Towards a Configurable Crossover Operator for Model-Driven Optimization

Stefan John
johns@mathematik.uni-marburg.de
Philipps-Universität Marburg
Marburg, Germany

Jens Kosiol
kosiolje@mathematik.uni-marburg.de
Philipps-Universität Marburg
Marburg, Germany

Gabriele Taentzer
taentzer@mathematik.uni-marburg.de
Philipps-Universität Marburg
Marburg, Germany

ABSTRACT

In evolutionary algorithms, mutation and crossover are used to explore a search space for solutions. For the model-based approach to model-driven optimization, where models are used to represent solutions, no crossover operator has been introduced yet. However, theoretical and experimental evidence shows that evolutionary search can benefit from the use of crossover. We present a configurable crossover operator for models defined in the Eclipse Modeling Framework (EMF), discuss several variants of incorporating domain knowledge into this operator, and argue that it produces EMF models again. We also present a prototype implementation of our crossover operator and conduct an initial evaluation to investigate the effectiveness of evolutionary computations that use both mutation and crossover.

CCS CONCEPTS

• **Software and its engineering** → **Search-based software engineering**.

KEYWORDS

search-based software engineering, model-driven optimization, evolutionary computation, crossover

ACM Reference Format:
Stefan John, Jens Kosiol, and Gabriele Taentzer. 2022. Towards a Configurable Crossover Operator for Model-Driven Optimization. In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3550356.3561603

1 INTRODUCTION

A variety of software engineering problems such as software modularization [4], software testing [33], and release planning [2] can be viewed as optimization problems. *Search-based software engineering* (SBSE) [16] 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 [17].

MODELS '22 Companion, October 23–28, 2022, Montreal, QC, Canada

ACM ISBN 978-1-4503-9467-3/22/10…$15.00
https://doi.org/10.1145/3550356.3561603

With reference to e.g. [13, 35], 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) [28] 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. This combination of SBSE and MDE is lately referred to as *model-driven optimization* (MDO) [19]. Two main approaches have emerged in MDO: The *model-based* approach [7, 35] performs optimization directly on models, while the *rule-based* approach [1, 3] 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 [10, 18, 30]. Since the model-based approach tends to be more effective than the rule-based approach [19], even without crossover, we take initial steps to implement and apply crossover in the model-based approach to MDO (hereafter referred to as MDO for short).

In 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 [21, 25, 31] to implement crossover. However, these crossover constructions are very generic and do not consider model-

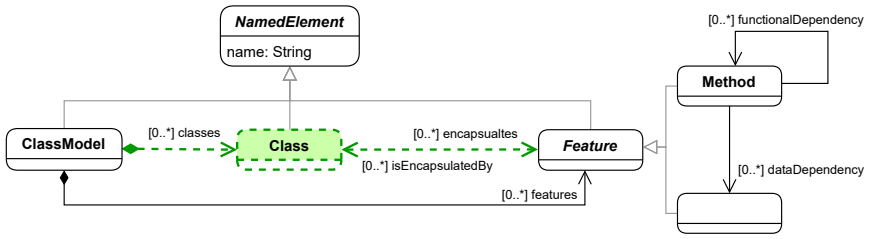


Figure 1: Meta-model for the CRA case (slightly adapted from [15])

or even domain-specific aspects. Since the Eclipse Modeling Framework (EMF) [11] has become a de facto standard technology for defining models and modeling languages, and MDO concepts have been developed and implemented for EMF models [5], a crossover operator for EMF models should also be available.

In this paper, 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 [12], 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 MDO.

2 RELATED WORK

In the rule-based approach to MDO, the solutions are represented as sequences of transformation units [1, 3]. 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 [1]. A non-executable unit can alternatively be repaired by, for example, replacing it with a random executable unit [3]. 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 [6]. Their specific use case allows adaptation of one-point crossover through model transformations. However, their crossover implementation is not described in detail. In [35], 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 [5]. 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.

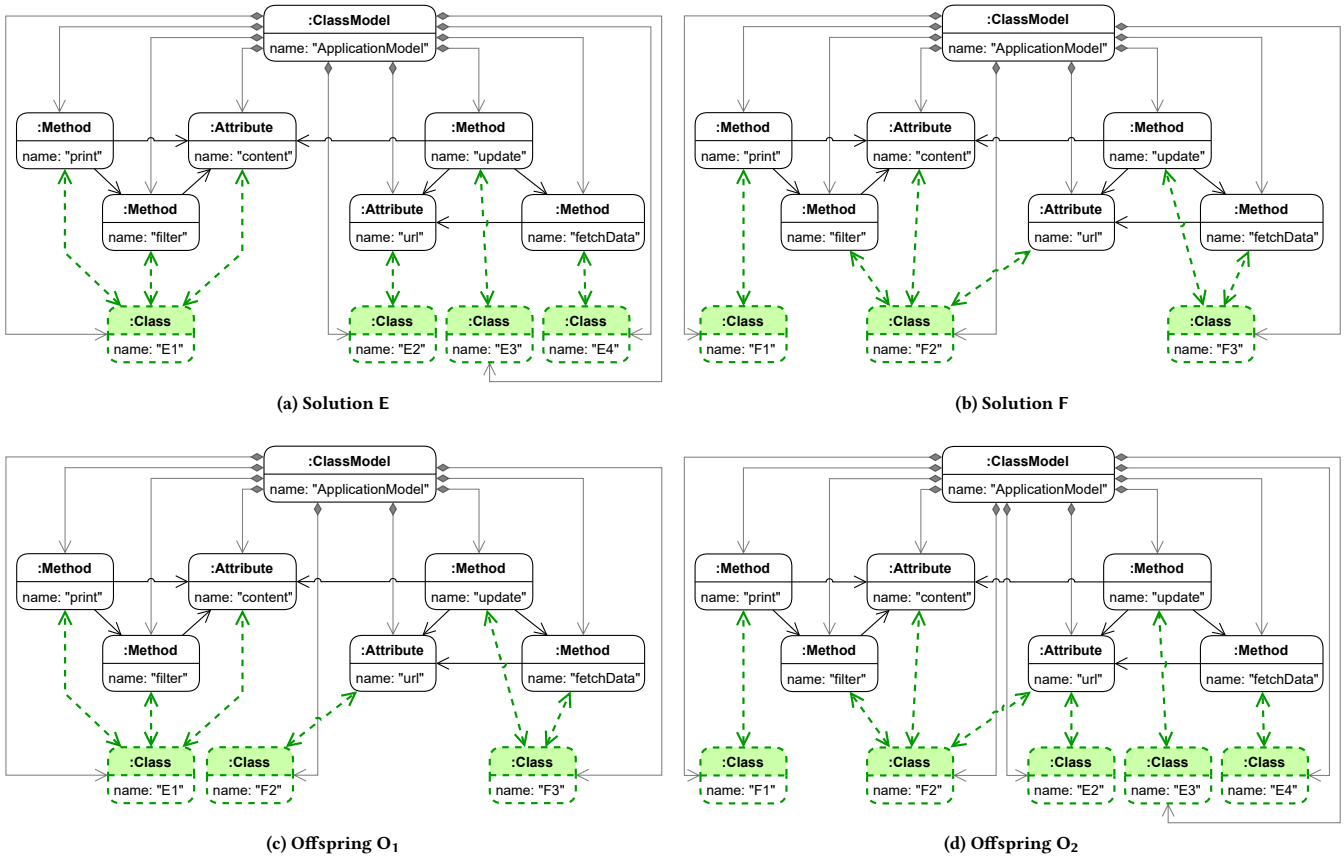
3 RUNNING EXAMPLE

The class responsibility assignment (CRA) case [4] 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 the core concepts of 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 [15] and recalled, with slightly adapted multiplicities, in Fig. 1. 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. 1 (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 [15]. 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).

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. 2a and 2b) 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. 2c. In fact, we can assign a descriptive name to each of the classes in O_1 that reflects its purpose from a



(a) Solution E

(b) Solution F

(c) Offspring O_1

(d) Offspring O_2

Figure 2: 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.)

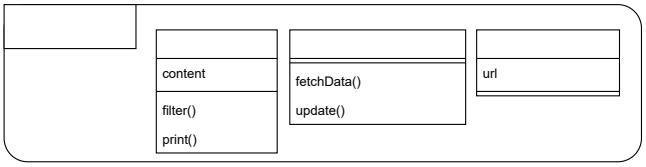


Figure 3: Class diagram represented by offspring O_1 with class names adapted to reflect their purpose.

semantic point of view; Fig. 3 shows the resulting class model in concrete syntax. Offspring O_2 (in Fig. 2d) 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 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.

4.1 Setting

EMF models. The *Eclipse Modeling Framework* (EMF) [11] 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) [26]. 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 MDO. Metamodels are used to define the underlying modeling language; a metamodel 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 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* [31]. 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. 1 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. 2. 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 paper, 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., [8, 24]). 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., [10, 27]). 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 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 [31] 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 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 1: Splitting algorithm

```
1  input:  solution model E                    //E is problem-rooted
2  output: models E_1 and E_2                  //split parts
3          model E_I                           //split point
4
5  E_1, E_2, E_I ← PI_P
6  compute C_temp as the set of containment edges outside of E_I whose
       source nodes belong to E_I
7
8  while C_temp is not empty
9    if |C_temp| = 1
10     include that containment edge in E_1, E_2, and E_I
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 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 (Sect. 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.

4.3 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 the definition from [31], 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 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 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 (Sect. 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.

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. For the configuration points of the split, we chose a random distribution of containment subtrees (line 13 in Alg. 1) and non-containment references (line 22 in Alg. 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 [12] 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 [22].

6 INITIAL EVALUATION

We conducted experiments focusing on the following research question; all evaluation data can be found at [14]:

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

Set up. Our running example (Sect. 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 E). To perform optimizations we use the framework MDEOptimiser [5, 23]. It supports Henshin transformation rules as mutation operators; to apply crossover, we integrated our crossover operator presented in Sec. 5. As the underlying evolutionary algorithm we choose NSGA-II [9], which is generally well established in both MDO [1, 5] and SBSE [17]. 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*.

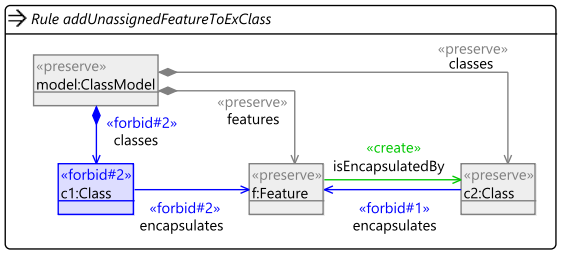


Figure 4: Mutation rule (2) in Henshin syntax. The green edge (as well as its opposite edge encapsulates) is created. The blue elements prevent application of the rule if Feature f is already assigned to the same or another Class.

For the mutations, we rely on the transformation rules proposed for the CRA case in [5, 19]. 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), 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 *consistency-sustaining* in the sense of [20].

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 [9]. 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 [13] 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 [34] 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 [34]. 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 1 summarizes the mean hypervolumes for what we found the most interesting algorithm variants. For four of them, Fig. 5 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. 5). 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.

Threats to Validity. Being a descriptive example, the CRA case exhibits only a low structural complexity and can also be solved with traditional encodings [29] . 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 [5]) 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.

Table 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**

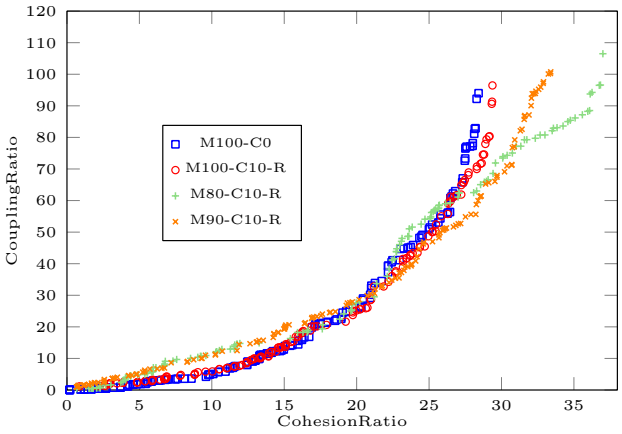


Figure 5: Cumulative approximation sets of selected algorithm variants for model D.

7 CONCLUSION

In this paper, 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.

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 [29]. 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 [32]. 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.

ACKNOWLEDGMENTS

This work has been partially supported by the Deutsche Forschungsgemeinschaft (DFG), grant TA 294/19-1.

REFERENCES

[1] Hani Abdeen, Dániel Varró, Houari Sahraoui, András Szabolcs Nagy, Csaba Debreceni, Ábel Hegedüs, and Ákos Horváth. 2014. Multi-objective Optimization in Rule-based Design Space Exploration. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering* (Vasteras, Sweden) *(ASE '14)*. ACM, 289–300. https://doi.org/10.1145/2642937.2643005

[2] A.J. Bagnall, V.J. Rayward-Smith, and I.M. Whittley. 2001. The Next Release Problem. *Information and Software Technology* 43, 14 (2001), 883–890. https://doi.org/10.1016/S0950-5849(01)00194-X

[3] Robert Bill, Martin Fleck, Javier Troya, Tanja Mayerhofer, and Manuel Wimmer. 2019. A Local and Global Tour on Momot. *Software & Systems Modeling* 18, 2 (01 April 2019), 1017–1046. https://doi.org/10.1007/s10270-017-0644-3

[4] Michael Bowman, Lionel C. Briand, and Yvan Labiche. 2010. Solving the Class Responsibility Assignment Problem in Object-oriented Analysis with Multi-objective Genetic Algorithms. *IEEE Trans. Softw. Eng.* 36, 6 (Nov. 2010), 817–837. https://doi.org/10.1109/TSE.2010.70

[5] Alexandru Burdusel, Steffen Zschaler, and Stefan John. 2021. Automatic Generation of Atomic Multiplicity-preserving Search Operators for Search-based Model Engineering. *Software and Systems Modeling* 20, 6 (Dec. 2021), 1857–1887. https://doi.org/10.1007/s10270-021-00914-w

[6] Frank R. Burton, Richard F. Paige, Louis M. Rose, Dimitrios S. Kolovos, Simon M. Poulding, and Simon Smith. 2012. 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 (Lecture Notes in Computer Science, Vol. 7349)*, Antonio Vallecillo, Juha-Pekka Tolvanen, Ekkart Kindler, Harald Störrle, and Dimitrios S. Kolovos (Eds.). Springer, 428–443. https://doi.org/10.1007/978-3-642-31491-9_32

[7] Frank R. Burton and Simon Poulding. 2013. Complementing Metaheuristic Search with Higher Abstraction Techniques. In *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering* (San Francisco, California) *(CMSBSE '13)*. IEEE Press, 45–48. http://dl.acm.org/citation.cfm?id=2662572.2662586

[8] Carlos A. Coello Coello. 2010. 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*, Martin Pelikan and Jürgen Branke (Eds.). ACM, 2603–2624. https://doi.org/10.1145/1830761.1830910

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (April 2002), 182–197. https://doi.org/10.1109/4235.996017

[10] Benjamin Doerr, Daniel Johannsen, Timo Kötzing, Frank Neumann, and Madeleine Theile. 2013. More Effective Crossover Operators for the All-pairs Shortest Path Problem. *Theor. Comput. Sci.* 471 (2013), 12–26. https://doi.org/10.1016/j.tcs.2012.10.059

[11] Eclipse. 2022. Eclipse Modeling Framework (emf). http://www.eclipse.org/emf.

[12] Eclipse. 2022. Henshin. http://www.eclipse.org/henshin/.

[13] A. E. Eiben and James E. Smith. 2015. *Introduction to Evolutionary Computing* (2nd ed.). Springer Publishing Company, Incorporated.

[14] Evaluation Data. 2022. Evaluation Data: Results and Artifacts. https://github.com/Leative/MDEIntelligence22-MDO-crossover-evaluation.

[15] Martin Fleck, Javier Troya, and Manuel Wimmer. 2016. The Class Responsibility Assignment Case. In *Proceedings of the 9th Transformation Tool Contest (TTC 2016)*. 1–10. http://ceur-ws.org/Vol-1758/paper1.pdf

[16] Mark Harman and Bryan F. Jones. 2001. Search-based Software Engineering. *Information and Software Technology* 43, 14 (2001), 833–839. https://doi.org/10.1016/S0950-5849(01)00189-6

[17] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based Software Engineering: Trends, Techniques and Applications. *ACM Comput. Surv.* 45, 1, Article 11 (Dec. 2012), 61 pages. https://doi.org/10.1145/2379776.2379787

[18] Jakub Husa and Roman Kalkreuth. 2018. A Comparative Study on Crossover in Cartesian Genetic Programming. In *Genetic Programming – 21st European Conference, EuroGP 2018, Parma, Italy, April 4–6, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10781)*, Mauro Castelli, Lukás Sekanina, Mengjie Zhang, Stefano Cagnoni, and Pablo García-Sánchez (Eds.). Springer, 203–219. https://doi.org/10.1007/978-3-319-77553-1_13

[19] Stefan John, Robert Bill Alexandru Burdusel, Daniel Strüber, Gabriele Taentzer, Steffen Zschaler, and Manuel Wimmer. 2019. Searching for Optimal Models: Comparing Two Encoding Approaches. *Journal of Object Technology* 18, 3 (2019), 6:1–22. https://doi.org/10.5381/jot.2019.18.3.a6

[20] Jens Kosiol, Daniel Strüber, Gabriele Taentzer, and Steffen Zschaler. 2022. Sustaining and Improving Graduated Graph Consistency: A Static Analysis of Graph Transformations. *Sci. Comput. Program.* 214 (2022), 102729. https://doi.org/10.1016/j.scico.2021.102729

[21] Penousal Machado, Henrique Nunes, and Juan Romero. 2010. 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 (Lecture Notes in Computer Science, Vol. 6025)*, Cecilia Di Chio, Anthony Brabazon, Gianni A. Di Caro, Marc Ebner, Muddassar Farooq, Andreas Fink, Jörn Grahl, Gary Greenfield, Penousal Machado, Michael O'Neill, Ernesto Tarantino, and Neil Urquhart (Eds.). Springer, 271–280. https://doi.org/10.1007/978-3-642-12242-2_28

[22] MD©ver. 2022. Crossover Implementation. https://mdo-over.github.io/.

[23] MDEO. 2022. Mde Optimiser. http://mde-optimiser.github.io.

[24] Zbigniew Michalewicz. 1995. 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*, John R. McDonnell, Robert G. Reynolds, and David B. Fogel (Eds.). A Bradford Book, MIT Press. Cambridge, Massachusetts., 135–155.

[25] Jens Niehaus. 2003. *Graphbasierte Genetische Programmierung*. Ph. D. Dissertation. Tech. University of Dortmund, Germany. http://hdl.handle.net/2003/2744

[26] OMG 2014. *Object Constraint Language*. OMG. http://www.omg.org/spec/OCL/

[27] Jean-Yves Potvin. 1996. Genetic Algorithms for the Traveling Salesman Problem. *Ann. Oper. Res.* 63, 3 (1996), 337–370. https://doi.org/10.1007/BF02125403

[28] D. C. Schmidt. 2006. Guest Editor's Introduction: Model-driven Engineering. *Computer* 39, 2 (Feb. 2006), 25–31. https://doi.org/10.1109/MC.2006.58

[29] Christopher L. Simons, Ian C. Parmee, and Rhys Gwynllyw. 2010. Interactive, Evolutionary Search in Upstream Object-oriented Class Design. *IEEE Transactions on Software Engineering* 36, 6 (2010), 798–816. https://doi.org/10.1109/TSE.2010.34

[30] Dirk Sudholt. 2017. How Crossover Speeds up Building Block Assembly in Genetic Algorithms. *Evolutionary Computation* 25, 2 (2017), 237–274. https://doi.org/10.1162/EVCO_a_00171

[31] Gabriele Taentzer, Stefan John, and Jens Kosiol. 2022. 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 (Lecture Notes in Computer Science, Vol. 13349)*, Nicolas Behr and Daniel Strüber (Eds.). Springer, 97–117. https://doi.org/10.1007/978-3-031-09843-7_6

[32] Henry Thölke and Jens Kosiol. 2022. A Multiplicity-preserving Crossover Operator on Graphs. In *Int. Workshop on "Model Driven Engineering, Verification, and Validation", Satellite Event of MoDELS 2022*. Accepted.

[33] Stefan Wappler and Frank Lammermann. 2005. Using Evolutionary Algorithms for the Unit Testing of Object-oriented Software. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation* (Washington DC, USA) *(GECCO '05)*. ACM, 1053–1060. https://doi.org/10.1145/1068009.1068187

[34] Eckart Zitzler, Dimo Brockhoff, and Lothar Thiele. 2007. The Hypervolume Indicator Revisited: On the Design of Pareto-compliant Indicators Via Weighted Integration. In *Proceedings of the 4th International Conference on Evolutionary Multi-Criterion Optimization* (Matsushima, Japan) *(EMO'07)*. Springer-Verlag, 862–876.

[35] Steffen Zschaler and Lawrence Mandow. 2016. *Towards Model-based Optimisation: Using Domain Knowledge Explicitly*. Lecture Notes in Computer Science, Vol. 9946. Springer International Publishing, 317–329. https://doi.org/10.1007/978-3-319-50230-4_24