based approach profit from excluding destructive transformation rules?

In the following, we present the experimental setup, the results we obtained and discuss threats to the validity of our study.

2.5.1 Experimental Setup

For each problem instance of the use cases presented above, we performed 30 optimization runs using NSGA-II [Deb+02] a prominent representative of genetic algorithms [BSA17] supported by both frameworks. With a population of 100 solutions each optimization instance was executed on Amazon Web Services relying on c4.2xlarge Spot instances running openjdk 1.8.0_191 and Amazon Linux release 2. Apart from these common settings, both encoding approaches differed in their parameters, runtime, and even the supported evolutionary operators.

2.5.1.1 Termination Condition

As a termination criterion for our experiments, we chose specific timelimits for each problem instance (Table 2.3). We systematically determined sensible limits by performing 10 runs with MDEOptimiser for each problem instance of each use case. The runs were stopped when no improvement took place for 100 evolutions to guarantee that a good level of convergence can be reached within the limit. The average runtime of these runs was chosen as the termination criterion for both approaches in the final experiments.

	CRA	CRA (ext.)	NRP	REF	REF (ext.)
A	5s	5s	120s	60s	800s
B	15s	15s	500s	800s	800s
C	30s	30s	-	-	-
D	300s	120s	-	-	-
E	2500s	1200s	-	-	-

Table 2.3 – Timeouts in seconds used as termination condition for each problem instance. A through E refer to the different instances of a problem. For NRP and REF, only two instances were used.

2.5.1.2 Operators

For both approaches, we let mutation perform exactly one change per evolution step for each solution (100% mutation rate). This is done by either executing a transformation rule (model-based) or changing the content of one sequence

slot (rule-based). To answer the above research questions, we conducted experiments with different MOMoT configurations. For the direct comparison of both approaches (C_2Q1), we configured MOMoT to use mutation only (MO). To get insights into how crossover influences the performance of the rule-based approach (C_2Q2), another variant combining mutation with one-point crossover (MC) was used. Crossover was also applied once for each solution and step (100% crossover rate). For both approaches, we used the same set of transformation rules for each use case. To answer the last research question (C_2Q3), in the CRA and NRP cases, we analyzed additional MOMoT configurations refraining from the use of destructive rules (e.g., *deleteEmptyClass* and *moveFeature* in the CRA case; see Fig. 2.3), flagging them as non-destructive (ND). In total, we tested four configurations of MOMoT as can be seen in Table 2.4.

To repair solutions in the rule-based approach, invalid rule calls were removed from a solution sequence.

2.5.1.3 Solution Length

As briefly discussed in Section 2.3.3, for MOMoT a fixed solution length determining the maximum number of possible rule calls of a solution needs to be chosen. For the CRA case, to allow exploration of the whole solution space, one has to guarantee that each *feature* can be assigned to a separate *class*. To achieve that, the solution length must match at least twice the number of features in the specific problem instance. As the transformation rules included in NRP allow multiple *software artifacts* to be selected in one step, the situation changes in favor of a smaller solution length. Based on these considerations, we conducted preliminary experiments with solution lengths of 1x, 2x, 4x and 8x the number of key elements of the specific use case. Interestingly, both use cases behaved similar, with best values for 4x and 8x variants. We chose 8x for the final experiments as it was slightly dominant. Regarding the REF case, we did not have any expectations on the length of solutions, as the number of possible refactorings can hardly be guessed from the number of elements. Therefore, we experimented with a solution length of 10, 20, 40, 80 and 160. All lengths lead to a very similar solution quality, 160 being slighty benefitial.

2.5.2 Quality Criteria

We compare the encoding approaches on two criteria: their *solution quality* and their associated *runtime*. In single-objective scenarios the fitness of solutions can directly be used as a quality metric. In multi-objective optimization, this approach is generally not applicable. When objectives are conflicting, tradeoffs need to be considered, captured by the concept of *Pareto optimality*. A solution is said to be *Pareto optimal* if one of its objectives cannot be improved without degrading another objective. As a result, sets of mutually incomparable solutions, called Pareto front approximations, need to be considered. Additionally, a solution *dominates* another solution, if it is better in at least one objective and not worse in any of the others. *Capacity*, *convergence* and *diversity* are the main dimensions along which the quality of such sets is commonly assessed [Jia+14]. In our work we rely on two metrics measuring these dimensions for Pareto front approximations.

2.5.2.1 Ratio Of Best Solutions Found

To quantify the capacity of a solution set S we measure the *Best Solution Ratio* based on the cardinal *C1* metric presented by [HJ98]

$$BSR = \frac{|S \cap PF_{pseudo}|}{|PF_{pseudo}|} \tag{2.1}$$

PF_{pseudo} denotes the set of non-dominated solutions in the union of all solutions found for a particular problem instance. In other words, the best solutions found for that problem by any configuration used in the experiments.

2.5.2.2 Hypervolume

The Hypervolume indicator (HV) [ZBT07] is one of the most popular metrics used in multi-objective optimization [RLB15]. It measures the size of the area enclosed by the objective vectors of a solution set with respect to a given reference point. In our experiments, for the latter we have chosen a point slightly worse than the nadir point given by all solutions found for a particular problem.

This nadir point combines the worst objective values found so far in one vector. We use PF_{pseudo} to normalize HV. As such HV increases towards 1 as solution sets converge towards PF_{pseudo}. The distribution of the objective vectors of the solutions among the objective space (the vector space of objective values) also influences the HV. Thereby, HV incorporates convergence and diversity in one measure.

2.5.3 Results

In the following we discuss the results of our experiments in terms of the research questions we seek to answer. Table 2.4 summarizes the outcome of the experiments with regard to the median quality and standard deviation reached by each configuration for each problem instance. The complete data set discussed in this section can be downloaded from [Joh+19b].

	MDEO		MOMoT							
			MO		MO, ND		MC		MC, ND	
	MED	SD	MED	SD	MED	SD	MED	SD	MED	SD
CRA-A	2.33	0.45	**3.00**	0.47	2.67	0.67	**3.00**	0.49	**3.00**	0.72
CRA-B	1.82	0.53	2.16	0.50	1.83	0.46	**2.72**	0.61	1.50	0.64
CRA-C	**2.22**	0.54	-0.10	1.36	-1.52	1.74	-2.90	2.54	-5.71	3.49
CRA-D	**5.44**	0.88	-4.34	3.45	-5.82	3.12	-6.41	3.31	-15.45	5.29
CRA-E	**11.30**	0.95	-8.42	3.14	-10.04	3.97	-9.98	2.96	-20.13	4.83
CRA-EXT-A	0.89	0.04	**0.97**	0.05	0.90	0.07	**0.97**	0.06	**0.97**	0.06
CRA-EXT-B	0.89	0.02	**0.91**	0.02	0.89	0.03	0.90	0.03	0.88	0.03
CRA-EXT-C	**0.97**	0.02	0.94	0.04	0.93	0.03	0.86	0.06	0.87	0.06
CRA-EXT-D	**0.93**	0.03	0.81	0.05	0.87	0.04	0.71	0.06	0.76	0.05
CRA-EXT-E	0.90	0.04	0.82	0.05	**0.91**	0.03	0.71	0.08	0.81	0.04
NRP-A	**0.79**	0.00	0.76	0.02	0.77	0.02	0.75	0.03	0.75	0.02
NRP-B	**0.71**	0.01	0.48	0.04	0.49	0.04	0.42	0.05	0.38	0.06
REF-A	28.00	0.00	28.00	0.00	-	-	28.00	0.00	-	-
REF-B	51.60	0.00	51.60	0.00	-	-	51.60	0.00	-	-
REF-EXT-A	0.46	0.00	**0.53**	0.00	-	-	**0.53**	0.00	-	-
REF-EXT-B	0.50	0.00	0.50	0.00	-	-	0.50	0.00	-	-

Table 2.4 – Median results (MED) and standard deviations (SD) over 30 runs. The median objective value is shown for the single-objective, the median Hypervolume for the dual-objective variants. Generally, higher values are better. Only for single-objective REF, lower values are better. ND variants are not available for the REF case.

Comparing Approaches (C_2Q1). Both tools have been configured to have equal runtime across similar configurations. Despite this limit, due to the differences in the solution encodings and evolutionary operators, the number of algorithm steps performed by each implementation differs ([Joh+19b]). Across most cases, MDEOptimiser is able to run more than twice the number of steps than MOMoT. In a few cases, however, most notably model E of the multi-objective CRA case, MOMoT outperforms MDEOptimiser with regard to evolution speed.

Despite this difference in the number of steps, for small models MOMoT is on par with or even better than MDEOptimiser in terms of solution quality (Table 2.4). In the CRA case, MOMoT finds a slightly higher objective median for input models A and B in the single-objective as well as in the multi-objective variants. The same can be observed for model A of the multi-objective REF case. However, as the size of the evaluated models increases, the quality of the solutions found by MOMoT decreases compared to solutions generated by MDEOptimiser. Most notably this can be seen for models C, D, and E of the CRA case and model B of the NRP case. In the single-objective variants this effect is also accompanied by a higher standard deviation.

The quality of the solutions is confirmed by the reference set contributions observed for the multi-objective configurations included in Table 2.5. The table shows the total size of the PF_{pseudo} found for each problem instance, and the BSR rate, indicating the percentage of the PF_{pseudo} solutions found by each configuration.

Impact of Rule-based Crossover (C_2Q2). While applying crossover affects the runtime, it does not do so in a consistent way ([Joh+19b]). In the CRA case, fewer steps could be performed when crossover was used. This effect is stronger for smaller models where mutation-only allows up to 50% more steps to be executed. For the largest models a difference of 15-20% can still be observed and the effect is a bit stronger for multi-objective variants than for single-objective ones. In contrast, applying crossover allows for up to 50% more steps in the NRP case. In the Refactoring case, there is no clear trend.

As shown in Table 2.4, in terms of solution quality, crossover is not beneficial for any but a single configuration (model B of the single-objective CRA case).

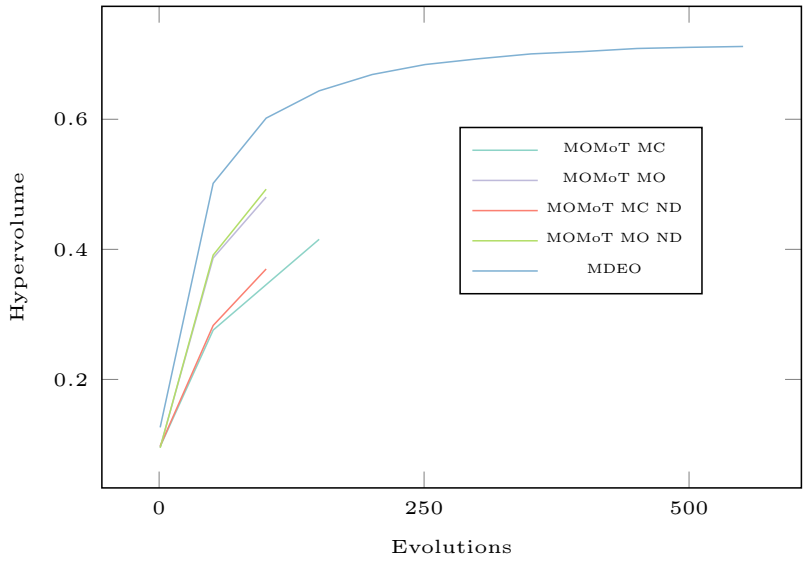
Configurations	PFS	PFC	BSR	Configurations	PFS	PFC	BSR
MOMoT MC CRA A	1	1	1	MOMoT MO CRA E	54	0	0
MOMoT MO CRA A	1	1	1	MOMoT MC ND CRA E	54	0	0
MOMoT MC ND CRA A	1	1	1	MOMoT MO ND CRA E	54	1	0.019
MOMoT MO ND CRA A	1	1	1	MDEO CRA E	54	53	0.981
MDEO CRA A	1	0	0	MOMoT MC NRP A	32	24	0.75
MOMoT MC CRA B	4	0	0	MOMoT MO NRP A	32	28	0.875
MOMoT MO CRA B	4	0	0	MOMoT MC ND NRP A	32	25	0.781
MOMoT MC ND CRA B	4	0	0	MOMoT MO ND NRP A	32	30	0.938
MOMoT MO ND CRA B	4	1	0.25	MDEO NRP A	32	31	0.969
MDEO CRA B	4	3	0.75	MOMoT MC NRP B	245	29	0.118
MOMoT MC CRA C	10	0	0	MOMoT MO NRP B	245	19	0.078
MOMoT MO CRA C	10	0	0	MOMoT MC ND NRP B	245	28	0.114
MOMoT MC ND CRA C	10	0	0	MOMoT MO ND NRP B	245	30	0.122
MOMoT MO ND CRA C	10	1	0.1	MDEO NRP B	245	245	1
MDEO CRA C	10	9	0.9	MOMoT MC Ref A	7	7	1
MOMoT MC CRA D	44	0	0	MOMoT MO Ref A	7	7	1
MOMoT MO CRA D	44	0	0	MDEO Ref A	7	5	0.714
MOMoT MC ND CRA D	44	0	0	MOMoT MC Ref B	24	24	1
MOMoT MO ND CRA D	44	1	0.023	MOMoT MO Ref B	24	24	1
MDEO CRA D	44	43	0.977	MDEO Ref B	24	22	0.917
MOMoT MC CRA E	54	0	0				

Table 2.5 – Summary of the PF_{pseudo} Size (PFS), number of PF_{pseudo} Contributions (PFC) and Ratios of Best Solutions found (BSR) for MDEO and MOMoT in all configurations.

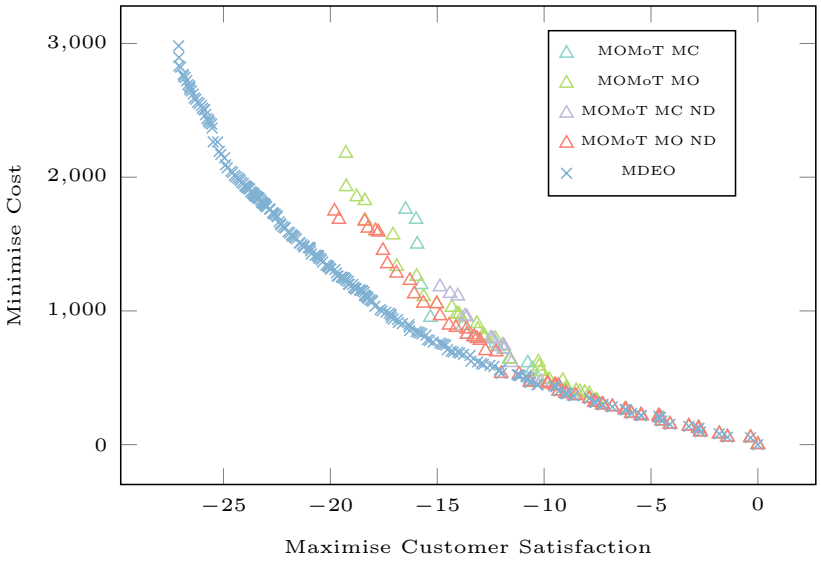
Even in the NRP case, when more steps are executed with crossover, the results are worse.

Impact of Destructive Rules (C_2Q3). The rejection of destructive rules has contrasting effects. Although the optimization is faster in the single-objective CRA case, solution quality decreases considerably. For the larger models of the multi-objective CRA version, however, we find the opposite to be true. In the NRP case, the rule execution speed decreases but the quality is barely affected (Table 2.4 and [Joh+19b]).

General Findings. In general, the convergence rate differs between MDEOptimiser and MOMoT. MDEOptimiser needs fewer evolution steps to develop good solutions in all cases studied. Regarding the BSR, MDEOptimiser outperforms MOMoT clearly for the multi-objective CRA cases and model B of the NRP (Table 2.5). Additionally, the objective vectors of solutions of the found Pareto



(a) Median HV growth



(b) Pareto front approximations after timeouts were reached for each evaluated configuration

Figure 2.4 – Summary of the median HV increase over the number of evolution steps and Pareto front approximations for all evaluated configurations for model B of the NRP.

front approximations spread wider across the objective space, a condition usually considered favorable in multi-objective optimization [ČLM13]. Figure 2.4 shows the convergence behavior and the spread of the Pareto front approximations on the example of model B of the NRP case. For the other cases we refer the reader to the online archive [Joh+19b].

2.6 Discussion

As the rule-based approach allows more options (crossover, reconsidering past decisions, specialized rule sets) in supporting a search, one might expect it to dominate the model-based approach in at least one of the tested configurations. However, the evaluation results paint a different picture. In the following, we will discuss possible reasons for these observations along the associated research questions.

Preface. As discussed in the qualitative comparison, in the rule-based approach rule calls may become invalid when changes are applied to their containing sequence. We find this to be a central aspect for the explanation of observations regarding runtime and solution quality. Due to the repair strategy used in the experiments, as rule calls become invalid in an evolution step the number of rule calls in a solution decreases. This positively affects runtime as fewer rules need to be applied when evaluating the fitness of that solution. We refer to this as *Invalidation Runtime Effect (IRE)* in the following. On the other hand, important evolutionary information might get lost and quality might degrade. We call this the *Invalidation Quality Effect (IQE)*.

With an increasing size of sequences invalidations become more likely. As such, both effects become stronger for larger models where longer solution sequences are needed.

Comparing Approaches (C_2Q1). The rule-based approach suffers from a slower evolution compared to that of the model-based approach. This is mainly caused by the overhead in applying all rule calls stored in a solution before the solution's fitness can be calculated. The overhead becomes more prominent for

larger models where longer solution sequences are needed. This behavior reflects in the solution quality. MOMoT outperforms MDEOptimiser for most of the smaller models because it seems to have enough time to properly converge. As models get larger, aggravated by a stronger IQE, MOMoT is not able to converge equally well. A relatively high standard deviation in these cases underpins this theory.

Impact of Rule-based Crossover (C_2Q2). In most of the cases, crossover was detrimental to the solution quality obtained by the rule-based approach. We attribute this effect to the destructive nature of the traditional crossover. By possibly invalidating many rule calls at once when mixing up sequences of rule calls, a high IQE kicks in. In the multi-objective REF case and the NRP case, applying crossover allowed to perform a higher number of evolution steps. Two possible reasons come to mind. Because the rules of these cases are potentially changing a large number of model elements at once, the invalidation of a single rule call might cause a snowball effect. Accordingly, the IRE might be more visible here than in the other use cases. Additionally, the IQE might degrade the quality of solutions so frequently that only a very small set of Pareto optimal solutions needs to be maintained. A faster selection process might be the result.

Impact of Destructive Rules (C_2Q3). In the CRA and NRP cases, any optimal solution would certainly not need destructive rules. However, without them, solutions might have to be degraded first (e.g. by substituting a feature assignment with a class creation in the CRA case) before a better solution can be reached. Here, destructive rules might help in overcoming local optima. Unfortunately, this does not explain why the absence of such rules is beneficial for some of the use cases. More research is needed to discover the causalities of the observed behavior.

General Findings. Generally, the rule-based approach converges slower than the model-based approach, i.e., more evolution steps are needed to reach a comparable solution quality. We attribute this to the IQE which may introduce steps of quality regression into the optimization process. As discussed for C_2Q2, crossover adds to that problem in most cases.

2.7 Threats To Validity

Regarding research question C_2Q1, the validity of our results depends highly on whether the observed differences in performance can be attributed to the differences in encoding. To mitigate the risk of side effects caused by the implementation of both encoding approaches we have chosen frameworks which are built on a similar basis. Both are implemented in Java, rely on EMF and Henshin for the modelling part, and use the same NSGA-II implementation for running the optimization. For common parameters, we used the same values and the same transformation rules were used to explore the search space. Where necessary, we also aligned the implementation of the tools: both starting from the same initial population in our study and performing mutations of equal size. However, although we checked key parts of both implementations, differences influencing the runtime cannot be ruled out completely.

The generalizability of our findings is limited as both approaches may expose different optimization behavior when other configurations, evolutionary operators and problems are selected. We are confident, though, that the characteristics of our use cases cover a wide range of problems of practical importance.

2.8 Related Work

Our work is related to various approaches to encode search problems, including conventional ones such as binary and integer representations. We discuss relevant work below.

2.8.1 Conventional encodings

Several types of encodings have been proposed for finding optimal genotype representations in evolutionary computation [ES15]. *Binary representation* uses a bit-string in which values of bits are interchanged and switched by the evolutionary operators to change the control variables. *Integer representation* and *real-valued representation* are similar to the previous category, however,

represent the control variables as integers and real values, respectively. *Tree representation* uses a tree to represent the genotype. This encoding is common to represent syntax trees of programs, when using genetic algorithms to improve programs. *Graph representations* use a graph to represent both the genotype and the phenotype.

In many cases, there is an additional genotype-phenotype translation step required to convert an encoding to the solution candidate, such that it can be evaluated by the fitness functions. Usually, an additional repair step is needed after the application of evolutionary operators, in order to repair syntactically correct solutions which are semantically invalid.

2.8.2 Encodings in Model-Driven Optimization

Model-based encodings have been introduced by Burton et. al [Bur+12], who used models and transformations to encode and evolve solution candidates for the NRP problem. The Crepé Complete framework is a more general approach that employs models to represent solutions for any given search problem [EWZ14]. However, this approach converts the models to an integer representation. FitnessStudio [Str17] uses the model-based encoding to generate efficient mutation operators. It evolves the mutation operators on a given training model using higher-order transformations.

2.8.3 Ruled-based encoding

The rule-based encoding has been introduced by Kessentini et. al [KLW13], who propose formulating search problems as a search of optimal transformation chains resulting from rule call sequences. Abdeen et al. [Abd+14] follow this idea in their VIATRA-DSE framework, calling their approach *rule-based design space exploration*. Whereas VIATRA-DSE employs the VIATRA model transformation language to specify transformations, our comparison focuses on the MOMoT [Bil+19; momot] and MDEOptimiser [BZS18; mdeo] tools, since they both use Henshin as the underlying transformation language.

2.8.4 Comparisons between encodings

Previous efforts to compare different encodings have focused on conventional encodings. Janikov et al. [JM91] experimentally compare a binary and a floating point encoding of a dynamic control problem. They find that some benefits of the binary encoding can be countered with refined problem-specific operators for the floating point encoding. Wu and Lindsay [WL96] compare two different flavors of linear representation, one with fixed and one with floating locations for the building blocks for individuals. They find that the floating representation enables the algorithm to better maintain diversity of individuals, and suggest to use both encodings in combination. Kantschik and Banzhaf [KB01] compare a text-based, a linear, and a hybrid representation and show that the hybrid outperforms the former two in most cases. To our knowledge, no work has compared different encodings for SBMDE problems, yet.

2.9 Conclusion

In this chapter, we performed a qualitative and quantitative comparison of the two main encoding approaches in model-driven optimization, the model-based and the rule-based one. The quantitative comparison showed that the model-based approach tends to be more effective, except for the smallest considered models. While the ability of the rule-based approach to reconsider past decisions may be beneficial on the long run, it is quite likely the cause of a slower convergence compared to the model-based approach.

Interestingly, the main distinguishing features of the rule-based encoding, do not help very much. While the rejection of destructive rules caused a slight improvement in some problem cases, it is not a game changing factor in general. For traditional crossover, the result is even worse as it had a detrimental effect on solution quality most of the time. Typically, crossover can help if there are good parts in solutions which can be recombined. This can be problematic in the case of model transformations, as the dependency of rule calls need to be considered when multiple rule calls are exchanged during recombination. A crossover operator more tailored to the needs of MDO is needed here.

Finally, our analysis raised a couple of questions left for future work. How do the specific characteristics of a use case influence the performance of each encoding approach? How can the traditional crossover be improved to be more effective in the rule-based approach? And can crossover be done effectively in the model-based approach?

3

A Formal Framework for Model-Based MDO

Preface: *This chapter corresponds to the publication* A Graph-Based Framework for Model-Driven Optimization Facilitating Impact Analysis of Mutation Operator Properties [Joh+23a].

3.1 Introduction

Various software engineering problems, such as software modularization [BBL10], software testing [WL05], and release planning [BRW01], can be viewed as optimization problems. *Search-based software engineering* (SBSE) [HJ01] explores the application of meta-heuristic techniques to such problems. One of the widely used approaches to efficiently explore a search space is the application of evolutionary algorithms [HMZ12]. In this approach, elements of the search space are generated from existing elements using evolutionary operators such as mutation operators. However, the proper application of SBSE techniques is often not an easy task. As pointed out in [ZM16], "the problem domains in software engineering are too complex to be effectively captured with traditional representations as they are typically used in search-based systems". Compared to traditional encodings, e.g., by vectors, domain-specific models allow to more easily capture structural information about the problem and solution domains. Thus, their use can facilitate the exploratory search for solutions using evolutionary operators, especially for structural software engineering problems.

Model-driven engineering (MDE) [Sch06] aims to represent domain knowledge in models and solve problems through model transformations. MDE can be used in the context of SBSE to minimize the expertise required of users of SBSE techniques. In Chapter 2, we coined the term *model-driven optimization* (MDO) for this combination of SBSE and MDE. Two main approaches have emerged in MDO: The model-based approach [BP13; ZM16] performs optimization directly on the models, while the rule-based approach [Abd+14; FTW15; Bil+19] searches for optimized model transformation sequences. In this chapter, we focus on the model-based approach to MDO and refer to it as MB-MDO for short. Problem instances and solutions are represented as models and the search space is explored by model transformations.

With reference to [ES15; ZM16], the definition of an *evolutionary algorithm* requires a representation of problem instances and search space elements (i.e., solutions). It also includes a formulated *optimization problem* that clarifies which of the solutions are *feasible* (i.e., satisfy all constraints of the optimization problem) and best satisfy the *objectives*. The key ingredients of an evolutionary algorithm are a procedure for generating an initial population of solutions, a mechanism for generating new solutions from existing ones (in this case, by mutation), a selection mechanism that typically implements the evolutionary concept of survival of the fittest, and a condition for stopping evolutionary computations. Selecting these ingredients so that an evolutionary algorithm is effective and efficient is a challenge.

Using MB-MDO the application of search-based techniques in software engineering can be simplified since the search space consists of models evolved with model transformations. However, this does not prevent us from creating sub-optimal specifications of evolutionary operators. For example, a particular set of mutation operators may not be *complete*, i.e., it may not reach all regions of the search space, so an optimum or a good enough solution may be missed. For optimization, however, it can be quite advantageous if the entire search space is reachable with a given set of evolutionary operators. Furthermore, too many of the possible mutations can lead to infeasible solutions, so that it may be advantageous if the given set of operators is *sound* in the sense that mutating a feasible solution yields a feasible solution again. In Chapter 2 and [Str17; BZJ21], several sets of mutation operators were evaluated for their *effectiveness*, i.e., ability to produce good results, and for their *efficiency*, i.e., low computational

cost. In particular, the results in [BZJ21] suggest that feasibility-preserving (i.e., sound) mutation operators can be advantageous. To clarify what soundness and completeness can mean in the first place, and what implications they can have for evolutionary computation, we need a formal basis.

According to Harman et al. [HMZ12], the initial excitement about SBSE is over; it is now time for consolidation, i.e., "to develop a deeper understanding and scientific basis for the results obtained so far." This statement has motivated us to develop a formal framework for MDO (in particular MB-MDO) that will hopefully lead to a deeper understanding of MDO, which combines SBSE with MDE. Our contributions are as follows:

(1) We present a *graph-based framework for (the model-based approach to) MDO* using mutation operators and evolutionary algorithms, and exemplify an instantiation based on the well-known NSGA-II algorithm [Deb+02]. We use the theory of graph transformation [Ehr+06] to define model-driven optimizations since graphs are a natural means to encode models of different types. Mutations of models can be formally defined as graph transformations. Our framework precisely defines all the relevant components of MB-MDO and is intended to assist the developer in using MB-MDO to solve optimization problems.

(2) We identify and define *soundness and completeness* as interesting properties of mutation operator sets. We select these properties because previous evaluations suggest that they may play a role and because these properties can be analyzed statically for certain types of mutation operator sets.

(3) In an *evaluation*, we investigate the impact of soundness and completeness on the effectiveness and efficiency of evolutionary algorithms. We use the framework to clarify all critical factors for conducting a reproducible experiment. In the experiment conducted, we stick to three state-of-the-art evolutionary algorithms (NSGA-II [Deb+02], PESA-II [Cor+01], and SPEA2 [ZLT01]) and we study different sets of mutation operators for three optimization problems: the Class Responsibility Assignment problem (CRA case) [BBL10; MJ14; FTW16; BZJ21], the problem of Scrum Planning [BZJ21], and the Next Release Problem [BRW01; BZJ21]. The experiment is based on the tools MDEOptimiser [BZJ21; mdeo] and Henshin [Are+10].

In the next section, the model-based approach to MDO is presented using an example. Section 3.3 considers the state of the art of MDO and other work related to our framework. Then, in Section 3.4, we present our graph-based framework for MB-MDO. Soundness and completeness of evolutionary operators are defined in Section 3.5. To enable comparison of evolutionary algorithms, we define their effectiveness and efficiency with respect to our framework in Section 3.6. The evaluation is presented in Section 3.7. We conclude in Section 3.8. All proofs and additional material for the evaluation can be found in Appendices A.1 and A.2.

3.2 Running example

Since the CRA case [BBL10] is a structural optimization problem in software engineering and has become one of the most well-known cases when considering MDO, we use it to recall the core concepts of MB-MDO, illustrate our formalization, and conduct a set of experiments. The CRA case aims to provide a high-level design for object-oriented systems. Given a class model with features (i.e., attributes and methods) and their usage relationships as the problem instance, each partial assignment of features to classes forms a solution. What is sought is a complete assignment of features to classes such that coupling between classes is low and cohesion within classes is high.

Meta-modeling is used to define what kind of domain models are considered for optimization. A suitable meta-model for the CRA case is presented in [FTW16] and has been introduced in Section 2.3. A slightly adapted version is reproduced in Fig. 3.1. Since the CRA case is a structural problem, we neglect all meta-attributes (which are attributes of meta-model classes) such as class and feature names to keep the running example as simple as possible. Additionally, we drop the isEncapsulated edge. Multiplicities were also removed, as we will formulate and consider the associated constraints explicitly later on. The meta-model specifies class models that contain features (i.e., attributes and methods) and prescribes the possible usage relationships. While methods can use attributes and methods, attributes are used only by methods. To represent solutions, a class model may contain classes to encapsulate features. A solution model is

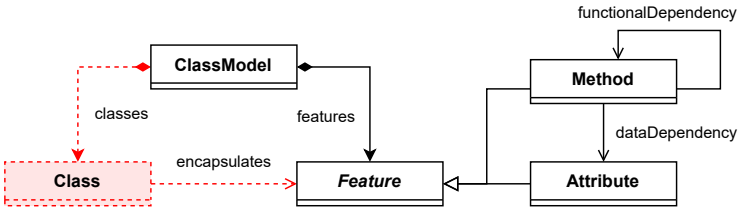


Figure 3.1 – Meta-model of the CRA case as in [FTW16], slightly adapted. White solid elements are invariant problem parts, the colored, dashed class element and its incoming and outgoing references are solution-related.

considered to be *feasible* if *each feature is assigned to exactly one class* (the feasibility constraint).

To assess the quality of a class design, two quality aspects are important: cohesion and coupling. While cohesion confirms that dependent features are within a single class, coupling refers to the dependencies of features between different classes. Good solutions exhibit a class design with high cohesion and low coupling because it is considered easy to understand and maintain. Cohesion and coupling are measured by the *CohesionRatio* and *CouplingRatio* presented in [FTW16], respectively.

3.3 Related work

Recently, several papers have been published on MDO optimizing models or rule-based model transformation sequences. We consider related work on both approaches below. Since our main contribution is a graph-based framework for MB-MDO, we also consider related work on evolving graphs and other frameworks for (evolutionary) optimization.

3.3.1 The rule-based approach to MDO

Early approaches combining SBSE with MDE seek optimized model transformation sequences [Abd+14; FTW15; Bil+19]. More precisely, a solution is a sequence of rule calls that is to be applied to a given input model. The successful

application of such a sequence then yields a solution model. The sequences are optimized using local search algorithms and evolutionary algorithms. While a mutation operator can change sequence slots, a crossover operator splits sequences into parts and recombines them in a different order. The behavior of these operators is largely similar to the operation of traditional variants for sequential encoding. As sequences of rule calls do not per se satisfy consistency constraints, they can easily become inapplicable to the input model after mutation or crossover has taken place. Thus, a disruptive repair step (e.g., truncation of a sequence) is typically needed to regain applicable sequences.

To our knowledge, soundness and completeness have not yet been formally defined in the rule-based approach. In addition, the effects of soundness and completeness on the effectiveness and efficiency of evolutionary algorithms have not been investigated in the rule-based approach.

The comparison of the rule-based approach with the model-based approach in Chapter 2 revealed that the model-based approach tends to be more effective than the rule-based approach. For that reason, we decided to first develop a framework for the model-based approach to MDO and plan to extend the framework toward the rule-based approach in future work.

3.3.2 The model-based approach to MDO

Since our framework follows the model-based approach to MDO, we consider related work in more detail. We consider our work in Chapter 2 and additionally selected the following papers on the *model-based approach to MDO* that have been published in journals, at conferences, and at workshops on modeling and SBSE: [KLW13; Bur+12; BP13; ZM16; Str17; BZJ21; Hor+22]. We investigate which core concepts of MB-MDO were considered. In addition, we compare MB-MDO approaches by describing how they account for the soundness and completeness of mutation operators.

An early work proposing an MDE-based framework to facilitate the application of SBSE to MDE problems is given in [KLW13]. A generic meta-model for encoding is presented that can be extended for specific optimization problems.

However, only preliminary ideas are presented for how to specify an evolutionary algorithm based on that encoding.

Generally, early papers such as [KLW13; BP13; Bur+12] mainly discuss the representation of a problem and the search space. The computation space is specified with meta-models that cover the representation of problems and solutions. A distinction between problem and solution models was introduced in [Bur+12; BP13]. Feasibility constraints and objectives for comparing search space elements are not explicitly distinguished in the early papers. In [KLW13; ZM16], the authors consider objectives and fitness functions and suggest that, for example, a Java implementation can be used to measure the quality of solutions.

Almost all papers on model-based MDO propose to define mutations of solution models as model transformations. Later works such as [ZM16; Str17; BZJ21] and our comparison in Chapter 2, go into more detail and define as well as evaluate concrete mutation operators. In our comparison and in [Str17; BZJ21], models are selected according to how well they meet the objectives or by giving preference to feasible solutions. To terminate an evolutionary computation, various forms of termination conditions were presented in [BZJ21; Hor+22] and Chapter 2: Simple conditions limit the number of evolutionary iterations performed in an evolutionary computation or the total runtime of these computations. Alternatively, the computation can be terminated if a certain number of iterations does not yield a sufficient improvement regarding the quality of solutions.

In Chapter 2 and [Str17; BZJ21], several groups of mutation operators are compared with respect to effectiveness and efficiency. In [Str17], mutation operators are generated by higher-order transformations and compared to manually constructed operators. Since the higher-order transformations generate larger rules than the manually constructed ones, the evolutions can be shortened, resulting in higher efficiency. In Chapter 2, the effects of destructive mutation operators were studied. Evolutionary computations using these rules were faster but resulted in models of less quality. In [BZJ21], the generation of consistency-preserving (i.e. sound) mutation operators with regard to multiplicity constraints is presented. The generated operators were compared to manually designed operators; some effects on effectiveness and efficiency were noted. In [Hor+22], the authors use

techniques from model-based MDO to tackle the problem of optimal configuration of product lines. Concretely, from a feature model with basic constraints they derive *consistency-preserving configuration operators* that transform valid configurations into valid configurations, i.e., they also construct sound operators. These are used as mutation operators in a genetic algorithm to search for optimal configurations. In an evaluation, in particular, concerning effectiveness, their approach outperforms other approaches that do not use sound mutation operators but rely on repair techniques instead. They mention that their technique might not result in a complete set of operators. Generally, the completeness of operator sets has not yet been explicitly investigated.

In summary, related approaches to MB-MDO consider several evolutionary operators and compare them in experiments with respect to their effects on the effectiveness and efficiency of evolutionary algorithms. The understanding of the core concepts of MB-MDO remains implicit or problem-specific. In contrast, we will present a graph-based framework for MB-MDO that concisely defines the core concepts. It will help clarify the critical factors for conducting experiments in MB-MDO and increase the reproducibility of experiments. In particular, the soundness and completeness of mutation operators have only been roughly discussed. We will precisely define these properties with the help of our framework and investigate whether they have an impact on the effectiveness and efficiency of evolutionary algorithms.

3.3.3 Evolving graphs

Since our framework is based on graphs and graph transformation, a closely related approach is *Evolving Graphs by Graph Programming* (EGGP) by Atkinson et al. [APS18; APS21; APS20]. The general motivation is to increase the effectiveness of evolutionary algorithms that operate on graph-like structures by operating directly on graphs as genotypes (as opposed to a linear encoding of graphs). In EGGP, so-called *function graphs* serve as genotypes, and graph programs based on graph transformation rules serve as mutation operators. The mutations are designed to respect certain constraints, namely acyclicity, the arity of nodes (since a node represents a function of a certain arity), and the maximal depth. No other constraints are discussed, and the effects of unsound

mutations are not measured. Our framework does not generally restrict the type of constraints used to specify feasibility, but we propose to stick with graph constraints [HP09] since the soundness of mutation operators for graphs can be statically shown for this kind of constraints. In this sense, we consider a more general form of soundness than that shown for EGGP. To our knowledge, the completeness of operator sets has not yet been considered for EGGP.

3.3.4 Other formal frameworks

While MOMoT [Bil+19; momot] and MDEOptimiser [BZS18; mdeo] are tooling frameworks for combining search-based optimization and model transformations, we are developing a *formal* framework for the model-based approach to MDO. While we are the first to develop a formal framework for this particular area, there are other formal frameworks for evolutionary computation or processes. Two such frameworks are discussed below as examples.

The first approach, which encompasses a large class of evolutionary (and other randomized search) algorithms, regards population-based algorithms as algorithms that generate a new population at each iteration such that each individual is selected from the pool of all possible individuals according to a probability distribution that depends on the current population [Cor+18]. This highly abstract view on evolutionary computation allows the development of common formal techniques that can be applied to analyze different kinds of evolutionary algorithms on different problems (see, e.g., the results in [Cor+18; DLN19; CL20]). However, this framework is too abstract for our purposes; it does not provide support for determining the ingredients of an evolutionary algorithm for which we need to find model-based implementations. Furthermore, this framework does not capture elitism or methods used to preserve the diversity of a population during evolutionary computation.

The most recent theoretical framework for evolutionary processes that we are aware of is [Pai+15]; we also refer the reader to this paper for an overview of attempts to develop such frameworks. In this work, the authors define in modular terms the parts that make up evolutionary processes, namely, *selection* and *variation operators*. To show the adequacy of their framework, they demonstrate how various evolutionary models from the domain of population genetics and various

evolutionary algorithms can be instantiated within it. While soundness is not discussed in [Pai+15], completeness (in our terminology) serves as the defining property that a mutation operator should have. They deal with recombination operators, a topic we leave for future work, and do not yet cover multi-objective problems that MDO regularly addresses.

Because the existing frameworks do not adequately fit our purposes, we develop our own framework in the following instead of presenting our framework as an instantiation of an existing one.

3.4 A graph-based framework for MB-MDO

MB-MDO has been used in Chapter 2 of this thesis and in the literature to solve a variety of optimization problems [BP13; FTW16; ZM16; BZJ21; Hor+22], and the key ideas of the evolutionary algorithms presented are similar. To *clarify the design space* of MB-MDO problems and evolutionary algorithms that solve them, we present a framework for MB-MDO below. The definition of the framework is deliberately *generic* since we want to include the existing variants (of the model-based approach to MDO) into the framework. The framework is also intended to be *formal* to allow for formal reasoning on MB-MDO. In particular, we want to facilitate impact analysis of important properties of evolutionary operators since domain experts can easily specify suboptimal operators. In Section 3.5, we define two properties of mutation operator sets, soundness and completeness, and in Section 3.7 we study their impact on effectiveness and efficiency of evolutionary algorithms.

We will present the framework in two steps: First, the key concepts of the framework are presented using a *meta-model*. It represents these key concepts as interface classes and captures their structure and relations.

In a second step, all the key concepts identified in the meta-model are defined. The definitions capture various requirements that have to be met when using the framework to define concrete MB-MDO problems, develop appropriate evolutionary algorithms, and design suitable experiments in MB-MDO. Since models in general have a graph structure and evolutionary operators generate

new models, it is natural to define the framework based on graph transformation theory.

The presentation of the formal framework is accompanied by a discussion of the characteristics of each key concept and examples of how the concept can be instantiated. The nature of instantiation varies depending on the concept: while the underlying computation space determines the type of models we work with, an instantiation of the term "optimization problem" is primarily concerned with an appropriate formulation of constraints and objectives. When instantiating the term "evolutionary algorithm", we must specify how to generate an initial population, which evolutionary operators are to be applied, how they are applied, and when to terminate.

3.4.1 Meta-model for MB-MDO

Figure 3.2 presents a meta-model that contains the core concepts of MB-MDO and their interrelationships. We use this meta-model to remind us what evolutionary optimization is and to facilitate the understanding of our framework, which defines each of the concepts appearing in this meta-model.

To formulate an *optimization problem*, we encode it in terms of a *computation space* that defines all *computation models* that can occur in the context of the optimization problem. A *problem instance* spans a *search space* that contains only those computation models that represent solutions to this problem instance, the *solution models*. In addition, there are *objective relations* (which may be realized by functions) to evaluate how well the solution models satisfy the optimization objectives. Also, there may be *feasibility constraints*; a solution model that satisfies all feasibility constraints is said to be feasible. A problem instance is itself considered a (potentially infeasible) solution model. A *population* is a finite multi-set of solution models over a common search space; a sequence of populations is called *evolutionary sequence*.

To solve an optimization problem with respect to a particular problem instance, an *evolutionary algorithm* iteratively evolves a population. A *population generator* first creates an initial population for a given problem instance. Given a set of *evolutionary operators*, the evolution is performed following a *computation*

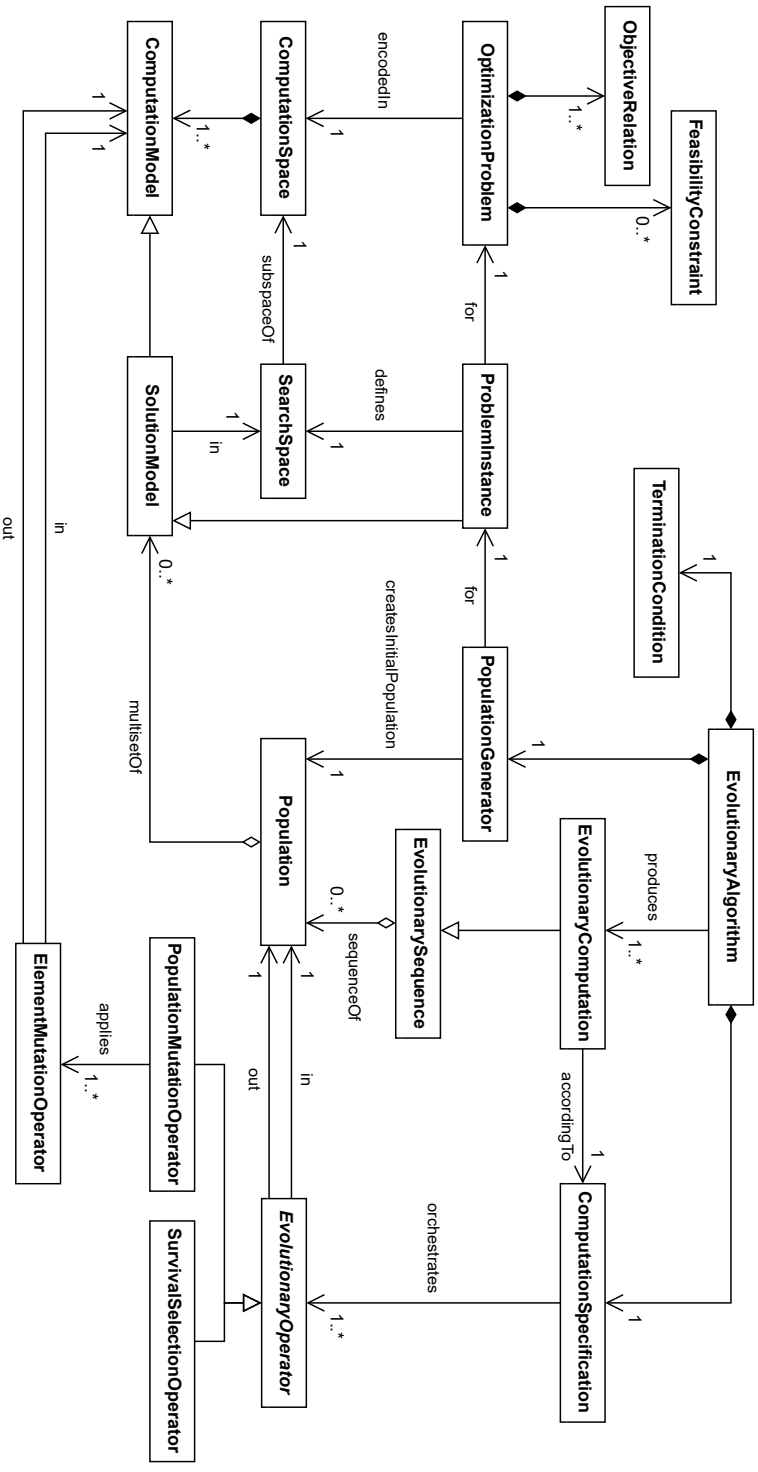


Figure 3.2 – Meta-model for MB-MDO.

specification that dictates how the evolutionary operators are applied in each iteration. An *evolutionary computation* of an evolutionary algorithm is represented as an evolutionary sequence which can be generated by that specific algorithm. While *element mutation operators* are used to perform changes on computation models, a *population mutation operator* organizes at the population level how a new population is created by applying element mutation operators. *Survival selection operators* decide which elements from a population survive and are candidates for further evolution. Evolutionary operators usually make decisions based on the given objective relations and constraints. Finally, an evolution is terminated when a given *termination condition* is met.

3.4.2 Computation space

A *computation space* for MB-MDO forms the basis for encoding and solving an optimization problem (see Fig. 3.2). Basically, it defines a domain-specific modeling language to specify both optimization problems and their solutions. In evolutionary computing, phenotypes, which represent elements externally, are distinguished from genotypes, which represent their internal encodings. In MB-MDO, we do not consider this distinction when formulating an evolutionary algorithm. Problem instances and search space elements are all domain-specific models. It is the task of future work to compare the efficiency of model-based encodings with traditional encodings and to translate models into more efficient encodings as needed. (An example where models are translated to bit strings is given in [Hor+22].)

However, to show formal properties and implement evolutionary computation, we choose appropriate formal representations such as typed graphs for formalizing models (see below) and Ecore models for implementing them (in Section 3.7). Since there are several modeling languages used in software engineering, we are looking for a formalization that is *generic* such that it is not restricted to models of a particular type. Graph-like structures are a natural way to formalize models of different types.

In MB-MDO, computation spaces are defined based on modeling languages, typically specified with meta-models. In Chapter 2, like in several MB-MDO approaches in the literature [ZM16; BZJ21], we have chosen to represent problem

instances by models and solutions by models with dedicated *problem models*. This means that each model of a computation space has a particular part, the *problem model*, which represents important information of the given problem instance and is invariant throughout evolutionary computations. Typical examples of such encoding are the CRA case [Str17; BZJ21], the NRP case [BZJ21], and the SCRUM case [BZJ21]. Accordingly, the meta-model for a computation space, called *computation meta-model*, contains a dedicated *problem meta-model*. All solution models must conform to the computation meta-model. A problem model is a solution model that is fully typed over the problem meta-model.

A meta-model contains typing information as well as multiplicities and optionally other constraints. We distinguish constraints that restrict the language (called *language constraints*) from constraints that specify the feasibility of solutions (called *feasibility constraints*). The problem meta-model induces a subset of the language constraints as *problem constraints*, namely the constraints that affect only the problem elements. Feasibility constraints specify properties that are expected to be satisfied by reasonable solutions, i.e., they constitute side conditions for the optimization. In contrast, language constraints serve to exclude instances that would not constitute a well-posed optimization problem, or to exclude instances for technical reasons. We distinguish problem constraints because, as will be shown in Proposition 3.1, they are particularly easy to use in optimization. In the context of graph transformation, constraints are formalized with nested graph constraints [HP09].

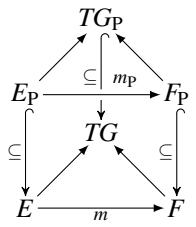


Figure 3.3 – Computation models and cm-morphism.

A *computation model* is given by an instance model that conforms to a computation meta-model. Its problem model and solution part are fully specified by the types in the computation meta-model. This basic structuring is reflected in the following definitions and is shown in Fig. 3.3. It is based on graphs typed over type graphs; a type graph contains a node for each node type and an edge for each edge type. The parallels to the structural part of meta-models are therefore striking. Typed graphs are presented in [Ehr+06] (and recalled in the appendix). For simplicity reasons, we neglect the handling of attributes here.

Appendix A.1 presents a generalized form of computation space based on category theory. A generalized computation space can have various instantiations. For example, it allows to define models as typed, attributed graphs using type inheritance and model changes as typed, attributed graph transformations. All the propositions presented in this section are proven in the appendix based on category theory.

Definition 3.1 (Computation space)**.** A *computation meta-model* is a pair $MM = (\subseteq\colon TG_{\mathrm{P}} \hookrightarrow TG, LC)$ where $\subseteq$ is an inclusion between type graphs TG_{P} and TG, and LC is a set of graph constraints typed over TG, called *language constraints*. The set $PC \subseteq LC$, called *problem constraints*, is the subset of constraints that can be considered as being already typed over TG_{P}. (TG_{P}, PC) is called *problem meta-model*. A *computation element* or *computation model* $(E, type_E)$ over MM is a graph E together with a graph morphism $type_E : E \to TG$ such that $E \models LC$. The *computation space* over MM is

$$CS \coloneqq \{(E, type_E) \mid \\ (E, type_E) \text{ is a computation model over } MM\}.$$

Given a computation model $(E, type_E)$ over MM, the model $(E_{\mathrm{P}}, type_{E_{\mathrm{P}}})$ where

$$E_{\mathrm{P}} \coloneqq E \cap type_E^{-1}(TG_{\mathrm{P}}) \text{ and } type_{E_{\mathrm{P}}} \coloneqq type_E|_{E_{\mathrm{P}}}$$

is the *problem model* and $E \setminus E_{\mathrm{P}}$ is the *solution part* of $(E, type_E)$.

A *computation-model morphism*, short *cm-morphism*, m between computation models $(E, type_E)$ and $(F, type_F)$ is a graph morphism $m : E \to F$ such that m is compatible with typing, i.e., $type_F \circ m = type_E$ (Fig. 3.3). A cm-morphism m is *problem-invariant* if m_{P}, the restriction of m to the problem model of E, is an isomorphism between E_{P} and F_{P}.

Characteristics. The definition of a computation model reflects the core idea that it consists of a problem model (which specifies a problem) and a solution part (which contains information about the solution). The problem model should stay invariant during evolution (which must be ensured); the solution part is developed during an evolutionary computation. A special case is an empty

problem model (and problem meta-model). A computation model (meta-model) would be a simple model (meta-model) in this case.

A meta-model may contain constraints to specify the well-formedness of the modeling language it defines. For example, certain elements of the solution part must always occur together or the number of instantiations of a certain problem type must be restricted. The latter would even be a problem constraint since it applies only to the problem model. Language constraints impose hard constraints that any computation model must satisfy at any point in an evolutionary computation.

Example 3.1 (Computation space)**.** The meta-model underlying the computation space for the CRA case is shown in Fig. 3.1. We consider the presence of exactly one class model as a problem constraint. Additionally, representing a dependency between two features by multiple edges is not useful. Parallel edges are thus forbidden by further language constraints; avoiding parallel edges between features can actually be realized by problem constraints.

For simplicity, the graph TG in Fig. 3.4 focuses on the core structural part and shows the computation meta-model for the CRA case without abstract types and names of edge types. Edge types are still distinguishable by the types of their source and target nodes. As there will always be one class model, we also neglect the node type ClassModel and its containment-edges shown in Fig. 3.1. As for type inheritance, the graph TG shows a flattened version where all inherited edges are shown. (For details on the flattening construction, see [Lar+07].) The black part of TG indicates the node and edge types of the problem meta-model $\mathsf{TG_P}$, while the red, dashed part with filled node rectangles indicates the components of the solution part of computation models. Note that the solution part itself is usually not a graph.

Figure 3.4 also shows two computation models E and S. They are both typed over TG and use the same color coding as TG. The type within each node indicates how it maps to the corresponding node in TG. S shows only a part of the problem model of E along with its solution part. It can be included in E; the inclusion morphism from S to E is indicated by numbers. Each node of S is mapped to the node in E with the same number. The mapping of edges is not shown explicitly but can be inferred from the node mapping. All morphisms between the graphs

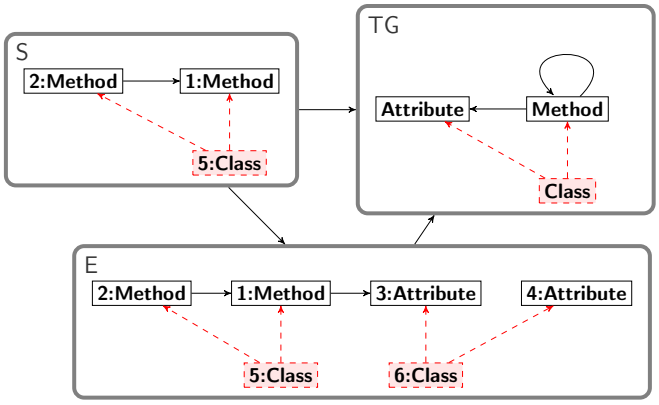


Figure 3.4 – Computation models as typed graphs.

E and S and to the type graph TG are shown by arrows between the graphs in Fig. 3.4. Note that due to the definition of cm-morphisms, the red and black parts are mapped separately. Problem-invariant morphisms will be used for the definition of element mutation operators in Def. 3.4.

Remark 3.1. In the remainder of this chapter, we assume that every computation model E and every cm-morphism is typed over a graph TG, even if this is not explicitly stated. For simplicity, we omit the definition of type inheritance in Section 3.4, but use it in the evaluation. The interested reader can find a suitable definition of type graphs with inheritance in [Löw+15]. The meta-model in Fig. 3.1 is an example of a type graph with inheritance. Class models and classes can refer to features, which can be attributes or methods. In that case, morphisms between computation models are allowed to map objects with compatible types. This means that objects in the image model may have more concrete types than their origins. For example, a Feature object may be mapped to an Attribute object. We explain and prove our formal results for type graphs without inheritance. However, all results can be applied to the case with inheritance as long as inheritance hierarchies are limited to either the problem or the solution part introduced in Def. 3.1 (i.e., problem elements cannot inherit from solution elements and vice versa). The inheritance hierarchy in Fig. 3.1, for example, is limited to the problem meta-model. When we prove our results in Appendix A.1, we also explain why the results are transferable in this way.

3.4.3 Optimization problem

To formulate an optimization problem, we need a computation space that contains problem instances, solutions, and any other models we need to perform evolutionary computations. As in the literature on evolutionary algorithms, we define an optimization problem in MB-MDO as containing both a set of *feasibility constraints* and a set of *objective relations* (see Fig. 3.2), so that it belongs to the category of Constrained Optimization Problems [ES15]. Unlike the language constraints, feasibility constraints can be violated by a model, and that model remains an element of the modeling language and the search space. However, a reasonable solution must not violate feasibility constraints. For each optimization objective, there is a corresponding relation that allows to compare solutions in terms of how well they satisfy the respective objective. Ultimately, the extent to which each of the objectives is met determines the perceived quality of a solution. A concrete problem to be optimized is given by a *problem instance*; it defines its *search space* within the computation space. This search space includes all models that have the same problem model as the given problem instance.

Definition 3.2 (Optimization problem. Search space)**.** Let a computation space CS over a meta-model $MM = (\subseteq\colon TG_{\mathrm{P}} \hookrightarrow TG, LC)$ be given. An *optimization problem* $\mathscr{P} = (FC, \leq_{\mathrm{O}})$ over CS consists of

- a set FC of graph constraints typed over TG, called *feasibility constraint set*, which defines

$$FE(CS, FC) := \{E \in CS \mid E \models FC\},$$

  the set of feasible elements of CS, and
- a finite set $\leq_{\mathrm{O}}$ of total preorders $\leq_j \subseteq CS \times CS$ for $j \in J$, where J indexes $\leq_{\mathrm{O}}$, which are called *objective relations*.

A *problem instance* PI for $\mathscr{P}$ is a computation model in CS. It defines the *search space*

$$S(PI) := \{E \in CS \mid E_{\mathrm{P}} \text{ is isomorphic to } PI_{\mathrm{P}}\}.$$

Each element of a search space is called *solution model* for PI. A solution model E with $E \models FC$ is called *feasible*; we also write $E \in FE(S(PI), FC)$.

Characteristics. There are two special cases for an optimization problem, both of which are covered by the definition: If FC is empty, it is an *unconstrained optimization problem*. All computation models are then automatically feasible, i.e., $FE(CS, \emptyset) = CS$. If $\leq_O$ is empty (but FC is not), we get a classical *constraint satisfaction problem* that only looks for a feasible element in the computation space.

When $|\leq_O| = 1$, we have a single-objective problem. An objective is often given as a function and is referred to as an *objective function* or *fitness function* in the literature. A metric that measures the ratio of coupling and cohesion can be defined as an objective function in the CRA case. We use objective relations instead of functions because they are more general. We also avoid the term "fitness function" because it is used variously in the literature.

For $|\leq_O| > 1$, a multi-objective problem is defined. In this case, it can happen that the objectives are contradictory. This means that a solution may be better than another with respect to one objective but at the same time worse with respect to another objective. As stated in [Zit+03] by Zitzler et al., "we consider the most general case, in which all objectives are considered equally important – no additional knowledge about the problem is available." Thus, to compare two elements with regard to multiple objectives, we can use the *dominance relation*. We say that a solution model E dominates a model F if it is as good as the other in all defined objective relations and, in addition, E is better in at least one of these relations. Formally this means that, if $E \leq_j F$ for all $j \in J$ and there is at least one $k \in J$ with $F \not\leq_k E$, then E *dominates* F. A solution model that is not dominated by any other model is called *Pareto optimum*. The set of all Pareto optima of a search space forms the *Pareto front*. The set of non-dominated solutions of a population is called *approximation set*.

Example 3.2 (CRA problem)**.** In the CRA case, each feature must be assigned to exactly one class, which represents two feasibility constraints, an assignment to at least one class and at most one class. For each constraint, the extent of its

violation in a solution can be determined by counting the number of features that violate the constraint.

Two computation models are compared using two metrics for coupling and cohesion. Combining both metrics into a single one (which is called CRA Index in [MJ14]), the CRA case can be considered as single-objective problem. These metrics can also be used to define two objective relations, which would then lead to a multi-objective problem. Class models can then easily become incomparable because one model has better coupling and another model has better cohesion. Only if both the coupling and cohesion of model A are better than those of model B, does A dominate B, so B is likely to be discarded.

A problem instance is given by a class model that contains a set of features that can use each other (and usually no class is given). It conforms to the type graph in Fig. 3.4. The problem models of the two computation models S and E shown in Fig. 3.4 can be used as problem instances. All the models in this figure formalize feasible solutions (albeit for different problem instances) because they provide a single class assignment for each of their features, implying that the feasibility constraints are satisfied.

The following lemma states that the validity of problem constraints only depends on the problem model of a computation model. For arbitrary constraints (typed over the entire given meta-model), an analogous statement is obviously false; their validity depends on the entire computation model, not just its problem part. This result especially means that, only depending on the problem model of a problem instance PI, either every element of its search space $S(PI)$ satisfies the problem constraints or none of it does.

Lemma 3.1. *Given a computation meta-model* $MM = (\subseteq\colon TG_{\mathrm{P}} \hookrightarrow TG, LC)$ *with a set of problem constraints* $PC \subseteq LC$*, a typed graph* $(E, type_E)$ *satisfies the problem constraints from* PC *if and only if* $(E_{\mathrm{P}}, type_{E_{\mathrm{P}}})$ *satisfies them.*

Later, in order to think about the quality of evolutionary computations, we need the notion of an evolutionary sequence for a given problem instance. Evolutionary sequences are computed by evolutionary algorithms (see Def. 3.7).

Definition 3.3 (Population. Evolutionary sequence)**.** Given an optimization problem $\mathscr{P}$ over *CS* and a problem instance *PI* for $\mathscr{P}$, a finite multi-set over $S(PI)$ is called a *population for PI*. $\mathscr{Q}(PI)$ denotes the set of all populations for *PI*. An *evolutionary sequence* for *PI* is a sequence of populations $Q_0Q_1Q_2\ldots$ with $Q_j \in \mathscr{Q}(PI)$ for $j = 0,1,2,\ldots$ The set $\mathscr{E}(PI)$ consists of all evolutionary sequences for *PI*.

3.4.4 Evolutionary operators

One of the most important configuration parameters of an evolutionary algorithm are the evolutionary operators. These can be divided into change (or variation) operators and selection operators. Evolutionary operators control the evolution with the goal of finding solutions of ever better quality. In order to not leave the search space of a problem instance, evolutionary operators need to be problem-invariant, i.e., they may not change the problem model of solutions.

In this chapter, we stick to mutation operators as the only kind of change operators; crossover operators will be studied in the following chapters. Mutations are usually considered as local changes of search space elements. Therefore, in the context of MB-MDO, it is natural to define so-called element mutations as model transformations, as done in Chapter 2 and in literature [ZM16; BZJ21].

We specify an element mutation operator for computation models as a model transformation rule with a pre-condition (L) and a post-condition (R). Nodes and edges of $L \setminus R$ are deleted, while nodes and edges of $R \setminus L$ are created. In addition, the application of a rule can be prohibited by negative application conditions (NACs). A NAC N is an extension of L and the pattern $N \setminus L$ is forbidden to occur. A concrete mutation of a computation model is realized as a rule application.

In the formal specification of element mutations, we follow the algebraic approach to graph transformation, which takes a transformation rule with a match and performs a transformation step on a graph representing a solution model. In the following, we give a set-theoretic definition (omitting many details) and stick to simple application conditions for mutation operators. A generalized form of transformation that allows more complex forms of application conditions is

defined in the appendix. A detailed definition of graph transformation based on set and category theory can be found, for example, in [Ehr+06]. All models and their relations used to define an element mutation are shown in Fig. 3.5.

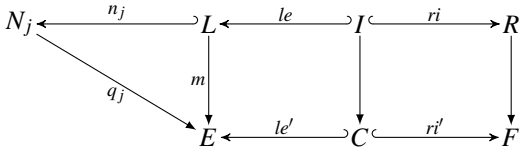


Figure 3.5 – Mutation of element E to element F.

Definition 3.4 (Element mutation operator. Element mutation)**.** Given a computation space CS over a meta-model $MM = (\subseteq\colon TG_{\mathrm{P}} \hookrightarrow TG, LC)$ including problem constraints $PC \subseteq LC$, an *(element) mutation operator mo* is defined by $mo = (L \overset{le}{\hookleftarrow} I \overset{ri}{\hookrightarrow} R, \mathscr{N})$, where L, I, and R are typed graphs over TG with *le* and *ri* being injective, typed morphisms. $\mathscr{N}$ is a set of negative application conditions defined by injective, typed morphisms $n_j : L \hookrightarrow N_j$ with $j \in J$ (where J enumerates the elements of $\mathscr{N}$).

Given a computation model E, an element mutation operator *mo* is *applicable* at an injective cm-morphism $m : L \to E$ if the *dangling condition* holds: A node $n \in E$ must not be deleted if there is an edge $e \in E \setminus m(L)$ which would dangle afterwards. In addition, there must not be a cm-morphism $q_j : N_j \hookrightarrow E$ with $q_j \circ n_j = m$ for a $j \in J$; m is then called *match*. An *element mutation* $E \Longrightarrow_{mo} F$ using *mo* at match m is defined as follows: If *mo* is applicable at m, construct graph $C = E \setminus m(L \setminus le(I))$. Then, $F = C \cup (R \setminus ri(I))$, i.e., a new copy of $R \setminus ri(I)$ is added disjointly to graph C, so that the dangling edges of $R \setminus ri(I)$ are connected to the nodes in C as prescribed by their preimages in I (see Fig. 3.5).

An element mutation $E \Longrightarrow_{mo} F$ is called *pc-preserving* if $E \models PC$ implies $F \models PC$. An element mutation $E \Longrightarrow_{mo} F$ is called *lc-preserving* if $E \models LC$ implies $F \models LC$. An element mutation $E \Longrightarrow_{mo} F$ is called *problem-invariant* if $E_{\mathrm{P}} \cong F_{\mathrm{P}}$. An element mutation operator *mo* is called *pc-preserving (lc-preserving)* if every element mutation $E \Longrightarrow_{mo} F$ with $E \models PC$ ($E \models LC$) is pc-preserving (lc-preserving). An element mutation operator *mo* is called *problem-invariant* if every element mutation $E \Longrightarrow_{mo} F$ is problem-invariant.

A sequence $E = E_0 \Longrightarrow_{mo_1} E_1 \Longrightarrow_{mo_2} \dots E_n = F$ of element mutations (where mutation operators mo_i and mo_j are allowed to coincide for $1 \leq i \neq j \leq n$) is denoted by $E \Longrightarrow_M^* F$, where M is a set containing all mutation operators that occur. For $n = 0$, we have $E = F$.

Remark 3.2. Element mutation operators must not change the types of nodes or edges. This also applies to node types in type hierarchies. Any nodes that are newly created must not have abstract types (such as Feature) since these types must not be instantiated. However, abstract types are useful in the pre-condition of a mutation operator, or in its NACs. For example, in the CRA case, the moving of a method or attribute from one class to another can be specified with only one operator if the abstract type Feature is used. Otherwise, two operators would be required, one for moving a method and one for moving an attribute. When applying a mutation operator, a node can be mapped to a node with a more concrete type if they are compatible with related edge types.

Characteristics. In order to not leave the computation space, element mutations must be lc-preserving. In principle, this condition can be satisfied in two ways: Either the system checks after each mutation whether the resulting model satisfies *LC* if the input model does and retracts the mutation result if it does not, or the modeler ensures that the underlying element mutation operator is designed to be lc-preserving.

We will see below that preservation of problem constraints *PC* can be easily ensured by not creating, deleting or changing elements that are typed over the problem meta-model. In this case, the resulting computation model F has the same problem model as E and if E satisfies *PC*, so does F. Proposition 3.1 states that problem invariance of mutation operators can be statically characterized by problem-invariant morphisms *le* and *ri* in the mutation operator. Hence, Proposition 3.1 provides a static analysis check that can be easily performed.

However, preserving *PC* still does not ensure that an element mutation remains in the computation space. For this, one must additionally ensure that an element mutation operator cannot introduce violations of the remaining language constraints, i.e., for constraints from $LC \setminus PC$. There are several approaches to (semi-)automatically check whether a transformation rule preserves a given

graph constraint [Pen09; Bec+11; Nas+18; Nas+20; Kos+22]. If a constraint is first-order, it can be expressed as a *nested graph constraint* and then ensured by integrating it as an *application condition* into a transformation rule [HP09; Rad+18]; the resulting rule forms an element mutation operator that preserves the given constraint. This constraint integration was automated in the tool *OCL2AC* by Nassar et al. [Nas+18; Nas+20]. However, the resulting element mutation operator is more restricted than the original one since it is only applicable if the mutations do not violate the integrated constraints. Consequently, it may not or hardly be applicable to a computation model anymore. When checking a computed application condition, it may be subsumed by an already existing application condition of the respective operator or it may cover a case that is known to not occur in solutions at all (such as multiple class models in the CRA case). In these cases, the integration of the computed application condition is not necessary.

Example 3.3 (Element mutation)**.** As a concrete example, we consider the element mutation operator moveFeatureToExClass in Fig. 3.6; it moves a feature from one existing class to another. Given the computation model E in Fig. 3.4 and applying moveFeatureToExClass to 3:Attribute (an instance of the abstract type Feature), 5:Class, and 6:Class and the included edge, we get model F in Fig. 3.7 as a result. Note that it represents a computation model with the same problem model as E. Based on model E, four different element mutations can be performed with moveFeatureToExClass: either 1:Method, 2:Method, 3:Attribute, or 4:Attribute can change their encapsulating class.

Operator moveFeatureToExClass is problem-invariant since $L_{\mathrm{P}} \cong R_{\mathrm{P}}$ and therefore, is pc-preserving (as shown below). However, it can introduce a language constraint violation by introducing a parallel edge between the nodes matched by f:Feature and c2:Class. This can happen in infeasible solution models where a feature is contained in more than one class. To preserve the language constraint, the operator can be extended with a negative application condition that checks whether f:Feature is already assigned to c2:Class.

The next proposition ensures that an element mutation $E \Longrightarrow_{mo} F$ returns a computation model F that has the same problem model as E (formally: there exists an isomorphism between E_{P} and F_{P}) if operator *mo* is problem-invariant.

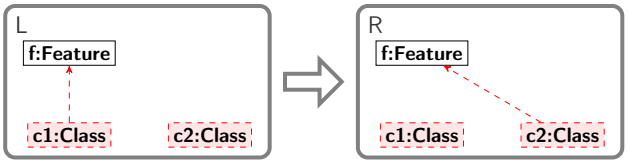


Figure 3.6 – Element mutation operator moveFeatureToExClass.

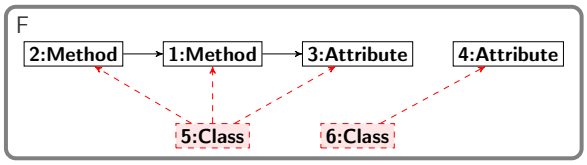


Figure 3.7 – Result of an element mutation of solution model E in Fig. 3.4.

This is not obvious in MB-MDO, since unrestricted element mutations can easily change the problem model.

Proposition 3.1. Let $mo = (L \overset{le}{\hookleftarrow} I \overset{ri}{\hookrightarrow} R, \mathcal{N})$ be an element mutation operator, and let $E, F \in CS$ be computation models such that there is an element mutation $E \Longrightarrow_{mo} F$ (compare Fig. 3.5). Then the operator *mo* is problem-invariant if the morphisms *le* and *ri* in *mo* are problem-invariant.

Together with Lemma 3.1, the above proposition clarifies that problem constraints are trivial to treat in optimization: If the given problem instance PI (or equivalently, its problem model PI_{P}) satisfies the problem constraints, Lemma 3.1 ensures that each computation model E of the search space $S(PI)$ does. Proposition 3.1 then ensures that this also holds for any computation model F obtained by an element mutation $E \Longrightarrow_{mo} F$. Thus, one only needs to verify (i) that a given problem instance satisfies the problem constraints (otherwise it specifies an ill-posed optimization problem) and (ii) that the morphisms used to specify the element mutation operators are indeed problem-invariant (an easy condition to verify).

Next, we consider mutations of populations. Normally, not the entire population is mutated, but a so-called parent selection decides whether and how often an element is mutated. A parent selection can be considered as the first step of mutating a population. After the selection of elements to be mutated, the actual

mutations take place, which are primarily element mutations but can also be sequences of element mutations. Since there are innumerable variations in the literature on how populations can be mutated, especially how parents can be selected, the following definition is deliberately generic.

Definition 3.5 (Population mutation. Population mutation operator)**.** Let a problem instance PI for an optimization problem $\mathscr{P}$ over the computation space CS and a set of element mutation operators M be given. A *population mutation operator* MO is a binary relation over $\mathscr{Q}(PI)$ with $Q \subseteq Q'$ for each $(Q,Q') \in MO$. For each $F \in Q'$ there is a (possibly empty) finite sequence of element mutations $E \Longrightarrow_M^* F$ with $E \in Q$. Each $(Q,Q') \in MO$ is called a *population mutation* of Q to Q' via MO.

Characteristics. Usually, it is not desired that each element of a population is evolved to an offspring solution. A *parent selection* decides whether an element is evolved once, several times or not at all. In addition, population mutations can differ in how many offspring solutions they generate and which and how many element mutations they apply to evolve an existing solution, i.e., what the finite transformation sequences that produce offspring are. The framework leaves open how these sequences are specified. In particular, such a sequence might be a complex programmed graph transformation sequence.

All of these decisions can be based on the fitness of solutions, which is usually determined by the objective relations (e.g., considering the dominance relation) and the satisfaction of the feasibility constraints. However, meta-information about the population can also be taken into account. Due to its generality, our definition supports all these variants.

Example 3.4 (Population mutation)**.** In addition to the element mutation operator moveFeatureToExClass shown in Ex. 3.3, other element mutation operators can be addUnassignedFeatureToNewClass, which creates a class and assigns an unassigned feature to it, addUnassignedFeatureToExClass, to assign an unassigned feature to an existing class, and deleteEmptyClass to remove an empty class from the model. Apart from their names, which were chosen for clarity, these mutation operators are analogous to those in [BZ16; BZJ21]. All the element mutation operators presented form the set M.

For parent selection, the NSGA-II algorithm [Deb+02] uses, for example, binary tournament selection, which we reproduce below: Let a population mutation operator CraMutation rely on n binary tournament selections (where n is the size of the population) to decide which solutions to evolve. In each tournament, two randomly chosen elements of the input population compete with regard to the following *fitness* specification: (1) Given two feasible solutions (assigning each feature to exactly one class), a solution that dominates the other in terms of the coupling and cohesion metrics is considered fitter. (2) A feasible solution is automatically fitter than a non-feasible one. (3) If two infeasible solutions are compared, the one with a lower degree of constraint violations is the fitter one. The extent to which a constraint is violated can be determined by the number of violations of the constraint. Alternatively, if a constraint is defined by attribute values, the extent to which a desired value is missed can also be considered. The degree of violation of a constraint can be calculated by summing up the extent of violation of each constraint.

For the CRA case, if two infeasible solutions are equal in terms of the degree of constraint violations, the solution that dominates the other in terms of the coupling and cohesion metrics is the fitter solution. Solutions of equal fitness with respect to the aforementioned rules can be further distinguished by their *crowding distance* [Deb+02]. The crowding distance estimates the proximity of a solution to other solutions in a population. To maintain diversity in a population, solutions with a high crowding distance are considered fitter. (For more details, we refer to [Deb+02].) The solution with the higher fitness wins the tournament. It is cloned, and the clone is mutated with an arbitrarily chosen *applicable* element mutation operator of M. If none of the element mutation operators is applicable, the clone remains unchanged. When n tournaments have been run, all clones are merged with the elements of the input population to form the output population.

In a population, not all its elements are required or desired to form the next generation. A survival selection filters the next generation from a population according to certain criteria. The following definition is also deliberately generic, as we do not want to exclude certain implementations from our framework.

Definition 3.6 (Survival selection. Survival selection operator)**.** Given a problem

instance *PI* for an optimization problem $\mathscr{P}$, a *survival selection operator SO* is a binary relation over $\mathscr{Q}(PI)$ such that for all $(Q, Q') \in SO$ holds: $Q' \subseteq Q \in \mathscr{Q}(PI)$. Each $(Q, Q') \in SO$ is called a *survival selection* that selects Q' from Q.

Characteristics. Following nature's example, survival selection is used to realize the concept of survival of the fittest. The concept of fitness often coincides with that used in parent selection of population mutation; the objective relations and feasibility constraints are the most influencing criteria for defining fitness. However, fitness in survival selection can also be viewed from a different perspective. Factors such as the age of solutions, i.e., how many generations they have survived, or the diversity of the resulting next generation can be considered. To avoid losing the most valuable solutions achieved so far, survival selection typically employs elitism, i.e., the fittest solutions are (partially) preserved for the next generation. On the other hand, survival of less optimal or even infeasible solutions may allow broader exploration of the search space. As with population mutation, there are many ways to implement survival selection.

Example 3.5 (CRA survival selection)**.** A survival selection operator corresponding to the one used in the NSGA-II algorithm can be based on the notion of fitness presented in Ex. 3.4. First, the three rules that take the constraints and objectives into account are used to create a pre-order (rank) of population elements, i.e., each rank contains solutions that have the same overall constraint violation and do not dominate each other. Beginning with the rank of best solutions, all solutions from subsequent ranks are selected until the selection of the entire next rank would exceed the number of desired solutions in the output population. At this point, solutions from the next rank are selected, taking their crowding distance (introduced in Ex. 3.4) into account, until the desired size of the output population is reached.

3.4.5 Evolutionary algorithm

A given instance of an optimization problem is solved using an evolutionary algorithm. Since MB-MDO has been performed with several evolutionary algorithms and there are many more variants of evolutionary algorithms in the

literature, we want to keep the definition of evolutionary algorithm deliberately generic. Actually, we define a skeleton of an evolutionary algorithm as shown in the pseudo code in Algorithm 1. The minimal set of parameters for an evolutionary algorithm includes a problem instance and a set of evolutionary operators. Concrete evolutionary algorithms may have further parameters, such as an initial population. If the algorithm does not receive the initial population as input, it first generates an initial population for the given problem instance PI. Then it iteratively applies operators from the given set OP of evolutionary operators to evolve this population. To this end, the computation specification C specifies how the operators of OP are orchestrated over the course of an iteration. After each iteration, a termination condition t determines whether further iterations are performed. Algorithm 1 leaves G, C, and t completely generic since the framework is intended to support all kinds of evolutionary algorithms. In the following, we define such a generic algorithm and its semantics.

Algorithm 1 Pseudo code for an evolutionary algorithm.

```
1: procedure EVOLUTIONARY ALGORITHM(PI, OP)
2:     Generate an initial population with G based on PI
3:     while t is not fulfilled do
4:         C                          ▷ Apply operators from OP following C.
5:     end while
6: end procedure
```

Definition 3.7 (Evolutionary algorithm and its semantics)**.** Given a problem instance PI for an optimization problem $\mathscr{P}$ and a set of evolutionary operators OP for $\mathscr{P}$, let $\rightarrow_{OP} \subseteq \mathscr{Q}(PI) \times \mathscr{Q}(PI)$ be defined as

$$\rightarrow_{OP} := \{(Q,Q') \mid (Q,Q') \in op, op \in OP\}.$$

An *evolutionary algorithm* $\mathscr{A}(PI,OP) = (G,C,t)$ consists of a *population generator* G to generate a starting population Q_0 based on PI, a *computation specification* C based on OP, and a *termination condition* t.

The semantics $Sem(C)$ of C is a subset of the binary relation $\rightarrow_{OP}^{*}$. The termination condition t is a predicate over $\mathscr{E}(PI)$, the set of evolutionary sequences for PI. An *evolutionary computation* of $\mathscr{A}(PI,OP)$ is an evolutionary sequence

$Q_0Q_1Q_2\ldots \in \mathscr{E}(PI)$ with $(Q_j, Q_{j+1}) \in Sem(C)$ for $j = 0, 1, \ldots$. Each (Q_j, Q_{j+1}) is called an iteration. The *semantics* $Sem(\mathscr{A}(PI, OP))$ of $\mathscr{A}(PI, OP)$ is the set of all its evolutionary computations satisfying t. An *execution* of $\mathscr{A}(PI, OP)$ results in an evolutionary computation of $Sem(\mathscr{A}(PI, OP))$. A set of executions of $\mathscr{A}(PI, OP)$, called *execution batch* of $\mathscr{A}(PI, OP)$, yields a *batch result*, a multi-set of evolutionary computations of $Sem(\mathscr{A}(PI, OP))$.

Characteristics. In general, evolutionary algorithms are non-deterministic since the generator for initial populations, the computation specification, and the evolutionary operators can introduce probabilistic behavior. Therefore, an evolutionary algorithm generally determines a set of possible evolutionary computations. When experimenting with an evolutionary algorithm, it is executed a certain number of times. Each execution of the algorithm results in an evolutionary computation being part of its semantics. An experiment usually includes a whole batch of executions. It may happen that the same computation is obtained several times in one experiment. Therefore, an execution batch may result in a multi-set of evolutionary computations. As a special case, algorithms with deterministic behavior (leading to a single computation) are also included.

The generator G, the computation specification C and the termination condition t can be instantiated by familiar patterns for existing evolutionary algorithms, adapted to MB-MDO. In most cases, a random initialization procedure is used to generate an initial population from a given problem instance. Properties such as feasibility or diversity of elements in the initial population can affect the efficiency and effectiveness of an evolutionary algorithm. Ideally, the entire search space should be reachable from an initial population. However, depending on the problem, specifying diversity, implementing an efficient generator, and analyzing reachability can be challenges in themselves. Traditionally, the set of operators consists of at least one population change operator and a survival selection; the computation specification combines them by first applying the change operators sequentially, followed by the survival selection to choose the population for the next iteration. More sophisticated concepts (such as self-adaptive evolutionary algorithms [ES15]) can also be expressed using our framework but may require the implementation of more complex computation specifications.

Example 3.6 (CRA evolutionary algorithm)**.** For the CRA case, consider an example evolutionary algorithm with an initial population of 100 models, all equal to the given problem instance. The set *OP* contains the population mutation operator CraMutation introduced in Example 3.4. Also, *OP* contains the selection operator presented in Example 3.5. The computation specification *C* prescribes that both operators in *OP* are applied alternately, starting with the mutation of a population, followed by the selection of a population for the next iteration.

An example termination condition t is the following: It monitors the progress of an evolutionary computation with respect to the improvement of the approximation set. The improvement between the approximation sets A_1,A_2 of the populations of two iterations is measured by the Euclidean distance of the solution vectors of their solutions. For each solution E, its *solution vector* $v_S(E)$ consists of the values of the cohesion and coupling metrics and two values reflecting the extent to which the two feasibility constraints are violated. For $E_2 \in A_2$, let $d_{\min}^{A_1}(E_2)$ be the minimum Euclidean distance of $v_S(E_2)$ to the solution vectors of all solutions in A_1. The distance between A_1 and A_2 is then defined as

$$d(A_1,A_2) \coloneqq \sum_{E_2 \in A_2} d_{\min}^{A_1}(E_2).$$

Let a finite evolutionary computation $Q_0Q_1 \ldots Q_k$ and a corresponding sequence of approximation sets $A_0A_1 \ldots A_k$ be given. Furthermore, let a sequence of *progression indices* $x_1x_2 \ldots x_i$ be defined with $1 \leq x_j \leq k$ for all $1 \leq j \leq i$ as follows. The first progression index x_1 represents the iteration at which the approximation set changed for the first time. Further progression indices represent iterations where the approximation set has improved by at least 3 percent compared to the approximation set of the iteration represented by the previous progression index. Thus, for any progression index x_j with $1 < j \leq i$, it holds that

$$d(A_{x_{j-1}},A_{x_j}) \geq 0.03 * d(A_0,A_{x_{j-1}}).$$

Let the sequence of progression indices be complete, i.e.,

$$d(A_{x_{j-1}}, A_m) < 0.03 * d(A_0, A_{x_{j-1}})$$
$$d(A_{x_i}, A_n) < 0.03 * d(A_0, A_{x_i})$$

hold for all m, n with $x_{j-1} < m < x_j$ and $x_i < n \leq k$. The termination condition t is satisfied if $k - x_i = 100$.

3.5 Soundness and completeness

Evolutionary algorithms aim at finding optimal solutions or at least approximating them. The operators chosen determine the effectiveness and efficiency of the search. In this section, we will consider two fundamental properties of sets of element mutation operators that can affect the effectiveness and efficiency of evolutionary algorithms: *soundness* and *completeness*. Soundness refers to preserving the feasibility of solutions and completeness refers to preserving the reachability of all feasible solutions. Investigating whether properties such as soundness and completeness actually have an impact on the effectiveness and efficiency of evolutionary search is of particular interest in the context of MDO. One of its promises is that domain knowledge can be integrated into problem-specific evolutionary operators, thereby improving evolutionary search. Determining general properties that operator sets should have provides guidelines and limits for constructing problem-specific operators.

While evolutionary algorithms typically operate on populations, in this chapter we first introduce our notions at the level of element operators. There are two reasons for this: First, we deliberately introduced the population mutation operators in general terms. At the level of element operators, it is still clear what soundness and completeness should mean. At the level of population operators, it becomes more complex; we think that it might be more promising to look for appropriate definitions for more concrete (classes of) population operators. Second, we also develop our definitions from the point of view of analyzability. The properties we define can be analyzed statically since they are based on element operators. Analyzing comparable properties at the level of population operators would likely be more difficult and costly. The fact that an

element operator has a particular property with respect to all or certain classes of solutions needs to be checked only once. At the level of population operators, run-time verification might be required to obtain a particular property. We will outline the possibilities for static analysis after the respective definitions and give some hints on how to define the respective properties for the more general level of population operators.

We consider an element mutation operator *sound* if it ensures the feasibility of all generated solution models under the assumption that the input models are already feasible.

Definition 3.8 (Soundness of element mutation (operator sets))**.** Let $\mathscr{P} = (FC, \leq_O)$ be an optimization problem for a computation space CS. Assuming $E \in FE(CS,FC)$, an element mutation $E \Longrightarrow_{mo} F$ is *sound*, where *mo* is an element mutation operator, if $F \in FE(CS,FC)$.

An element mutation operator *mo* is *sound* if every element mutation $E \Longrightarrow_{mo} F$ via *mo* with $E \in FE(CS,FC)$ is sound. A set of element mutation operators is *sound* if each of its operators is sound.

Note that a population mutation that starts with feasible solutions can lead to a population with only feasible solutions, even if the element mutation operators are not sound. This is because the soundness of the element mutation operators is a sufficient condition for preserving the feasibility of solutions but not a necessary condition.

Example 3.7 (CRA soundness)**.** In the CRA case, the element mutation operator moveFeatureToExClass in Fig. 3.6 is sound because it reassigns a feature to another class, i.e., the feature is neither left without a class assignment nor given a second one. In contrast, an operator that simply removes a feature from a class is obviously unsound.

Static analysis. In the formal framework developed so far, element mutation operators are not generally sound. It is up to the modeler to prove that concrete element mutation operators are sound. We discussed above (after Def. 3.4) which

approaches exist in the literature to (semi-)automatically check whether a transformation rule preserves a given graph constraint. Since feasibility constraints are also specified as graph constraints, these approaches can also be used for checking soundness. In principle, it is also possible to check at runtime whether unsound operators preserve feasibility (and if not, to undo their application), but this would require additional computational effort at runtime.

Outlook on population operators and runtime verification. Since feasibility is a property of an individual solution, it is not immediately obvious how soundness should be defined for population mutation or survival selection (operators). One might think about requiring that at least the proportion of feasible solutions be preserved in a population. Preserving the proportion of feasible solutions (while allowing enough newly computed solutions to be selected for the next population) might require sophisticated control of the selection of solutions and element mutation operators used during population mutation to ensure that enough feasible offspring are computed. Note that the probability for this could be increased by the use of sound element mutation operators (without further control): In (most) evolutionary algorithms, feasible solutions are preferred over infeasible ones in both selection for reproduction (here: for mutation) and survival selection.

In our evaluation, we investigate whether the use of sound element mutation operators supports the finding of optimal solutions more efficiently and effectively.

Next, we consider the *completeness* of a set of element mutation operators. It is satisfied if, for a given problem instance, all feasible solutions of its search space can be generated at each point of an optimization.

Definition 3.9 (Completeness of element mutation operator sets)**.** Let $\mathscr{P} = (FC, \leq_O)$ be an optimization problem, PI a problem instance for $\mathscr{P}$, and M a set of element mutation operators. The set M is *complete* if, for every solution model $E \in S(PI)$ and every feasible solution model $F \in FE(S(PI), FC)$, there exists a finite sequence of element mutations $E \Longrightarrow_M^* F$.

Only considering element mutations based on a set of mutation operators, the completeness of that set is a sufficient (but not a necessary) condition for the reachability of all optimal solutions. It should be noted that the evolutionary algorithm may still miss these optimal solutions since it further constrains which element mutations are actually applied. Therefore, the completeness of a set of element mutation operators does not imply that an evolutionary algorithm using that set definitely reaches all optimal solutions. Conversely, an evolutionary algorithm that uses an incomplete set of element mutation operators can, in principle, still find optimal solutions. However, using an incomplete set of element mutation operators bares the risk of completely truncating regions of the search space that contain optimal solutions. Knowing whether a set of element mutation operators is complete or not allows the developer to make an informed decision about it.

Example 3.8 (CRA completeness)**.** The set of element mutation operators discussed in Ex. 3.4 is not complete since no new classes can be created after all features are assigned. To obtain a complete set of element mutation operators, we make a small change. We add an additional element mutation operator moveFeatureToNewClass that can move an already assigned feature to a new class. The resulting set of element mutation operators is complete since any computation model (feasible or not) can be converted into a feasible model with one class using the operators addUnassignedFeatureToNewClass at most once, the operator addUnassignedFeatureToExClass as often as possible, and operators moveFeatureToExClass and deleteEmptyClass as often as needed. This feasible model with one class can then be transformed into any feasible model using the operators moveFeatureToExClass and moveFeatureToNewClass. Summarizing, every feasible model is reachable from any model. Note that this argumentation (implicitly) uses the fact that computation models satisfy the language constraints of the CRA case: General graphs that are typed over the meta-model of the CRA case could contain more than one class model or classes that are not contained in a class model. In such situations, we could, for example, not assign two features that are contained in different class models to the same class. Hence, our (extended) set of operators is not complete for general graphs.

Static analysis. We are not aware of any formal approach that automatically checks the completeness of sets of element mutation operators as defined above. Basically, the problem is a reachability problem that can be analyzed for a single model using model checking for graph transformation [RSV04]. However, we do not ask about the reachability of a single model from a single model but about the reachability of a (possibly infinite) set of models from each possible model. Maybe surprisingly, in all three example cases of our evaluation the same simple technique can be used to manually prove completeness of a set of element mutation operators. As sketched for the CRA case in the example above, one looks for one model for which one can argue that it can be transformed (via the given element mutation operators) into any feasible model, and that any model can be transformed into it. This (manual) analysis is still static in the sense that it only needs to be done once for a given set of element mutation operators. Since we selected our case studies before developing operator sets for them, for which we then proved completeness, and the same simple proof technique worked in all cases, we are confident that proving completeness manually will be feasible also in other cases.

Outlook on population operators and runtime verification. If a chosen set of element mutation operators is complete, we can be sure that in principle all optimal feasible solutions can be found using these operators. However, much weaker forms of completeness suffice to obtain this property. It is sufficient that during an evolutionary computation every feasible solution remains reachable from one element of the current population, but it need not be reachable from all elements. Defining such a population as complete, a survival selection operator would be *completeness-preserving* if it transforms complete populations to complete populations. Note that completeness of a set of element mutation operators, as introduced above, ensures this milder notion and, as mentioned above, can be argued statically.

Note also that the completeness of element mutation operator sets we introduced also allows a more free choice of initial populations. While an initial population can still affect the search, a complete set of element mutation operators ensures that, in principle, every feasible element can be reached, regardless of the initial population chosen.

In our evaluation, we investigate whether the use of a complete set of element mutation operators supports finding optimal solutions more efficiently and effectively. While we believe that soundness and completeness are interesting properties, they also serve to show that the formal framework we developed in the previous section allows one to define and reason about such properties.

3.6 Effective and efficient algorithms

A major concern in the configuration of evolutionary algorithms is to make them solve optimization problems effectively and efficiently. To that end, we want to investigate how the choice of sound and/or complete sets of element mutation operators affects the effectiveness and efficiency of evolutionary algorithms. To compare evolutionary algorithms in this regard, we start with the definition of quality relations based on our formal framework. Due to their probabilistic nature, evolutionary algorithms cannot be directly compared regarding our definition of their semantics. Instead, we compare execution batches that represent evolutionary computations that result from conducting experiments.

Definition 3.10 (Quality relation)**.** Given a problem instance PI for an optimization problem $\mathscr{P}$, a *quality relation* $\leq_{\mathrm{Q}}$ is a total preorder over multi-sets over $\mathscr{E}(PI)$. Let further two evolutionary algorithms $\mathscr{A}_i$ for PI and two corresponding execution batches EB_i, with $i = 1,2$, be given. Then algorithm $\mathscr{A}_1$ has a *better or equal quality than* $\mathscr{A}_2$ w.r.t $\leq_{\mathrm{Q}}$ and the considered execution batches if $EB_2 \leq_{\mathrm{Q}} EB_1$.

In the following, we show examples for quality relations. To make statements about the effectiveness and efficiency of evolutionary computations, evolutionary computations are compared with respect to a so-called quality indicator, which relates populations according to their quality.

Definition 3.11 (Quality indicator)**.** Given an optimization problem $\mathscr{P}$, a problem instance PI for $\mathscr{P}$, and a set T endowed with a total order $<$, the *quality indicator* $I : \mathscr{Q}(PI) \to T$ is a function that assigns a value $I(Q)$ in T to each population $Q \in \mathscr{Q}(PI)$.

In a multi-objective setting, a quality indicator should be Pareto-compliant such that it does not contradict the order induced by the dominance relation. Pareto compliance means the following: For two populations Q_1 and Q_2, if E_1 dominates E_2 and E_2 does not dominate E_1 for all search space elements $E_1 \in Q_1$ and $E_2 \in Q_2$, then $I(Q_1) < I(Q_2)$. Typically, the hypervolume indicator is used as quality indicator; it is known to be Pareto-compliant [ZBT07].

In practice, algorithms often need to be compared by performing a limited number of optimizations; one uses statistical methods to validate conclusions drawn from the generated execution batches. Consequently, as in the following examples, quality relations can be restricted to execution batches that are non-empty, finite, of equal size, and contain only finite evolutionary computations.

Example 3.9 (Effectiveness as quality relation)**.** The effectiveness of an algorithm can be considered in each iteration of an evolutionary computation, with the last iteration usually being the most interesting. To capture the effect of outliers, non-robust measures (e.g., *mean*) may be specifically favored over robust ones (e.g., *median)* when aggregating quality indicator values from multiple populations. In this case, it may also be useful to consider the standard deviation of quality indicator values. Low deviations indicate greater robustness of the computations than higher deviations.

For a problem instance of a multi-objective optimization problem, we use a normalized hypervolume indicator $h : \mathscr{Q}(PI) \to [0,1]$ to determine the quality of a population. Two vectors are used for its calculation. A vector consisting of the worst values found for each objective (also known as *nadir point*). And an artificial Pareto optimum consisting of the best values found for each objective (called *ideal point*). While the nadir point, which is degraded by a fixed value of 1 for each objective, is used as the reference point for calculating the area of the search space dominated by a solution, the ideal point is used to normalize the results. For the construction of both vectors, the approximation sets of the populations of the last iterations of all evolutionary computations generated by all evolutionary algorithms for this specific problem instance are considered.

Let mean and sd be functions that compute the mean and standard deviation of a set of real values, respectively. Given $h_{\text{last}}(Q_0 Q_1 \dots Q_k) = h(Q_k)$ for $Q_0 Q_1 \dots Q_k \in \mathscr{E}(PI)$ and $h_n(Q_0 Q_1 \dots Q_n \dots Q_k) = h(Q_n)$ for $0 \leq n \leq k$,

$h_{\text{last}}(Q_0 Q_1 \dots Q_k)$ otherwise (i.e., given projections onto the hypervolume of the last or the n-th population of a finite evolutionary sequence), we define three quality relations to compare the effectiveness of two algorithms $\mathscr{A}_1(PI, OP)$ and $\mathscr{A}_2(PI, OP)$ regarding two execution batches EB_1 of $\mathscr{A}_1$ and EB_2 of $\mathscr{A}_2$ with $EB_1, EB_2 \subseteq \mathscr{E}(PI)$.

- The *mean-effectiveness* $\leq_{\text{mean}}$ with $EB_2 \leq_{\text{mean}} EB_1$ if $\text{mean}\{h_{\text{last}}(e_2) \mid e_2 \in EB_2\} \leq \text{mean}\{h_{\text{last}}(e_1) \mid e_1 \in EB_1\}$,
- the *mean-n-effectiveness* $\leq_{\text{mean},n}$ with $EB_2 \leq_{\text{mean},n} EB_1$ if $\text{mean}\{h_n(e_2) \mid e_2 \in EB_2\} \leq \text{mean}\{h_n(e_1) \mid e_1 \in EB_1\}$ for $n > 0$, and
- the *sd-effectiveness* $\leq_{\text{sd}}$ with $EB_2 \leq_{\text{sd}} EB_1$ if $\text{sd}\{h_{\text{last}}(e_2) \mid e_2 \in EB_2\} \leq \text{sd}\{h_{\text{last}}(e_1) \mid e_1 \in EB_1\}$.

Example 3.10 (Efficiency as quality relation)**.** With respect to the efficiency of evolutionary computations, the length of evolutionary sequences, i.e., the number of iterations needed to satisfy the termination condition, and the mean runtime of these iterations are of interest. For the latter, let a runtime function $rt : \mathscr{E}(PI) \to \mathbb{R}$ be given which computes the mean runtime in ms per iteration for an evolutionary sequence. Analogously to Ex. 3.9 we define two quality relations to compare efficiency of two algorithms $\mathscr{A}_1(PI, OP)$ and $\mathscr{A}_2(PI, OP)$ regarding two execution batches EB_1 of $\mathscr{A}_1$ and EB_2 of $\mathscr{A}_2$ with $EB_1, EB_2 \subseteq \mathscr{E}(PI)$.

- The *iteration-efficiency* $\leq_{\text{it}}$ with $EB_2 \leq_{\text{it}} EB_1$ if $\text{mean}\{\text{length}(e_1) \mid e_1 \in EB_1\} \leq \text{mean}\{\text{length}(e_2) \mid e_2 \in EB_2\}$,
- and the *runtime-efficiency* $\leq_{\text{rt}}$ with $EB_2 \leq_{\text{rt}} EB_1$ if $\text{mean}\{\text{rt}(e_1) \mid e_1 \in EB_1\} \leq \text{mean}\{\text{rt}(e_2) \mid e_2 \dots \in EB_2\}$.

3.7 Evaluation

In Section 3.5, we briefly discussed how sound and complete operator sets may influence evolutionary computations. Consequently, we also expect these operator sets to have an impact on the outcome of optimizations in quantitative evaluations. To investigate this assumption, we conduct experiments focusing on the following two research questions (subscripts are indicating the associated chapter):

C_3Q1 Does the soundness of element mutation operators have an impact on the effectiveness or efficiency of evolutionary algorithms?

C_3Q2 Does the completeness of the set of element mutation operators have an impact on the effectiveness or efficiency of evolutionary algorithms?

In the following, we first introduce the context and use cases of the evaluation according to the structure presented in our framework. Then, we present the evaluation setup and results. All evaluation data can be found at [Joh+23b]. They include the results of our experiments and all artifacts needed to reproduce them.

3.7.1 Implementation of the framework

To conduct the experiments, we use MDEOptimiser [BZS18; mdeo], a tool that implements the framework presented. It allows users to configure, instantiate, and run evolutionary algorithms as defined in our framework. MDEOptimiser relies on the MOEAFramework [moea] to provide a selection of *evolutionary algorithms* from which users can choose. The algorithm implementations provide the *computation specification* and the *selection operators*. They also predefine most parts of the *population mutation operators*. However, the user can choose between different variants of how to apply element mutation operators when a solution needs to be evolved. These variants differ, for example, in how they select which element mutation operators to apply and how many element mutations to use. Furthermore, the *set of element mutation operators* used by the population mutation operators can be specified.

The realization of the *computation space* is based on the Eclipse Modeling Framework (EMF) [Ste+08; emf]. Accordingly, the *computation meta-models* and *problem instances* are implemented as EMF Ecore and instance models, respectively. By default, an *initial population* is created by first replicating the provided problem instance. Each replica is then modified by two applications of the specified population mutation operator (ignoring parent selection). Alternatively, a user can provide an initialization procedure implemented in Java to generate a user-defined initial population. *Feasibility constraints* and *objective relations* are induced from constraint functions and objective functions

implemented in OCL or Java. The user must specify a set of *element mutation operators* as rules or units of the Henshin model transformation language [Are+10]. Henshin units allow complex transformations to be composed of multiple rules using control flow elements. Regarding the *termination condition*, the user can choose between predefined variants.

3.7.2 Optimization problems selected

The evaluation considers three multi-objective problems: the Class Responsibility Assignment [BBL10; MJ14; FTW16; BZJ21] problem (CRA), the problem of Scrum Planning [BZJ21] (SCRUM), and the Next Release Problem [BRW01; BZJ21] (NRP). The reasons for this choice are twofold. First, these use cases cover different aspects that might be relevant for optimization problems in MBMDO since they differ in their structural complexity, the number and kind of their feasibility constraints (more structural or more attribute-oriented), and the number and complexity of element mutation operators needed to perform meaningful optimizations. Second, the number and complexity of the required element mutation operators is low enough to allow for their (partially manual) analysis with regard to soundness and completeness. In the absence of real world examples, all problem instances of the chosen use cases were generated.

The same use cases were already considered by Burdusel et al. [BZJ21]. However, some adjustments to the objectives and constraints were necessary to allow comparison of operator sets implementing different degrees of soundness and completeness. Therefore, the evaluation results are not directly comparable to those in [BZJ21].

The computation space of each of the optimization problems is defined by an EMF meta-model. The meta-model of the running example is shown in Fig. 3.1. The meta-models of the other optimization problems can be found in Appendix A.2. Since the experiments are based on EMF as the underlying modeling framework, EMF-specific language constraints apply to all of the following optimization problems: Each computation model has exactly one root node that (transitively) contains all other nodes, each node is contained in at most one container, there are no containment cycles, and there are no parallel edges of the same type between two objects. In our use cases, the root node is always part

of the problem model. Thus, its unique existence is even a problem constraint. The SCRUM and NRP cases demand further language constraints. Here, we will only discuss constraints stemming from the semantics of the respective optimization problem. For a discussion of constraints tightly connected to the design of the meta-model please also refer to Appendix A.2.

CRA. Our running example (Section 3.2), the CRA case, also serves as the initial use case for the experiments. In summary, in this case, a unique class model serves as the root node to which classes can be added. The assignment of interdependent features to classes must be done in compliance with two feasibility constraints: each feature must be assigned to one class and must not be assigned to more than one class. The objectives are to minimize the coupling between classes and maximize their cohesion. In the experiments, problem instances range from 9 features and 14 dependencies (Model A) to 160 features and 600 dependencies (Model E), as introduced in [FTW16]. All problem instances are infeasible because none of their features are initially assigned.

SCRUM. Scrum [Rub12] is an agile software development technique. In Scrum, a software product is defined by a set of work items each representing a feature desired by one of the project stakeholders. The work items of all stakeholders are collected in a backlog. Work items can be of varying importance to their stakeholder and can also differ in the estimated effort required to implement them. In order to partition the development of all features into manageable units, the work items are assigned to so-called sprints. Sprints represent timed iterations and are implemented one after the other.

The SCRUM case goes back to the idea of using Scrum to manage maintenance tasks for existing projects. In this scenario, sprints are not decided on in an ad hoc basis. Instead, an optimal plan for distributing the work items across the sprints must be found. Two objectives are considered when evaluating the quality of a solution. First, the total effort of all work items should be distributed evenly across sprints by minimizing effort variance. Second, the requirements of each stakeholder should be distributed evenly across the sprints in terms of their importance. For each stakeholder, the variance in the sum of the importance of their work items in each sprint is considered. The objective is implemented

by minimizing the average of this deviation across all stakeholders. Each work item must be assigned to exactly one sprint (a feasibility constraint similar to class assignment in the CRA case). Additionally, a sprint should neither be too easy nor too complex. Thus, the plan must respect a minimum and maximum number of desired sprints for each problem instance. These limits are expressed via additional feasibility constraints. For reasonable problem instances, the minimum and maximum must be at least zero and must not be greater than the number of available work items. Additionally, the minimum must be less than or equal to the maximum. These requirements are formulated as problem constraints.

In the experiments, we consider two problem instances: Model A with 5 stakeholders and 119 work items and Model B with 10 stakeholders and 254 work items.

NRP. When planning the next release of a software system, customer satisfaction must be weighed against the costs associated with developing new software artifacts. This struggle is known as the Next Release Problem [BRW01]. In the formulation considered here, requirements may be abstract and depend on other requirements. By assigning a value to a requirement, customers can specify how important a requirement is for them. A requirement can be satisfied to varying degrees by different sets of software artifacts. In addition, some customers may be more important to the software development company than others. Ultimately, software artifacts may depend on each other and complex dependency hierarchies may even emerge. Each software artifact has a development cost associated with it. The cost of the next release is determined by the sum of the costs of all artifacts selected for that release.

The goal of the NRP case is to select a set of software artifacts while considering two conflicting objectives: minimizing the total development cost of the selection while maximizing customer satisfaction by satisfying the requirements. The dependency hierarchy between software artifacts imposes a structural feasibility constraint; an artifact can only be selected for the next release if all of its (transitive) dependencies are also selected. In addition, a predefined budget limits the cost allowed for a feasible solution. In reasonable problem instances no cyclic dependencies may occur in the dependency structure of requirements

or software artifacts. Additionally, the values assigned to requirements, the costs associated with software artifacts, the degrees to which sets of software artifacts satisfy requirements, and the importance of customers need to be positive. These requirements constitute problem constraints of the NRP case.

Experiments are conducted with three models that differ in size and complexity of the underlying dependency hierarchy. Model A contains 25 customers, 25 requirements, and 200 software artifacts with a median of 11 (transitive) dependencies per artifact. Model B has twice as many requirements and a slightly higher median of 15 (transitive) dependencies per artifact. Also, Model C contains twice as many artifacts (25 customers, 50 requirement, 400 software artifacts) and a median of 16 (transitive) dependencies per artifact.

3.7.3 Evolutionary operators

In MDEOptimiser the selection operators and most parts of the population mutation operators are given by the choice of the evolutionary algorithm used to perform an optimization (as described in Section 3.7.1). Therefore, we will discuss this choice in the next section. However, regardless of the used evolutionary algorithm, in our experiments the evolution of a solution is always performed as presented in Ex. 3.4. A single arbitrarily chosen applicable element mutation operator is applied. If none of the available element mutation operators is applicable, the solution remains unchanged. In the following, we present the sets of element mutation operators used by the population mutation operators.

For each use case, we consider three variants of sets of element mutation operators: a set *SC* that is *sound and complete*, a set *SIC* that is *sound but incomplete*, and a set *UC* that is *unsound but complete*. We checked each set for soundness and completeness. Since all the element mutation operators are formalized as transformation rules in Henshin, we used the tool *OCL2AC* [Nas+18; Nas+20] to support the soundness checks for all three cases. With regard to the common EMF-specific language constraints, we designed our rules according to the guidelines developed in [BET12], which were proven there to be sufficient conditions to preserve these constraints. Additionally, the implementation of the transformation language Henshin does not support parallel edges of the same type, i.e., attempts to introduce a parallel edge are ignored. Therefore, we do not

need additional negative application conditions (as proposed in Ex. 3.3) for our element mutation operators to preserve the respective language constraint. As the root nodes are part of the problem model in all use cases, Proposition 3.1 and Lemma 3.1 guarantee their unique existence in all solutions throughout the evolutionary computations. Overall, all of the following element mutation operators are lc-preserving with regard to the EMF-specific language constraints.

The operator sets used in the experiments are discussed in detail using the CRA case as example. For the other use cases, please refer to Appendix A.2 for more illustrations and details regarding their preservation of further language constraints, their soundness, and their completeness.

CRA. Figure 3.8 shows the element mutation operators considered in the CRA case as Henshin rules in graphical syntax. Nodes and edges are preserved, deleted, created, or forbidden as annotated (and encoded via a color scheme). Apart from the EMF-specific language constraints no language constraints are specified for the CRA case. Therefore, all operators are *lc-preserving*.

The *SC* variant includes five operators. The operators 3.8a and 3.8b can only add an assignment to a feature. To prevent features from being assigned to multiple classes, application conditions ensure that only features that have not yet been assigned are considered. If a feature is already assigned to a class, the operators 3.8c (introduced in Ex. 3.4) and 3.8d can move it to another class. The new assignment replaces the existing one in each case. In doing so, features cannot be assigned to more than one class, nor can they remain unassigned. Since the last operator (3.8e) only deletes empty classes and does not change assignments, all operators in the set are *sound*. For a given solution model, any feature assignment can be reached as argued in Ex. 3.8. Consequently, the set *SC* is *complete*.

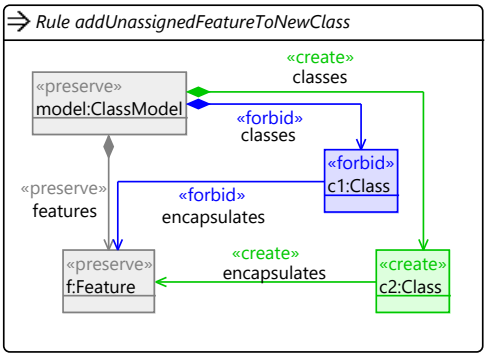
The operator set *SIC* corresponds to the operator set introduced in Section 2.3. It largely coincides with the SC variant, but the designer forgot to implement the operator 3.8d, which moves a feature to a new class. Since this is a subset of SC, the set is also *sound*. However, once all features have been assigned to classes, no new classes can be created. Therefore, the set is *not complete*.

UC shares with its competitors only the deletion operator. Unlike their SC counterparts (3.8a and 3.8b), the operators responsible for adding assignments (3.8f and 3.8g) do not check if a feature is already assigned. Their application may result in features being assigned to multiple classes. The operator 3.8h can be used to unassign a feature which may result in unassigned features. Since both feasibility constraints can be violated, the set UC is *not sound*. By removing an assignment of a feature and reassigning it to another class, both move rules of the SC variant can be mimicked. As the operators for adding assignments are also more general than in the SC variant, the argumentation for completeness presented in Ex. 3.8 can also be adapted to the UC variant. Thus, the set is *complete*.

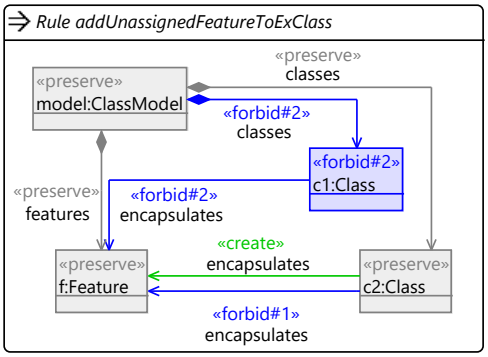
SCRUM. Conceptually, the SCRUM and CRA cases are similar: Certain objects (work items/features) must be assigned to containers (sprints/classes). Therefore, it is not surprising that the operator sets also have similarities. The operator set for SCRUM is shown in Fig. A.5. The *SC* variant of the SCRUM case contains two operators for adding unassigned work items to new or existing sprints. Two other operators move work items from one sprint to another (existing or new) sprint. Empty sprints can be deleted by the last operator. The reasoning about completeness and soundness, at least with respect to the feasibility constraints related to the assignment of work items, is analogous to the CRA case. In addition, rules that create/delete new sprints are only applied when the maximum/minimum number of allowed sprints has not yet been reached.

In the *SIC* variant, we again simulate a designer who forgot to design an operator (as in the CRA case). Unlike the CRA case, where at some point no new containers (i.e., classes) could be created, the rule for deleting containers (i.e., sprints) is completely missing. Otherwise, the set is equal to SC. Obviously, this set is not complete. For example, if all solutions in the population have reached the maximum number of sprints allowed, there is no way to create new solutions with fewer sprints.

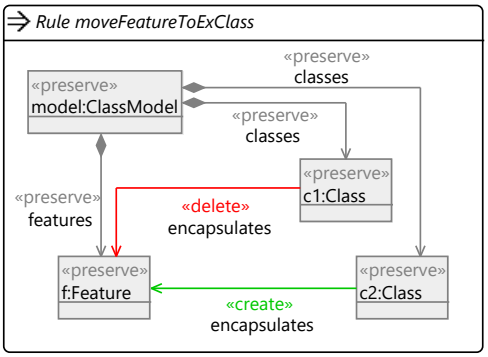
The set *UC* is similar to the SC variant. However, the operators that create or delete sprints do so regardless of the current number of sprints in a solution.



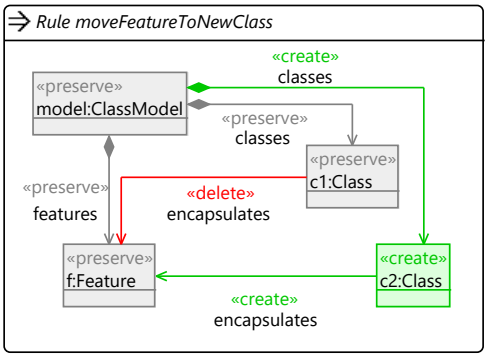
(a) addUnassignedFeatureToNewClass (SC, SIC)



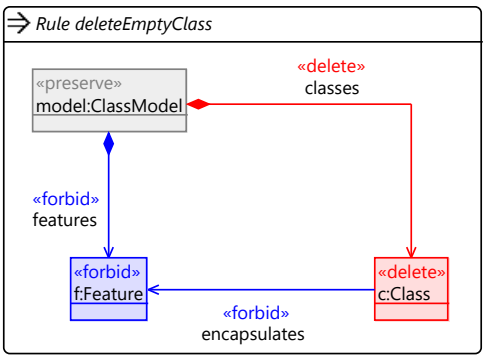
(b) addUnassignedFeatureToExClass (SC, SIC)



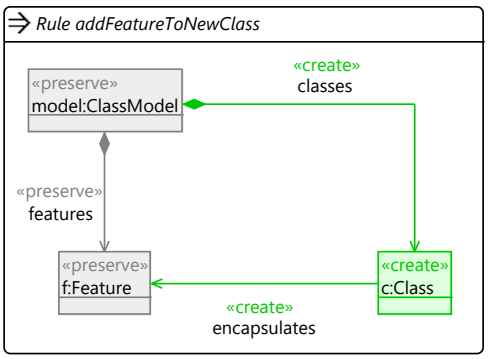
(c) moveFeatureToExClass (SC, SIC)



(d) moveFeatureToNewClass (SC)

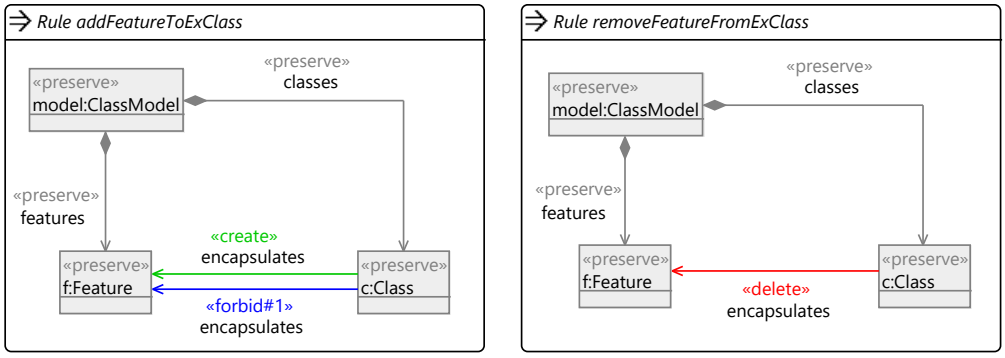


(e) deleteEmptyClass (SC, SIC, UC)



(f) addFeatureToNewClass (UC)

Figure 3.8 – The set of all element mutation operators used for the CRA case. The operator set variants to which an operator belongs are given in parenthesis. (part 1/2)



(g) addFeatureToExClass (UC) **(h)** removeFeatureFromExClass (UC)

Figure 3.8 – The set of all element mutation operators used for the CRA case. The operator set variants to which an operator belongs are given in parenthesis. (part 2/2)

NRP. Each of the operator set variants in the NRP case (in Fig. A.7) contains two element mutation operators, one for adding an artifact to a release and one for removing an artifact. To not violate the constraints on the dependency hierarchy of artifacts, the *SC* variant follows a bottom-up construction of a release. Starting from artifacts without dependencies, artifacts are added only if all their dependencies are already part of the release. Complementarily, artifacts are removed in a top-down manner.

SIC shares the bottom-up approach for adding artifacts. However, important artifacts (representing a dependency for three or more other artifacts) are never removed once they are in the release. Compared to SC, the rules in SIC are more restrictive and therefore this set is still sound. However, it is not complete. If an important artifact is added, feasible solutions without that artifact can no longer be reached.

The *UC* variant randomly adds and removes artifacts, regardless of their dependency structure. Randomly, all feasible and infeasible solutions can be reached at any time.

3.7.4 Evolutionary algorithms

We consider three common evolutionary algorithms: NSGA-II [Deb+02], PESA-II [Cor+01], and SPEA2 [ZLT01]. NSGA-II is most commonly used in the

MDO literature and is also well represented in the literature combining SBSE and MDE [BSA17]. Therefore, we already used it to exemplify our framework in Section 3.4. It uses the population mutation operator explained in Ex. 3.4 and the survival selection operator discussed in Ex. 3.5. Its computation specification is described in Ex. 3.6. For details on PESA-II and SPEA2, we refer the reader to [Cor+01] and [ZLT01], respectively. All algorithms are used with the population size and termination condition of Ex. 3.6. The initial population is always generated by mutating replicas of the problem instance twice, the standard initialization procedure of MDEOptimiser.

For PESA-II, two additional parameters must be specified: the size of an archive of non-dominated solutions and the number of regions into which the search space is partitioned when selecting parents. In agreement with [Cor+01], we use an archive of 100 elements and 32 regions. For SPEA2, an additional factor k (to estimate the uniqueness of solutions) must be specified. For performance reasons and in accordance with a recommendation of the MOEA framework, we set $k = 1$.

For each use case, each evolutionary algorithm is run in three configurations depending on the operator sets used: an SC, SIC, and UC variant. According to Def. 3.7, we denote each of these configurations by $\mathscr{A}(PI_{\mathscr{P}}, OP_{\mathscr{P}})$, where $\mathscr{A}$ is the name of the evolutionary algorithm, $PI_{\mathscr{P}}$ indicates a problem instance of the optimization problem $\mathscr{P}$, and $OP_{\mathscr{P}}$ uniquely identifies the set of evolutionary operators. Although each set of evolutionary operators contains not only mutation operators but also a selection operator, we name the set of operators only after the mutation operators since the selection operator is always the same. For ease of reading, the name of the algorithm is given by its initial letter only. For example, $N(A_{\text{CRA}}, SC_{\text{CRA}})$ represents NSGA-II applied to model A of the CRA case using the sound and complete operator set for the CRA case. For each of our use cases, we run an execution batch of 30 executions for each combination of problem instance and operator set.

3.7.5 Effectiveness and efficiency

For each evolutionary algorithm separately, we compare the variants of the mutation operator sets with respect to the quality relations presented in the

Table 3.1 – For each use case, problem instance, and configuration based on NSGA-II, the following is presented: the mean normalized hypervolume (for which higher values are better), the standard deviation with respect to this mean, the mean number of iterations required to terminate and the mean runtime in ms required for an iteration (with lower values being better).

		CRA					SCRUM		NRP		
	Metric	A	B	C	D	E	A	B	A	B	C
SC	Hypervolume	0.904	0.878	0.797	0.709	0.703	0.959	0.937	0.434	0.745	0.775
	SD	0.007	0.03	0.03	0.043	0.05	0.011	0.028	0.001	0.007	0.003
	Iterations	127	289	578	993	1250	407	497	174	184	285
	ms/Iteration	31	34	89	651	4098	730	4223	252	326	454
UC	Hypervolume	0.506	0.24	0.148	0.055	0.026	0.915	0.831	0.402	0.738	0.735
	SD	0.079	0.111	0.055	0.021	0.012	0.018	0.043	0.037	0.006	0.034
	Iterations	119	131	164	244	377	461	556	238	445	489
	ms/Iteration	26	31	54	177	515	527	2985	222	298	434
SIC	Hypervolume	0.881	0.809	0.757	0.661	0.676	0.942	0.902	0.588	0.734	0.766
	SD	0.039	0.068	0.043	0.054	0.045	0.014	0.028	0.042	0.014	0.018
	Iterations	121	274	481	769	992	352	452	150	163	269
	ms/Iteration	31	34	84	456	2704	699	3822	255	331	452

examples 3.9 and 3.10. This means that we do not compare the results between different evolutionary algorithms; for example, we do not compare an algorithm configuration based on NSGA-II with a configuration based on PESA-II or SPEA2. This allows us to attribute the differences in results to a single parameter: the differences in mutation operator sets. Using the hypervolume [ZBT07] as a quality indicator for each algorithm configuration and problem instance, we recorded the mean effectiveness of the last population, the standard deviation from this mean, the mean length of evolutionary computations performed, and the mean runtime per iteration. Table 3.1 summarizes the measurements for all configurations based on NSGA-II. Since the results for the other algorithms are very similar, we will not list them explicitly here, but will discuss the observed differences.

In general, the results of optimization algorithms cannot be assumed to be normally distributed. Therefore, pairwise *Mann-Whitney U tests* [MW47] on the mean effectiveness of the last populations are performed to test the significance (with p-value < 0.05) of the differences between two algorithm configurations. We analyze the effect size of the observed differences using *Cliff's delta* [Cli93], which takes values between 0.0 (no effect at all) and ± 1.0 (the hypervolume of

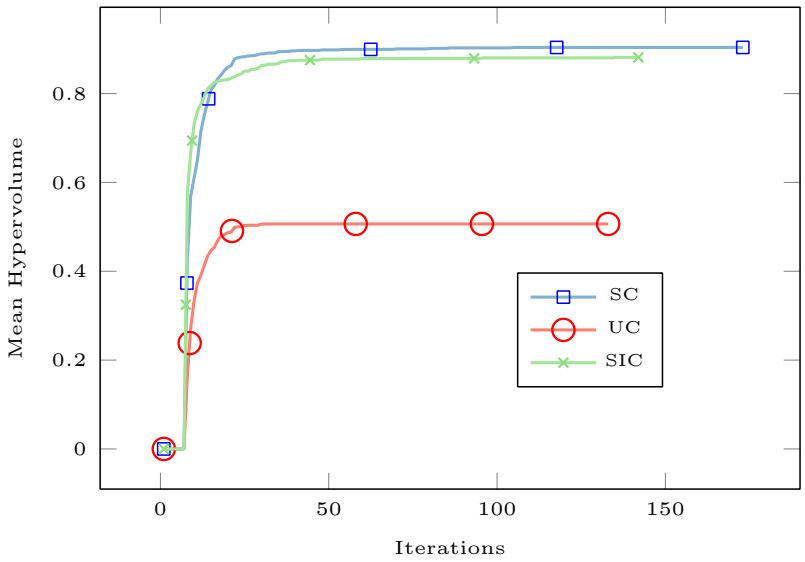
each evolutionary computation of one algorithm configuration is larger than the hypervolumes of all evolutionary computations of the other). We will use only positive values and rely on context to clarify which algorithm configuration is the dominating one.

To discuss the *mean-n-effectiveness*, for NSGA-II based configurations, Figures 3.9 to 3.11 show the *development of the mean hypervolume* for the smallest and largest problem instance of each use case, respectively. Consistent with Ex. 3.9, the mean hypervolume of each iteration of an algorithm configuration is computed with regard to all populations generated in that iteration in all evolutionary computations of the respective configuration. Figure 3.12 illustrates, as an example, the *cumulative approximation sets* achieved by each NSGA-II-based algorithm configuration for Model E of the CRA case and Model A of the NRP case. A cumulative approximation set is formed by combining the approximation sets of the last populations of all evolutionary computations of an algorithm configuration into one set of non-dominated solutions. Note that the diagrams in Figure 3.12 depict objective values along the axis and not the normalized values used for the hypervolume calculation. Also, note that the cumulative approximation sets shown in no way reflect how many evolutionary computations of an algorithm configuration yielded a particular solution.

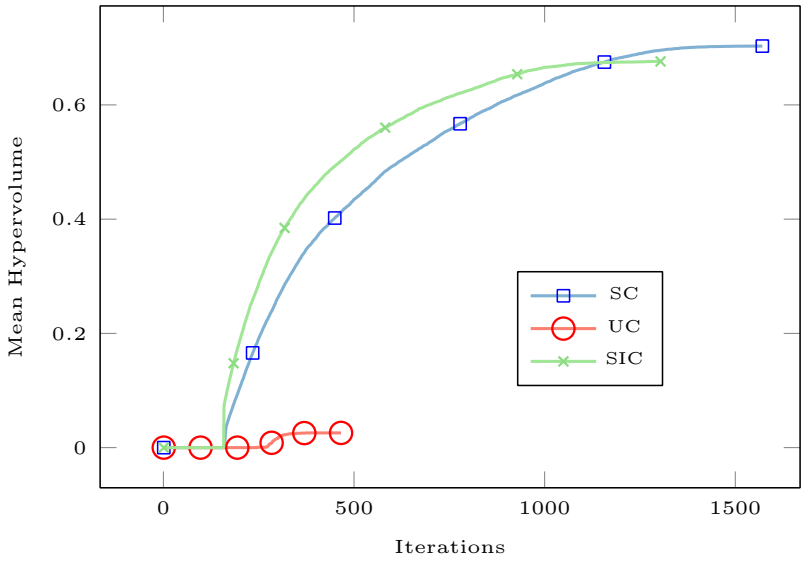
3.7.6 Results

In the following, we compare the UC and SIC variants of all evolutionary algorithms with their respective SC variant for all cases.

UC effectiveness. Regardless of the evolutionary algorithm, the UC variant turns out to be less effective than its two sound counterparts in terms of mean-effectiveness in almost all cases. Model B of the NRP case seems to be a special case, as UC is almost on par with SC and SIC here. Only the configurations $N(B_{\text{NRP}}, UC_{\text{NRP}})$, $P(A_{\text{NRP}}, UC_{\text{NRP}})$, $P(B_{\text{NRP}}, UC_{\text{NRP}})$, and $P(C_{\text{NRP}}, UC_{\text{NRP}})$ are better than their respective SIC variants. The differences between a UC variant and its corresponding SC and SIC variants are significant with a few exceptions. These are the SC variants $P(A_{\text{NRP}}, SC_{\text{NRP}})$, $P(B_{\text{NRP}}, SC_{\text{NRP}})$, and $S(A_{\text{NRP}}, SC_{\text{NRP}})$ as well as the SIC variant $S(B_{\text{NRP}}, SIC_{\text{NRP}})$.

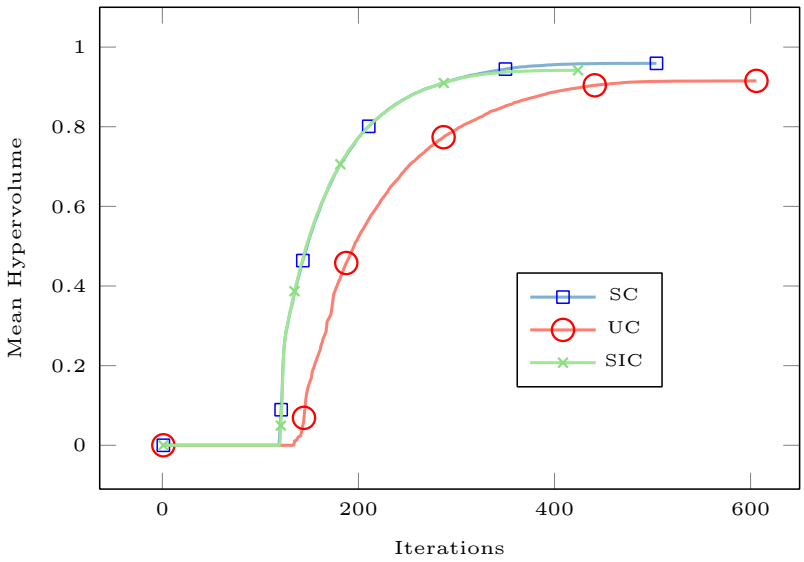


(a) CRA A

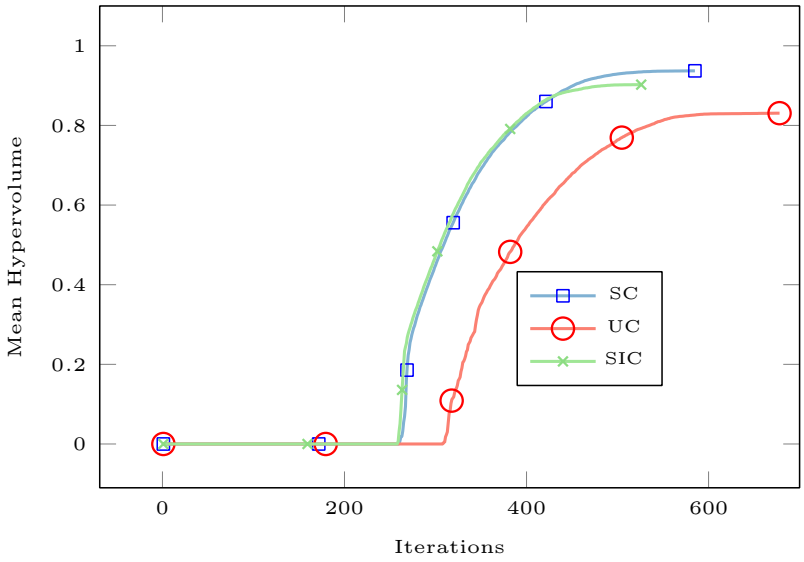


(b) CRA E

Figure 3.9 – Development of the mean hypervolume (defined in Section 3.7.5) for all NSGA-II based algorithm configurations for problem instances A and E of the CRA case.

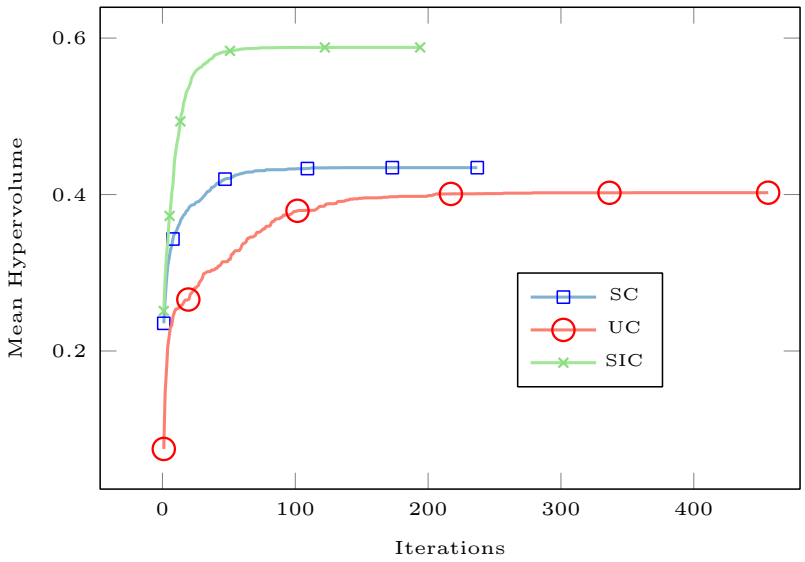


(a) SCRUM A

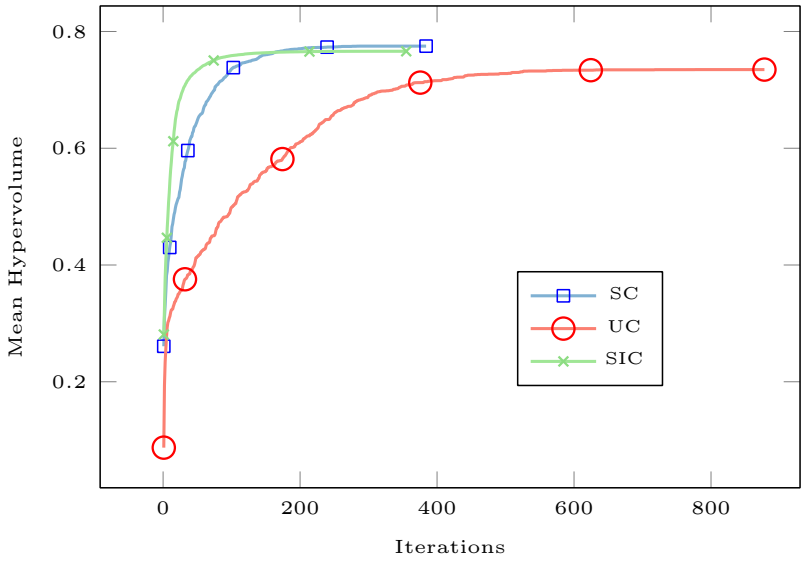


(b) SCRUM B

Figure 3.10 – Development of the mean hypervolume (defined in Section 3.7.5) for all NSGA-II based algorithm configurations for problem instances A and B of the SCRUM case.

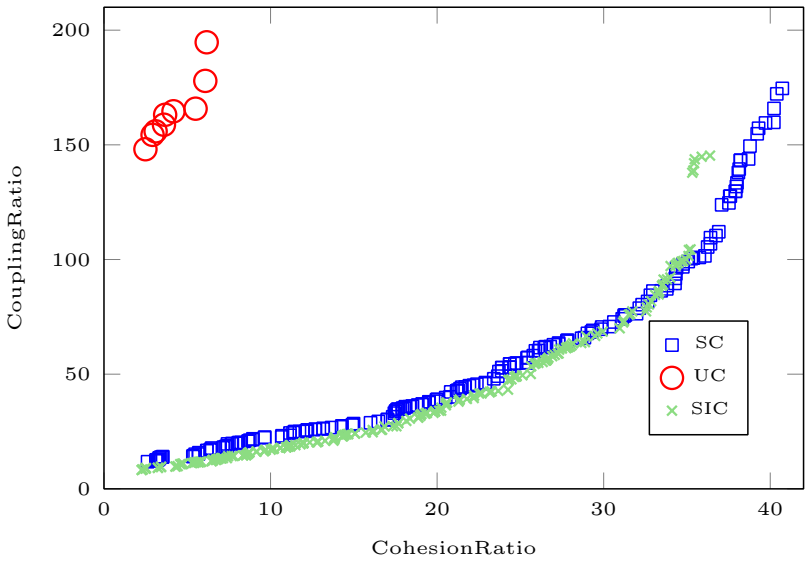


(a) NRP A

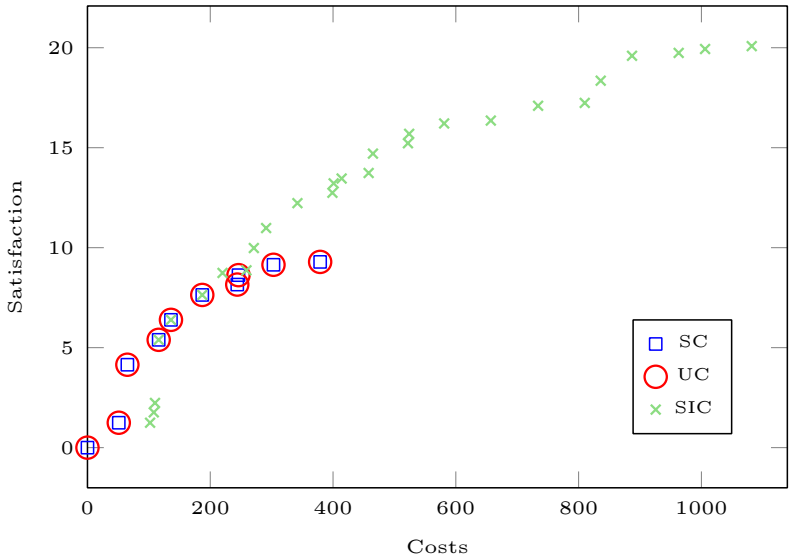


(b) NRP C

Figure 3.11 – Development of the mean hypervolume (defined in Section 3.7.5) for all NSGA-II based algorithm configurations for problem instances A and C of the NRP case.



(a) CRA E



(b) NRP A

Figure 3.12 – Cumulative approximation sets (defined in Section 3.7.5) for all NSGA-II based algorithm configurations for model E of the CRA case and model A of the NRP case, respectively.

Apart from the few insignificant differences, high effect sizes can be observed. For all problem instances of the CRA case, the highest possible effect size of 1.0 is achieved regardless of the algorithm used.

Apart from the NRP case where UC eventually surpasses SIC when PESA-II is used, UC always shows the lowest mean-n-effectiveness. For larger models of the CRA case, UC stands out with a low standard deviation. However, considering the poor quality of the solutions found, the robustness indicated by the low standard deviation can hardly be considered an advantage.

UC efficiency. In the CRA case, UC trades effectiveness for efficiency and requires far fewer iterations than the other variants (clearly seen in Fig. 3.9b). The iterations are also performed faster. In the NRP and SCRUM cases, UC requires more iterations than its competitors and its runtime efficiency is closer to (in some PESA-II and SPEA2 cases even worse than) that of SC and SIC. In general, UC converges more slowly than the other variants, i.e., the hypervolume improves in smaller steps (most evident in Fig. 3.11b).

SIC effectiveness. Comparing the incomplete variant SIC with its complete counterpart SC, the situation is not as clear as for UC. The result differs between the use cases and even between problem instances of the same use case. In terms of mean-effectiveness, SC is significantly better than SIC for models B, C, and D of the CRA case regardless of the evolutionary algorithm used. Effect sizes for models B and C lie between 0.55 to 0.96; for model D they range from 0.33 with SPEA2 to 0.55 with NSGA-II. For models A and E of the CRA case, the differences are not significant regarding NSGA-II and PESA-II; however, $S(E_{\mathrm{CRA}}, SIC_{\mathrm{CRA}})$ outperforms its SC counterpart significantly.

For both models of the SCRUM case, SC outperforms SIC. However, the differences are only significant for NSGA-II and SPEA2. In the NRP case, the differences between SC and SIC are significant except for $S(C_{\mathrm{NRP}}, SIC_{\mathrm{NRP}})$. SC performs slightly better than SIC in most cases (with effect sizes up to 1.0 for PESA-II and medium effect sizes for NSGA-II and SPEA2). However, $N(A_{\mathrm{NRP}}, SIC_{\mathrm{NRP}})$ and $S(A_{\mathrm{NRP}}, SIC_{\mathrm{NRP}})$ surpass their respective SC variants with effect sizes close to 1.0.

Looking at the standard deviation, SC produces slightly more consistent results in most cases. Regarding the mean-n-effectiveness, SIC performs better than SC at the beginning of an optimization in the CRA and NRP cases. For the SCRUM case, both variants behave nearly identically.

SIC efficiency. Most of the time SIC requires fewer iterations than SC to terminate. Only when using PESA-2 to solve the NRP case, it is the other way around. Although this result is not well reflected in Table 3.1, there is no clear winner in terms of runtime; the winner depends heavily on the algorithm considered and the problem instance.

3.7.7 Discussion

UC. Since unsound operators allow the introduction of constraint violations, their effect on optimization depends heavily on the constraint handling mechanisms of the underlying evolutionary algorithm. As described in Ex. 3.5, the selection operator of NSGA-II discriminates hard against infeasible solutions. The same holds for PESA-II and SPEA2. As a result, infeasible solutions in the population are discarded as soon as a feasible substitute is found. Especially in scenarios where the optimization starts from a population containing feasible or nearly feasible solutions (as in the NRP case), the introduction of constraint violations caused by unsound operators usually wastes evolution steps. The resulting infeasible solutions are replaced early on by newly found or existing feasible solutions. This behavior cannot only slow down the optimization process (as seen in Fig. 3.11) but also increases the probability of getting stuck in local optima (see Fig. 3.9).

The extent to which negative effects of an unsound operator become apparent depends on other factors: (1) how often and at which stage of an optimization it is applicable, (2) whether it necessarily produces infeasible solutions, and (3) whether other operators are given a chance to counteract its negative effects. In the CRA case, the unsound operators addFeatureToNewClass, addFeatureToExClass, and removeFeatureFromExClass are all applicable to the vast majority of solutions. At the beginning of an optimization, only a few features have a

chance to be assigned to a class (most of them to exactly one class). Therefore, an application of removeFeatureFromExClass is likely to leave a feature unassigned. The more features are assigned, the more likely multi-assignments will occur when addFeatureToNewClass or addFeatureToExClass are applied. Although removeFeatureFromExClass can potentially resolve multi-assignments, the corresponding solutions are often discarded before such a resolution can occur due to the constraint handling used by the considered evolutionary algorithms. Since the combination of the discussed operators repeatedly leads to constraint violations in the course of an optimization, their negative effects are clearly reflected in the observed effectiveness of the UC variant. In contrast, in the SCRUM case, the negative effects of the unsound operators are less noticeable. The unsound operators for creating/deleting sprints do not necessarily lead to constraint violations as long as the maximum/minimum number of allowed sprints is not exceeded after their application. Furthermore, the effect of creating a sprint can be neutralized by subsequent deletion of a sprint and vice versa, before the respective limit is reached. Finally, one of the unsound operators that creates sprints can no longer be applied once all work items have been assigned.

In cases were UC is nearly as effective as its competitors, more iterations are required to compensate for the wasted mutations caused by unsound operators. On the other hand, UC also reaches the termination condition when it gets stuck in local optima. Therefore, fewer iterations are required than for the other variants. Checking the application condition of an operator and finding a match of the operator in the solution model can be a time-consuming task. Unsound operators restrict their applicability less than their sound counterparts. Consequently, UC is often the fastest variant in terms of time required per iteration.

SIC. The performance of the SIC variant depends largely on which part of the search space becomes unreachable due to the incompleteness of the operator set. When (near) optimal solutions become unavailable, the mean-effectiveness of SIC likely suffers (as for the CRA case in Table 3.1). Comparing the cumulative approximation sets generated by the NSGA-II variants for models C, D, and E of the CRA case, we found that SC generated solutions with high cohesion and high coupling that SIC could not find (e.g., Fig. 3.12a). We attribute this to the

inability of the SIC variant to create new classes once all features are assigned. Obviously, SIC was unable to maintain solutions with the number of classes necessary to induce such a high coupling. However, ignoring certain parts of the search space during the course of an optimization can also serve to take shortcuts, speed up the optimization process, or even overcome local optima. With NSGA-II, this can clearly be observed for model A of the NRP case (see Fig. 3.12b). While complete variants get stuck at solutions of low satisfaction, SIC manages to explore a much larger portion of the search space. For some reason, however, the same set of operators is less successful in SPEA2 and even worse than the other variants in PESA-II.

Note that the impact of an incomplete operator set may depend on the problem instance at hand. While SIC performs better than SC on model A of the NRP case, it is the opposite for the other problem instances. Moreover, small changes in the introduced incompleteness can have large effects. In the SIC variant of the NRP case, we do not allow an artifact to be removed if it serves as a dependency for $x = 3$ other artifacts. Choosing other values for x (e.g., 2 or 4), we found that the aforementioned superiority of SIC for Model A disappears completely.

Summary. In summary, we observe that the use of unsound operator sets in our experiments has a mostly negative impact on optimization (C_3Q1). While they are sometimes more efficient than their sound counterparts, their lack of effectiveness outweighs this advantage. An effect on the optimization was also observed for the completeness of operator sets (C_3Q2). In our use cases, the incomplete variants were slightly more efficient than the sound and complete variants, but less effective in most cases. However, in a few cases, they managed to perform significantly better than their competitors regarding effectiveness.

3.7.8 Threats to validity

Although the selected use cases already differ in various aspects covering a range of problems relevant in practice, they represent only a small part of the broad spectrum of optimization problems in software engineering. While the problem models of the considered cases have rather complex structures, the complexity of the solution parts is rather limited. Consequently, the complexity of changes

made by element mutation operators is also limited; a fact that allows us to reason about the soundness and completeness of the operator sets in the first place. Unfortunately, only a relatively small number of use cases have been prepared for MB-MDO so far, i.e., with problem instances formalized over metamodels and element mutation operators implemented as model transformations; elaborating a suitable use case is a non-trivial task. Other optimization problems, including those that allow for more complex mutations, will need to be addressed in the future.

While the use cases considered reflect practical software engineering problems, all problem instances were generated. The extent to which these are representative of real-world cases remains unknown. We have attempted to address this problem by considering problem instances of varying sizes and complexity for each use case.

We attribute the observed behavior of the mutation operator sets to their soundness and completeness. However, there may also be hidden side effects on the optimization caused by other differences in the implementation of the mutation operators. Due to the simplicity of the use cases mentioned above, we attempted to mitigate this risk by keeping the changes between operator sets as small as possible and maintaining a similar granularity of model changes made by the mutation operators.

Finding the best parametrization of an evolutionary algorithm (e.g., the initial population, the operators, and the termination criterion) can be considered as an optimization problem in its own right. We chose the size and generation process of the population based on our experience from Chapter 2 and [BZJ21]. By design, our termination criterion allows a fair comparison as all evolutionary computations converge in same way. However, in the experiments for each evolutionary algorithm, we kept all variation points constant except for the mutation operator sets. This allows us to attribute differences to a single independent variable, but neglects possible synergies between specific parameters and variants of mutation operator sets. Like NSGA-II, the selection mechanisms of PESA-II and SPEA2 also discriminate infeasible solutions (see Ex. 3.5). Other selection and constraint-handling mechanisms may lead to different observations and need to be investigated. However, since NSGA-II, PESA-II, and SPEA2 can be considered state of the art, we believe that our results are very relevant.

3.8 Conclusion

Developing effective and efficient algorithms in MDO and, in particular, MB-MDO requires not only domain expertise but also in-depth knowledge of evolutionary algorithm design. We have presented a graph-based framework that identifies and clarifies the core concepts of MB-MDO. It is intended to assist the domain expert in using MB-MDO to solve optimization problems. It can also help in clarifying the critical factors for conducting reproducible experiments in MB-MDO. We have used this framework to conduct a series of experiments on optimizing software modularization and release planning using the CRA, SCRUM, and NRP cases.

In particular, since our formal framework puts the focus on the specification of mutation operators, it facilitates impact analysis of the properties of operators. As a showcase, we consider the soundness and completeness of element mutation operator sets. Our experiments provide a first insight into the effects of these properties on the effectiveness and efficiency of evolutionary algorithms in MB-MDO. We found that for a selection operator that strictly discriminates infeasible solutions, unsound mutation operators can slow down the optimization process as infeasible solutions are generated and discarded; moreover, unsound operators increase the probability of getting stuck in local optima. The performance of a set of incomplete mutation operators depends largely on which part of the search space becomes unreachable due to the incompleteness of the operator set. When (near) optimal solutions become unreachable, the effectiveness of the corresponding algorithm may suffer. On the other hand, ignoring parts of a search space that do not contain interesting solutions can speed up the optimization process. Since a search space also depends on the given problem instance, the effect of completeness may vary from instance to instance of the same optimization problem. Therefore, it can be very important to make an informed decision about the soundness and completeness of change operators.

To confirm our observations, the task of future work is to consider other optimization problems and further vary the determining factors. More tool support for analyzing soundness and completeness would facilitate further experimentation. Investigating runtime verification techniques for analyzing soundness and completeness on the population level is also part of future work.

In addition, it is interesting to consider crossover (or breeding) operators for MB-MDO and investigate the effects of soundness and completeness for these as well. In Section 4.1, we take a first step in this direction by defining a crossover operator for graph-like structures. It forms the formal basis for a crossover operator for EMF models; we present a first draft and implementation in 4.2. Furthermore, in [TK22], we develop a (not yet implemented) approach to crossover on graphs that is sound with respect to multiplicity constraints. We intend to use these works to extend our framework to crossover operators and to extend our impact analysis of soundness and completeness also to evolutionary algorithms that use both mutation and crossover operators.

A fundamental design decision of our framework is to distinguish problem and solution parts in models: it can be advantageously used in the CRA, NRP, and SCRUM optimization problems that we have considered throughout the chapter. There may also be optimization problems where the separation of problem and solution parts in models is not clear. For example, the Refactoring case [LK13; Kol+14] starts from a class model and searches for an optimal refactoring sequence for that model; during evolution, all parts of the class model may change. This case would be specified in our framework with empty problem models. It is left to future work to investigate the consequences of this design decision.

Our graph-based framework is deliberately generic in terms of modeling languages and specification options for evolutionary algorithms. It is incumbent on future work to investigate how it relates to other existing optimization approaches based on graphs or models. This is particularly true for the evolution of graphs through graph programs by Atkinson et al. [APS18; APS21; APS20] and the rule-based approach to MDO [Abd+14; FTW15; Bil+19].

Our long-term goal is to consolidate MDO so that it is well-suited for optimizations in complex problem domains such as those encountered in search-based software engineering and beyond.

4

A Crossover Approach for MB-MDO

Preface: *This chapter comprises two publications. Section 4.1 corresponds to the publication* A Generic Construction for Crossovers of Graph-like Structures [TJK22]. *It introduces a generic approach for performing crossover on graph-like structures. Section 4.2 concretizes this approach in the context of the Eclipse Modeling Framework. It corresponds to the publication* Towards a Configurable Crossover Operator for Model-Driven Optimization [JKT22a].

4.1 Formal Approach to Crossover

4.1.1 Introduction

In software development, software engineers often make design decisions in the context of competing constraints ranging from requirements to technology. To efficiently find optimal solutions, Search-Based Software Engineering (SBSE) [HJ01] attempts to formulate software engineering problems as optimization problems that capture the constraints of interest as objectives. By using meta-heuristic search techniques, good solutions can often be found with reasonable effort. Because of their generality, evolutionary algorithms, and in particular genetic algorithms [HMZ12; BSA17] that use mutation, crossover, and selection to perform a guided search over the search space, are a technique of particular relevance. According to e.g. [ES15], the definition of an evolutionary algorithm requires a representation of problem instances and search space

elements (i.e., solutions). It also includes a formulated optimization problem that clarifies which of the solutions are *feasible* (i.e., satisfy all constraints of the optimization problem) and best satisfy the objectives. The key ingredients of the optimization process are a procedure for generating a start population of solutions, a mechanism for generating new solutions from existing ones (e.g., by mutation and crossover), a selection mechanism that typically establishes the evolutionary concept of survival of the fittest, and a condition for stopping evolutionary computations. Selecting these ingredients so that an evolutionary algorithm is effective and efficient is usually a challenge.

Model-driven optimization (MDO) aims at reducing the required level of expertise of users of meta-heuristic techniques. Two main approaches have emerged in MDO: the model-based approach [Bur+12; BZJ21] performs optimization directly on models, while the rule-based approach [Abd+14; Bil+19] searches for optimized model transformation sequences. In this chapter, we focus on the model-based approach since it tends to be more effective (see Chapter 2) and refer to it as MB-MDO for short. In MB-MDO, optimization problems are specified as models that capture domain-specific information about a problem and its solutions. In that way, users can interact with a domain-specific formulation of their problem, rather than traditional encodings that are typically closer to implementation. While the search space consists of models, the mutation of search space elements is specified by model transformations. In sophisticated evolutionary algorithms, mutations typically perform local changes, while crossovers are used to generate offspring by recombining existing search space elements. For (the model-based approach to) MDO, no crossover mechanism has been worked out yet. This chapter fills this research gap and presents a crossover construction for graph-based models. Several graph-based approaches to crossover have been suggested in the literature, e.g. [Per+99; Nie03]. In most cases, these crossovers are not *generic* (in the sense of different kinds of graphs), but are designed with specific semantics of the underlying graphs in mind. We aim to develop a generic construction of crossovers that can be applied to different kinds of graph-like structures. Moreover, this construction of crossovers is applicable regardless of the semantics of the graphs of interest. We also prove the correctness and completeness of our crossover construction.

The chapter is organized as follows: We start with an example MB-MDO problem and discuss a possible crossover in this context in Section 4.1.2. Section 4.1.3

recalls preliminaries. The main contribution of this chapter, a pushout-based crossover construction, is presented in Section 4.1.4. In Section 4.1.5, we explain how our new crossover construction encompasses important, more specific approaches to crossover (on graph-like structures) that have been suggested in the literature. We close with a discussion of related work and a conclusion in Sects. 4.1.6 and 4.1.7. All proofs are given in Appendix B.1.

4.1.2 Running Example

The CRA case [BBL10] is an optimization problem from the domain of software design that has recently established itself as an easily understood use case in the context of MDO. Given a software product represented by a set of features (i.e., attributes and methods) and dependency relations between them, the task is to modularize the software by encapsulating its features into classes. Two well-known quality aspects are used to evaluate the quality of solutions: cohesion and coupling. Cohesion rewards classes in which features are highly interdependent, while coupling captures the interdependencies of features that exist between classes. A highly cohesive design with low coupling is considered easy to understand and maintain. Therefore, maximizing cohesion and minimizing coupling are the opposing objectives of the CRA case.

The structure of models in the CRA case can be defined by the type graph shown in Fig. 4.1. A problem instance consists at least of the features and their dependencies. These elements form the invariant part of a concrete problem. Classes (and their relationships), on the other hand, can be added, modified and removed to explore the search space and create new solutions. Typical mutations for the CRA case include small changes like adding or removing a class, assigning a feature to a class, or changing the assignment of a feature from one class to another. Mutation usually does not consider already well optimized substructures that might be worth being shared with other solutions.

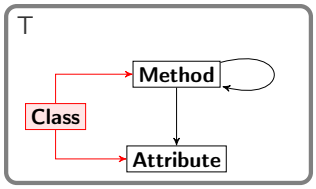


Figure 4.1 – Type graph of the CRA case. White solid elements specify invariant problem parts, the red colored class element and its relations are solution specific.

In the CRA case, a subset of features, along with their current assignment to classes, contains potentially valuable information. The exchange of this information between two solutions represents a promising crossover as we will see in the following example. Consider solutions $\overline{\mathsf{E}}$ and $\overline{\mathsf{F}}$ in Fig. 4.2, for a problem instance consisting of four methods and two attributes. Let a crossover choose to recombine them by exchanging their assignment information for the features 1:Method, 2:Attribute and 3:Method. This results in two offspring solutions. Solution $\overline{\mathsf{E}^1\mathsf{F}^2}$ keeps the original assignments of 4:Method, 5:Attribute, and 6:Method as found in solution $\overline{\mathsf{F}}$ and combines them with the assignments of $\overline{\mathsf{E}}$ for the exchanged features. The solution $\overline{\mathsf{E}^2\mathsf{F}^1}$ is constructed in the opposite way.

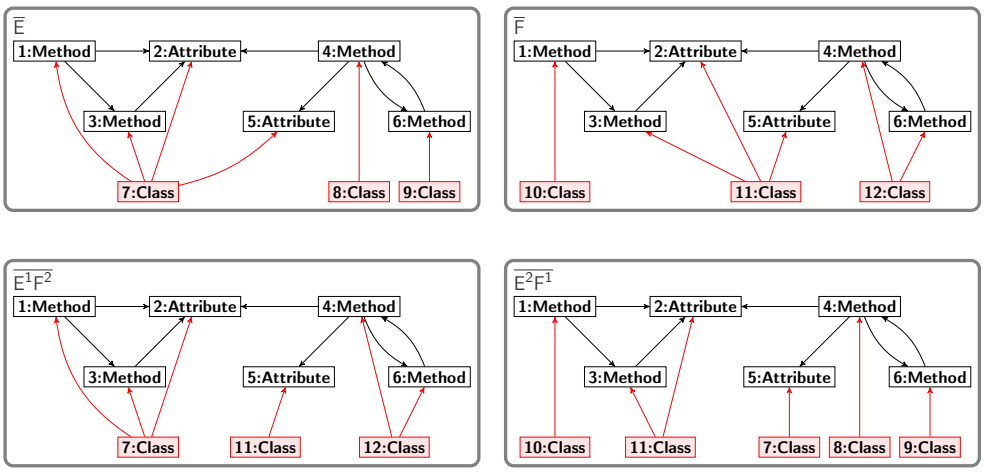


Figure 4.2 – Example crossover in the CRA case that creates the offspring $\overline{\mathsf{E}^1\mathsf{F}^2}$ and $\overline{\mathsf{E}^2\mathsf{F}^1}$ by exchanging the assignments of features 1:Method, 2:Attribute, and 3:Method between the solutions $\overline{\mathsf{E}}$ and $\overline{\mathsf{F}}$.

Note that combining 1:Method, 2:Attribute and 3:Method into one class (as done in solution $\overline{\mathsf{E}}$) seems a reasonable choice. Their pairwise dependencies promote cohesion, while splitting them would lead to coupling. The same is true for the features of class 12: Class in solution $\overline{\mathsf{F}}$. Consequently, the offspring $\overline{\mathsf{E}^1\mathsf{F}^2}$ combines the best of both worlds.

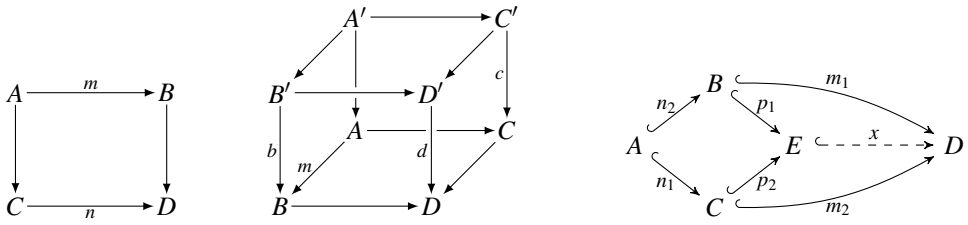
4.1.3 Preliminaries: $\mathscr{M}$-Adhesive Categories

In this section, we briefly recall our central formal preliminaries, namely *$\mathscr{M}$-adhesive categories* and *$\mathscr{M}$-effective unions* [Ehr+15], which provide the setting in which we formulate our contribution. $\mathscr{M}$-adhesive categories with $\mathscr{M}$-effective unions are categories where pushouts along certain monomorphisms interact in a particularly nice way with pullbacks. This is of importance because our construction of crossovers is based on pushouts. Moreover, working in the framework of $\mathscr{M}$-adhesive categories allows us to easily abstract from the concrete choice of graphs used to formalize the models of interest (such as typed, labeled, and attributed graphs). We only use category-theoretic concepts that are common in the context of algebraic graph transformation, and refer to [Ehr+06; Ehr+15] for introductions.

Definition 4.1 ($\mathscr{M}$-adhesive category)**.** A category $\mathscr{C}$ with a morphism class $\mathscr{M}$ is an *$\mathscr{M}$-adhesive category* if the following properties hold:

- $\mathscr{M}$ is a class of monomorphisms closed under isomorphisms (f isomorphism implies that $f \in \mathscr{M}$), composition ($f, g \in \mathscr{M}$ implies $g \circ f \in \mathscr{M}$), and decomposition ($g \circ f, g \in \mathscr{M}$ implies $f \in \mathscr{M}$).
- $\mathscr{C}$ has pushouts and pullbacks along $\mathscr{M}$-morphisms, i.e., pushouts and pullbacks where at least one of the given morphisms is in $\mathscr{M}$, and $\mathscr{M}$-morphisms are closed under pushouts and pullbacks, i.e., given a pushout like the left square in Fig. 4.3a, $m \in \mathscr{M}$ implies $n \in \mathscr{M}$ and, given a pullback, $n \in \mathscr{M}$ implies $m \in \mathscr{M}$.
- Pushouts in $\mathscr{C}$ along $\mathscr{M}$-morphisms are *vertical weak van Kampen squares*, i.e., for any commutative cube in $\mathscr{C}$ (as in the right part of Fig. 4.3a) where we have the pushout with $m \in \mathscr{M}$ in the bottom, $b, c, d \in \mathscr{M}$, and pullbacks as back faces, the top is a pushout if and only if the front faces are pullbacks.

We speak of $\mathscr{M}$-adhesive categories $(\mathscr{C}, \mathscr{M})$ and indicate arrows from $\mathscr{M}$ as hooked arrows in diagrams. Examples of categories that are $\mathscr{M}$-adhesive include sets with injective functions, graphs with injective graph morphisms and various varieties of graphs with special forms of injective graph morphisms. In particular,



(a) Vertical weak van Kampen square **(b)** $\mathscr{M}$-effective union

Figure 4.3 – Defining $\mathscr{M}$-adhesive categories with $\mathscr{M}$-effective unions

typed attributed graphs form an $\mathscr{M}$-adhesive category (where the class $\mathscr{M}$ consists of injective morphisms where the attribute part is an isomorphism).

The existence of *$\mathscr{M}$-effective unions* ensures that the $\mathscr{M}$-subobjects of a given object form a lattice.

Definition 4.2 ($\mathscr{M}$-effective unions)**.** An $\mathscr{M}$-adhesive category $(\mathscr{C},\mathscr{M})$ has *$\mathscr{M}$-effective unions* if for each pushout of a pullback of a pair of $\mathscr{M}$-morphisms the induced mediating morphism belongs to $\mathscr{M}$ as well, i.e., if in each diagram like the one depicted in Fig. 4.3b where the outer square is a pullback of $\mathscr{M}$-morphisms and the inner one a pushout, the induced morphism x is an $\mathscr{M}$-morphism.

4.1.4 A Pushout-Based Crossover Construction

In this section, we develop our approach to crossover. Recall that we introduced our formal framework in Section 3.4 based on graphs and type graphs, but presented a generalization in Appendix A.1. In the following, we will present our crossover construction based on this general setting of category theory. We start with introducing the objects to which crossover will be applied.

In MDO, optimization problems are defined based on modeling languages, typically specified with meta-models. Various MDO approaches in the literature such as [Bur+12; BZJ21] have chosen to represent problem instances and solutions by models. Both can contain invariant problem parts as well as solution specific parts, a distinction typically embedded in the associated meta-model.

In our formalization, this is reflected in the fact that a *computation element* is given by an object that conforms to a *computation type object*. The type object specifies which parts of a computation element are invariant and which parts contribute to the solution. A concrete problem to be optimized is given by a *problem instance*; every computation element can serve as such. The *search space* of a problem instance includes all computation elements with the same problem object as specified by the given problem instance. In MDO, problem instances and solutions are typically further constrained by additional conditions. We leave this refinement to future work.

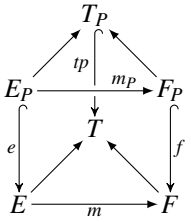


Figure 4.4 – Computation elements and ce-morphism

Definition 4.3 (Computation element. Problem instance. Search space)**.** Let $(\mathscr{C},\mathscr{M})$ be an $\mathscr{M}$-adhesive category. A *computation type object* in $\mathscr{C}$ is an $\mathscr{M}$-morphism $tp\colon T_P \hookrightarrow T$; T_P is called the *problem type object*. A *computation element* $\overline{E} = (e\colon E_\mathrm{P} \hookrightarrow E, t_{E_\mathrm{P}}, t_E)$ over tp is an $\mathscr{M}$-morphism e together with *typing morphisms* $t_{E_\mathrm{P}}\colon E_\mathrm{P} \to T_P$ and $t_E\colon E \to T$ such that the induced square (over tp) is a pullback. The pair $(E_\mathrm{P}, t_{E_\mathrm{P}})$ is the *problem object* of $\overline{E}$. If defined, the initial pushout over e yields the *solution part* of $\overline{E}$, written $E \setminus E_P$.

A *computation-element morphism* $\overline{m} = (m_P, m)$, short *ce-morphism*, from computation element $\overline{E}$ to computation element $\overline{F}$ is a pair of morphisms $m_P\colon E_\mathrm{P} \to F_\mathrm{P}$ and $m\colon E \to F$ that are compatible with typing, i.e., $t_{F_\mathrm{P}} \circ m_P = t_{E_\mathrm{P}}$ and $t_F \circ m = t_E$ (see Fig. 4.4). A ce-morphism $\overline{m}$ is *problem-invariant* if m_P is an isomorphism between E_P and F_P.

Given a computation type object $tp\colon T_P \hookrightarrow T$ in $\mathscr{C}$, a *problem instance* $\overline{PI}$ of tp is a computation element $\overline{PI} = (p\colon PI_\mathrm{P} \hookrightarrow PI, t_{PI_\mathrm{P}}, t_{PI})$ over tp. It defines the *search space*

$$S(\overline{PI}) := \{\overline{E} = (e\colon E_\mathrm{P} \hookrightarrow E, t_{E_\mathrm{P}}, t_E) \in CS \mid \text{there exists an isomorphism } a_P\colon PI_\mathrm{P} \xrightarrow{\sim} E_\mathrm{P} \text{ s.t. } t_{E_\mathrm{P}} \circ a_P = t_{PI_\mathrm{P}}\}.$$

Each element of the search space $S(\overline{PI})$ is called *solution (object)* for $\overline{PI}$.

Given a solution $\overline{E}$ for $\overline{PI}$, a *subsolution* of $\overline{E}$ is a solution $\overline{E^1}$ from the search space $S(\overline{PI})$ such that there exists a problem-invariant ce-morphism $\overline{s^1}$ from $\overline{E^1}$ to $\overline{E}$ where $s^1 \in \mathscr{M}$.

Before providing an example, some remarks with respect to the above definition and notation are in order. Since the typing of the problem object of a computation element is defined via a pullback, pullback decomposition implies that a ce-morphism is indeed a pullback square (compare Fig. 4.4). Thus, in abstract terms, we fix an $\mathscr{M}$-morphism $T_P \hookrightarrow T$ from a given $\mathscr{M}$-adhesive category $\mathscr{C}$. We then work in the category that has pullback squares over $T_P \hookrightarrow T$ as objects and pullbacks between such pullback squares as arrows. The results in [Kos+20a, Theorem 1] ensure that this category is again $\mathscr{M}$-adhesive, provided that the original category $\mathscr{C}$ is also partial-map adhesive (as defined in [Hei12]); a property that is satisfied by the category of attributed graphs; see, for example, [Kos+20a, Corollary 1]. However, in this chapter it will suffice to consider the arising diagrams as diagrams in the $\mathscr{M}$-adhesive category $\mathscr{C}$.

To shorten the presentation, we often only speak of computation elements $\overline{E}$ and ce-morphisms $\overline{m}$ and use their components (such as E_{P}, $t_{E_{\mathrm{P}}}$, or m) freely without introducing them explicitly. Furthermore, we often let the typing be implicit; in particular, we omit it in almost all diagrams. In our examples, we use the category of graphs as the underlying $\mathscr{M}$-adhesive category $\mathscr{C}$. Finally, we specify problem instances in terms of the actual computation elements (and not just in terms of their problem objects) to account for the fact that in practice the problem of interest may be given as part of a (suboptimal) solution.

Example 4.1. The graph T in Fig. 4.1 can be viewed as a compact representation of a computation type graph where the black part marks the embedded problem type graph. Similarly, the typed graphs of Fig. 4.2 are interpreted as computation elements over T, with the black parts typed over the problem type graph; the typing is indicated by the names of the nodes. Since the typing morphisms form pullbacks, these black parts represent the problem graphs of the respective computation elements. Having identical problem graphs, all four graphs belong to the same search space, which can be defined using either of them. This reflects that a user might want to optimize an existing assignment of features to classes, rather than just specifying the features and their interdependencies.

Taking two computation elements (from the same search space) and splitting their solution parts, two offspring solutions are constructed by recombining the resulting subsolutions crosswise. In the following, we formally develop this intuition (based on the category-theoretic concept of pushouts) and prove basic properties of this construction of crossovers. We begin by defining the *split* of a given solution.

Definition 4.4 (Split)**.** Given a problem instance $\overline{PI}$ and a solution $\overline{E}$ for $\overline{PI}$, a *split* of $\overline{E}$ is a commuting cube as depicted in Fig. 4.5 where the bottom square is a pushout, the vertical squares constitute ce-morphisms, all morphisms come from $\mathscr{M}$, and all problem objects (the objects in the square at the top) are isomorphic to PI_{P}. The bottom square is called *solution split* and $\overline{E^I}$ is a *split point* of $\overline{E}$. The subsolutions $\overline{E^1}$ and $\overline{E^2}$ of $\overline{E}$ are called *(solution) split objects* of $\overline{E}$.

A solution can be split in several ways; the central idea is that each solution item of $\overline{E}$ occurs in (at least) one of the solution parts of $\overline{E^1}$ or $\overline{E^2}$. We next present a concrete construction that implements the above declarative definition.

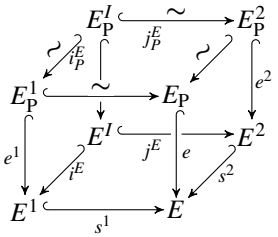


Figure 4.5 – Split of solution $\overline{E}$

Definition 4.5 (Split construction)**.** Given a solution $\overline{E}$, the *split construction* consists of the following steps:

(1) Choose an $\mathscr{M}$-subobject $s^1 : E^1 \hookrightarrow E$ from E (in $\mathscr{C}$) such that when pulling back s^1 along e, the morphism s^1_P opposite to s^1 is an isomorphism (in particular, $E^1_{\mathrm{P}} \cong E_{\mathrm{P}} \cong PI_{\mathrm{P}}$, where E^1_{P} is the object computed by this pullback). The typing morphisms $t_{E^1_{\mathrm{P}}}$ and t_{E^1} are defined as $t_{E_{\mathrm{P}}} \circ s^1_P$ and $t_E \circ s^1$, respectively.

(2) Choose another such $\mathscr{M}$-subobject $s^2 : E^2 \hookrightarrow E$ from E such that s^1, s^2 are jointly epi (again, typing is defined by composition).

(3) Complete the cube by constructing pullbacks. That is, determine E^I as the pullback of s^1 and s^2, E^I_{P} as the pullback of the isomorphisms at the top of the cube, and $e^I : E^I_{\mathrm{P}} \hookrightarrow E^I$ as the morphism that is induced by the

universal property of the bottom pullback. Again, when considered as computation element, the typing of $\overline{E^I}$ is defined by composition.

Remark 4.1. While in general categories the above construction need not be constructive, it is when the underlying category is one of the familiar categories of graphs (being, e.g. typed, labeled, or attributed). Then, the choice of E^1 amounts to extending (an isomorphic copy of) E_P by a choice of solution elements from E; s^1 extends the isomorphism accordingly. Since pullbacks of injective morphisms compute intersections, the pullback of s^1 along e computes the chosen isomorphic copy (up to unique isomorphism). For the choice of E^2, one again extends an isomorphic copy of E_P by a choice of solution elements from E. To ensure that s^1 and s^2 become jointly epi (that is, jointly surjective in our case), one must include at least all solution elements of E not chosen in the construction of E^1.

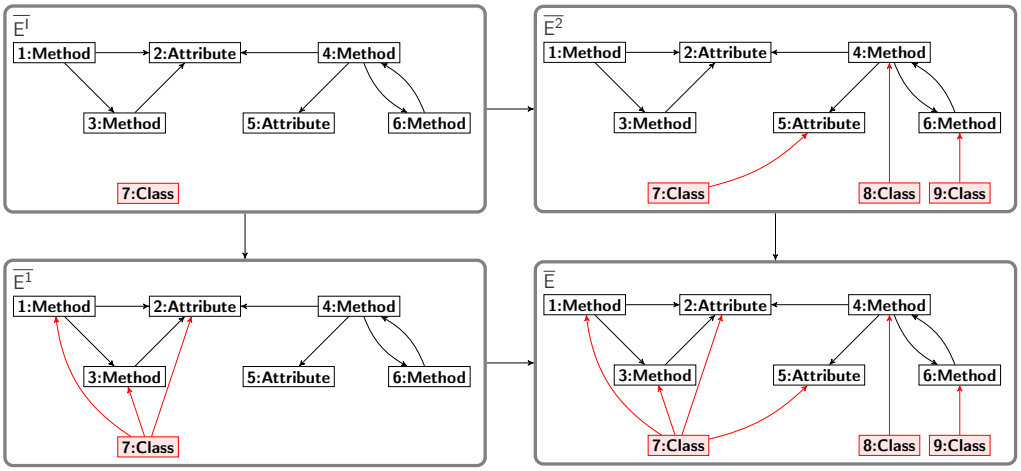


Figure 4.6 – A split of solution $\overline{\mathsf{E}}$

Example 4.2. Given the two degrees of freedom for a split, different splits can be constructed from solution $\overline{\mathsf{E}}$ shown in Fig. 4.2. In steps (1) and (2) we have all possibilities to extend its problem graph $\mathsf{E_P}$ (or an isomorphic copy) with solution parts that yield E^1 and E^2 as long as E^1 and E^2 form graphs and jointly cover E.

A possible split of the solution $\overline{\mathsf{E}}$ is shown in Fig. 4.6. Here, $\overline{\mathsf{E}}$ is split by first inserting the assignment relations of 1:Method, 2:Attribute, and 3:Method into E^1 along with the associated class 7:Class. The rest of the feature assignments and the necessary classes become part of E^2. The pullback E^I of E^1 and E^2 contains their common solution element 7:Class. To simplify the presentation, the problem graph E_P is reused in all four graphs. Note that the morphisms in Fig. 4.6 are indicated by equal numbers in the corresponding nodes. They uniquely induce the mapping of edges. We use these conventions in all of the following examples.

Proposition 4.1 (Correctness and completeness of split construction)**.** In an $\mathscr{M}$-adhesive category with $\mathscr{M}$-effective unions, the split construction in Definition 4.5 is *correct* and *complete*: it always yields a split of the given solution and every possible split can be realized through it. Moreover, for each choice of an $\mathscr{M}$-subobject $s^1 : E^1 \hookrightarrow E$ there exists at least one possible split.

Given a problem instance $\overline{PI}$ and two solutions $\overline{E}$ and $\overline{F}$ for it, a crossover of $\overline{E}$ and $\overline{F}$ can be performed. Their offspring are basically constructed by recombining solution split objects crosswise. Variations of recombinations are possible, since solution-split objects resulting from solution splits of $\overline{E}$ and $\overline{F}$ can be recombined with more or less overlap. To uniquely determine a crossover of $\overline{E}$ and $\overline{F}$, we define a crossover point that specifies the overlap of their solution split objects.

Definition 4.6 (Crossover point)**.** Given a problem instance $\overline{PI}$, two solutions $\overline{E}$ and $\overline{F}$ for $\overline{PI}$, with splits having split points $\overline{E^I}$ and $\overline{F^I}$, respectively, a *crossover point* $\overline{CP}$ is a common subsolution of $\overline{E^I}$ and $\overline{F^I}$. That is, a crossover point is a span of problem-invariant ce-morphisms as depicted in Fig. 4.7 (with bottom components coming from $\mathscr{M}$).

We will explain crossover points later along with the crossover operation as such. Next we briefly mention that it is always possible to find a crossover point in a trivial way – the problem object of the given problem instance can always serve as such.

Lemma 4.1 (Existence of crossover points)**.** *Given a problem instance* $\overline{PI} = (p\colon PI_{\mathrm{P}} \hookrightarrow PI, t_{PI_{\mathrm{P}}}, t_{PI})$ *over type object tp, two solutions* $\overline{E}$ *and* $\overline{F}$ *for* $\overline{PI}$*, and splits with split points* $\overline{E^I}$ *and* $\overline{F^I}$*, respectively,* $\overline{CP} := (id : PI_{\mathrm{P}} \hookrightarrow PI_{\mathrm{P}}, t_{PI_{\mathrm{P}}}, tp \circ t_{PI_{\mathrm{P}}})$ *is always a crossover point for them. In particular, for each two splits of solutions for the same problem instance there always exists a crossover point.*

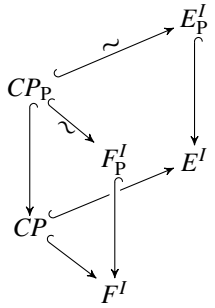


Figure 4.7 – Crossover point

Taking two solutions $\overline{E}$ and $\overline{F}$ for a common problem instance and splitting them into subsolutions $\overline{E^1}, \overline{E^2}$ and $\overline{F^1}, \overline{F^2}$, we choose a crossover point for these splits and now define a crossover of these solutions. It basically recombines the subsolutions of $\overline{E}$ and $\overline{F}$ crosswise at the crossover point and yields the computation elements $\overline{E^1F^2}$ and $\overline{E^2F^1}$. We show in Prop. 4.2 that these two offspring are also solutions to the joint problem instance.

Definition 4.7 (Crossover)**.** Let a problem instance $\overline{PI}$, two solutions $\overline{E}$ and $\overline{F}$ for $\overline{PI}$, splits of these two solutions with split objects $\overline{E^1}, \overline{E^2}, \overline{F^1}, \overline{F^2}$ and split points $\overline{E^I}$ and $\overline{F^I}$, respectively, and a *crossover point* $\overline{CP}$ for these splits be given. Then, a *crossover* of solutions $\overline{E}$ and $\overline{F}$ (at $\overline{CP}$ and these splits) yields the two *offspring solutions* $\overline{O_1}$ and $\overline{O_2}$ of $\overline{E}$ and $\overline{F}$ that are shown in Fig. 4.8 and constructed as follows:

(1) The ce-morphisms from $\overline{CP}$ to $\overline{E^1}$ and $\overline{E^2}$ are obtained by composing the ce-morphism from $\overline{CP}$ to $\overline{E^I}$ (given by the crossover point) with the ce-morphisms from $\overline{E^I}$ to $\overline{E^1}$ and $\overline{E^2}$ (given by the solution split of $\overline{E}$), respectively. The ce-morphisms from $\overline{CP}$ to $\overline{F^1}$ and $\overline{F^2}$ are obtained analogously.

(2) The top and bottom squares of the cubes are computed as pushouts (in $\mathscr{C}$) yielding the objects $(E^1F^2)_{\mathrm{P}}$, E^1F^2, $(E^2F^1)_{\mathrm{P}}$, and E^2F^1. The typing morphisms for these objects are obtained from the universal properties of the respective pushout.

(3) The morphisms $o_1\colon (E^1F^2)_{\mathrm{P}} \hookrightarrow E^1F^2$ and $o_2\colon (E^2F^1)_{\mathrm{P}} \hookrightarrow E^2F^1$ are also induced by the universal property of the pushout squares at the top of the cubes. These morphisms form the objects of $\overline{O_1}$ and $\overline{O_2}$.

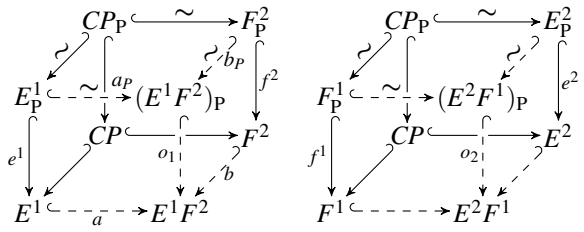


Figure 4.8 – Crossover of solutions $\overline{E}$ and $\overline{F}$

We illustrate the construction before establishing some of its basic properties such as its correctness.

Example 4.3. A split of solution $\overline{\mathsf{F}}$ (introduced in Fig. 4.2) is shown in Fig. 4.9. Again, the split point extends the problem graph by a Class element. Therefore, a crossover point for $\overline{\mathsf{E}}$ and $\overline{\mathsf{F}}$ (with the splits given in Figs. 4.6 and 4.9) consists either of their common problem graph only, or of this problem graph extended by a single Class. Figure 4.2 already shows the two offspring graphs that result from applying crossover to $\overline{\mathsf{E}}$ and $\overline{\mathsf{F}}$ where the problem graph is chosen as crossover point. In contrast, adding a Class to the crossover point would merge 7:Class and 11:Class during the recombination and result in the offspring shown in Fig. 4.10.

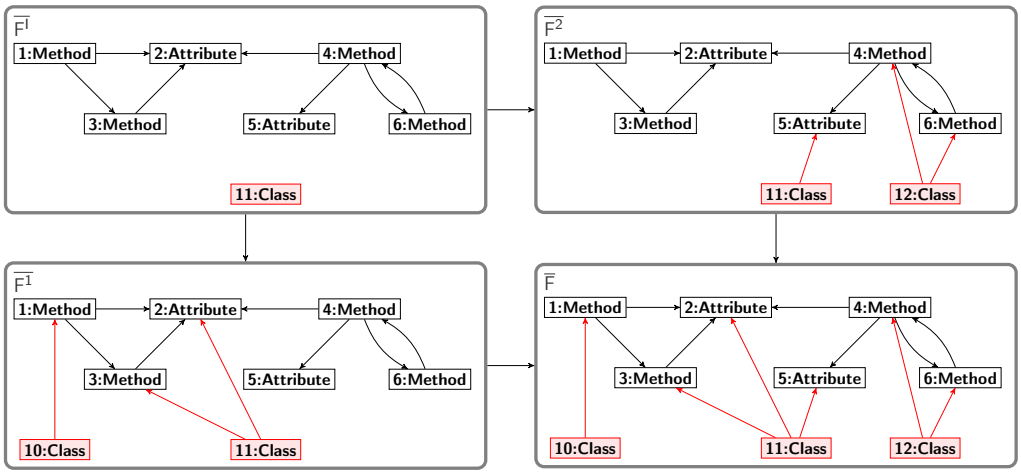


Figure 4.9 – A split of solution $\overline{\mathsf{F}}$ originally presented in Fig. 4.2

The next proposition shows that a crossover calculate the offspring correctly, i.e. all offspring calculated represent solutions (for the given problem instance).

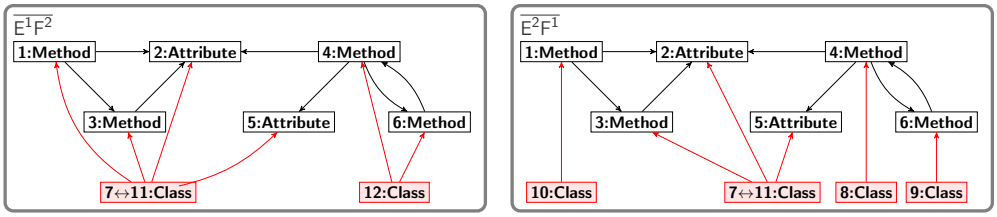


Figure 4.10 – Two offspring models $\overline{\mathsf{E}^1\mathsf{F}^2}$, $\overline{\mathsf{E}^2\mathsf{F}^1}$, based on the splits of Figs. 4.6 and 4.9 and a crossover point containing an additional class

Proposition 4.2 (Correctness of offspring)**.** Given a problem instance $\overline{PI}$, two solutions $\overline{E}$ and $\overline{F}$ for $\overline{PI}$, splits with split objects $\overline{E^1}, \overline{E^2}, \overline{F^1}, \overline{F^2}$ and split points $\overline{E^I}$ and $\overline{F^I}$, respectively, and a crossover point $\overline{CP}$ for these splits, then there is always a crossover and the two offspring solutions $\overline{O_1}$ and $\overline{O_2}$ are solutions for $\overline{PI}$.

Next we characterize the expressiveness of the presented crossover construction: Given two solutions $\overline{E}$ and $\overline{F}$, all solutions that can be understood as results of splitting $\overline{E}$ and $\overline{F}$ and their recombination can indeed be generated as offspring of the construction in Def. 4.7 (by different choices of solution splits and crossover points). This is reminiscent of the expressiveness of *uniform crossover* when using arrays of, e.g., bits as genotype [ES15].

Proposition 4.3 (Completeness of crossover)**.** Let the underlying $\mathscr{M}$-adhesive category $\mathscr{C}$ have $\mathscr{M}$-effective unions, and let a problem instance $\overline{PI}$ and solutions $\overline{E}$, $\overline{F}$, and $\overline{O}$ for $\overline{PI}$ be given. The solution $\overline{O}$ can be obtained as offspring from a crossover of $\overline{E}$ and $\overline{F}$ if and only if there are subsolutions $\overline{E^1}$ of $\overline{E}$ and $\overline{F^2}$ of $\overline{F}$ with problem-invariant ce-morphisms $\bar{i}\colon \overline{E^1} \to \overline{O}$ and $\bar{j}\colon \overline{F^2} \to \overline{O}$ such that i and j are jointly epic $\mathscr{M}$-morphisms.

Discussion. As mentioned earlier, $\mathscr{M}$-adhesive categories include various categories of (typed, labeled, or attributed) graphs that can be used to formalize modeling approaches. In particular, our construction supports crossovers of graphs with inheritance and attribution – concepts that are regularly used in modeling. As for the construction of splits and crossover points, our approach provides several degrees of freedom. In principle, for any implementation of

these variation points, the definitions and results in this section are sufficient to complement evolutionary computations in model-based MDO with crossovers. Moreover, our proposed crossover construction is *generic* in the sense that it can be applied to any meta-model; it only needs to be possible to formalize the optimization problem of interest and its search space according to Definition 4.3. Then, whenever two solution models are chosen for crossover, Proposition 4.1 ensures that both can be split. Next, Lemma 4.1 ensures that regardless of which splits are chosen, a crossover point exists for these splits. Finally, Proposition 4.2 ensures that, for two splits and a crossover point, there is always a crossover that provides solutions of the search space.

Beyond typing, meta-modeling typically employs *integrity constraints* that express further requirements for instances being considered well-formed; *multiplicities* are a typical example. We do not consider such constraints so far. This means that given a meta-model with additional integrity constraints and two of its instance models satisfying these constraints, computing crossover as specified in this work may result in offspring models that violate the constraints. We illustrate this with our running example: In practical applications, the meta-model (type graph) from Fig. 4.1 would have a constraint requiring each Method and each Attribute to be associated with at most one Class. A slight adjustment of the split and crossover points in Examples 4.2 and 4.3 results in the offspring shown in Fig. 4.11; both graphs violate the considered constraint. The splits of $\overline{E}$ and $\overline{F}$ were adjusted to additionally include the edge to 5:Attribute in $\overline{E^1}$ as well as in $\overline{F^1}$ (from 7:Class and 11:Class, respectively); the problem part served as the crossover point. Computing offspring that violate such additional constraints is not in itself a problem; several methods have been developed in evolutionary algorithm research to deal with this. For example, such *infeasible solutions* can be eliminated by the selection operator, or they can be tolerated (with a reduced fitness assigned to them); after all, even an infeasible solution can lead to a feasible solution of high quality later during the evolutionary computation. However, producing too many infeasible solutions can waste valuable resources and slow down the evolutionary computation process.

Summarizing, we expect evolutionary search to profit most if domain-specific knowledge is used to direct the choices of splits and crossover points, that is, if these choices are adapted to the problem at hand (possibly including the preservation of additional constraints). Thus, while our construction can

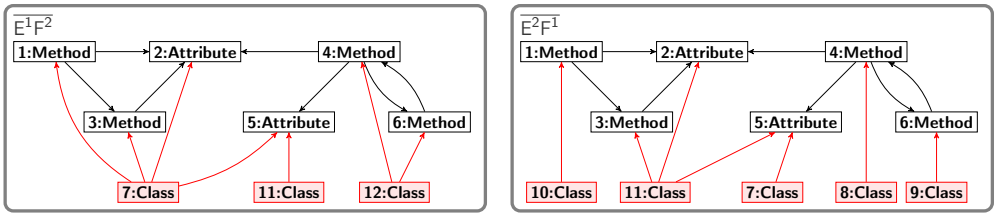


Figure 4.11 – Offspring violating an integrity constraint

principally yield problem-agnostic crossovers, it can also (and maybe better) be understood as a *generic* construction that offers a unifying framework for the implementation of specific crossovers on graph-like structures. In the next section, we substantiate the claim that our construction offers such a unifying framework.

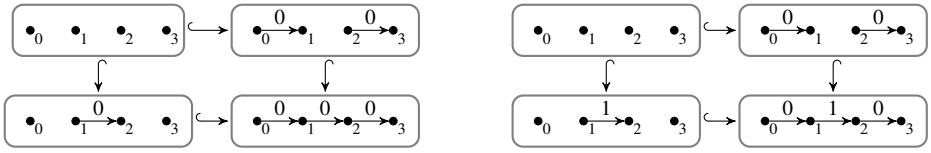
4.1.5 Instantiating Existing Approaches to Graph-Based Crossover

In this section, we exemplify how our generic construction includes existing crossover operators that can be applied to graph-like structures. We discuss *uniform*, *k-point* and *subtree crossover*, as these are classic operators that are commonly applied [Koz92; ES15]. In addition, we consider *horizontal gene transfer* (HGT), which was recently introduced in a setting similar to ours [APS20].

Uniform and k-point crossover. These are crossover operators commonly used when solutions are encoded as strings (arrays) of bits (or other alphabets) [ES15]. In k-point crossover, two given parent strings of equal length are split into $k+1$ substrings at k randomly selected crossover points (at equal positions in both strings). The two offspring solutions are obtained by alternately concatenating a substring from each parent, resulting in solutions of the same length as the given parents. In uniform crossover, a new decision is made at each position (according to a given probability) which offspring gets the entry from which parent. This can be understood as k-point crossover with varying k.

Strings can be represented as graphs by simply considering each character of a string as an edge typed or labeled with that character; see, e.g., [Plu98]. Using

this representation, our construction of crossovers can be used to implement uniform and k-point crossover. Here, the problem object (graph) is given by the nodes of the graphs (which encode the length of the given strings). The splits are chosen such that (i) the edges are partitioned (disjointly) into the solution splits and (ii) the same partitions are chosen for both parents (i.e., if the first edge of the first parent is included in its first subsolution, the first edge of the second parent is also included in its first subsolution). This partitioning can be done according to the rules of k-point or uniform crossover. The only available crossover point is the set of nodes (i.e. the problem graph), since the edges are distributed disjointly. The calculation of the crossover, i.e.performing the two pushouts, results in two offspring solutions with the same length as the parents.



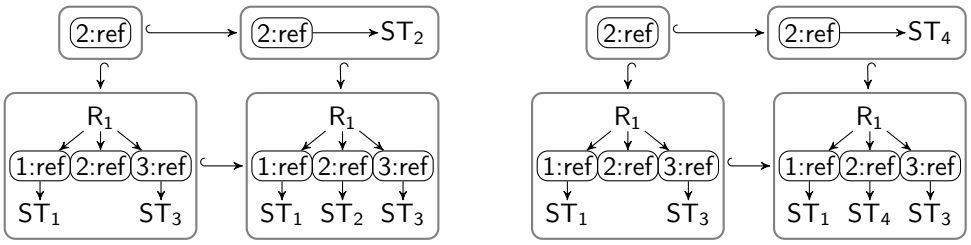
(a) Splitting s_1 **(b)** Computing o_1

Figure 4.12 – Implementing classic 2-point crossover

For the k-point crossover, we consider the concrete example of a 2-point crossover of the strings $s_1 : 0|0|0$ and $s_2 : 1|1|1$, where $|$ represents the chosen crossover points. The computed offspring strings are $o_1 : 010$ and $o_2 : 101$. Figure 4.12 outlines how this calculation is implemented in our approach.

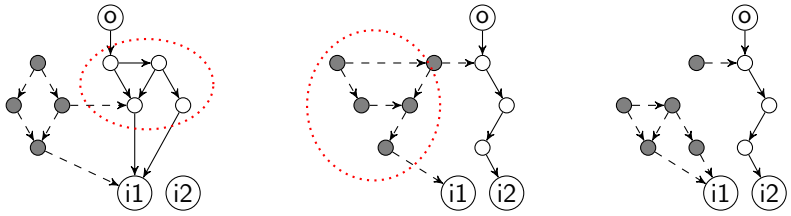
Subtree crossover. Subtree crossover is the recombination operator commonly used in genetic programming [Koz92]. In genetic programming, a program is represented by its syntax tree. Such a tree serves as a genotype for an evolutionary computation that aims at finding an (optimal) program for the given task. Given two syntax trees, subtree crossover (randomly) selects and exchanges one subtree from each of them. With our approach, we can implement subtree crossover if we use a little trick in representing the trees: We explicitly encode the edges of the trees as nodes (for a representation of (hyper)edges as special kinds of nodes, see, e.g., their (visual) representation in [Plu99]). The problem tree (graph) is always empty. A split divides a tree into a subtree and the

remaining tree, where the node encoding the reference to the subtree is common in both split objects. This node serves as a crossover point to exchange subtrees crosswise at the correct positions. Figure 4.13 schematically represents a subtree crossover, where R_1 is the root node of the first tree, all ST_i represent subtrees, and nodes of type ref represent edges. Note that representing edges as nodes allows us to split an edge into two parts and distribute it between the two split parts. In this way, we can redirect edges.



(a) Splitting a tree **(b)** Performing subtree crossover

Figure 4.13 – Implementing subtree crossover



(a) Donor **(b)** Receiver **(c)** Result graph

Figure 4.14 – Example of the horizontal gene transfer (HGT) proposed in [APS20]. o is the fixed output node. Active nodes are depicted in white, passive nodes are gray. i1 and i2 are input nodes. The marked nodes of the receiver (including outgoing edges) are substituted by the marked parts of the donor.

Horizontal gene transfer (HGT). HGT was proposed by Atkinson et al. in [APS20] as a non-recombinative method for transferring genetic information between individuals. In their work, graphs are used to represent functions (or, with small adaptations, neural networks); the reachability of fixed output nodes determines the *active component* of a graph. As indicated in Fig. 4.14, HGT takes the active component of one graph (the donor) and copies it to the passive

component of another graph (the receiver); to maintain a fixed number of nodes, an appropriate number of passive nodes is deleted from the receiver beforehand. Input nodes representing parameters are identified during that process. In our construction the output and input nodes would be considered the problem part. Choosing the active component as the solution split for the donor, the subgraph that remains after deleting the passive nodes as the solution split of the receiver, and the problem part as crossover point, our approach can compute HGT as a crossover.

4.1.6 Related Work

In addition to the approaches presented in detail above in Section 4.1.5, we now relate our crossover construction to other variants of crossover on graph-like structures. For each approach, we clarify whether it can be simulated by our approach and how expressive it is. We then discuss the crossover variants used so far in MDO.

4.1.6.1 Further Approaches for Graph-Based Crossover

The two most general crossover variants on graph-like structures that we are aware of are those proposed by Niehaus [Nie03] and Machado et al. [MNR10]. Niehaus introduces *random crossover* on directed graphs, where a subgraph of one graph is removed and replaced by a subgraph of another graph; in particular, only one offspring is computed. To avoid dangling edges, the exchanged subgraphs must have the same in- and out-degrees with respect to the edges that connect them to the rest of the graph. By using the trick of representing edges as a special kind of node, we can realize this crossover with our approach.

Machado et al. [MNR10] also exchange subgraphs between graphs. The subgraphs are constructed as radii around randomly chosen nodes. To connect the exchanged subgraphs to their new host graphs, a correspondence is established between the nodes that were adjacent to them in their former host graphs. If this correspondence is one-to-one, we can implement this operator in our approach by again representing edges by a special type of node. However, Machado et al. also allow for correspondences that are not one-to-one. To implement this

feature, we would need to allow non-injective mappings from the crossover point to the splits in our approach. Unlike this approach, our approach is not limited to choosing subgraphs as radii around randomly chosen nodes.

Other approaches are less general since they depend to a greater extent on the chosen representation or semantics of the graphs used [KB02; DZB10; DHK12; Nob+13; KRD17; HK18]. In these cases, it does not seem straightforward to apply the proposed crossovers in other contexts. The kind of computations that can be performed using crossover may also be less expressive than those in the approaches already discussed [Per+99; KB02; Nob+13; KRD17; HK18]. We can implement the crossovers proposed in [Per+99; DZB10; DHK12; Nob+13; HK18] in our approach, often by representing edges as a special type of node. The approach by Kantschik and Banzhaf [KB02] cannot be implemented for reasons similar to those discussed for [MNR10]. Furthermore, we cannot implement the *subgraph crossover* proposed in [KRD17], because this approach allows random insertion of new edges into an offspring and these edges do not come from any parent.

In summary, our generic approach to crossover on graph-like structures encompasses most of the approaches proposed for more specific situations. Our approach allows more general exchanges of subgraphs than most of the approaches discussed. Moreover, our Proposition 4.3 is the first result (that we know of) that formally clarifies the expressiveness of the proposed crossover. We have identified two reasons why our approach is not able to encompass an existing approach: First, crossover could cause two (or more) edges that targeted different nodes in their original graph to target the same node in their new context. Second, elements that do not originate from either parent are reintroduced in the offspring. However, both kinds of changes can be realized in our approach by the subsequent application of mutation operators. We could also solve the first problem by allowing non-injective mappings from crossover points to the splits when performing crossover. However, this would complicate the theory we can provide for our construction: Pushouts along any two morphisms need not exist in $\mathscr{M}$-adhesive categories, and even if the necessary pushouts did exist, ensuring that the computed results come from the search space under consideration (i.e., represent an $\mathscr{M}$-morphism) would only be possible for certain morphisms.

4.1.6.2 Crossover in MDO

In the rule-based approach to MDO, the solutions are represented as sequences of model transformations [Abd+14; Bil+19]. This allows traditional crossovers (e.g., k-point crossover, uniform crossover) to be applied seamlessly. However, they have been shown to be disruptive because the transformations can depend on each other and repair strategies must be used to mitigate this problem (see Chapter 2). As for the effects of crossover in the rule-based approach, no theoretical results are available. To date, neither a formal basis nor alternatives to traditional crossover have been developed in this context.

Burton et al. were the first to perform optimization directly on models as search space elements [Bur+12]. Their specific use case allows for the adaptation of single-point crossover through model transformations. However, their crossover implementation is not described in detail. Recent applications of the model-based approach neglect crossover and stick to mutation as their only change operator, such as [BZJ21]. In [ZM16], Zschaler and Mandow present a generalized view on the model-based approach to MDO and point out the challenge of specifying crossover in such a setting. They briefly discuss model differencing and model merging as related concepts, but do not elaborate on this idea. To our knowledge, this work presents the first approach to address this issue.

4.1.7 Conclusion

There is theoretical and practical evidence that evolutionary algorithms in general benefit from the use of crossover [DHK12; HK18; APS20] in the sense that the search for optimal solutions can be more effective and efficient. However, in the absence of suitable crossover approaches for (the model-based approach to) MDO, the effect of crossover in this context has not yet been studied. Our proposed generic crossover construction can serve as a basis to start with.

How existing solutions are split and the selection of common crossover points for such splits are critical design decisions. Which of these decisions are beneficial to the effectiveness and efficiency of an optimization remains to be explored. Apart from the typing of objects, our approach neglects additional constraints of an optimization problem, i.e., crossover may lead to violations of constraints.

Whether our approach needs to be refined to guarantee constraint-preserving offspring remains for future work. In addition to theoretical exploration of our approach, an implementation is needed to enable empirical analysis. Additionally, specification concepts need to be elaborated to allow users to conveniently specify different split strategies and crossover points that fit their domain.

4.2 An EMF-Based Crossover Operator

4.2.1 Introduction

A variety of software engineering problems such as software modularization [BBL10], software testing [WL05], and release planning [BRW01] can be viewed as optimization problems. *Search-based software engineering* (SBSE) [HJ01] explores the application of meta-heuristic techniques to such problems. One of the widely used approaches to efficiently explore a search space is the application of evolutionary algorithms [HMZ12].

With reference to e.g. [ES15; ZM16], the definition of an *evolutionary algorithm* requires a representation of problem instances and solutions. Formulated constraints and objectives determine the quality of solutions. A mechanism for generating new solutions from existing ones (typically involving mutation and crossover) and a selection mechanism (typically based on the concept of survival of the fittest) drive the evolution. Finally, a population of solutions to start from and a condition for stopping evolutionary computations are needed. It is a challenge to select these ingredients so that an evolutionary algorithm is effective and efficient.

The use of domain-specific models can facilitate the exploratory search for solutions, especially for structural software engineering problems where traditional encodings (e.g., as vectors) are hard to apply. Model-driven engineering (MDE) [Sch06] aims to represent domain knowledge in models and solve problems through model transformations. MDE can be used in the context of SBSE to minimize the expertise required of users of SBSE techniques. In Chapter 2, we referred to this combination of SBSE and MDE as *model-driven optimization* (MDO). Two main approaches have emerged in MDO: The *model-based*

approach [BP13; ZM16] performs optimization directly on models, while the *rule-based* approach [Abd+14; Bil+19] searches for optimized model transformation sequences.

Both use mutations to make local changes. Crossover, i.e., splitting and recombining existing solutions, has only been applied in the rule-based approach, because the concepts for conventional crossovers (such as k-point crossover and uniform crossover) can be seemlessly applied to rule call sequences. Crossover of models is not so obvious, since models are multidimensional structures. This is probably the reason why a crossover operator for the model-based approach to MDO is still missing. However, there is theoretical and practical evidence that evolutionary algorithms can benefit from the use of crossover [Doe+13; Sud17; HK18]. Since the model-based approach tends to be more effective than the rule-based approach (see Chapter 2), even without crossover, we take initial steps to implement and apply crossover in the model-based approach to MDO (hereafter referred to as MB-MDO for short).

In MB-MDO, problem instances and solutions are represented as domain-specific models, each containing a problem part with all relevant information about the given problem instance and a solution part with solution information. All models in the initial population have the same problem part. Model mutations are rule-based model transformations that preserve the problem part. Since models are close to graphs, it is quite obvious to use a graph-based construction such as [Nie03; MNR10] or our approach in Section 4.1 to implement crossover. However, these crossover constructions are very generic and do not consider model- or even domain-specific aspects. Since the Eclipse Modeling Framework (EMF) [Ste+08; emf] has become a de facto standard technology for defining models and modeling languages, and MB-MDO concepts have been developed and implemented for EMF models [BZJ21], a crossover operator for EMF models should also be available.

In this chapter, we present (1) an algorithmic design for a crossover operator for EMF models that generates EMF models as offspring. It is designed to be configurable so that the user can incorporate domain-specific knowledge. (2) We provide a prototypical implementation of our crossover operator based on Henshin [henshin], a model transformation engine for EMF models. (3) Initial

experiments show that the combination of mutation and crossover can be more effective than using only mutation in MB-MDO.

4.2.2 Related Work

In the rule-based approach to MDO, the solutions are represented as sequences of transformation units [Abd+14; Bil+19]. This allows conventional crossovers such as k-point and uniform crossover to be applied seamlessly. The crossover operators considered so far do not take domain-specific knowledge into account. An offspring sequence may contain unit applications that are no longer executable. By default, such units are simply ignored; they can also be deleted from the sequence [Abd+14]. A non-executable unit can alternatively be repaired by, for example, replacing it with a random executable unit [Bil+19]. Since there is no guarantee that the offspring solutions are feasible, more sophisticated repair strategies are needed to mitigate this problem.

Burton et al. were the first to perform optimization directly on models as search space elements [Bur+12]. Their specific use case allows adaptation of one-point crossover through model transformations. However, their crossover implementation is not described in detail. In [ZM16], Zschaler and Mandow present a generalized view on the model-based approach to MDO and point out the challenge of specifying crossover in such an environment. They briefly discuss model differencing and model merging as related concepts but do not elaborate on this idea. Recent applications of the model-based approach neglect crossover and stick to mutation as the only change operator, such as [BZJ21]. To our knowledge, this is the first approach to a crossover operator for the model-based approach to MDO and the first approach to a configurable crossover operator for MDO in general that will be able to incorporate domain-specific knowledge.

4.2.3 Running Example

The class responsibility assignment (CRA) case [BBL10] is a structural optimization problem in software engineering that has become one of the most well-known cases when considering MDO. Therefore, we use this case to recall

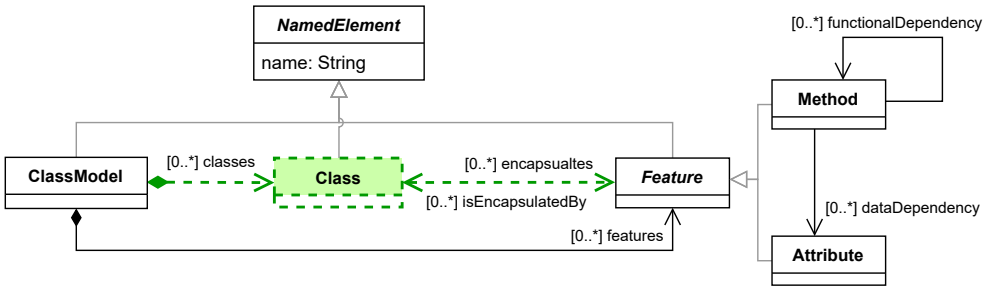


Figure 4.15 – Meta-model for the CRA case (slightly adapted from [FTW16])

the core concepts of MB-MDO and illustrate our crossover operator. The CRA case aims to create a high-level design for object-oriented systems. Starting from a class model with features and their usage relations as a problem instance, each partial assignment of features to classes forms a solution. What is sought is a complete mapping of features to classes such that coupling between classes is low and cohesion within classes is high.

A suitable meta-model for the CRA case is presented in [FTW16] and has been introduced in Section 2.3. Figure 4.15 recalls the meta-model with slightly adapted multiplicities. The meta-model specifies ClassModels that contain Features (i.e., Attributes and Methods) and prescribes the possible usage relations. Methods can use Attributes and Methods. To form a solution, Classes can be used to encapsulate Features. In Fig. 4.15 (and subsequent figures), uncolored solid elements are used to describe the optimization problem. They remain invariant during optimization, while the colored Class element with dashed border and its incoming and outgoing references can be created or removed.

We treat the CRA case as a *multi-objective problem* where two aspects of quality are important: cohesion and coupling. While cohesion means that dependent Features are within a single Class, coupling refers to the dependencies of Features between different Classes. Good solutions exhibit a class design with high cohesion and low coupling because it is considered easy to understand and maintain. Cohesion and coupling are measured by the *CohesionRatio* and *CouplingRatio* both presented in [FTW16]. Furthermore, the CRA case is a *constrained optimization problem*: A solution model is said to be *feasible* if *each Feature is associated with exactly one Class* (the feasibility constraints).

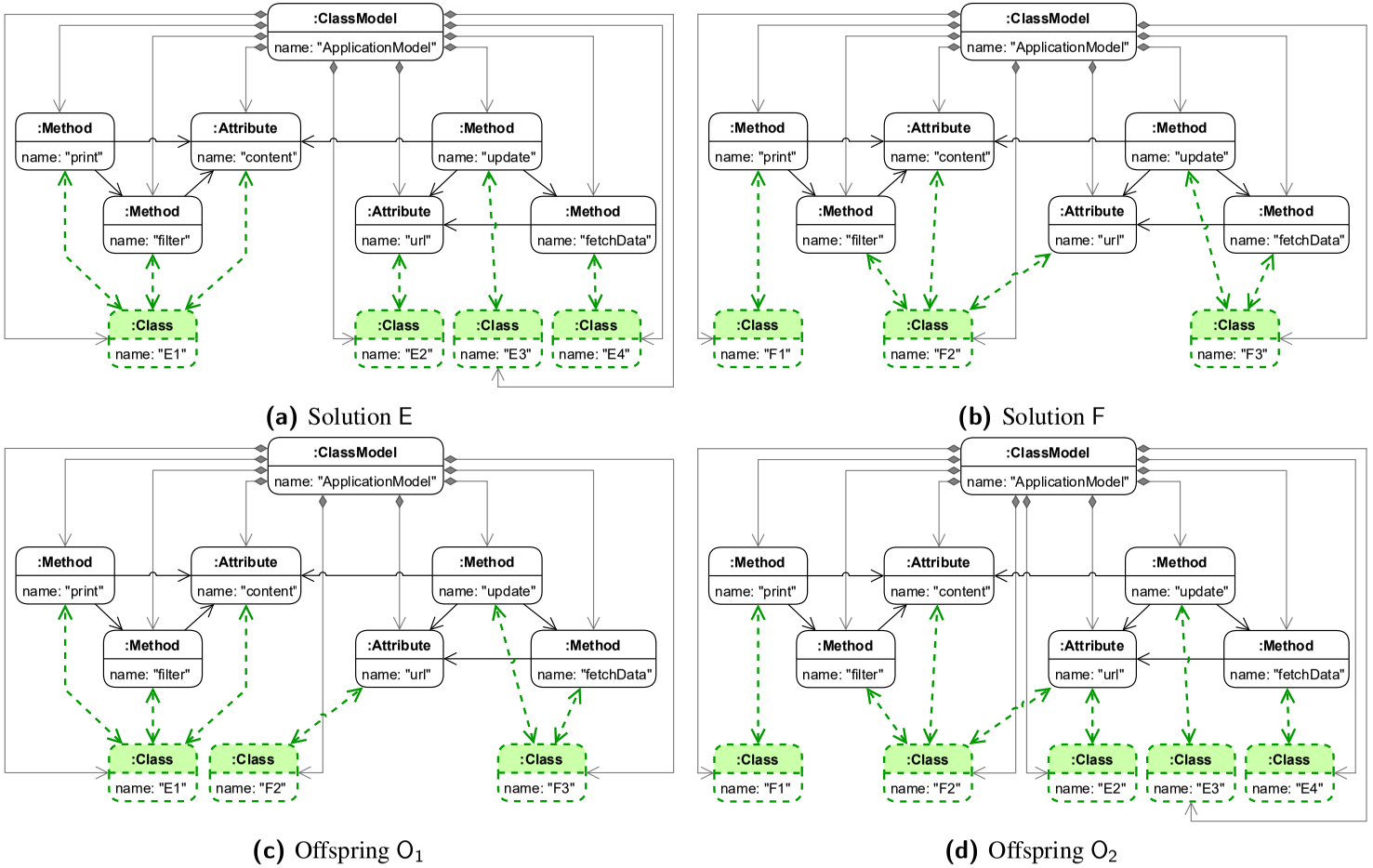


Figure 4.16 – Example of two solution models E and F of the CRA case and their offspring O_1 and O_2 generated by applying crossover. (Reference types are not shown.)

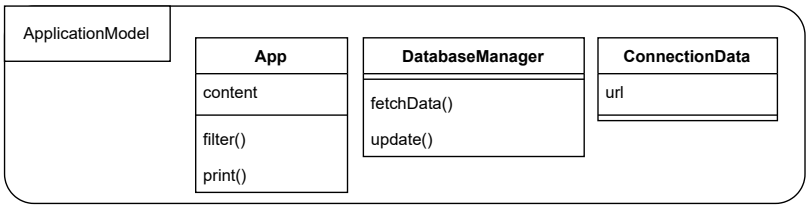


Figure 4.17 – Class diagram represented by offspring O_1 with class names adapted to reflect their purpose.

Mutations can locally change solutions, such as assigning a Feature to a new or existing Class or moving a Feature from one Class to another. Crossover of solutions is of interest when parts of them are already promising, so that decomposing and recombining them can lead to solutions with better fitness. In the following, we present a crossover example on two solutions E and F (Figs. 4.16a and 4.16b) for the same problem instance, which consists of six interdependent Features: two Attributes and four Methods. Combining Attribute content and Methods print and filter in one Class (as in solution E) seems reasonable. Their pairwise dependencies promote cohesion, while splitting them would lead to coupling. Similarly, combining Methods update and fetchData like in solution F makes sense. A possible crossover of E and F that combines those assignments combines the best of both worlds. The resulting offspring solution, called O_1, is shown in Fig. 4.16c. In fact, we can assign a descriptive name to each of the classes in O_1 that reflects its purpose from a semantic point of view; Fig. 4.17 shows the resulting class model in concrete syntax. Offspring O_2 (in Fig. 4.16d) is constructed by combining the opposite assignments. Note that the assignments of Attribute url have been transferred to O_2 from both parents. This shows that crossover may generate infeasible solutions. Infeasible offspring can either be repaired after crossover or are discarded due to infeasibility or low fitness.

4.2.4 A Configurable Crossover Operator for EMF Models

In this section, we introduce our crossover operator for EMF models and argue that this operator always generates EMF models as offspring, i.e., models that

satisfy general EMF constraints. We also discuss how it preserves a specific form of rootedness expected from the input models. First, we briefly introduce the general setting for our work.

4.2.4.1 Setting

EMF models. The *Eclipse Modeling Framework* (EMF) [Ste+08; emf] has established itself as a leading framework for model-driven engineering. In EMF, *meta-models* are used to specify the syntax of domain-specific (modeling) languages. To further constrain meta-model *instances* that are considered *valid*, a meta-model can be additionally equipped with *multiplicities* and further *well-formedness constraints*, usually defined in the *Object Constraint Language* (OCL) [ocl]. Recall that any EMF model has a hierarchical containment structure over its elements to support its efficient processing. Specifically, an EMF model must satisfy the following *EMF constraints*: (1) It is *concretely typed*, i.e., none of its object nodes has an abstract type. (2) Each object node has at most one container, i.e., at most one incoming reference of a containment type. (3) There are no containment cycles. (4) There are necessary opposite references, i.e., if a reference has a type with opposite type (in the meta-model), there is also a reference of the opposite type pointing into the opposite direction. (5) There are no parallel references of the same type between the same two nodes. We call an EMF model *rooted* if its containment structure is a tree, i.e., if it has a single, non-contained node (called *root node*) that (transitively) contains all other nodes.

EMF models as representations for MB-MDO. Meta-models are used to define the underlying modeling language; a meta-model generally consists of structural part specifying the used types and references, as well as a set of language constraints including EMF constraints, multiplicities, and further well-formedness constraints. In MB-MDO, models directly encode optimization problems and their solutions. To define these, a meta-model is divided into two parts. The *problem part* is a sub-meta-model for defining an optimization problem; the remaining meta-model part is used to model solutions; we call a meta-model with this distinction a *computation meta-model* (see Def. 3.1). The *problem model* of an instance model consists of all those elements that are typed

over the problem part of the meta-model. An *optimization problem* consists of a computation meta-model, a (possibly empty) set of feasibility constraints, and a non-empty set of objective functions. A concrete *problem instance* of an optimization problem is an instance model, denoted *PI*; its *problem model* is denoted PI_{P}. The *search space* of a problem instance *PI* consists of all instance models whose problem model matches PI_{P}. An element of the search space is called *solution* (*model*). A solution is *feasible* if it satisfies all feasibility constraints. The objective functions are used to determine the quality of a solution with respect to the given optimization problem.

To recall, Fig. 4.15 shows a meta-model for the CRA case. Uncolored solid elements constitute the problem part; accordingly, they form the problem models in the solutions depicted in Fig. 4.16. Any solution can serve as a problem instance as its problem model defines a search space.

Requirements for the crossover operator and assumptions. Usually, a crossover operator takes two solutions, splits them, and recombines the parts crosswise. A such computed solution model can show two forms of infeasibility: In the extreme case, even validity of EMF constraints is lost; in the moderate case, the solution merely violates feasibility constraints. In this chapter, we develop a crossover operator that generates at least rooted EMF models. To deal with the other feasibility constraints, standard techniques for constraint handling in the context of evolutionary algorithms can be used (see, e.g., [Mic95; Coe10]). Those include not selecting infeasible solutions for the next population, reducing their fitness (proportional to the degree of infeasibility) to reduce the probability of their selection, or applying additional repair operators. By preserving the EMF constraints, we ensure that EMF-based tooling can be used for evolutionary computations. Furthermore, crossover operators that preserve at least basic structures of solutions have proved to increase the efficiency of evolutionary search in other contexts (e.g., [Pot96; Doe+13]). To develop a crossover operator which can produce rooted EMF models efficiently, we assume the problem model of an input solution to be a non-empty, rooted EMF model that forms the beginning of the containment hierarchy of the whole solution model. That is, its root coincides with the root of the complete solution model and no problem part element is contained by an element of the solution part. We call such solution models *problem-rooted*.

4.2.4.2 Splitting of Solution Models

We now show how we split a solution model into two parts for crosswise recombination. The *input* of the splitting algorithm is a solution model E for a given problem instance PI; we assume that E is a problem-rooted EMF model. As *output*, the splitting algorithm computes two models E_1 and E_2, called *split parts*. Since models are graph-like structures, it may be useful to look at crossover for graphs. In Section 4.1, we have developed a generic crossover operator for graph-like structures that can preserve basic graph structures. We can use this approach as a generic framework for the algorithmic design of our crossover operator for EMF models. Consequently, a split of a solution model must satisfy the following properties:

(1) Each split part is a *sub-model* of E, i.e., it contains only elements and attribute values from E.
(2) Each split part is a solution model for PI.
(3) E_1 and E_2 together *cover* E, i.e., every element of E occurs in at least one of them.

Additionally for EMF models:

(4) Each split part is a problem-rooted EMF model.

Default splitting algorithm for EMF models. The key idea of Algorithm 4.1 is to split model E in such a way that the branches of the containment tree of E that leave PI_P are distributed to E_1 or E_2.

Algorithm 4.1 – Splitting algorithm

```
1  input:  solution model E                    //E is problem–rooted
2  output: models E1 and E2                    //split parts
3          model EI                            //split point
4
5  E1,E2,EI ← PIP
6  compute Ctemp as the set of containment edges outside of EI whose
       source nodes belong to EI
7
8  while Ctemp is not empty
9    if |Ctemp| = 1
10     include that containment edge in E1, E2, and EI
```

```
11      recompute C_temp
12    else
13      C_1,C_2 ← split(C_temp)
14      include subtrees starting within C_1 in E_1
15      include subtrees starting within C_2 in E_2
16      break
17  end while
18
19  for all non-containment references r in E
20    if r is opposite to a reference in E_i (with i = 1,2)
21      include r in E_i
22    else include r in E_1 or E_2 (or both)
23  end for
```

In the first iteration, C_{temp} is the set of containment edges whose source nodes belong to PI_{P} but whose target nodes do not. Inclusion of containment edges comprises also the inclusion of corresponding target nodes (line 10). Splitting divides C_{temp} into two (not necessarily disjoint) sets (line 13). Inclusion of subtrees means that each object node contained (transitively) in E by a containment edge from C_i is assigned (along with its incoming containment edge) to E_i (where $i = 1,2$) (lines 14,15). When including non-containment references, a split part is preferred if it contains both adjacent nodes of the reference and the other does not (line 22). If one of the adjacent nodes of a reference is missing from the model E_i to which it is assigned, this node (together with the containment structure leading to it) is additionally included in the respective split part. If an element (object node or reference) is included in both split parts, it is also included in the *split point* E_I. Thus, E_I is the largest common sub-model of E_1 and E_2 (i.e., their intersection in E); it always contains at least the given problem model PI_{P}. We will use these split points in the definition of our crossover operator.

Algorithm 4.1 computes two problem-rooted EMF models with the split properties above: Each element of E is assigned to at least one of the split parts E_1 and E_2 (Prop. 3), and E_1 and E_2 receive only elements from E, i.e., they form sub-models of E (Prop. 1). Since E_1 and E_2 both extend PI_P, they are solution models (Prop. 2). Regarding Prop. 4, E_1 and E_2 cannot contain abstract types, multiple containers for a node, containment cycles, and parallel edges of the same type (as such violations do not occur in E). Furthermore, references

are always assigned together with their opposite counterpart (if existent) and each object node always together with its container, preventing gaps in the containment hierarchy.

In our running example (Section 4.2.3), the splitting algorithm computes as C_{temp} the set of containment edges of type classes. One possible split of E into split parts E_1 and E_2 would be to put Class EC1 into E_1 and Classes EC2, EC3, and EC4 into E_2. When distributing all encapsulation references along with their classes, the resulting split point would be the problem model.

Configuration points. In the splitting algorithm, the split of C_{temp} and the distribution of non-containment references can be configured by controlling the assignment of elements. Selected elements can either be separated or added to the same split part; the split point size can be adjusted by either adding an element to just one or to both split parts. Furthermore, the size of both split parts can be chosen to be similar or significantly different.

4.2.4.3 Recombining Two Solution Models

Next, we explain how the splits of two models are recombined crosswise. We assume that two problem-rooted solution models E and F are given for the problem instance PI, together with split parts E_i, F_i (where $i = 1,2$) and split points E_I, F_I computed as introduced above. The split parts E_1 and F_2 as well as E_2 and F_1 are recombined to compute the *offspring models* O_1 and O_2. Formally, this recombination is the union of the respective split parts over a common sub-model CP, called *crossover point*. CP must be a common sub-model of the two split points E_I and F_I and must minimally include the problem model PI_{P}. Following Def. 4.7, O_1 and O_2 must satisfy the following properties:

- Both E_1 and F_2 are sub-models of O_1, i.e., all their elements (objects and references) occur in O_1. Similarly, E_2 and F_1 are sub-models of O_2. In particular, both O_1 and O_2 contain PI_{P}, i.e., they are solutions.
- E_1 and F_2 together cover O_1, i.e., every element of O_1 occurs in at least one of them; elements occurring in both are exactly the elements from CP. Similarly, E_2 and F_1 cover O_2.

Additionally for EMF models:

- Both offspring models O_1 and O_2 are problem-rooted EMF models.
- Except for the objects from PI_P, the attribute values in O_1 and O_2 are allowed to differ from their counterparts in the model splits.

Default recombination algorithm for EMF models. The core of the recombination algorithm is the determination of a crossover point *CP*. To ensure that the containment structure of the offspring forms a tree, we initialize *CP* with the given problem model PI_P, which is the beginning of the containment structure, and then extend *CP* from top to bottom. Specifically, the recombination algorithm is described as follows.

Algorithm 4.2 – Recombination algorithm

```
1  input:  E1, E2, F1, F2                    //model splits
2          EI, FI                            //split points
3  output: models O1 and O2                  //problem–rooted solution models
4
5  CP ← PIP
6  compute CB as pairs of identifiable cont. edges from EI and FI
7
8  while (CB is not empty and StopCrit = false)
9    include a pair (eE,eF) from CB in CP
10   recompute CB
11 end while
12
13 include induced non-containment references in CP
14
15 O1 ← union of E1 and F2 over CP
16 O2 ← union of E2 and F1 over CP
17
18 recompute attribute values in solution parts of O1 and O2
```

In Algorithm 4.2, the set C_B of *pairs of identifiable containment edges* is defined as follows: The *border B* of *CP* consists of all object nodes *n* of *CP* whose counterparts in both E_I and F_I have outgoing containment edges to target nodes not yet included in *CP*. A pair (e_E, e_F) from E_I and F_I of such outgoing containment edges belongs to C_B if they have the same type, the same source node in *B*, and the types of their target nodes are the same or one inherits from the

other (line 6). Another check, whose technical details we omit, ensures that their identification cannot introduce parallel edges. The inclusion of a pair (e_E, e_F) from C_B in *CP* means that a containment edge e_{CP} of the same type as e_E and e_F is included in *CP* along with its target node and is mapped to e_E and e_F (line 9). The type of the target node in *CP* is the higher of the corresponding types in E_I and F_I. `StopCrit` refers to a user-defined stopping criterion (line 8). To avoid parallel edges after recombination, for each pair of nodes (n_1, n_2) from *CP* (where $n_1 = n_2$ is allowed): If there are two non-containment references of the same type between n_1 and n_2 in E_I and F_I, a reference of this type must be included in *CP* (line 13). "Union over *CP*" means that elements from E_1 and F_2 for which each has a counterpart in *CP* should coincide and occur only once in the offspring (lines 15,16). If the types of such identified object nodes differ, the smaller of the two is chosen in the offspring to ensure that all necessary reference types are defined.

Both offspring are problem-rooted, since the containment hierarchy of the crossover point *CP* is the beginning of the containment structures of E_I and of F_I. Thus, the containment structure of offspring O_1 arises from the extension of *CP* by the branches from E_1 and from F_2; the extension of a tree by branches cannot destroy the tree structure.

In our example (Section 4.2.3), the crossover point consists only of the problem model. It cannot contain any other elements because already the split points coincide with the problem model. To identify Classes FC2 and EC2 in O_2, one would first need to assign them (and their references to attribute url) to their respective split points during the split algorithm. If this is the case, the containment edges leading to those Classes become part of C_B and can be included in *CP* during the recombination algorithm.

Configuration points. In the recombination algorithm, both the inclusion of containment edges into *CP* and the final setting of attribute values can be further configured. With respect to the inclusion of containment edges, domain-specific information could be used to favor or avoid certain identifications. For example, pairs of containment edges could be removed from C_B after a certain number of rounds if they were not selected for identification. Resetting attribute values can be done according to user-defined instructions. In particular, if

finding appropriate values for particular attributes is part of the optimization task, it might be beneficial to include ideas designed for crossover on data, e.g., calculating the value of a numeric attribute in the offspring as the average of the values in the parents.

4.2.5 Implementation

We have implemented a prototype crossover operator for EMF models to conduct experiments. It implements the algorithms for splitting and recombining EMF models presented in Sec. 4.2.4. For the configuration points of the split, we chose a random distribution of containment subtrees (line 13 in Alg. 4.1) and non-containment references (line 22 in Alg. 4.1) among the split parts. A distribution ratio can be specified to skew this distribution. In our prototypical implementation of the recombination algorithm, we neglect the option to extend the crossover point and immediately set the stop criterion to `true`, i.e., we restrict ourselves to the inclusion of the problem part for the time being. Note that the problem parts of two solution models are usually not identical in terms of object identity, but merely isomorphic. We use the model transformation language Henshin [henshin] in the construction of the crossover point to identify the necessary mappings between them. Henshin is also used to implement the actual recombination of two split parts by model transformations. The implementation can be found at [mdover].

4.2.6 Initial Evaluation

We conducted experiments focusing on the following research question; all evaluation data can be found at [JKT22b]:

RQ: Can evolutionary search of models be more effective if it uses mutation and crossover instead of just mutation?

Set up. Our running example (Section 4.2.3), the CRA case, also serves as a use case for the initial experiments. Problem instances range from 9 features and 14 dependencies (Model A) to 160 features and 600 dependencies (Model

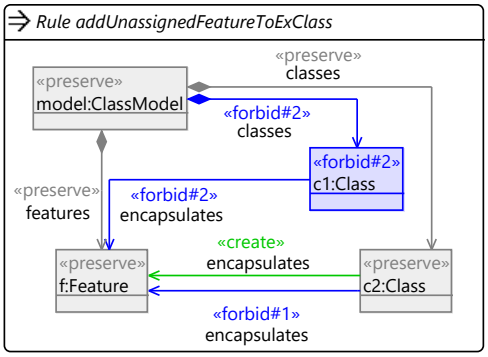


Figure 4.18 – Mutation rule (2) in Henshin syntax. The green edge (as well as its opposite edge isEncapsulated) is created. The blue elements prevent application of the rule if Feature f is already assigned to the same or another Class.

E). To perform optimizations we use the framework MDEOptimiser [BZJ21; mdeo]. It supports Henshin transformation rules as mutation operators; to apply crossover, we integrated our crossover operator presented in Sec. 4.2.5. As the underlying evolutionary algorithm we choose NSGA-II [Deb+02], which is generally well established in both MDO [Abd+14; BZJ21] and SBSE [HMZ12]. In our configuration, depending on a certain *crossover rate*, a pair of solutions can participate once in crossover. Both (possibly unchanged) models can additionally be subjected to the application of a single mutation operator, depending on a certain *mutation rate*.

For the mutations, we rely on the transformation rules proposed for the CRA case in [BZJ21], which we introduced in Section 2.3 and later discussed in more detail in Section 3.7. They implement the following operations: (1) Create a new class and assign an unassigned feature to that class. (2) Assign an unassigned feature to an existing class. (3) Unassign a feature and assign it to another existing class. (4) Delete a class to which no features are assigned. As illustrated at the example of the second rule (in Fig 4.18), all rules are designed so that they do not violate the feasibility constraints of the CRA case. Specifically, applications of the rules can never cause a feature to be assigned multiply or to not be assigned at all. Formally, w.r.t. the feasibility constraints, the selected mutation rules are *sound* in the sense of Def. 3.8 and even *consistency-sustaining* in the sense of [Kos+22].

Crossover can easily introduce new violations of feasibility constraints. For example, for the CRA case our crossover operator can introduce features which are assigned to multiple classes. To cope with these new violations, we optionally allow a repair step after crossover that removes multiple assignments of features: it deletes (randomly) incoming isEncapsulatedBy-edges of a feature until only one is left. Since the initial population does not contain multiply assigned features and the mutation rules cannot introduce such, it is important to note that this repair process only repairs violations that have just been introduced by the application of the crossover operator.

After an evolution of the population, the solutions for the next population are selected according to their fitness and feasibility. For details we refer to [Deb+02]. For all problem instances, an initial population of 100 models is used, each of which is generated by the standard initialization procedure of MDEOptimiser: a replica of the problem instance is mutated twice. The search stops if no relevant improvement has occurred in the last 100 iterations.

In our experiments, we varied the crossover and mutation rates and also whether or not the repair step is applied. For each of these algorithm variants and each problem instance, we performed 30 evolutionary computations. Below, we present the results for a selection of these variants, highlighting the most relevant findings.

Since cohesion and coupling of a class are conflicting objectives, in the CRA case there is no single solution that is better than all the others. Instead, an evolutionary search produces a set of pairwise incomparable non-dominated solutions; we call this an approximation set. A solution is said to dominate another one if it is as good as the other one w.r.t. all objective functions and better w.r.t. at least one of them. We refer the reader to [ES15] for more details on multi-objective optimization and the concepts of dominance and Pareto optimality. To compare the quality of approximation sets, we use a normalized version of the well-known hypervolume indicator [ZBT07] where one solution set is considered better than another if its hypervolume is closer to 1. The reason for considering hypervolume is that, among the common metrics, hypervolume is the only one that is Pareto compliant [ZBT07]. Also, in contrast to other metrics, it considers not only convergence but also diversity of solution sets.

Results. In this initial evaluation, we compare the effectiveness of algorithm variants based on the mean hypervolume of their approximation sets and their coverage of the search space. Table 4.1 summarizes the mean hypervolumes for what we found the most interesting algorithm variants. For four of them, Fig. 4.19 depicts the cumulative approximation sets obtained for model D. A cumulative approximation set combines the approximation sets of all evolutionary computations performed for an algorithm variant into a set of non-dominated solutions. The labels indicate the mutation (Mxxx) and crossover rates (Cxxx) used in percent. The application of the repair step is indicated by a trailing R. M100-C0 is the basic variant without crossover. We first tried adding crossover at a rate of 100 percent and without repair (M100-C100). Obviously, the effect on the hypervolume is devastating for all but the smallest problem instance. This did not hit us unexpectedly, since our crossover operator can make large changes to parent solutions. If applied too often, this can hinder the gradual improvement of the population and lead to premature stagnation of the search. For this reason, we tried several variants with lower crossover rates. One promising variant is M100-C10. In terms of mean hypervolume, it is close to M100-C0 for smaller models, but less effective for larger models. We attribute this in part to the fact that our crossover operator can lead to constraint violations. This can lead to wasted evolution steps, as the resulting infeasible solutions are discarded. By adding a repair step, variant M100-C10-R outperforms the basic variant in terms of mean hypervolume for all models. Note that the observed differences are not statistically significant considering a *p-value* of 0.05. Variants with higher crossover rates and repair, i.e., M100-C20-R and M100-C30-R, perform even better on smaller models (for models B and C significantly better than M100-C0). With increasing model size, however, variants with lower mutation rate are more effective. While M90-C10-R significantly outperforms M100-C0 on model D, M80-C10-R does so for model E. Most interestingly, these variants appear to be more effective in certain regions of the search space (see Fig. 4.19). They produce highly cohesive solutions with low coupling which cannot be found by other variants, particularly not by M100-C0.

In summary, even with our prototypical crossover implementation, using crossover (plus repair) and mutation to perform evolutionary search can be more effective and can cover a larger part of the search space than using mutation only.

Table 4.1 – Mean normalized hypervolume (higher values are better) of selected algorithm variants. Bold numbers mark the best results for each problem instance.

Algorithm	Metric	A	B	C	D	E
M100-C0	Mean HV	0.924	0.841	0.738	0.574	0.596
M100-C100	Mean HV	0.846	0.332	0	0	0
M100-C10	Mean HV	0.915	0.838	0.743	0.538	0.455
M100-C10-R	Mean HV	0.929	0.871	0.759	0.584	0.609
M100-C20-R	Mean HV	**0.934**	**0.883**	0.77	0.576	0.595
M100-C30-R	Mean HV	0.933	0.879	**0.779**	0.583	0.579
M90-C10-R	Mean HV	0.89	0.603	0.506	**0.601**	0.594
M80-C10-R	Mean HV	0.774	0.586	0.538	0.483	**0.618**

Threats to Validity. Being a descriptive example, the CRA case exhibits only a low structural complexity and can also be solved with traditional encodings [SPG10] . The initial evaluation includes only a single use case and NSGA-II as the only underlying algorithm. Of course, this is not sufficient to generalize our findings. For this purpose, additional use cases and algorithms need to be considered. In particular, how well the proposed crossover works on use cases with a more complex structure (e.g., the next release problem [BZJ21]) remains an open question. However, from the perspective of the research question, the evaluation is still meaningful. It shows that the application of our crossover operator, even in its prototypical form, can be beneficial, at least in the configurations considered. This motivates and justifies further analysis of (variants of) our crossover operator.

4.2.7 Conclusion

In this chapter, we have presented a crossover operator for EMF models that always generates EMF models as offspring. Since this operator is configurable, it can also accommodate domain-specific knowledge. Based on our initial results, we outline several topics for future work.

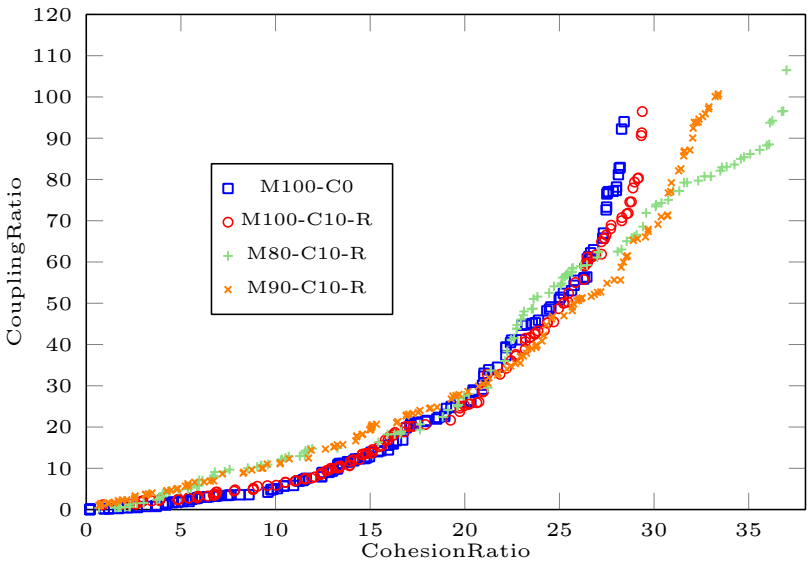


Figure 4.19 – Cumulative approximation sets of selected algorithm variants for model D.

To illustrate our crossover operator, we applied it to the class responsibility assignment problem. The CRA case was also used to illustrate the rule-based approach to MDO and problem-specific approaches, such as evolutionary search in [SPG10]. Future work is needed to compare to existing approaches for crossover operators in terms of effectiveness, efficiency, and simplicity of their specification.

The proposed crossover operator may lead to violations of feasibility constraints. It is an open question how to configure a crossover operator to *preserve* (most of) *the constraints* of a given language. To find feasible offspring models, it might be helpful to first choose the crossover point and select the split parts accordingly. Upper bounds could be satisfied, for example, by identifying common nodes and edges in the crossover point [TK22]. It could also be helpful to focus on computing only one offspring (instead of two) to obtain good offspring.

More generally, we can ask: *What is the configuration space for crossover operators and how can it be controlled from a domain-specific perspective?* A domain-specific language (DSL) for configuring the crossover operator could help the user specify the split and crossover points in a way that leverages

domain-specific knowledge. The configuration facilities of such a DSL need to be evaluated at a selected set of optimization problems.

5 Conclusion

In the following we conclude this work by summarizing our main findings and recapping how they contribute to the research questions identified in section 1.2. Finally, we will also discuss research directions for future work.

5.1 Summary

By using models and model transformations to specify and solve optimization problems, MDO aims to make SBSE techniques more accessible to both domain experts and software engineers. However, how to develop efficient and effective evolutionary algorithms with MDO is still a central question. This thesis addresses seven research questions related to this topic.

Two approaches to encode solutions have been developed in MDO. RB-MDO represents solutions as sequences of rule calls while MB-MDO works directly on models. So far, it has been unclear what their advantages and drawbacks are (RQ1) and how they compare with regard to their optimization effectiveness and efficiency (RQ2). In a systematic comparison of the two approaches we provide insights into their features and the implications of using one or the other. We conclude that although RB-MDO is more flexible with respect to evolutionary operators it has two major drawbacks. While the quality of solutions can directly be assessed in MB-MDO, in RB-MDO rule calls of a sequence need to be applied to an input model to obtain an assessable solution model. This overhead slows down the optimization process. Furthermore, in RB-MDO rule calls can depend on each other. These dependencies are easily broken by evolutionary operators and lead to epistatic effects which are known to negatively

affect optimization [WCT12]. Traditional crossover operators showed to be particularly problematic in this regard. We hold these drawbacks responsible for MB-MDO outperforming RB-MDO especially for larger input models; better solutions are found in less time. As real-world problems tend to be large and MB-MDO performed well even without a yet missing crossover concept, we considered MB-MDO to be the more promising approach to investigate.

However, we found that a formal foundation for rigorous analysis and extension of MB-MDO was lacking. Consequently, we developed a formal framework for MB-MDO (without crossover) based on graph transformation theory. In accordance with RQ3, the framework precisely specifies the key concepts of evolutionary algorithms in MB-MDO; specifically, it details the use of graph transformation rules as *element mutation operators*, i.e., as operators that implement a change of a single model. At the same time, the framework abstracts from the concrete implementation of evolutionary operators to allow the instantiation of a wide variety of evolutionary algorithms found in practice.

To investigate the importance of evolutionary operator properties on the effectiveness and efficiency of evolutionary algorithms in MB-MDO (RQ5), we defined *soundness* and *completeness* as two properties of (sets of) element mutation operators. Typically, an optimization problem specifies a set of so-called *feasibility constraints*, which may be violated in the course of an optimization, but must be satisfied by feasible solution models in the end. We consider an element mutation operator to be sound if its application cannot introduce violations of feasibility constraints. Consequently, a set of element mutation operators is sound if all of its operators are sound. A set of element mutation operators is complete if all feasible solutions can be reached from any given solution (feasible or not) by only applying (possibly multiple times) element mutation operators from that set. We evaluated the impact of both properties on the efficiency and effectiveness of three state-of-the-art evolutionary algorithms and found soundness to be generally desirable. The use of unsound element mutation operators slows down the evolution towards better solutions or even leads to stagnation and premature termination of an optimization. The effect of incomplete sets of element mutation operators is generally hard to predict. We recommend using complete sets of element mutation operators as a baseline when tackling new optimization problems. Afterwards, incompleteness should only be introduced by well-considered and purposeful decisions. Overall, we have shown that both

soundness and completeness are important properties of element mutation operators that need to be considered when designing evolutionary algorithms in MB-MDO.

To demonstrate how our framework can be used to specify experiments in MB-MDO in a precise and reproducible way (RQ4), we detailed all aspects of the experiments of the above evaluation using the concepts introduced by our framework.

To address the problem of MB-MDO lacking a concept of how to perform crossover on models (RQ6), we complemented our framework with a formalization of a generic construction for crossover operators on graph-like structures. The approach is generic in the sense that it can be applied regardless of the optimization problem at hand, i.e., problem-agnostic crossover operators can be constructed. It also offers several configuration points to embed domain-specific information and tailor crossover operators to specific optimization problems. However, apart from the correct typing of elements and the preservation of a graph structure, the generic construction does not consider constraints. Therefore, to facilitate the practical application of our crossover construction, we also developed an EMF-based [Ste+08; emf] concretization of our generic approach. EMF is an established modeling framework in the MB-MDO community [Bil+19; Str17; BZS18; Hor+22] and imposes various constraints that need to be fulfilled by models in order to be processable. Our EMF-based approach preservers these constraints and guarantees to produce EMF models as offspring.

Finally, following this EMF-based approach, we provide a prototypical and problem independent implementation of a crossover operator. To get first insights on whether or not crossover can be beneficial for the effectiveness and efficiency of evolutionary algorithms in MB-MDO (RQ7), we performed an initial evaluation with various configurations of evolutionary operators using this prototype. The results are promising. We observe that even the application of this still rather unpolished crossover can contribute to the quality of the solutions found. More interestingly, certain regions of the search space seem to be explored more thoroughly with crossover than with mutation alone.

Overall, we believe that this thesis significantly advances MB-MDO by providing new insights and opportunities to develop efficient and effective evolutionary algorithms. In particular, it completes the concept of evolutionary algorithms in

MB-MDO through the theoretical and practical implementation of a crossover operator. Regarding the availability of typical evolutionary operators, MB-MDO is now on par with RB-MDO. Furthermore, the analysis of soundness and completeness provides a valuable guideline for the design of effective and efficient mutation operators in MB-MDO; a similar guidance is still missing in RB-MDO. Our formal framework showed to be crucial to enable this development and will certainly prove useful for future enhancements of MB-MDO.

5.2 Future Work

Although our work already provides some insight into MB-MDO, we view it primarily as a foundation for the further investigation and development of MB-MDO. While some questions have been answered, many others have emerged that need to be addressed in future work. Most prominently, properties of evolutionary operators, their impact on optimization, and their realization in the design of the operators remain a major topic.

Element mutation operators, which are implemented as graph transformation rules in our framework, are of particular interest as they can be specified and analyzed drawing on the rich theory of graph transformation. So far we showed that soundness and completeness are important properties of (sets of) element mutation operators. We plan to extend this work by considering further properties related to the preservation of constraints. With regard to graph constraints our notion of soundness is equivalent to the notion of *consistency-preserving* graph transformation rules [HP09]. That is, a graph already satisfying the considered constraints will not violate them after the application of such rules. Recently, Kosiol et al. defined additional properties of graph transformation rules concerned with constraint violations [Kos+20b; Kos+22]. *Consistency-sustaining* rules, for example, refrain from introducing new constraint violations, while *consistency-improving* rules decrease the number of constraint violations. Given our results on soundness, investigating these properties in the context of (sets of) element of mutation operators seems a logical next step. Furthermore, as briefly discussed in Section 3.7, the effect of soundness as well as of other properties

concerned with constraint violations depends on the constraint handling mechanism employed by an evolutionary algorithm. The interplay between these properties and different constraint handling mechanisms needs to be studied.

We speculate that the effect of such properties might also be related to the different stages of an optimization. Different (sets of) element mutation operators might be suited for different stages. Optimization often starts from solutions with many violations of the respective feasibility constraints. In these cases, it may be favorable to begin with a (possibly incomplete) set of consistency-improving element mutation operators to speed up the discovery of feasible solutions. As soon as feasible solutions are found, switching to a complete set of sound operators may become advantageous. To the best of our knowledge, such a context dependent switch of operators has not yet been considered in MB-MDO.

With respect to the practicality of our findings to date, we are aware that soundness and completeness checks are not yet fully automated. For soundness checks we used existing tools from Nassar et al. [Nas+18; Nas+20]. Some manual steps, choices, and an interpretation of the results are needed though. Completeness entirely needs to be proven manually. While in our use cases these proofs turned out to be not too complicated, this might not generally be the case. A coherent, tool-supported approach for analyzing the soundness and completeness of (sets of) element mutation operators would greatly support users of MB-MDO in designing mutation operators.

Beyond the study of element mutation operators, the introduction of crossover operators for MB-MDO opens up additional research possibilities. Our prototypical implementation of crossover has already yielded promising results. However, the initial evaluation considered only one use case. Clearly, a more representative set of use cases needs to be considered in the future. Furthermore, we believe that the true potential of our approach lies in the integration of domain knowledge. Due to the abstract nature of our approach, integrating domain knowledge may not be trivial. To that end, a comprehensive way of specifying the configurable parts of our crossover (i.e., the split and crossover points) needs to be found. Additionally, optimization problems often share structural similarities (e.g., the existence of container objects in assignment problems). Whether or not this knowledge can be used to prepare reusable patterns for the integration of domain

knowledge has to be investigated. In summary, a thorough study is needed to determine the potentials and limitations of our crossover construction.

While we have studied soundness and completeness on (sets of) element mutations operators only, the reachability and feasibility of solutions are typically subject to all evolutionary operators. In particular, the role of soundness and completeness in the design of crossover operators and also in the interplay between mutation, crossover, and selection needs to be investigated. As a first step, Thölke and Kosiol have recently developed a multiplicity preserving crossover operator [TK22] based on our generic crossover construction, but their approach has yet to be implemented and evaluated. How other types of constraints can be preserved by crossover operators in MB-MDO is still an open research question.

A question not specific to MB-MDO, but to MDO in general, is how the efficiency of MDO compares to traditional encoding approaches. Querying and copying large models or performing model transformations on them can be costly and become a bottleneck in the optimization process. Encodings specifically suited for certain optimization problems, combined with specialized operators are likely to be more efficient. A study comparing the model-driven and the traditional encoding approaches is needed to validate this assumption. The automatic translation of models into more concrete artifacts (e.g., platform-specific source code) is as typical use case in MDE. Thus, future work might consider the automatic generation of more efficient representations from the models and model transformations used in MDO. A first paper in that direction is currently under review [ADS23].

We see parallels between our research on soundness and completeness and the problems currently associated with RB-MDO. The probably most pressing issue in RB-MDO is the handling of dependencies between rule calls. Traditional mutation and crossover operators are likely to break such dependencies ruining the progress of an optimization. So far only trivial repair steps have been considered to mitigate the problem [Bil+19]. While more sophisticated repair strategies might be a route to go, we see more potential in evolutionary operators tailored towards RB-MDO. Respecting dependencies can be considered a feasibility constraint in RB-MDO. If evolutionary operators are sound in that regard, i.e., respect dependencies between rule calls in the first place, repair steps might

become superfluous. Certainly, whether or not such sound operators can still explore all relevant parts of the search space (i.e., are complete) will also be important. It is up to future work to precisely define soundness an completeness in the context of RB-MDO and to develop operators implementing these properties.

Finally, we note that the development of a formal framework for MB-MDO has been a driving force in extending and analyzing the approach. We are confident that for advancing RB-MDO a similar framework would be of great value. In fact, a comprehensive framework for MDO that can be instantiated to MB-MDO, RB-MDO, and possibly other new encoding approaches is one of our long-term goals.

A

Appendix to Chapter 3

This appendix complements the presentation of the formal framework presented in Chapter 3. It provides formal preliminaries, proofs, and details of the SCRUM and NRP use cases which have been used in the evaluation.

A.1 Additional formal preliminaries and proofs

A.1.1 Formal preliminaries

Graphs. We first recall the definitions of *graphs* and *typed graphs* and their morphisms.

Definition A.1 (Graph)**.** A *graph* $G = (G_V, G_E, src_G, tgt_G)$ consists of a set G_V of vertices (or nodes), a set G_E of edges, and two maps $src_G, tgt_G \colon G_E \to G_V$ assigning the source and target to each edge, respectively. By $e \colon x \to y$ we denote an edge $e \in G_E$ with $src_G(e) = x$ and $tgt_G(e) = y$.

Definition A.2 (Graph morphism)**.** A *graph morphism* $f \colon G \to H$ consists of a pair of functions $f_V \colon G_V \to H_V$, $f_E \colon G_E \to H_E$ preserving the graph structure: For each edge $e \colon x \to y$ in G_E it holds that $f_E(e) \colon f_V(x) \to f_V(y)$ in H, i.e., we have $f_V \circ src_G = src_H \circ f_E$ and $f_V \circ tgt_G = tgt_H \circ f_E$. Morphism f is *injective* if f_V and f_E are injective.

A typed graph is a graph that is mapped to a given type graph. A mapping between two typed graphs over one and the same type graph has to be type-conformant.

Definition A.3 (Type graph, typed graph and typed morphism)**.** A *type graph* is a distinguished graph $TG = (TG_V, TG_E, src_{TG}, tgt_{TG})$. A *typed graph* $(G, type_G\colon G \to TG)$ which is typed by TG is a graph G together with a graph morphism $type_G$ from G to TG. A typed graph G is also called *instance graph* of graph TG and the morphism $type_G$ is called *typing morphism*.

Given a type graph TG, a *typed graph morphism* $f\colon G \to H$ between typed graphs $(G, type_G)$ and $(H, type_H)$ is a graph morphism $f\colon G \to H$ such that $type_G = type_H \circ f$.

Categories. In this section, we give a short and semi-formal introduction into all those notions of category theory we need for our approach. For more details see, e.g., [Mac71], [AHS90], [Ehr+14], and [Ehr+12].

A *category* $\mathbf{C}$ is a mathematical structure that has objects collected in Ob_C and morphisms $Mor_C(A,B)$ relating pairs of objects $A, B \in Ob_C$ in some way. There needs to be a composition operation $\circ$ for morphisms $f \in Mor_C(A,B)$ and $g \in Mor_C(B,D)$ as well as an identity morphism id_A for each object $A \in Ob_C$. The composition $\circ$ has to be associative and composition with identities has to be neutral.

Examples are the category **Set** of all sets and functions, the category **Poset** of all partially ordered sets and order-preserving mappings, and the category **Graph** of all graphs and graph morphisms.

There are special types of morphisms: An *isomorphism* is a morphism to which an inverse morphism exists, i.e., composing them in either order leads to identities. Objects related by an isomorphism exhibit exactly the same structure and can thus be considered as equal in many contexts. If we have $m \circ f = m \circ g \implies f = g$ for any two morphisms f and g such that the composition is defined, m is called *monomorphism*. In the category **Set**, isomorphisms are the bijective functions and monomorphisms are the injective ones.

Pushouts and pullbacks. A *pushout* can be considered as a kind of union of two objects over a common one. Given two morphisms $g\colon A \to B$ and $h\colon A \to C$, a pushout, if it exists, consists of an object D and two morphisms $k\colon B \to D$ and $l\colon C \to D$ such that (1) $k \circ g = l \circ h$ and (2) the following *universal property*

holds: If there are morphisms $k' \colon B \to X$ and $l' \colon C \to X$ with $k' \circ g = l' \circ h$, then there is a unique morphism $x \colon D \to X$ with $x \circ k = k'$ and $x \circ l = l'$ (see the left diagram in Fig. A.1).

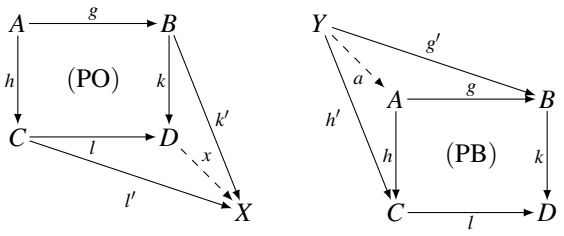


Figure A.1 – A schematic depiction of a pushout (left) and a pullback (right).

Reversing the direction of all morphisms, a *pullback* can be seen as a generalized intersection of two objects over a common object. Given two morphisms $k \colon B \to D$ and $l \colon C \to D$, a pullback consists of an object A and morphisms $g \colon A \to B$ and $h \colon A \to C$ such that $k \circ g = l \circ h$ and the following *universal property* holds: If there are morphisms $g' \colon Y \to B$ and $h' \colon Y \to C$ with $k \circ g' = l \circ h'$, then there is a unique morphism $a \colon Y \to A$ with $g \circ a = g'$ and $h \circ a = h'$ (see the right diagram in Fig. A.1).

In the category **Set**, if a morphism g is injective, the pushout object is $D = C \cup (B - g(A))$. Since a pushout is unique up to isomorphism, any set isomorphic to D would also be a pushout object. A pullback object, for l being injective, is constructed by $A = k(B) \cap l(C)$. In the category **Graph**, pushouts and pullbacks can be constructed componentwise on node and edge sets. For these, and more general computations, compare, e.g., [Ehr+06, Fact 2.17, Fact 2.23, and Remark 2.24].

$\mathscr{M}$-adhesive categories. We will prove our statements in a quite abstract setting, namely the one of *$\mathscr{M}$-adhesive categories* [Ehr+14; Ehr+12]. These are categories where pushouts along monomorphisms interact in a particularly nice way with pullbacks and encompass the categories of sets, of graphs, and many graph-like structures, including typed attributed graphs. A category $\mathbf{C}$ with a morphism class $\mathscr{M}$ is an *$\mathscr{M}$-adhesive category* if the following properties hold:

- $\mathscr{M}$ is a class of monomorphisms closed under isomorphisms (f isomorphism implies that $f \in \mathscr{M}$), composition ($f, g \in \mathscr{M}$ implies $g \circ f \in \mathscr{M}$), and decomposition ($g \circ f, g \in \mathscr{M}$ implies $f \in \mathscr{M}$).
- **C** has pushouts and pullbacks along $\mathscr{M}$-morphisms, i.e., pushouts and pullbacks, where at least one of the given morphisms is in $\mathscr{M}$, and $\mathscr{M}$-morphisms are closed under pushouts and pullbacks, i.e., given a pushout like the left diagram in Fig. A.2, $m \in \mathscr{M}$ implies $n \in \mathscr{M}$ and, given a pullback (1), $n \in \mathscr{M}$ implies $m \in \mathscr{M}$.
- Pushouts in **C** along $\mathscr{M}$-morphisms are so-called *vertical weak van Kampen squares*, i.e., for any commutative cube in **C** where we have the pushout with $m \in \mathscr{M}$ in the bottom, $b, c, d \in \mathscr{M}$, and pullbacks as back faces, the top is a pushout if and only if the front faces are pullbacks.

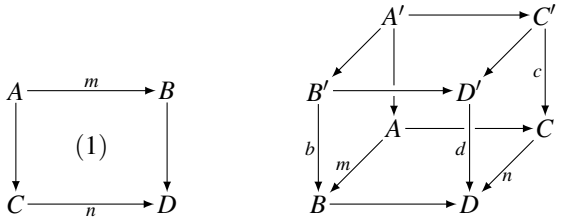


Figure A.2 – A schematic depiction of an $\mathscr{M}$-VK square.

Examples for categories that are $\mathscr{M}$-adhesive are sets with injective functions, graphs with injective graph morphisms and several variants of graphs with special forms of injective graph morphisms. In particular, typed attributed graphs constitute an $\mathscr{M}$-adhesive category (where the class $\mathscr{M}$ consists of injective morphisms where the attribute part is an isomorphism).

The definition of element mutation operators (i.e., rules) can easily be lifted to this more general setting: An element mutation operator then is a span of $\mathscr{M}$-morphisms and a NAC is an $\mathscr{M}$-morphism with domain L. The application of such an element mutation operator is defined via the diagram depicted in Fig. 3.5, requiring that both squares are pushouts. For details and, in particular, a proof that the set-theoretic approach from Definition 3.4 coincides with the here discussed category-theoretic one, we refer to [Ehr+06].

Next, we introduce a logic in $\mathscr{M}$-adhesive categories that allows to reason about objects as well as about morphisms in it. In the category of graphs,

this logic turns out to be expressively equivalent to the ordinary first-order logic on graphs [HP09]. *Nested conditions* express properties of morphisms; the definition of *nested constraints* is based on these and allows to express properties of objects. Constraints and conditions are defined recursively as trees of morphisms. For the definition of constraints, we assume the existence of an initial object $\emptyset$ in the given category, i.e., of an object $\emptyset$ such that for every object A of the given category there is a unique morphism $i_A : \emptyset \to A$. In the category of graphs, this is just the empty graph.

Definition A.4 ((Nested) conditions and constraints)**.** Let $\mathscr{C}$ be an $\mathscr{M}$-adhesive category with initial object $\emptyset$. Given an object P, a *(nested) condition* over P is defined recursively as follows: true is a condition over P. If $a\colon P \to C$ is a morphism and d is a condition over C, $\exists(a\colon P \to C, d)$ is a condition over P again. Moreover, Boolean combinations of conditions over P are conditions over P. A *(nested) constraint* is a condition over the initial object $\emptyset$.

Satisfaction of a nested condition c over P for a morphism $g\colon P \to G$, denoted as $g \models c$, is defined as follows: Every morphism satisfies true. The morphism g satisfies a condition of the form $c = \exists(a\colon P \to C, d)$ if there exists an $\mathscr{M}$-morphism $q\colon C \hookrightarrow G$ such that $g = q \circ a$ and $q \models d$. For Boolean operators, satisfaction is defined as usual. An object G satisfies a constraint c, denoted as $G \models c$, if the initial morphism to G does so.

Remark A.1. When considering typed graphs with inheritance, the definition of satisfaction of a nested condition should be adapted. The category of typed graphs with inheritance is $\mathscr{M}$-adhesive with $\mathscr{M}$ being the class of injective, type-strict morphisms [Löw+15]. When simultaneously matches are not required to be type-strict – as is usually the case in applications – evaluating conditions via type-strict morphisms leads to an undesired semantics. Therefore, the semantics of conditions should be defined using the same class of morphisms that is used as matches, e.g., via injective morphisms (that are allowed to be down-typing). Alternatively, instead of interpreting an application condition as a single application condition, one could interpret it as representing all of its flattened versions and check the validity of every flattened version via a type-strict morphism. The same suggestions to deal with the semantics of conditions (i.e., adapting the class of morphisms that defines their semantics to be the class of morphisms that

is also used for matches or interpreting a condition as a class of conditions) have been made and formalized for attributed structures [Her+14; Ehr+15]. In the following proofs, we will always remark why our result remains true in case one considers typed graphs with inheritance and defines the satisfaction of conditions via injective morphisms (that are allowed to be down-typing).

A.1.2 Proofs

We now present the proofs of the statements in Chapter 3. We do so in the more general setting of $\mathscr{M}$-adhesive categories. This means, we obtain our results in greater generality as stated in the chapter above. In particular, the case of typed attributed graphs is covered by our proofs. For this, we first need to generalize the notion of computation space to arbitrary $\mathscr{M}$-adhesive categories.

Definition A.5 (Generalized computation space (cf. Definition 3.1))**.** Let $\mathscr{C}$ be any $\mathscr{M}$-adhesive category. A *computation meta-model in* $\mathscr{C}$ is a pair $MM = (\subseteq\colon TG_{\mathrm{P}} \hookrightarrow TG, LC)$ where $\subseteq$ is an $\mathscr{M}$-morphism from TG_{P} to TG, and LC is a set of nested constraints typed over TG, called *language constraints*. The set $PC \subseteq LC$, called *problem constraints*, is the subset of constraints that can be considered as already typed over TG_{P}. (TG_{P}, PC) is called *problem meta-model*. A *computation element* or *computation model* $(E, type_E)$ over MM is an object E together with a morphism $type_E\colon E \to TG$ such that $E \models LC$. The *computation space* over MM is

$$CS \coloneqq \{(E, type_E) \mid (E, type_E) \text{ is a computation model over } MM\}.$$

Given a computation model $(E, type_E)$ over MM, the model $(E_{\mathrm{P}}, type_{E_{\mathrm{P}}})$ where $type_{E_{\mathrm{P}}}\colon E_{\mathrm{P}} \to TG_{\mathrm{P}}$ and $\subseteq_E\colon E_{\mathrm{P}} \hookrightarrow E$ are obtained by pulling back $type_E$ along $\subseteq$ is the *problem model* and $E \setminus E_{\mathrm{P}}$ (if defined) is the *solution part* of $(E, type_E)$ (where *initial pushouts* [Ehr+06] can be used to lift the definition of the set-theoretic difference operator $\setminus$ to the categorical level).

A *computation-model morphism*, short *cm-morphism*, m between computation models $(E, type_E)$ and $(F, type_F)$ is a morphism $m\colon E \to F$ such that m is compatible with typing, i.e., $type_F \circ m = type_E$. A cm-morphism m is *problem-invariant*

if m_P, the restriction of m to the problem model of E, is an isomorphism between E_P and F_P.

By the above description of pullbacks, since $\subseteq\colon TG_\mathrm{P} \hookrightarrow TG$ is an inclusion, the definition of E_P as $E \cap type_E^{-1}(TG_\mathrm{P})$ and of $type_{E_\mathrm{P}}$ via restriction ensures that the typing morphism of a computation model (considered as a pair of morphisms) constitutes a pullback square. In particular, the above definition indeed generalizes the set-theoretic Definition 3.1. One important observation for the following proof of Proposition 3.1 is that, therefore, cm-morphisms between computation models constitute pullback squares, as well (compare Fig. 3.3): Since both typing morphisms are pullbacks, pullback decomposition [Ehr+06, Fact 2.27] implies that a cm-morphism is a pullback square. This means, all squares depicted in Fig. 3.3 are pullbacks. Another important observation is that the morphism $\subseteq_E\colon E_\mathrm{P} \hookrightarrow E$ is always an $\mathscr{M}$-morphism as it arises by pullback along one.

We first prove that validity of the problem constraints only depends on the problem part of a computation model.

Lemma A.1. *In any $\mathscr{M}$-adhesive category $\mathscr{C}$, given a computation meta-model $MM = (\subseteq\colon TG_\mathrm{P} \hookrightarrow TG, LC)$ in $\mathscr{C}$ with a set of problem constraints $PC \subseteq LC$, a typed object $(E, type_E)$ satisfies the problem constraints from PC if and only if $(E_P, type_{E_\mathrm{P}})$ satisfies them.*

Proof. We show more generally that the corresponding statement holds for *conditions*, not only for constraints. For this, let c be a nested condition over a computation model $(X, type_X)$ where c is typed over TG_P. In particular, $type_X$ can be considered to already have codomain TG_P. First, it is easy to check that this implies that $\subseteq_X\colon X_\mathrm{P} \hookrightarrow X$ is an isomorphism. Without loss of generality, we assume it to be the identity of X in the following; in particular $X_\mathrm{P} = X$. The same holds for any other model occurring in the condition c.

This implies that, for every model $(E, type_E)$, there is a one-to-one correspondence between cm-morphisms $g\colon X \to E$ and typed morphisms $g_\mathrm{P}\colon X_\mathrm{P} \to E_\mathrm{P}$ between the problem parts: Given g, g_P is obtained by pulling back g along $\subseteq_E\colon E_\mathrm{P} \to E$. Given a morphism $g_\mathrm{P}\colon X_\mathrm{P} \to E_\mathrm{P}$, $g := \subseteq_E \circ g_\mathrm{P}\colon X_\mathrm{P} \to E$ defines

a cm-morphism from X to E since $X_\mathrm{P} = X$ (checking the induced square to constitute a pullback square is routine). We show, via structural induction, for any cm-morphism $g\colon X \to E$ that $g \models c$ if and only if $g_\mathrm{P} \models c$ where $g_\mathrm{P}\colon X_\mathrm{P} \to E_\mathrm{P}$ is the restriction of g to the problem model.

The *induction basis* is trivial as every morphism satisfies true. In particular, $g \models \mathsf{true}$ if and only if $g_\mathrm{P} \models \mathsf{true}$.

Assume that the statement holds for conditions d_1, d_2, d. For the *induction step*, first let $c := d_1 \wedge d_2$. Then

$$\begin{aligned} g \models c &\iff g \models d_1 \text{ and } g \models d_2 \\ &\iff g_\mathrm{P} \models d_1 \text{ and } g_\mathrm{P} \models d_2 \\ &\iff g_\mathrm{P} \models c \ . \end{aligned}$$

Similarly, for $c := \neg d$

$$\begin{aligned} g \models c &\iff g \not\models d \\ &\iff g_\mathrm{P} \not\models d \\ &\iff g_\mathrm{P} \models c \ . \end{aligned}$$

Finally, for $c := \exists(a\colon X \to Y, d)$

$$\begin{aligned} g \models c \iff & \text{there exists an } \mathscr{M}\text{-morphism } q\colon Y \to E \\ & \text{such that } q \circ a = g \text{ and } q \models d \\ \iff & \text{there exists an } \mathscr{M}\text{-morphism } q\colon Y \to E \\ & \text{such that } \subseteq_E \circ q_\mathrm{P} \circ a = \subseteq_E \circ g_\mathrm{P} \text{ and } q \models d \\ \iff & \text{there exists an } \mathscr{M}\text{-morphism } q_\mathrm{P}\colon Y_\mathrm{P} \to E \\ & \text{such that } q_\mathrm{P} \circ a = g_\mathrm{P} \text{ and } q_\mathrm{P} \models d \\ \iff & g_\mathrm{P} \models c \end{aligned}$$

where the second equivalence holds by the above explained correspondence of morphisms and the third by monotonicity of $\subseteq_E$ and the induction hypothesis. □

Proof of Lemma 3.1. Just instantiate Lemma A.1 to typed graphs noting that graph inclusions are $\mathscr{M}$-morphisms. □

Remark A.2. The central ingredient for the proof of Lemma A.1 is the one-to-one correspondence between cm-morphisms and morphisms starting at the problem part (as used in the last induction step). While in the context of a type graph with inheritance this cannot any longer be argued for in the same way, the statement is still true as long as no solution element inherits from a problem element. Therefore, the lemma also holds for typed graphs with inheritance under the assumption of separate inheritance hierarchies for problem and solution elements.

Definition A.6 (Generalized element mutation operator. Generalized element mutation (cf. Definition 3.4))**.** Given a computation space *CS* over a meta-model $MM = (\subseteq\colon TG_{\mathrm{P}} \hookrightarrow TG, LC)$, a *generalized element mutation operator mo* is defined by $mo = (L \overset{le}{\hookleftarrow} I \overset{ri}{\hookrightarrow} R, ac)$, where L, I, and R are objects typed over TG, *le* and *ri* are $\mathscr{M}$-morphisms, and *ac* is a (nested) condition over L.

A *generalized element mutation* $E \Longrightarrow_{mo} F$ using *mo* at match *m* is defined as in the diagram of Fig. 3.5 such that both squares are pushouts and *m* satisfies *ac*.

A sequence $E = E_0 \Longrightarrow_{mo_1} E_1 \Longrightarrow_{mo_2} \ldots E_n = F$ of generalized element mutations (where mutation operators mo_i and mo_j are allowed to coincide for $1 \leq i \neq j \leq n$) is denoted by $E \Longrightarrow^*_M F$, where M is a set containing all generalized mutation operators that occur. For $n = 0$, we have $E = F$.

Instead of proving Proposition 3.1 as stated in the paper, we present a more precise statement in the general setting of $\mathscr{M}$-adhesive categories below.

Proposition A.1. Let $\mathscr{C}$ be an $\mathscr{M}$-adhesive category and *MM* a computation meta-model in $\mathscr{C}$. Let $mo = (L \overset{le}{\hookleftarrow} I \overset{ri}{\hookrightarrow} R, ac)$ be a generalized element mutation operator, and let $E, F \in CS$ be computation models such that there is a generalized element mutation $E \Longrightarrow_{mo} F$ (compare Fig. A.3). Then the morphisms le' and ri' in Fig. A.3 are cm-morphisms, and the operator *mo* is problem-invariant if the morphisms *le* and *ri* defining the operator *mo* are problem-invariant.

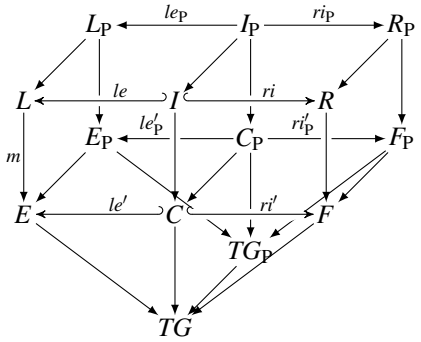


Figure A.3 – Mutation of solution model E to solution model F explicitly showing the problem model.

Proof. We have to show that (i) le' and ri' are cm-morphisms and (ii) the stated equivalence. To prove both (i) and (ii) compare Fig. A.3: The front of the diagram is basically the same as in Fig. 3.5; we just omit the application condition as we already assume the rule to be applicable at match m. The typing morphisms to TG are added; the ones for L,I,R are given by composition. The typing morphism $type_C$ is obtained by composition of le' with $type_E$; $type_F\colon F \to TG$ is obtained by the universal property of F as a pushout object (using $type_C$ and $type_R$ as comparison morphisms). The back of Fig. A.3 is then induced by pulling back the front diagram along $\subseteq\colon TG_P \hookrightarrow TG$. In particular, this makes both the squares (C_P, E_P, C, E) and (C_P, C, F_P, F) to pullback squares. This shows le' and ri' to be typed morphisms; i.e., (i) holds.

Concerning (ii), if we show both squares in the back to be pushouts as well as pullbacks, this implies (ii): Both pushouts and pullbacks of isomorphisms result in isomorphisms again. For this, observe that both front squares are pushouts by assumption. Moreover, since cm-morphisms constitute pullback squares, the further squares (except for the two in the back) are pullbacks. This implies, by the weak vertical van Kampen property of the front squares, that the squares in the back are pushouts as well. Furthermore, because the top squares are pullbacks, $le, ri \in \mathscr{M}$ implies that also $le_P, ri_P \in \mathscr{M}$. Therefore, as pushouts along $\mathscr{M}$-morphisms in an $\mathscr{M}$-adhesive category, the squares in the back are also pullbacks. Summarizing, le_P and ri_P are isomorphisms if and only if le'_P and ri'_P are isomorphisms. □

Proof of Proposition 3.1. Again, proving Proposition 3.1 just amounts to instantiating Proposition A.1 to the category of typed graphs noting that the inclusion of graphs is an $\mathscr{M}$-morphism. Also, every negative application condition is a nested condition. □

Remark A.3. The short category-theoretical proof we gave for Proposition A.1 does not carry over to the setting of a type graph with inheritance. The reason is that cm-morphisms between typed graphs with inheritance do not constitute pullback squares anymore (compare the discussion before Lemma A.1). But the claim of Proposition A.1 can still be proved elementary in that setting by excluding every way in which it would be possible for a rule (defined via problem-invariant morphisms) to alter the problem model. To exclude the deletion of problem elements, however, it is necessary to assume that problem elements cannot inherit from solution elements. Otherwise, a rule could specify the deletion of a solution element but be applied to a problem element (via down-typing).

A.2 Evaluation: Details of optimization problems SCRUM and NRP

In the following, we present the meta-models and element mutation operators of the optimization problems SCRUM and NRP. We also present their language constraints an discuss why these constraints are preserved by their element mutation operators. An argumentation for the preservation of language constraints shared by all use cases can be found in Sec. 3.7.3. Furthermore, we discuss the soundness and completeness properties of the different sets of element mutation operators used in the evaluation. Explanations of the operators can be found in Sec. 3.7.3.

A.2.1 SCRUM

Figure A.4 shows the meta-model of the SCRUM case. In the SCRUM case, the type Sprint (along with its incoming and outgoing edges) and the attribute

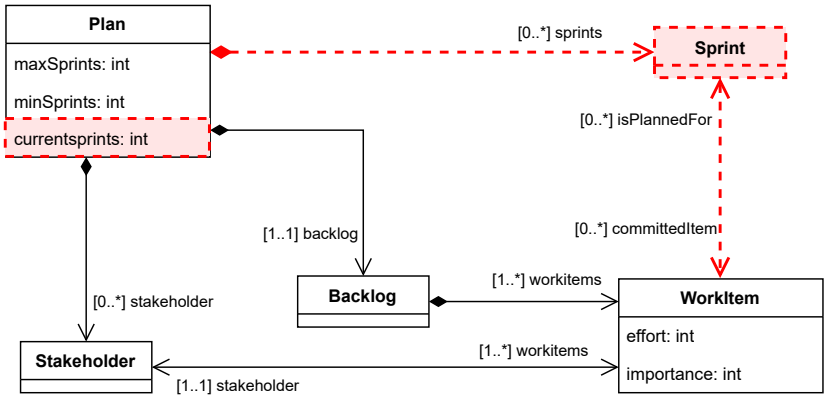
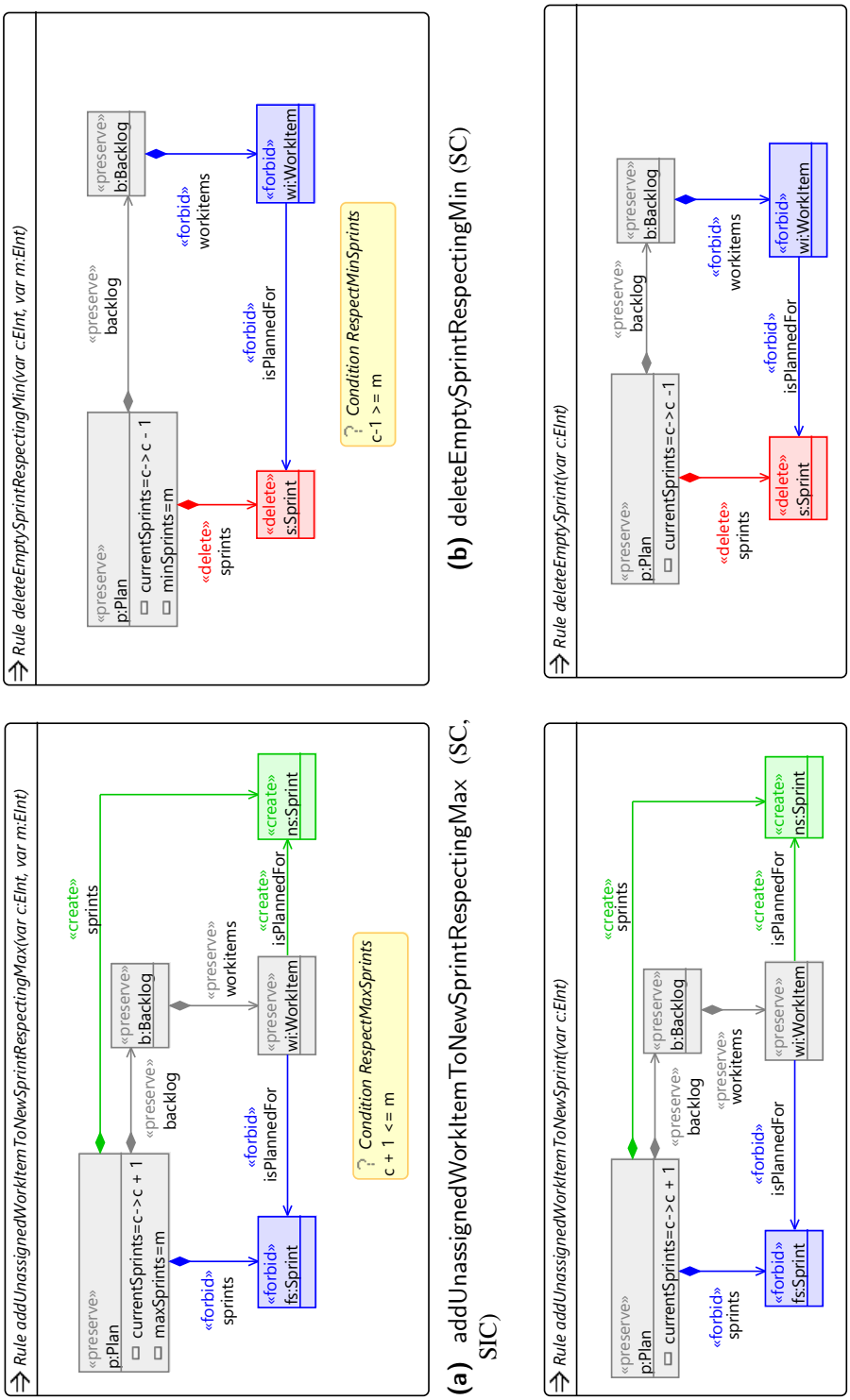


Figure A.4 – Meta-model of the SCRUM use case. White solid elements are invariant problem parts, the colored, dashed elements and references are solution-related.

currentSprints are part of the solution. All other elements are part of the problem meta-model. The element mutation operators of the use case are depicted in Figure A.5.

Preservation of language constraints in the SCRUM case. In addition to the EMF-specific language constraints common to all use cases (with Plan being the root node), we consider the following language constraints specific to the SCRUM case: The attributes minSprints and maxSprints require several problem constraints. Their values must be between zero to the number of work items, and minSprints must be less than (or equal to) maxSprints. Since the edges of the solution part are unbound, only problem constraints arise from the multiplicities of the meta-model (we will not enumerate all of them here). To save computation time, the current number of sprints is recorded in the attribute currentSprints. A language constraint requires that this attribute always reflects the correct number of sprints currently available.

All element mutation operators, including those belonging to the set UC, preserve the validity of these language constraints. With respect to the correct value of currentSprints, we see that every operator that creates or deletes a Sprint increments or decrements currentSprints accordingly; the other operators do not change its value. Since all other constraints are problem constraints, Proposition 3.1 and Lemma 3.1 ensure their preservation.



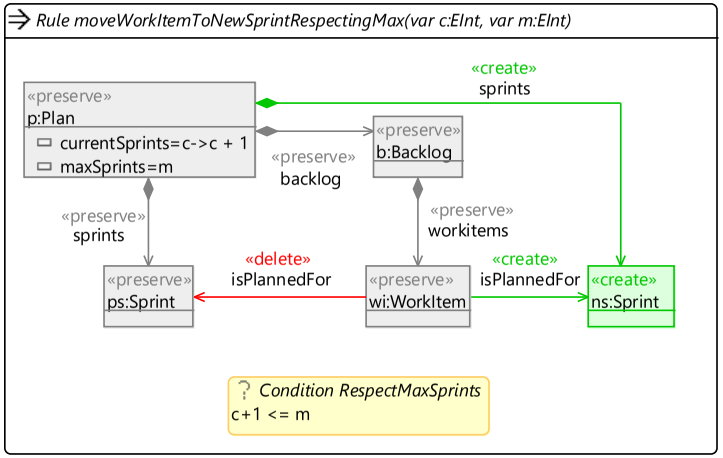
(a) addUnassignedWorkItemToNewSprintRespectingMax (SC, SIC)

(b) deleteEmptySprintRespectingMin (SC)

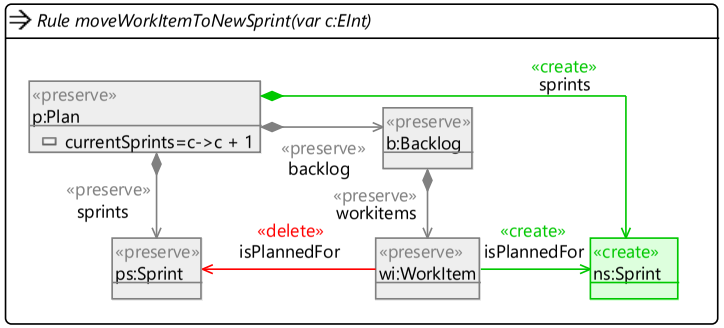
(c) addUnassignedWorkItemToNewSprint (UC)

(d) deleteEmptySprint (UC)

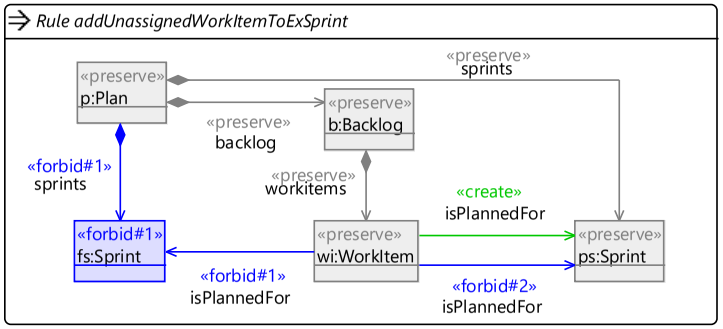
Figure A.5 – The set of element mutation operators used for the SCRUM case. The algorithm variants to which an operator belongs are given in parenthesis. (part 1 of 2)



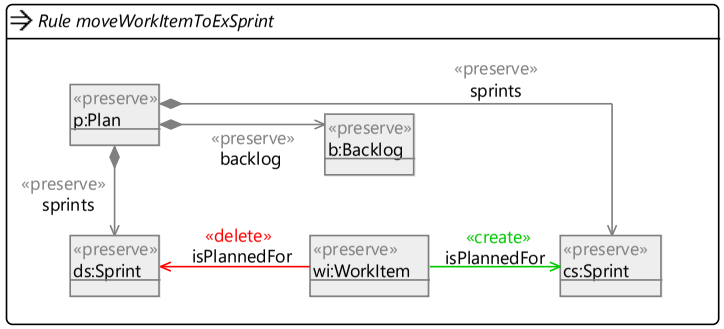
(e) moveWorkItemToNewSprintRespectingMax (SC, SIC)



(f) moveWorkItemToNewSprint (UC)



(g) addUnassignedWorkItemToExSprint (SC, UC, SIC)



(h) moveWorkItemToExSprint (SC, UC, SIC)

Figure A.5 – The set of element mutation operators used for the SCRUM case. The algorithm variants to which an operator belongs are given in parenthesis. (part 2 of 2)

Soundness for SCRUM case. In the SCRUM case, there are four feasibility constraints. First, two structural constraints require that each WorkItem is assigned to exactly one Sprint (this is a lower and an upper bound). Furthermore, it is required that the total number of sprints is between a minimum and a maximum value; this is expressed as an attribute constraint via a counter. The structural constraints of the SCRUM case have exactly the same structure as the feasibility constraints of the CRA case, where it is required that each feature is assigned to exactly one class. Moreover, each rule of the SCRUM case that belongs to the set SC and/or SIC structurally corresponds to such a rule in the CRA case (namely, addUnassignedWorkItemToNewSprintRespecting-Max to addUnassignedFeatureToNewClass, deleteEmptySprintRespectingMin to deleteEmptyClass, moveWorkItemToNewSprintRespectingMax to moveFeature-ToNewClass, addUnassignedWorkItemToExSprint to addUnassignedFeatureTo-ExClass, and moveWorkItemToExSprint to moveFeatureToExClass). Since the rules for the CRA case are sound, the rules for the SCRUM case are also sound with regard to the structural constraints.

With respect to the attribute constraints, it is sufficient to note that each rule in the sets SC and SIC that creates or deletes a sprint has an application condition that ensures that the number of sprints does not exceed the maximum and does not fall below the minimum. Therefore, these rules also cannot transform a model with a correct number of sprints to one with a number that violates the minimum or maximum number of sprints allowed. Overall, the rules for the SCRUM case that belong to the sets SC and/or SIC are sound with respect to all constraints.

In the case of UC, it is obvious that there are rules that can create computation models with too few or too many sprints, even if they start from feasible solutions. Hence, the set is unsound.

Completeness for SCRUM case. In the SCRUM use case, completeness can be argued for almost in the same way as in the CRA case. We just need to additionally consider the given minimum and maximum number of sprints. First, for any instance, we can use the rules addUnassignedWorkItemToNew-SprintRespectingMax (or addUnassignedWorkItemToNewSprint in case of UC), moveWorkItemToNewSprintRespectingMax (or moveWorkItemToNewSprint

in case of UC), moveWorkItemToExSprint, and deleteEmptySprintRespecting-Min (or deleteEmptySprint in case of UC) to transform any computation model (feasible or not) into the model that contains the minimal number of sprints and in which all WorkItems are assigned to the same Sprint. Using the rules moveWorkItemToNewSprintRespectingMax (or moveWorkItemToNewSprint in case of UC) and moveWorkItemToExSprint, this model can then be transformed into any feasible model. Therefore, both sets SC and UC are complete.

In case of SIC, there is no rule to delete a sprint. Therefore, no instance can be transformed into an instance with fewer sprints, making the rule set incomplete.

A.2.2 NRP

Figure A.6 shows the meta-model of the NRP case. In the NRP case, the solution part comprises only the edge between Solution and SoftwareArtifact as well as the attribute totalCosts. The element mutation operators of the use case are depicted in Figure A.7.

Preservation of language constraints in the NRP case. In addition to the EMF-specific language constraints common to all use cases (with NRP being the root node), we consider the following language constraints specific to the NRP case: Neither the dependency hierarchies of requirements nor of software artifacts may contain cycles. Furthermore, the attributes value, amount, percentage, and importance must be greater than zero. All of these requirements are problem constraints. Again, only problem constraints arise from the multiplicities of the meta-model. To save computations, similar to the SCRUM case, the attribute totalCosts is used to capture the sum of the costs of all selected software artifacts. A language constraint must guarantee the correctness of its value.

All element mutation operators, including those belonging to the set UC, preserve the validity of these language constraints. With respect to the correct value of totalCosts, all operators (de-)select a SoftwareArtifact, but also recompute totalCosts accordingly. Finally, Proposition 3.1 and Lemma 3.1 again guarantee that all problem constraints are preserved.

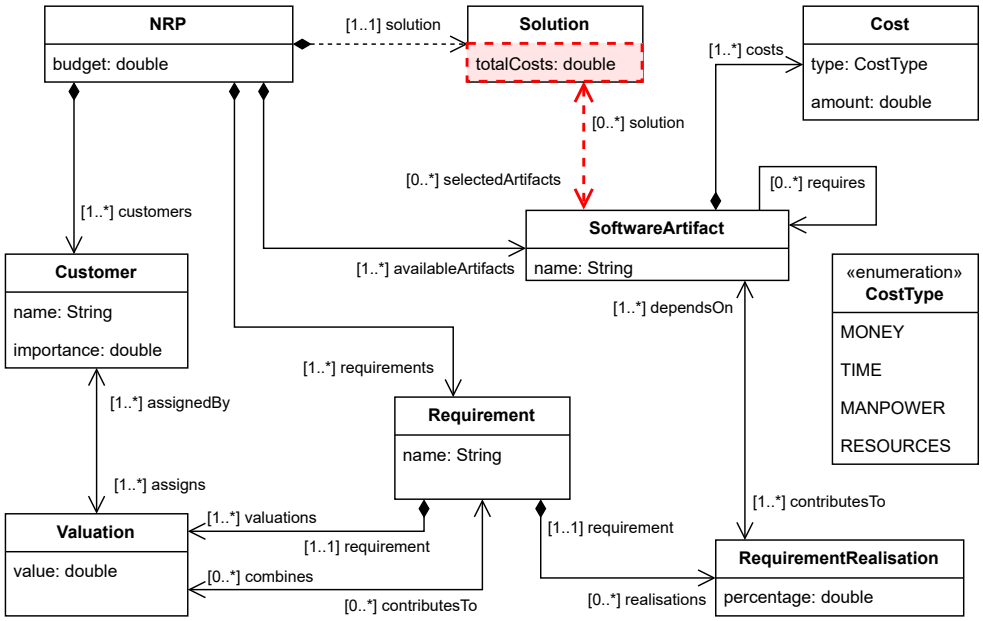
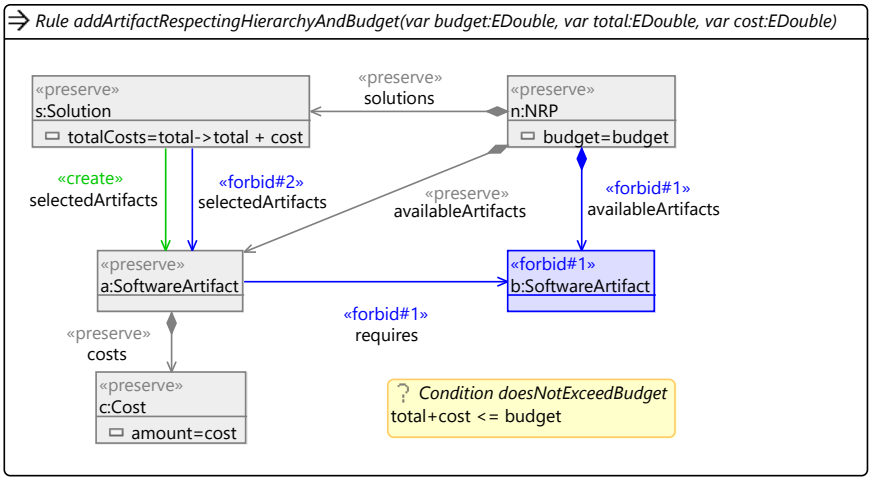


Figure A.6 – Meta-model of the NRP use case. White solid elements are invariant problem parts, the colored, dashed elements and references are solution-related.

Soundness for NRP case. In the NRP case, we consider two feasibility constraints. A structural constraint states that a solution that contains a SoftwareArtifact also contains all SoftwareArtifacts that this SoftwareArtifact (transitively) requires. An additional attribute constraint expresses that the total cost of a feasible solution does not exceed the budget. Since the depth of a requires-hierarchy can be arbitrary, the structural constraint is not first-order in this case. Therefore, we cannot use the tool OCL2AC and instead perform a manual analysis.

Let us assume that a feasible solution for the NRP case is given (for any problem instance). This means that for each selected SoftwareArtifact, all (transitively) required SoftwareArtifacts are also selected. The rules from SC and SIC that deselect SoftwareArtifacts (removeArtifactRespectingHierarchy resp. removeArtifactRespectingHierarchyAndDependents) both have a NAC that ensures that the rule is only applicable if no other selected SoftwareArtifacts require the one to be deleted. Therefore, only "leafs" of the requires-hierarchy can be deleted, resulting in offspring that again satisfies the constraint. For the rule that selects new SoftwareArtifacts (addArtifactRespectingHierarchyAnd-



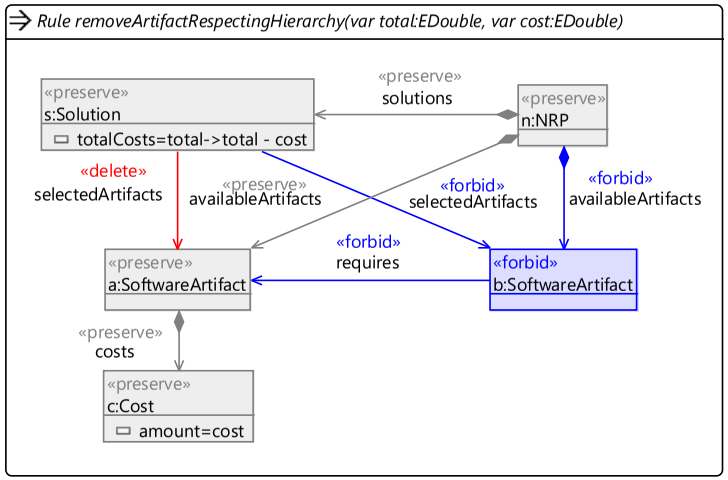
(a) addArtifactRespectingHierarchyAndBudget (SC, SIC). The rule contains a nested NAC which is not included in the visual syntax of Henshin. The nested application condition further constrains software artifact b to artifacts that are not already selected. Obviously, artifacts that are already selected for the next release should not prohibit the selection of dependent artifacts.

Figure A.7 – The set of element mutation operators used for the NRP case. The algorithm variants to which an operator belongs are given in parenthesis. (part 1/2)

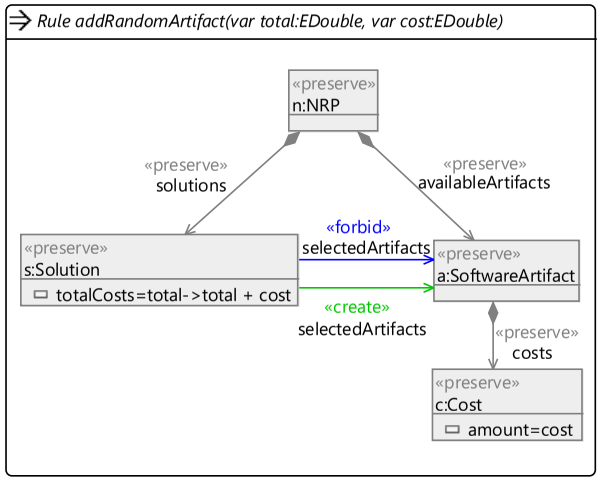
Budget), the first application condition (annotated with forbid#1) prohibits the SoftwareArtifact to be selected to require another SoftwareArtifact that is not already selected. (Note that the second part of this application condition has no counterpart in the visual representation of the rule in Fig. A.7a. However, it is present in the programmed rule.) This means that only SoftwareArtifacts that do not require other SoftwareArtifacts or for which all directly required SoftwareArtifacts are already selected can be selected. However, for feasible solutions, all SoftwareArtifacts required by them are also already selected (by feasibility), which means that the overall result of applying addArtifactRespectingHierarchyAndBudget to a feasible solution yields a solution where the structural constraints are satisfied.

With regard to the attribute constraint, addArtifactRespectingHierarchyAndBudget prohibits the selection of a SoftwareArtifact that would result in a budget overrun. In summary, both sets SC and SIC are sound.

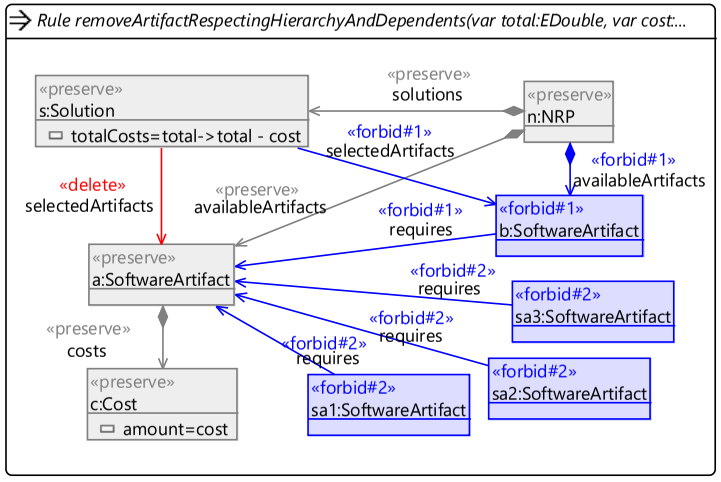
In addition, the set UC is obviously unsound. Both addRandomArtifact and



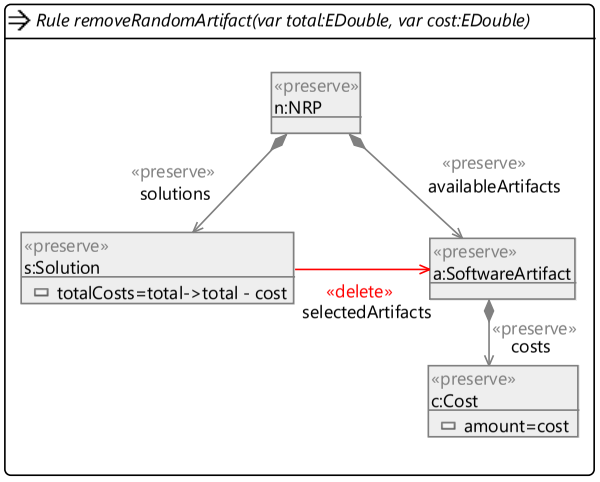
(b) removeArtifactRespectingHierarchy (SC)



(c) addRandomArtifact (UC)



(d) removeArtifactRespectingHierarchyAndDependents (SIC)



(e) removeRandomArtifact (UC)

Figure A.7 – The set of element mutation operators used for the NRP case. The algorithm variants to which an operator belongs are given in parenthesis. (part 2/2)

removeRandomArtifact can destroy an intact requires-hierarchy of the selected artifacts. Moreover, addRandomArtifact may lead to budget violations.

Completeness for NRP case. Both sets SC and UC are complete. First, with rule removeArtifactRespectingHierarchy resp. removeRandomArtifact, each computation model can be transformed to the instance where no SoftwareArtifact is selected at all. In the first case, this must be done from top to bottom along the requires-hierarchy; in the second case, the order can be arbitrary. Subsequently, the rule addArtifactRespectingHierarchyAndBudget resp. addRandomArtifact can be used to create any feasible instance by selecting the appropriate SoftwareArtifacts. Again, in the first case this must be done in a definite order, here from bottom to top, while in the second case, the order can be arbitrary.

With respect to the set SIC, it is not possible to deselect a SoftwareArtifact from an instance if it is required by at least three other SoftwareArtifacts (cf. the NAC annotated with forbid#2 in rule removeArtifactRespectingHierarchyAndDependents). In general, therefore, an instance in which such a SoftwareArtifact is selected cannot be transformed to every feasible solution using this rule set.

B

Appendix to Section 4.1

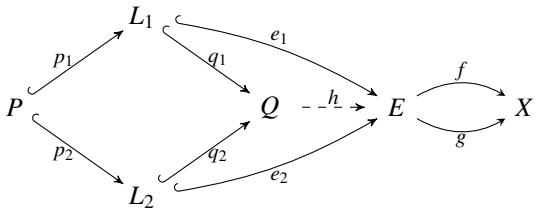
This appendix belongs to Section 4.1 which introduces the generic crossover approach and contains proofs for the presented propositions and lemmas.

B.1 Proofs

The following lemma is the central ingredient for the proof of Proposition 4.1 and also used in the one of Proposition 4.3. For adhesive categories, it has already been stated in the extended version of [Fri+18]. Here, we present it in the more general context of $\mathscr{M}$-adhesive categories. Because of that, we need to additionally assume the existence of $\mathscr{M}$-effective unions.

Lemma B.1 (Pullbacks as pushouts)**.** *In an $\mathscr{M}$-adhesive category $(\mathscr{C},\mathscr{M})$ with $\mathscr{M}$-effective unions, let $(e_1,e_2) : L_1, L_2 \hookrightarrow E$ be a pair of jointly epimorphic $\mathscr{M}$-morphisms. Then the pullback of (e_1,e_2) is also a pushout.*

Proof. Given the diagram below, where P arises as pullback of (e_1,e_2), Q as pushout of (p_1,p_2), and the morphism h from the universal property of Q, we show that h is an isomorphism.



First, since e_1, e_2 are $\mathscr{M}$-morphisms, the morphism h is an $\mathscr{M}$-morphism, assuming $\mathscr{M}$-effective unions. This means that h is a regular monomorphism (compare [LS05, Lemma 4.8], which is easily seen to also hold in $\mathscr{M}$-adhesive categories).

Secondly, given two morphisms $f, g : E \to X$ with $f \circ h = g \circ h$, it follows that $f \circ h \circ q_1 = g \circ h \circ q_1$ which implies $f \circ e_1 = g \circ e_1$; analogously, $f \circ e_2 = g \circ e_2$ holds. Since e_1, e_2 are jointly epimorphic, it follows that $f = g$, and h is an epimorphism. Thus, h is epi and regular mono and therefore an isomorphism. □

Prrof of Proposition 4.1. Given a solution split as depicted in Fig. 4.5, it is straightforward to realize this split via the split construction. One just chooses the already given morphisms s^1 and s^2. As the bottom square in Fig. 4.5 is a pushout, s^1 and s^2 are jointly epimorphic. Moreover, in an $\mathscr{M}$-adhesive category that square is also a pullback because $E^I \hookrightarrow E^1$ (or, equally, $E^I \hookrightarrow E^2$) $\in \mathscr{M}$.

To show that the construction always computes a solution split, we have to show that it produces a commuting cube of $\mathscr{M}$-morphisms (with isomorphisms at the top) such that the bottom square is a pushout and the four vertical squares constitute ce-morphisms (i.e., are also pullbacks and are compatible with typing). It is well-known that, in every category, in a cube that is computed via pullbacks as stipulated by our construction, all squares are pullbacks; see, e.g., [Awo10, 5.7 Exercises, 2. (b)]. By closedness of $\mathscr{M}$-morphism under pullbacks, this in turn implies that all morphisms are $\mathscr{M}$-morphisms (because e, s^1, and s^2 are). The two morphisms at the front of the top square are isomorphisms by assumption; the other two become isomorphisms by closedness of isomorphisms under pullback. Finally, in an $\mathscr{M}$-adhesive category with $\mathscr{M}$-effective unions, the pullback of jointly epimorphic $\mathscr{M}$-morphisms is always a pushout (see Lemma B.1 above). Therefore, the bottom square (computed as pullback of the jointly epic $\mathscr{M}$-morphisms s^1 and s^2) is a pushout as desired. The typing of $\overline{E^1}$ and $\overline{E^2}$ is compatible with the typing of $\overline{E}$ by definition; moreover, the squares obtained from the typing morphisms are pullbacks by pullback composition.

For the last statement, it suffices to observe that E^2 can always be chosen as E, embedded via the identity morphism (which then leads to $E^I \cong E^1$). □

Proof of Lemma 4.1. To prove the statement, we have to show that there exists a ce-morphism (a_P, a) from $\overline{CP} := (id : PI_{\mathrm{P}} \hookrightarrow PI_{\mathrm{P}}, t_{PI_{\mathrm{P}}}, tp \circ t_{PI_{\mathrm{P}}})$ to $\overline{E^I}$ such that a_P is an isomorphism and $a \in \mathscr{M}$; the analogous statement for $\overline{F^I}$ is proved in exactly the same way.

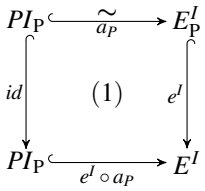


Figure B.1 – Showing $\overline{CP}$ to constitute a crossover point

We define such a ce-morphism using the isomorphism a_P with $t_{E^I_{\mathrm{P}}} \circ a_P = t_{PI_{\mathrm{P}}}$ that exists since $\overline{E^I}$ is an element of the search space of $\overline{PI}$. Figure B.1 depicts this. The square commutes and $a, e^I \circ a \in \mathscr{M}$ by closedness of $\mathscr{M}$ under isomorphisms and composition. Moreover, using the fact that e^I is a monomorphism, it is also easy to check that the square constitutes a pullback. Finally, using $t_{E^I_{\mathrm{P}}} \circ a_P = t_{PI_{\mathrm{P}}}$ we compute

$$\begin{aligned} t_{E^I} \circ e^I \circ a_P &= tp \circ t_{E^I_{\mathrm{P}}} \circ a_P \\ &= tp \circ t_{PI_{\mathrm{P}}} \end{aligned}$$

which shows $(a_P, e^I \circ a_P)$ to be type-compatible. □

Proof of Proposition 4.2. First, in an $\mathscr{M}$-adhesive category, pushouts along $\mathscr{M}$-morphisms exist. This means that, given two solution splits and a crossover point, crossover is always applicable. Since isomorphisms are closed under pushout, the top squares in the construction consist of isomorphisms only. In particular, $(E^1F^2)_{\mathrm{P}} \cong PI_{\mathrm{P}} \cong (E^2F^1)_{\mathrm{P}}$ (because $E^1_{\mathrm{P}} \cong PI_{\mathrm{P}} \cong E^2_{\mathrm{P}}$ by assumption).

By definition, o_1 is the unique morphism such that

$$o_1 \circ a_P = a \circ e^1 \text{ and } o_1 \circ b_P = b \circ f^2 \ ,$$

where (a_P, a) and (b_P, b) denote the ce-morphisms from e^1 and f^2 to o_1 (see Fig. 4.8). A standard diagram chase (using the facts that the top squares in

Fig. 4.8 consist of isomorphisms only and that diagrams remain commutative if one replaces isomorphisms by their inverses) then shows that $a \circ e^1 \circ a_P^{-1}$ (or, equally, $b \circ f^2 \circ b_P^{-1}$) exhibits this universal property. Therefore, $o_1 = a \circ e^1 \circ a_P^{-1} \in \mathscr{M}$ as composition of $\mathscr{M}$-morphisms. Again, this uses the fact that $\mathscr{M}$ contains all isomorphisms.

Finally, that the typing morphisms of $\overline{O_1}$ induce even a pullback square over tp (and not merely a commuting one) follows exactly as in the proof of Lemma 2.2 in [GL12], using the facts that the ambient category $\mathscr{C}$ is $\mathscr{M}$-adhesive and $tp \in \mathscr{M}$. □

Proof of Proposition 4.3. Let solution $\overline{O}$ be computed via a crossover from $\overline{E}$ and $\overline{F}$. It is immediately clear from the construction that there exist the two required ce-morphisms $\bar{i}$ and $\bar{j}$ such that i, j are jointly epic $\mathscr{M}$-morphisms because the projections of a pushout are jointly epi and $\mathscr{M}$-morphisms are closed under pushout.

For the converse direction, O is jointly covered by E^1 and F^2, which stem from subsolutions $\overline{E^1}$ and $\overline{F^1}$ of $\overline{E}$ and $\overline{F}$ by assumption. If the underlying category has $\mathscr{M}$-effective unions, pulling these morphisms back results in a pushout. Let $\overline{CP}$ be the object resulting from that pullback (exactly as in the proof of Proposition 4.1). We merely have to show that there exist solution splits of $\overline{E}$ and $\overline{F}$ that split up $\overline{E}$ into $\overline{E^1}$ and some suitable subsolution $\overline{E^2}$ of $\overline{E}$ and $\overline{F}$ into $\overline{F^2}$ and some suitable subsolution $\overline{F^1}$ of $\overline{F}$ for which $\overline{CP}$ can serve as a crossover point. As in (the proof of) the second part of Proposition 4.1, we can use $\overline{E}$ as $\overline{E^2}$ and, because of the symmetric nature of a solution split, $\overline{F}$ as $\overline{F^1}$ and obtain splits of $\overline{E}$ and $\overline{F}$ with $\overline{E^I} = \overline{E^1}$ and $\overline{F^I} = \overline{F^2}$. Hence, $\overline{CP}$, together with the morphisms that stem from its computation as a pullback, can serve as a crossover point for these splits, and applying the crossover construction computes the given solution $\overline{O}$. □

Bibliography

[Abd+14] H. Abdeen et al. “Multi-Objective Optimization in Rule-Based Design Space Exploration”. In: *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*. ASE ’14. ACM, 2014, pp. 289–300. DOI: 10.1145/2642937.2643005.

[ADS23] L. van Arragon, C. D. N. Damasceno, and D. Stüber. “Model-Driven Optimization: Towards Performance-Enhancing Low-Level Encodings”. In: *(under review)*. 2023.

[AHS90] J. Adámek, H. Herrlich, and G. E. Strecker. *Abstract and Concrete Categories. The Joy of Cats*. Wiley-Interscience, 1990.

[APS18] T. Atkinson, D. Plump, and S. Stepney. “Evolving Graphs by Graph Programming”. In: *Genetic Programming – 21st European Conference, EuroGP 2018, Parma, Italy, April 4–6, 2018, Proceedings*. Vol. 10781. Lecture Notes in Computer Science. Springer, 2018, pp. 35–51. DOI: 10.1007/978-3-319-77553-1_3.

[APS20] T. Atkinson, D. Plump, and S. Stepney. “Horizontal Gene Transfer for Recombining Graphs”. In: *Genetic Programming and Evolvable Machines* 21.3 (2020), pp. 321–347. DOI: 10.1007/s10710-020-09378-1.

[APS21] T. Atkinson, D. Plump, and S. Stepney. “Evolving Graphs with Semantic Neutral Drift”. In: *Natural Computing* 20.1 (2021), pp. 127–143. DOI: 10.1007/s11047-019-09772-4.

[Are+10] T. Arendt et al. “Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations”. In: *Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I*. Vol. 6394. Lecture Notes in Computer Science. Springer, 2010, pp. 121–135. DOI: 10.1007/978-3-642-16145-2_9.

[Awo10] S. Awodey. *Category Theory*. 2nd. Vol. 52. Oxford Logic Guides. Oxford University Press, 2010.

[BBL10] M. Bowman, L. C. Briand, and Y. Labiche. "Solving the Class Responsibility Assignment Problem in Object-Oriented Analysis with Multi-Objective Genetic Algorithms". In: *IEEE Transactions on Software Engineering* 36.6 (2010), pp. 817–837. DOI: 10.1109/TSE.2010.70.

[BDW98] L. C. Briand, J. W. Daly, and J. Wüst. "A Unified Framework for Cohesion Measurement in Object-Oriented Systems". In: *Empirical Software Engineering* 3.1 (1998), pp. 65–117. DOI: 10.1023/A:1009783721306.

[Bec+11] B. Becker et al. "Iterative Development of Consistency-Preserving Rule-Based Refactorings". In: *Theory and Practice of Model Transformations – 4th International Conference, ICMT@TOOLS 2011, Zurich, Switzerland, June 27–28, 2011. Proceedings*. Vol. 6707. Lecture Notes in Computer Science. Springer, 2011, pp. 123–137. DOI: 10.1007/978-3-642-21732-6_9.

[BET12] E. Biermann, C. Ermel, and G. Taentzer. "Formal Foundation of Consistent EMF Model Transformations by Algebraic Graph Transformation". In: *Software and Systems Modeling* 11.2 (2012), pp. 227–250. DOI: 10.1007/s10270-011-0199-7.

[Béz04] J. Bézivin. "In Search of a Basic Principle for Model Driven Engineering". In: *Novatica/Upgrade* 5.2 (2004).

[Bil+19] R. Bill et al. "A Local and Global Tour on MOMoT". In: *Software and Systems Modeling* 18.2 (2019), pp. 1017–1046. DOI: 10.1007/s10270-017-0644-3.

[BP13] F. R. Burton and S. Poulding. "Complementing Metaheuristic Search with Higher Abstraction Techniques". In: *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering*. CMSBSE '13. IEEE Press. IEEE, 2013, pp. 45–48. DOI: 10.1109/CMSBSE.2013.6604436.

[BRW01] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley. "The Next Release Problem". In: *Information and Software Technology* 43.14 (2001), pp. 883–890. DOI: 10.1016/S0950-5849(01)00194-X.

[BSA17] I. Boussaïd, P. Siarry, and M. Ahmed-Nacer. "A Survey on Search-Based Model-Driven Engineering". In: *Automated Software Engineering* 24.2 (2017), pp. 233–294. DOI: 10.1007/s10515-017-0215-4.

[Buc+20] A. Bucchiarone et al. "Grand Challenges in Model-Driven Engineering: An Analysis of the State of the Research". In: *Software and Systems Modeling* 19.1 (2020), pp. 5–13. DOI: 10.1007/s10270-019-00773-6.

[Bur+12] F. R. Burton et al. "Solving Acquisition Problems Using Model-Driven Engineering". In: *Modelling Foundations and Applications – 8th European Conference, ECMFA 2012, Kongens Lyngby, Denmark, July 2–5, 2012. Proceedings*. Vol. 7349. Lecture Notes in Computer Science. Springer, Jan. 1, 2012, pp. 428–443. DOI: 10.1007/978-3-642-31491-9_32.

[BZ16] A. Burdusel and S. Zschaler. "Model Optimisation for Feature Class Allocation Using MDEOptimiser: A TTC 2016 Submission". In: *Proceedings of the 9th Transformation Tool Contest, co-located with the 2016 Software Technologies: Applications and Foundations (STAF 2016), Vienna, Austria, July 8, 2016*. Vol. 1758. CEUR Workshop Proceedings. CEUR-WS.org, 2016, pp. 33–38.

[BZ18] A. Burdusel and S. Zschaler. "Towards Automatic Generation of Evolution Rules for Model-Driven Optimisation". In: *Software Technologies: Applications and Foundations* (Jan. 1, 2018). DOI: 10.1007/978-3-319-74730-9_6.

[BZJ21] A. Burdusel, S. Zschaler, and S. John. "Automatic Generation of Atomic Multiplicity-Preserving Search Operators for Search-Based Model Engineering". In: *Software and Systems Modeling* 20.6 (2021), pp. 1857–1887. DOI: 10.1007/s10270-021-00914-w.

[BZS18] A. Burdusel, S. Zschaler, and D. Strüber. "MDEoptimiser: A Search Based Model Engineering Tool". In: *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS '18. ACM, 2018, pp. 12–16. DOI: 10.1145/3270112.3270130.

[CL20] B. Case and P. K. Lehre. "Self-Adaptation in Nonelitist Evolutionary Algorithms on Discrete Problems with Unknown Structure". In: *IEEE Transactions on Evolutionary Computation* 24.4 (2020), pp. 650–663. DOI: 10.1109/TEVC.2020.2985450.

[Cli93] N. Cliff. "Dominance Statistics: Ordinal Analyses to Answer Ordinal Questions". In: *Psychological Bulletin* 114.3 (1993), pp. 494–509. DOI: 10.1037/0033-2909.114.3.494.

[ČLM13] M. Črepinšek, S.-H. Liu, and M. Mernik. "Exploration and Exploitation in Evolutionary Algorithms: A Survey". In: *ACM Comput. Surv.* 45.3 (2013), 35:1–35:33. DOI: 10.1145/2480741.2480752.

[Coe10] C. A. C. Coello. "Constraint-Handling Techniques Used with Evolutionary Algorithms". In: *Genetic and Evolutionary Computation Conference, GECCO 2010, Proceedings, Portland, Oregon, USA, July 7–11, 2010, Companion Material*. ACM, 2010, pp. 2603–2624. DOI: 10.1145/1830761.1830910.

[Cor+01] D. W. Corne et al. "PESA-II: Region-Based Selection in Evolutionary Multiobjective Optimization". In: *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. GECCO'01. Morgan Kaufmann Publishers Inc., 2001, pp. 283–290.

[Cor+18] D. Corus et al. "Level-Based Analysis of Genetic Algorithms and Other Search Processes". In: *IEEE Transactions on Evolutionary Computation* 22.5 (2018), pp. 707–719. DOI: 10.1109/TEVC.2017.2753538.

[CR04] C. A. Coello Coello and M. Reyes Sierra. "A Study of the Parallelization of a Coevolutionary Multi-Objective Evolutionary Algorithm". In: *MICAI 2004: Advances in Artificial Intelligence*. Lecture Notes in Computer Science. Springer, 2004, pp. 688–697.

[Deb+02] K. Deb et al. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197. DOI: 10.1109/4235.996017.

[DHK12] B. Doerr, E. Happ, and C. Klein. "Crossover Can Provably Be Useful in Evolutionary Computation". In: *Theoretical Computer Science* 425 (2012), pp. 17–33. DOI: 10.1016/j.tcs.2010.10.035.

[DLN19] D. Dang, P. K. Lehre, and P. T. H. Nguyen. "Level-Based Analysis of the Univariate Marginal Distribution Algorithm". In: *Algorithmica* 81.2 (2019), pp. 668–702. DOI: 10.1007/s00453-018-0507-5.

[Doe+13] B. Doerr et al. "More Effective Crossover Operators for the All-Pairs Shortest Path Problem". In: *Theoretical Computer Science* 471 (2013), pp. 12–26. DOI: 10.1016/j.tcs.2012.10.059.

[Dur+11] J. J. Durillo et al. "A Study of the Bi-Objective Next Release Problem". In: *Empirical Software Engineering* 16.1 (2011), pp. 29–60.

[DZB10] C. Downey, M. Zhang, and W. N. Browne. "New Crossover Operators in Linear Genetic Programming for Multiclass Object Classification". In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*. GECCO '10. ACM, 2010, pp. 885–892. DOI: 10.1145/1830483.1830644.

[Ehr+06] H. Ehrig et al. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. Springer, 2006. DOI: 10.1007/3-540-31188-2.

[Ehr+12] H. Ehrig et al. "$\mathscr{M}$-Adhesive Transformation Systems with Nested Application Conditions. Part 2: Embedding, Critical Pairs and Local Confluence". In: *Fundam. Informaticae* 118.1-2 (2012), pp. 35–63. DOI: 10.3233/FI-2012-705.

[Ehr+14] H. Ehrig et al. "$\mathscr{M}$-Adhesive Transformation Systems with Nested Application Conditions. Part 1: Parallelism, Concurrency and Amalgamation". In: *Mathematical Structures in Computer Science* 24.4 (2014). DOI: 10.1017/S0960129512000357.

[Ehr+15] H. Ehrig et al. *Graph and Model Transformation – General Framework and Applications*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2015. DOI: 10.1007/978-3-662-47980-3.

[emf] *Eclipse Modeling Framework (EMF)*. Eclipse Foundation. URL: https://www.eclipse.org/emf (visited on 06/2023).

[ES15] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. 2nd. Natural Computing Series. Springer, 2015. DOI: 10.1007/978-3-662-44874-8.

[EWZ14] D. Efstathiou, J. R. Williams, and S. Zschaler. "Crepe Complete: Multi-Objective Optimization for Your Models". In: *Proceedings of the First International Workshop on Combining Modelling with Search- and Example-Based Approaches co-located with 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014), Valencia, Spain, September 28, 2014*. Vol. 1340. CEUR Workshop Proceedings. CEUR-WS.org, 2014, pp. 25–34.

[Fle+17] M. Fleck et al. "Model Transformation Modularization As a Many-Objective Optimization Problem". In: *IEEE Transactions on Software Engineering* 43.11 (2017), pp. 1009–1032. DOI: 10.1109/TSE.2017.2654255.

[Fog97] D. B. Fogel. "The Advantages of Evolutionary Computation". In: *Biocomputing and emergent computation: Proceedings of BCEC97*. World Scientific Press, 1997, pp. 1–11.

[Fri+18] L. Fritsche et al. "Short-Cut Rules – Sequential Composition of Rules Avoiding Unnecessary Deletions". In: *Software Technologies: Applications and Foundations – STAF 2018 Collocated Workshops, Toulouse, France, June 25–29, 2018, Revised Selected Papers*. Vol. 11176. Lecture Notes in Computer Science. Springer, 2018, pp. 415–430. DOI: 10.1007/978-3-030-04771-9_30.

[FSK17] S. Faridmoayer, M. Sharbaf, and S. Kolahdouz-Rahimi. "Optimization of Model Transformation Output Using Genetic Algorithm". In: *2017 IEEE 4th International Conference on Knowledge-Based*

Engineering and Innovation (KBEI). 2017, pp. 0203–0209. DOI: 10.1109/KBEI.2017.8324973.

[FTW15] M. Fleck, J. Troya, and M. Wimmer. "Marrying Search-Based Optimization and Model Transformation Technology". In: *Proceedings of the First North American Search Based Software Engineering Symposium*. Elsevier, 2015, pp. 1–16.

[FTW16] M. Fleck, J. Troya, and M. Wimmer. "The Class Responsibility Assignment Case". In: *Proceedings of the 9th Transformation Tool Contest (TTC 2016)*. 2016, pp. 1–10.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[GL12] R. Garner and S. Lack. "On the Axioms for Adhesive and Quasiadhesive Categories". In: *Theory and Applications of Categories* 27.3 (2012), pp. 27–46.

[Gol89] D. E. Goldberg. "Genetic Algorithms in Search Optimization and Machine Learning". In: 1989.

[Har+12] M. Harman et al. "Dynamic Adaptive Search Based Software Engineering". In: *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*. ESEM '12. Association for Computing Machinery, 2012, pp. 1–8. DOI: 10.1145/2372251.2372253.

[Hei12] T. Heindel. "Adhesivity with Partial Maps Instead of Spans". In: *Fundam. Informaticae* 118.1-2 (2012), pp. 1–33. DOI: 10.3233/FI-2012-704.

[henshin] *Henshin*. Eclipse Foundation. URL: https://www.eclipse.org/henshin/ (visited on 06/2023).

[Her+14] F. Hermann et al. "Formal Analysis of Model Transformations Based on Triple Graph Grammars". In: *Mathematical Structures in Computer Science* 24.4 (2014). DOI: 10.1017/S0960129512000370.

[HJ01] M. Harman and B. F. Jones. “Search-Based Software Engineering”. In: *Information and Software Technology* 43.14 (2001), pp. 833–839. DOI: 10.1016/S0950-5849(01)00189-6.

[HJ98] M. P. Hansen and A. Jaszkiewicz. *Evaluating the Quality of Approximations to the Non-Dominated Set*. Tech. rep. IMM Technical Report IMM-REP-1998-7. Technical University of Denmark, 1998.

[HK18] J. Husa and R. Kalkreuth. “A Comparative Study on Crossover in Cartesian Genetic Programming”. In: *Genetic Programming – 21st European Conference, EuroGP 2018, Parma, Italy, April 4–6, 2018, Proceedings*. Vol. 10781. Lecture Notes in Computer Science. Springer, 2018, pp. 203–219. DOI: 10.1007/978-3-319-77553-1_13.

[HMZ12] M. Harman, S. A. Mansouri, and Y. Zhang. “Search-Based Software Engineering: Trends, Techniques and Applications”. In: *ACM Computing Surveys* 45.1 (2012), 11:1–11:61. DOI: 10.1145/2379776.2379787.

[Hor+22] J. M. Horcas et al. “We’re Not Gonna Break It! Consistency-Preserving Operators for Efficient Product Line Configuration”. In: *IEEE Transactions on Software Engineering* (2022). DOI: 10.1109/TSE.2022.3171404.

[HP09] A. Habel and K. Pennemann. “Correctness of High-Level Transformation Systems Relative to Nested Conditions”. In: *Mathematical Structures in Computer Science* 19.2 (2009), pp. 245–296. DOI: 10.1017/S0960129508007202.

[HWR14] J. Hutchinson, J. Whittle, and M. Rouncefield. “Model-Driven Engineering Practices in Industry: Social, Organizational and Managerial Factors That Lead to Success or Failure”. In: *Science of Computer Programming*. Special issue on Success Stories in Model Driven Engineering 89.B (2014), pp. 144–161. DOI: 10.1016/j.scico.2013.03.017.

[Jia+14] S. Jiang et al. “Consistencies and Contradictions of Performance Metrics in Multiobjective Optimization”. In: *IEEE Transactions on Cybernetics* 44.12 (2014), pp. 2391–2404. DOI: 10.1109/TCYB.2014.2307319.

[JKT22a] S. John, J. Kosiol, and G. Taentzer. "Towards a Configurable Crossover Operator for Model-Driven Optimization". In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS '22. ACM, 2022, pp. 388–395. DOI: 10.1145/3550356.3561603.

[JKT22b] S. John, J. Kosiol, and G. Taentzer. *Towards a Configurable Crossover Operator for Model-Driven Optimization - Accompanying Data*. URL: https://github.com/Leative/MDEIntelligence22-MDO-crossover-evaluation (visited on 06/2023).

[JM91] C. Janikow and Z. Michalewicz. "An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms". In: 1991.

[Joh+19a] S. John et al. "Searching for Optimal Models: Comparing Two Encoding Approaches". In: *Journal of Object Technology* 18.3 (2019), 6:1–22. DOI: 10.5381/jot.2019.18.3.a6.

[Joh+19b] S. John et al. *Searching for Optimal Models: Comparing Two Encoding Approaches - Accompanying Data*. DOI: 10.6084/m9.figshare.8236505. URL: http://dx.doi.org/10.6084/m9.figshare.8236505 (visited on 06/2023).

[Joh+23a] S. John et al. "A Graph-Based Framework for Model-Driven Optimization Facilitating Impact Analysis of Mutation Operator Properties". In: *Software and Systems Modeling* (2023). DOI: 10.1007/s10270-022-01078-x.

[Joh+23b] S. John et al. *A Graph-Based Framework for Model-Driven Optimization Facilitating Impact Analysis of Mutation Operator Properties - Accompanying Data*. URL: https://github.com/Leative/SoSyM22-MDO-framework-evaluation (visited on 06/2023).

[KB01] W. Kantschik and W. Banzhaf. "Linear-Tree GP and Its Comparison with Other GP Structures". In: *Genetic Programming*. Lecture Notes in Computer Science. Springer, 2001, pp. 302–312. DOI: 10.1007/3-540-45355-5_24.

[KB02] W. Kantschik and W. Banzhaf. “Linear-Graph GP – A New GP Structure”. In: *Genetic Programming, 5th European Conference, EuroGP 2002, Kinsale, Ireland, April 3–5, 2002, Proceedings*. Vol. 2278. Lecture Notes in Computer Science. Springer, 2002, pp. 83–92. DOI: 10.1007/3-540-45984-7_8.

[KLW13] M. Kessentini, P. Langer, and M. Wimmer. “Searching Models, Modeling Search: On the Synergies of SBSE and MDE”. In: *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering*. CMSBSE ’13. IEEE, 2013, pp. 51–54. DOI: 10.1109/cmsbse.2013.6604438.

[Kol+14] S. Kolahdouz-Rahimi et al. “Evaluation of Model Transformation Approaches for Model Refactoring”. In: *Science of Computer Programming* 85 (2014), pp. 5–40. DOI: 10.1016/j.scico.2013.07.013.

[Kos+20a] J. Kosiol et al. “Double-Pushout-Rewriting in *S*-Cartesian Functor Categories: Rewriting Theory and Application to Partial Triple Graphs”. In: *J. Log. Algebraic Methods Program.* 115 (2020), p. 100565. DOI: 10.1016/j.jlamp.2020.100565.

[Kos+20b] J. Kosiol et al. “Graph Consistency As a Graduated Property”. In: *Graph Transformation*. Lecture Notes in Computer Science. Springer, 2020, pp. 239–256. DOI: 10.1007/978-3-030-51372-6_14.

[Kos+22] J. Kosiol et al. “Sustaining and Improving Graduated Graph Consistency: A Static Analysis of Graph Transformations”. In: *Science of Computer Programming* 214.C (2022), p. 102729. DOI: 10.1016/j.scico.2021.102729.

[Koz92] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Complex adaptive systems. MIT Press, 1992.

[Kra07] J. Kramer. “Is Abstraction the Key to Computing?” In: *Communications of the ACM* 50.4 (2007), pp. 36–42. DOI: 10.1145/1232743.1232745.

[KRD17] R. Kalkreuth, G. Rudolph, and A. Droschinsky. "A New Subgraph Crossover for Cartesian Genetic Programming". In: *Genetic Programming – 20th European Conference, EuroGP 2017, Amsterdam, The Netherlands, April 19–21, 2017, Proceedings*. Vol. 10196. Lecture Notes in Computer Science. 2017, pp. 294–310. DOI: 10.1007/978-3-319-55696-3_19.

[Lar+07] J. de Lara et al. "Attributed Graph Transformation with Node Type Inheritance". In: *Theoretical Computer Science* 376.3 (2007), pp. 139–163. DOI: 10.1016/j.tcs.2007.02.001.

[Lie+14] G. Liebel et al. "Assessing the State-Of-Practice of Model-Based Engineering in the Embedded Systems Domain". In: *Model-Driven Engineering Languages and Systems*. Lecture Notes in Computer Science. Springer, 2014, pp. 166–182. DOI: 10.1007/978-3-319-11653-2_11.

[LK13] K. Lano and S. Kolahdouz Rahimi. "Case Study: Class Diagram Restructuring". In: *Electronic Proceedings in Theoretical Computer Science*. Vol. 135. EPTCS. 2013, pp. 8–15. DOI: 10.4204/EPTCS.135.2.

[Löw+15] M. Löwe et al. "Algebraic Graph Transformations with Inheritance and Abstraction". In: *Science of Computer Programming* 107-108 (2015), pp. 2–18. DOI: 10.1016/j.scico.2015.02.004.

[LS05] S. Lack and P. Sobociński. "Adhesive and Quasiadhesive Categories". In: *RAIRO Theor. Informatics Appl.* 39.3 (2005), pp. 511–545. DOI: 10.1051/ita:2005028.

[Mac71] S. MacLane. *Categories for the Working Mathematician*. Vol. 5. Graduate Texts in Mathematics. Springer, 1971.

[mdeo] A. Burdusel and S. Zschaler. *MDEOptimiser*. URL: https://mde-optimiser.github.io (visited on 06/2023).

[mdover] S. John, J. Kosiol, and G. Taentzer. *Md⊙ver: A Crossover Implementation for Model-Driven Optimization*. URL: https://mdo-over.github.io/ (visited on 06/2023).

[MHB14] W. J. Ma, M. Husain, and P. M. Bays. “Changing Concepts of Working Memory”. In: *Nature Neuroscience* 17.3 (2014), pp. 347–356. DOI: 10.1038/nn.3655.

[Mic95] Z. Michalewicz. “A Survey of Constraint Handling Techniques in Evolutionary Computation Methods”. In: *Proceedings of the Fourth Annual Conference on Evolutionary Programming, EP 1995, San Diego, CA, USA, March 1–3, 1995*. A Bradford Book, MIT Press. Cambridge, Massachusetts., 1995, pp. 135–155.

[Mil56] G. A. Miller. “The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information”. In: *Psychological Review* 63.2 (1956), pp. 81–97. DOI: 10.1037/h0043158.

[MJ14] H. Masoud and S. Jalili. “A Clustering-Based Model for Class Responsibility Assignment Problem in Object-Oriented Analysis”. In: *J. Syst. Softw.* 93 (2014), pp. 110–131. DOI: 10.1016/j.jss.2014.02.053.

[MNR10] P. Machado, H. Nunes, and J. Romero. “Graph-Based Evolution of Visual Languages”. In: *Applications of Evolutionary Computation, EvoApplications 2010: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoMUSART, and EvoTRANSLOG, Istanbul, Turkey, April 7–9, 2010, Proceedings, Part II*. Vol. 6025. Lecture Notes in Computer Science. Springer, 2010, pp. 271–280. DOI: 10.1007/978-3-642-12242-2_28.

[moea] *MOEA Framework*. URL: http://moeaframework.org/ (visited on 06/2023).

[Moh+13] P. Mohagheghi et al. “An Empirical Study of the State of the Practice and Acceptance of Model-Driven Engineering in Four Industrial Cases”. In: *Empirical Software Engineering* 18.1 (2013), pp. 89–116. DOI: 10.1007/s10664-012-9196-x.

[momot] M. Fleck, J. Troya, and M. Wimmer. *MOMoT*. URL: http://martin-fleck.github.io/momot/ (visited on 06/2023).

[MW47] H. B. Mann and D. R. Whitney. “On a Test of Whether One of Two Random Variables Is Stochastically Larger Than the Other”. In: *The Annals of Mathematical Statistics* 18.1 (1947), pp. 50–60.

[Nas+18] N. Nassar et al. “OCL2AC: Automatic Translation of OCL Constraints to Graph Constraints and Application Conditions for Transformation Rules”. In: *Graph Transformation*. Lecture Notes in Computer Science. Springer, 2018, pp. 171–177. DOI: 10.1007/978-3-319-92991-0_11.

[Nas+20] N. Nassar et al. “Constructing Optimized Constraint-Preserving Application Conditions for Model Transformation Rules”. In: *J. Log. Algebraic Methods Program.* 114 (2020), p. 100564. DOI: 10.1016/j.jlamp.2020.100564.

[Nie03] J. Niehaus. “Graphbasierte Genetische Programmierung”. PhD thesis. Technical University of Dortmund, Germany, 2003.

[Nob+13] M. S. Nobile et al. “The Foundation of Evolutionary Petri Nets”. In: *Proceedings of the International Workshop on Biological Processes & Petri Nets, Milano, Italy, June 24, 2013*. Vol. 988. CEUR Workshop Proceedings. CEUR-WS.org, 2013, pp. 60–74.

[ocl] *Object Constraint Language*. Object Management Group. URL: http://www.omg.org/spec/OCL/ (visited on 06/2023).

[Pai+15] T. Paixão et al. “Toward a Unifying Framework for Evolutionary Processes”. In: *Journal of Theoretical Biology* 383 (2015), pp. 28–43. DOI: 10.1016/j.jtbi.2015.07.011.

[PD18] E. C. Pinto and C. Doerr. “A Simple Proof for the Usefulness of Crossover in Black-Box Optimization”. In: *Parallel Problem Solving from Nature – PPSN XV*. Lecture Notes in Computer Science. Springer, 2018, pp. 29–41. DOI: 10.1007/978-3-319-99259-4_3.

[Pen09] K. Pennemann. “Development of Correct Graph Transformation Systems”. PhD thesis. University of Oldenburg, Germany, 2009.

[Per+99] F. B. Pereira et al. “Graph Based Crossover – a Case Study with the Busy Beaver Problem”. In: *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation – Volume 2*. GECCO’99. Morgan Kaufmann Publishers Inc., 1999, pp. 1149–1155.

[Plu98] D. Plump. “Termination of Graph Rewriting Is Undecidable”. In: *Fundam. Informaticae* 33.2 (1998), pp. 201–209. DOI: 10.3233/FI-1998-33204.

[Plu99] D. Plump. “Term Graph Rewriting”. In: *Handbook of Graph Grammars and Computing by Graph Transformation*. Vol. 2. World Scientific, 1999, pp. 3–61.

[Pot96] J. Potvin. “Genetic Algorithms for the Traveling Salesman Problem”. In: *Ann. Oper. Res.* 63.3 (1996), pp. 337–370. DOI: 10.1007/BF02125403.

[Rad+18] H. Radke et al. “Translating Essential OCL Invariants to Nested Graph Constraints for Generating Instances of Meta-Models”. In: *Science of Computer Programming* 152 (2018), pp. 38–62. DOI: 10.1016/j.scico.2017.08.006.

[RLB15] N. Riquelme, C. V. Lücken, and B. Baran. “Performance Metrics in Multi-Objective Optimization”. In: *Proc. Latin American Computing Conf. (CLEI)*. 2015, pp. 1–11. DOI: 10.1109/CLEI.2015.7360024.

[Rot+89] J. Rothenberg et al. “The Nature of Modeling”. In: *in Artificial Intelligence, Simulation and Modeling*. John Wiley & Sons, 1989, pp. 75–92.

[RSV04] A. Rensink, Á. Schmidt, and D. Varró. “Model Checking Graph Transformations: A Comparison of Two Approaches”. In: *Graph Transformations, Second International Conference, ICGT 2004, Rome, Italy, September 28 - October 2, 2004, Proceedings*. Vol. 3256. Lecture Notes in Computer Science. Springer, 2004, pp. 226–241. DOI: 10.1007/978-3-540-30203-2_17.

[Rub12] K. S. Rubin. *Essential Scrum. A Practical Guide to the Most Popular Agile Process*. Addison-Wesley, 2012.

[Rul+18] S. Ruland et al. “Controlling the Attack Surface of Object-Oriented Refactorings”. In: *Fundamental Approaches to Software Engineering*. Lecture Notes in Computer Science. Springer, 2018, pp. 38–55.

[Sch06] D. C. Schmidt. “Guest Editor’s Introduction: Model-Driven Engineering”. In: *Computer* 39.2 (2006), pp. 25–31. DOI: 10.1109/MC.2006.58.

[SNV18] O. Semeráth, A. S. Nagy, and D. Varró. “A Graph Solver for the Automated Generation of Consistent Domain-Specific Models”. In: *Proceedings of the 40th International Conference on Software Engineering*. ICSE ’18. Association for Computing Machinery, 2018, pp. 969–980. DOI: 10.1145/3180155.3180186.

[SPG10] C. L. Simons, I. C. Parmee, and R. Gwynllyw. “Interactive, Evolutionary Search in Upstream Object-Oriented Class Design”. In: *IEEE Transactions on Software Engineering* 36.6 (2010), pp. 798–816. DOI: 10.1109/TSE.2010.34.

[Ste+08] D. Steinberg et al. *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2008.

[Str+17] D. Strüber et al. “Henshin: A Usability-Focused Framework for EMF Model Transformation Development”. In: *Graph Transformation*. Lecture Notes in Computer Science. Springer, 2017, pp. 196–208. DOI: 10.1007/978-3-319-61470-0_12.

[Str17] D. Strüber. “Generating Efficient Mutation Operators for Search-Based Model-Driven Engineering”. In: *Theory and Practice of Model Transformation*. Lecture Notes in Computer Science (Jan. 1, 2017), pp. 121–137. DOI: 10.1007/978-3-319-61473-1_9.

[Sud17] D. Sudholt. “How Crossover Speeds up Building Block Assembly in Genetic Algorithms”. In: *Evolutionary Computation* 25.2 (2017), pp. 237–274. DOI: 10.1162/EVCO_a_00171.

[TJK22] G. Taentzer, S. John, and J. Kosiol. “A Generic Construction for Crossovers of Graph-Like Structures”. In: *Graph Transformation - 15th International Conference, ICGT 2022, Held as Part of STAF 2022, Nantes, France, July 7-8, 2022, Proceedings*. Vol. 13349.

Lecture Notes in Computer Science. Springer, 2022, pp. 97–117. DOI: 10.1007/978-3-031-09843-7_6.

[TK22] H. Thölke and J. Kosiol. "A Multiplicity-Preserving Crossover Operator on Graphs". In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS '22. Accepted. ACM, 2022, pp. 588–597. DOI: 10.1145/3550356.3561587.

[WCT12] T. Weise, R. Chiong, and K. Tang. "Evolutionary Optimization: Pitfalls and Booby Traps". In: *Journal of Computer Science and Technology* 27.5 (2012), pp. 907–936. DOI: 10.1007/s11390-012-1274-4.

[WL05] S. Wappler and F. Lammermann. "Using Evolutionary Algorithms for the Unit Testing of Object-Oriented Software". In: *Genetic and Evolutionary Computation Conference, GECCO 2005, Proceedings, Washington DC, USA, June 25–29, 2005*. GECCO '05. ACM, 2005, pp. 1053–1060. DOI: 10.1145/1068009.1068187.

[WL96] A. S. Wu and R. K. Lindsay. "A Comparison of the Fixed and Floating Building Block Representation in the Genetic Algorithm". In: *Evolutionary Computation* 4.2 (1996), pp. 169–193. DOI: 10.1162/evco.1996.4.2.169.

[ZBT07] E. Zitzler, D. Brockhoff, and L. Thiele. "The Hypervolume Indicator Revisited: On the Design of Pareto-Compliant Indicators Via Weighted Integration". In: *Evolutionary multi-criterion optimization*. Vol. 4403. Lecture Notes in Computer Science. Springer, 2007, pp. 862–876.

[Zit+03] E. Zitzler et al. "Performance Assessment of Multiobjective Optimizers: An Analysis and Review". In: *IEEE Transactions on Evolutionary Computation* 7.2 (2003), pp. 117–132. DOI: 10.1109/TEVC.2003.810758.

[ZLT01] E. Zitzler, M. Laumanns, and L. Thiele. "SPEA2: Improving the Strength Pareto Evolutionary Algorithm". In: 2001. DOI: 10.3929/ETHZ-A-004284029.

[ZM16] S. Zschaler and L. Mandow. “Towards Model-Based Optimisation: Using Domain Knowledge Explicitly”. In: *Software Technologies: Applications and Foundations*. Lecture Notes in Computer Science 9946 (Jan. 1, 2016), pp. 317–329. DOI: 10.1007/978-3-319-50230-4_24.