Noname manuscript No.
(will be inserted by the editor)

Handling nonconforming individuals in Search-Based Model-Driven Engineering

Nine generic strategies for Feature Location in the modeling space of the Meta Object Facility

Jaime Font · Lorena Arcega · Øystein Haugen · Carlos Cetina

Received: date / Accepted: date

Abstract Lately, the Model-Driven Engineering community has been paying more attention to the techniques offered by the Search-Based Software Engineering community. However, even though the conformance of models and metamodels is a topic of great interest for the modeling community, the works that address model-related problems through the use of search metaheuristics are not taking full advantage of the strategies for handling nonconforming individuals. The search space can be huge when searching in model artifacts (magnitudes of around 10^{150} for models of 500 elements). By handling the nonconforming individuals, the search space can be drastically reduced. In this work, we present a set of nine generic strategies for handling nonconforming individuals that are ready to be applied to model artifacts. The strategies are independent from the application domain and only include constraints derived from the Meta Object Facility. In addition, we evaluate the strategies with two industrial case studies using an evolutionary algorithm to locate features in models. The results show that the use of the strategies presented can reduce the number of generations needed to reach the solution by 90% of the original value. Generic strategies such as the ones presented in this work could lead to the emergence of more complex fitness functions for searches in models or even new applications for the search metaheuristics in model-related problems.

Keywords Model-Driven Engineering (MDE), Search-Based Software Engineering (SBSE), Feature Location (FL) , Evolutionary Algorithm (EA)

An open-source implementation of the strategies will be made publicly available once the paper has been published

J. Font, L. Arcega, and C. Cetina
SVIT Research Group, Universidad San Jorge, Spain
E-mail: jfont@usj.es, larcega@usj.es, ccetina@usj.es

Ø. Haugen
Department of Information Technology, Østfold University College, Norway.
E-mail: oystein.haugen@hiof.no

1 Introduction

Since the term Search-Based Software Engineering (SBSE) was coined [47] in 2001, it has attracted many research efforts from many different research fields such as testing [9,6], maintenance [48], requirements [48] and Software Product Lines [46]. Part of the success of SBSE resides in the fact that many of the problems present in the field of software engineering can be expressed in a way that can be successfully addressed by existing metaheuristic algorithms, such as evolutionary algorithms. In fact, only three key ingredients are needed to begin: 1) a representation (encoding) of the problem, 2) the definition of a fitness function, and 3) the definition of a set of operators. Then, candidate solutions (which are encoded following the representation chosen) are evolved (by applying the operators) and are evaluated (by the fitness function) in an iterative process until optimal solutions to the problem are found.

Similarly, Model-Driven Engineering (MDE) [53] aims to facilitate the development of complex systems by using models as the main artifacts of the software development process. However, with the widespread application of MDE to larger and more complex systems, new software engineering issues are emerging to support the development, evolution, and maintenance of large models. SBSE techniques are best applied in situations where a large search space is present with a set of conflicting constraints.

This has led to the combination of MDE and SBSE techniques into a new field of study known as Search-Based Model-Driven engineering (SBMDE) [17,54], where Search-Based techniques are applied for MDE related tasks, such as discovering or optimizing models, automatically generating test procedures, maintaining consistency between models and metamodels, etc.

However, when applying SBSE to model artifacts, the search space can grow too large (a model of 500 elements can yield a search space of around 10^{150} individuals [36]), making the search impractical if the search space is not reduced. Another problem that has originated when SBSE techniques are applied to MDE is the generation of models that do not conform to the metamodel. Conformance between the model and its metamodel has been widely studied [73,40] and is required by existing tools [2,82,81].

One solution for reducing the search space while managing the conformance between the metamodel and the models generated by the search metaheuristic is the use of strategies to handle nonconforming individuals. In other words, the conformance between a model and its metamodel can be formulated as a constraint that needs to be guaranteed by the metaheuristic algorithm being applied. Therefore, we will refer to conforming or nonconforming individuals, depending on whether or not the model encoded by the individual conforms to its metamodel. There are several methods proposed in the literature [61,22] to handle these constraints, which belong to different categories:

Penalty functions: The application of penalty functions to the nonconforming individuals that hinders their spread during the evolution. [13,89,52,76]

Strong encoding: The use of a representation for the problem that guarantees (by construction) that all the individuals are conforming individuals [15].

Closed operators: The use of operators that return conforming individuals as output when provided with conforming individuals as input [62].

Repair operators: The use of repair operators that transform nonconforming individuals into conforming ones. [21,67,66]

The application of these strategies will result in a reduction of the search space and fine-grained control over the conformance between the models and the metamodel. However, most of the works in the literature do not apply these strategies to MDE problems [61,22], or they do not encode model artifacts as individuals. This results in a lack of strategies to handle nonconforming individuals when working directly with model artifacts as individuals.

The Meta-Object Facility (MOF) [65] is a specification from the Object Management Group to define a universal metamodel for describing modeling languages. In this paper, we present and compare nine different generic strategies for coping with nonconforming individuals when applying SBSE techniques that encode model artifacts built within the MOF modeling space. Specifically, we present and compare: 1) a set of five different penalty functions; 2) a strong encoding and its associated operations; 3) a set of closed operations; and 4) a set of two repair operators. All of these strategies have been designed to work with MOF models as individuals, and, therefore, are generic in the sense that they do not contain any particularities of the application domain; they only include constraints derived from the definition of MOF models.

In our previous works [38,39] we have successfully applied SBSE techniques to perform Feature Location in Models (FLiM). Feature Location (FL) is one of the most common activities performed by developers during software maintenance and evolution [28] and is known as the process of finding the set of software artifacts that realize a specific feature. We use the FLiM problem as a running example throughout the paper.

We evaluate the different strategies for coping with nonconforming individuals by applying them to perform FLiM on the product models from two industrial domains: BSH, the leading manufacturer of home appliances in Europe; and CAF, a leading company that manufactures railway solutions all over the world. The evaluation is performed using two fitness functions, an optimal fitness based on an oracle and a state-of-the-art fitness based on textual similarity. The results show that these strategies for handling nonconforming individuals can reduce the number of generations needed to reach the solution to 90% of the original value. This can result in gains in performance of more than 20% for some of the metrics analyzed. In addition, we provide a statistical analysis to ensure the significance of the results obtained.

The community that is currently applying SBSE solutions to MDE problems is not taking full advantage of the improvements that the use of strategies such as the one presented in this paper can provide. Therefore, we want to provide evidence of their benefits and contribute to the community with a set of domain-independent strategies that have been evaluated on two industrial case studies of FLiM and can be applied by other researchers when applying SBSE to MDE problems. The strategies can be applied without modification to other FLiM problems whose models are created with any MOF Domain Specific Language expecting similar results. In addition, the encoding pre-

sented has been used in other SBMDE problems as Bug Location [10] and Traceability Link Recovery [71] and we expect similar results when applying the strategies. For SBMDE problems requiring a different encoding (with expressiveness to generate model fragments that are not part of the parent model), modifications of the strategies may be required, and further experimentation is needed to evaluate if the results are similar.

The rest of the paper is structured as follows. Section 2 discusses related work. Section 3 establishes the foundations for the rest of the paper, including the problem of FLiM and the Evolutionary Algorithm that we use to address it, the model and metamodel conformance, and the search space. Section 4 presents the nine generic strategies for handling the nonconforming individuals introduced in this work. Section 5 presents the evaluation performed with two industrial case studies. Section 6 provides a discussion of the results obtained. Section 7 discusses the threats to validity, and Section 8 concludes the paper.

2 Related Work

This section presents works from the literature that are related to the approach presented here. There are some works that apply SBSE strategies to address MDE problems. However, not all of them use models as the individuals; some apply the searches to model transformation rules, while others focus on the improvement of the metamodel through the use of Object Constraint Language (OCL) rules. We also present works about feature location in models. Finally, we discuss works that are related to models in the context of a Software Product Line.

2.1 Model transformation rules

Some works that apply EAs to models use model transformation rules to encode the individuals. Nonconforming individuals are mainly handled through repair operators or death penalties:

The work from [5] applies a Non-dominated Sorting Genetic Algorithm (NSGA) to the problem of rule-based, design-space exploration. The aim is to find the candidates that are reachable from an initial model by applying a sequence of exploration rules. In that work, the authors make use of a custom repair operator that fixes nonconforming individuals. However, their individuals are encoded as sequences of exploration rules, not models themselves, and, therefore, their repair operator is specific to their particular domain. In our work individuals are encoded as model fragments and the repair operators that we propose in this paper can be applied to individuals encoding models from any domain.

In [27], the authors apply search directly to model transformations, without the need for an intermediate representation. The approach proposes the creation of model transformation rules that are capable of performing the tasks associated with an Evolutionary Algorithm (EA), such as the creation of the initial population. The approach is applied to a problem of resource allocation, where the nonconforming individuals are pruned out through the use of one of these model transformation rules. Similarly to our work, the authors apply a death penalty to prune out nonconforming individuals. However, the rule that is used to identify those individuals is specific to their particular domain, and cannot be applied to identify nonconforming model fragments from other domains.

In [33], the authors present MOMoT, a tool that applies SBSE strategies to optimize the set of model transformation rules needed to maximize the requested quality goals of a given model. The approach is further refined in [32] to include support for many objectives and an exhaustive performance comparison of different search strategies is presented in [16]. The tool makes use of three different strategies to handle duplicated or non-executable sets of transformations that could arise when performing genetic operations. The first one is the use of a death penalty, removing those transformations sets. The second one is the replacement of the malformed transformation by a random transformation (or a placeholder transformation) so the set of transformations can be executed. The third strategy is the use of a dedicated re-combination operator (such as the partially matched crossover [43]) that is able to consider some constraints avoiding the creation of non-executable transformation sets. However, the strategies used in those works are designed to work on individuals encoding the order of the transformations, and cannot be applied to repair nonconforming individuals that encode model fragments. Furthermore, the impact of the use of those strategies on the performance of the approach is not evaluated.

In [18], the authors describe strategies for generating closed operators. They use graph transformation rules to encode the mutation operators that are then automatically generated in the form of transformations. These operators guarantee the consistency of the mutated models with the metamodel multiplicity constraints. The resulting operators are similar to the closed operators proposed in this work, but their operators are generated taking into account the multi-

plicities from the metamodel. In this work, we obtain the constraints for the closed operators from the inherent constraints of the metalanguage used to build the metamodels, instead of using the multiplicities of the metamodel. In addition, our operations are designed to work over EA encoding model fragments.

2.2 Metamodel enhancement

Other works try to enhance the metamodel to avoid the generation of models that should not be part of the modeling space for that metamodel. This is usually done through the use of the Object Constraint Language (OCL) rules defined throughout the metamodel.

In [44], the authors propose an approach that helps the modeller find the boundaries of the modeling space of a metamodel. To do this, the approach generates samples of all of the models that can be built with a given metamodel and iterates those samples (through a simulated annealing algorithm) to maximize the coverage of the sample. Then, the sample is presented to experts so that they can fix the metamodel if any of the presented models should not be allowed. By doing this, the gap between the modeling space (all of the models that are reachable from a metamodel) and the intended modeling space (the models that the experts want to be built with the metamodel) can be reduced, and the accuracy of the metamodel can be increased. Similar to our work, their work deals with nonconforming individuals. However, in [44], the undesired individuals are identified by experts and then turned into nonconforming by modifications of the metamodel that was used to create the individuals. In its current form, their approach cannot be applied to handle nonconforming individuals of a running EA.

In [30], the authors take two sets of models (one that includes valid models and another one that includes invalid models) and use an EA to automatically generate well-formedness rules that are derived from the two sets of models provided. As a result, they provide a set of OCL rules that can be used to improve the metamodel into a more precise metamodel + well-formedness rules.

Other works, such as [7], take into account the OCL constraints that are embedded throughout the metamodel and try to generate sets of parameters that fulfill the OCL constraints with the aim of using them as test data. The approach is further refined in [8] to include more types from OCL and heuristics to guide the search. They compare themselves with the most widely used OCL constraint solver, achieving better results. Similarly, the graph solver presented in [79,80] generate consistent models of a designated size from a specification defined by a metamodel and a set of well-formedness constraints. However, these approaches do not solve the problem of handling nonconforming individuals when using search strategies. They do take into account constraints that models should fulfill and use EAs or other generators to help in this task. In contrast, the strategies that we present in this paper are designed to be applied to existing searches in models that are not benefiting from the advantages associated with the proper management of nonconforming individuals.

In [86,87], the authors propose Crepe, a Domain-Specific Language (DSL) that can be used to specify individuals that represent any model conforming to a specific metamodel. Thus, they are able to encode individuals in the form of a model (or model fragments) as we do in our work. In [88], the authors report the generation of nonconforming individuals when applying their encoding for models as individuals, which allowed them to improve the DSL being used. In [57], the authors identify the generation of nonconforming individuals in Crepe, and propose a repair operator to address this issue. After an individual is modified, they use a re-coding operation to repair the individual, preserving the semantics of the model in those aspects not directly affected by the crossover and mutation operations. However, individuals are only partially repaired as the expressiveness of the operators and encoding being used allows for the generation of individuals that cannot be automatically repaired. We believe that approaches such as [88,57] could be improved through the use of the strategies to handle nonconforming individuals presented in this paper.

2.3 Feature Location in Models

Some works from the literature focus only on capturing guidelines and techniques for manual transformations of a set of existing products into assets of a Software Product Line. Those works are interesting because they are based on industrial experiences; however, there is almost no automation in the process.

Other works [85,50,90,91,59,41,36] focus on the location of features in models through comparisons with each other. As a result, the variability is expressed in the form of a model expressing the differences (which is eventually turned into a Software Product Line). These include the following:

- The authors in [90] present a generic approach that is able to perform comparisons of MOF models, resulting in the features being located in the form of a Common Variability Model [83]. The approach in

[90] was further refined in [91] to allow the extension of the model capturing the features, when new models are added to the comparison. This reduces the complexity of the process, avoiding the need for executing all of the comparisons from scratch and allowing them to be performed incrementally.
- Wille et al. [85] present an approach based on an exchangeable metric that is used to measure the similarity between different attributes of the models. The approach in [85] was further refined in [50] to minimize the number of comparisons needed to obtain the model representing the similarities and differences among the different models.
- Martinez et al. [59] propose an extensible approach based on models' comparisons that can obtain the features from a family of related models. The approach can be extended through a system of templates, allowing the identification of differences of any type of model-based content (as long as the comparison method is provided)

However, all of these approaches are based on mechanical comparisons among the models, classifying the elements based on their similarity, identifying the dissimilar elements and formalizing them as the feature realizations. In contrast, in our work the feature location is applied to a single model, so it does not rely on model comparisons to locate the features; instead it relies on searches across the modelling space performed by an EA.

Some of our previous works focus on the topic of feature location in models, ranging from approaches based on comparisons [41] to human-in-the-loop approaches [36] or searches based on metaheuristics [37,38,39,19]. One of them focus on the influence of genetic operations on the quality of the results [72]. Some of them focus on the possibility of sharing the information scattered among different engineers in order to empower them to produce better queries that guide the EA [69,70], while other works explore the use of Multi Objective Evolutionary Algorithms [19,71]. However, none of those works has ever investigated the possibility of handling nonconforming individuals to boost the search process, as is the case in this work.

2.4 Software Product Lines

Finally, some works report problems when nonconforming individuals are automatically generated by their genetic operations in the context of a software product line. In [23], the authors propose a representation of a software product line architecture that can be used by Search-Based techniques. This allows the optimization of the architecture model through the use of different search operations. The authors report the generation of some solutions that are non-consistent with their definition of the product line architecture that are repaired by a domain-dependant repair operator.

In [78], the authors present ETHOM, an EA that is capable of generating computationally-hard feature models in order to use them to feed analysis tools for feature models. To this end, the EA encodes feature models as a combination of a tree and the related cross-tree constraints. Since the use of this encoding leads to the generation of nonconforming individuals, the authors use a repair operator or discard the individual (death penalty), depending on the complexity of repairing the individual. However, since the encoding used by the authors is particular to their specific domain (their representation of feature models), the repair operator proposed is also particular to their domain and captures inconsistencies that only occur in their representation of feature models. In contrast, the strategies presented in our work are designed to work with models created with any Domain Specific Language created using the Meta Object Facility [65] metalanguage, improving its re-usability by different practitioners whose Domain Specific Languages are created using MOF.

3 Overview of the Problem

This section provides the foundation for the rest of the paper. It describes the following: 1) what Feature Location in Models is; 2) how it is achieved through an evolutionary algorithm; 3) what model and metamodel conformance is and what makes an individual nonconforming; 4) what the search space is when searching for model fragments and what it looks like.

3.1 Feature Location in Models (FLiM)

Feature Location [75,28] is the process of identifying the set of software artifacts that realize a specific feature. That is, Feature Location requires to find and indicate all the software artifacts that are used for the design, development and further maintenance of a specific feature (such as requirements, source code, documentation, or tests). Depending on the nature of the software artifacts and the features being located, a different granularity may be applied; for instance, when features are located across source code, a feature could correspond to a single class, a set of methods from different classes, some conditions inside a switch statement, or even a whole package.

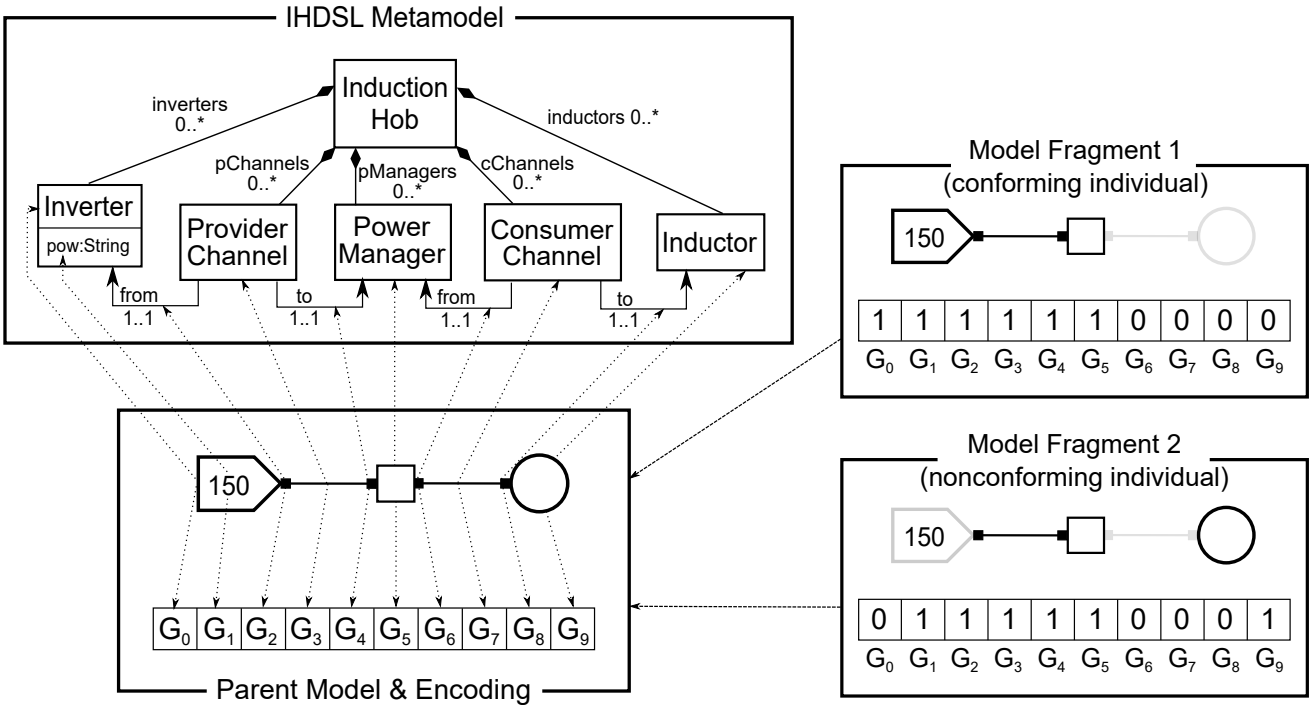

Fig. 1 Running example including the Induction Hob Domain-Specific Language (IHDSL) metamodel (top-left), the encoding of a parent model and its mapping to the metamodel (bottom-left), and two model fragments encoded as individuals, one that is conforming and one that is nonconforming (right)

We define the Feature Location in Models (FLiM) as the process of identifying the set of model elements that realize a specific feature. The results of the FLiM process are model fragments that represent a specific feature. At this point, it is important to define what a model fragment [38,37,39] is: A model fragment is always defined in reference to a parent model. A model fragment is a subset of the elements of the parent model. Therefore, all of the model fragments of a given parent model are subsets of the parent model.

Similarly to other software artifacts, the granularity can vary depending on the nature of the models and the features being located. Taking into account the MOF specification from the Object Management Group (OMG) and our experience with models from industrial domains [39,40,38], we divide the relevant elements of a model into a set of atomic elements (meta-class, meta-reference, and meta-property), and we do not consider further subdivisions of those units in this work.

To illustrate the elements, Fig. 1 (top-left) shows the Induction Hob Domain-Specific Language (IHDSL) metamodel, which is a simplification[1] of the DSL used by one of our industrial partners. The DSL is used to model the firmware of the Induction Hobs in the context of a Model-Based software product line, where some of the features are reused by different products. In the following we explain the concepts of meta-class, meta-reference and meta-property.

Meta-class is the core element, holds a set of meta-properties and meta-references, e.g., the *Inductor* meta-class element from the metamodel in Fig. 1 (top-left).

Meta-reference relates two meta-class elements and includes a source and a target meta-class element, a multiplicity for the target and the source meta-classes, and a name. Meta-references can also be distinguished by whether or not they are containment meta-references. For instance, the *inductors* meta-reference from the metamodel in Fig. 1 (top-left) is a containment meta-reference whose source is the Induction Hob meta-class (multiplicity 1) and whose target is the *Inductor* meta-class (multiplicity any), while the *from* meta-reference is a meta-reference (non-containment) whose source is the *Provider Channel* meta-class (multiplicity 1) and whose target is the *Inverter* meta-class (multiplicity 1).

Meta-property gives information about a meta-class, including the meta-property name, the type, and the value. For instance, the *Inverter* meta-class element from the metamodel in Fig. 1 (top-left) contains a meta-property named *pow* whose type is a *String*.

[1] We use a simplification as running example in order to increase legibility and due to intellectual property rights.

Based on this division, a model fragment is a subset of the model elements that are present in the parent model, with the granularity of the elements being meta-classes, meta-references, or meta-properties.

3.2 Feature Location in Models by an Evolutionary Algorithm (FLiMEA)

Fig. 2 depicts an activity diagram for a generic EA. First, a set of individuals is obtained (following a previously designed specific representation) to be the initial population of solution candidates for the EA. Then, a fitness function is designed to assess the quality of individuals as solutions to the problem. If the stop condition is met, the execution ends; otherwise, a set of operators that is compatible with the representation and capable of evolving the individuals is executed to evolve the population. The following subsections present each of the EA parts in detail.

3.2.1 Representation of the individuals

Traditionally, the representation used in EAs comes in the form of binary strings [74]. For this EA, the individuals encode model fragments that are defined in the context of a parent model. Therefore, the representation needs to be able to capture any model fragment that can be generated from a given parent model. We use a binary string where each bit in the sequence represents the presence or absence of one specific element of the candidate solution.

In our case, the different elements that may or may not be part of an individual are the ones defined in the previous subsection (class, reference, and property). Each of the elements present in the parent model will be "tagged" with a position in the binary string, and then the binary string will be filled with either 0 (to indicate the absence of that element in the model fragment) or 1 (to indicate the presence of that element in the model fragment).

Fig. 1 (bottom-left) shows a parent model of an example Induction Hob that contains one of each of the elements defined by the metamodel and the encoding associated to it. All of the individuals encoded in reference to this parent model will use a string of the same length, one gene for each of the elements[2] present in the parent model (G_0 to G_9, up to a total of ten elements that may or may not be part of model fragments of this parent model). It is important to note that all of the elements (classes, references, and properties) that are present in the parent model need to be present in the encoded binary string so that we are able to represent any possible model fragment using it.

Fig. 1 (right) shows two individuals that are encoded in reference to this parent model (Model Fragment 1 and Model Fragment 2). Below each individual, a string with ten genes where the presence or absence of each element can be indicated is depicted. For instance, Model Fragment 1 is composed of an *Inverter* class element (G_0), a *Provider Channel* class element (G_3), a *Power Manager* class element (G_5), a *from* reference element (G_2), a *to* reference element (G_4), and a *pow* property element (G_1).

The function $value(mf, i)$ is used to retrieve the value of a gene of a given model fragment (mf) and a given gene index i. For instance, the $value(MF1, 2)$ is 1 while $value(MF2, 0)$ is 0 (Fig. 1).

It is important to note that the presented encoding can be applied without changes to any MOF-compliant metamodel since it is expressed at the level of the building blocks of MOF metamodels. No domain-dependant information is embedded into the encoding (although it is presented in the context of a specific Induction Hob metamodel).

3.2.2 Fitness Function

The fitness function is used to evaluate how good each of the individuals is as a solution to the problem. In the past [38,39,34], we have successfully applied fitness functions based on textual similarity between a feature definition and a model fragment. However, we identified some issues that influenced negatively the value of the fitness when using textual similarity as the fitness for FLiM [19]: (i) some feature descriptions may be incomplete (guiding the search to an incomplete model fragment), (ii) there may be vocabulary mismatch (the specific concepts defining the feature are different from those present in the feature, even though both represent the same concept), and (iii) there may be some concepts related to the domain that are not present in the model fragments but that are present in the model transformation rules applied afterwards.

Therefore, in this work we perform the evaluation applying two different fitness functions. First, to isolate the effect of the strategies used to handle nonconforming individuals from the fitness function chosen, we make use of an optimal fitness function. Secondly, to study the effect of the strategies on a realistic scenario we apply a state-of-the-art fitness function based on textual similarity.

The **Optimal Fitness** function is used only for evaluation purposes; it relies on an oracle to guide the

[2] In this figure, the containment references and the root node are omitted for simplicity.

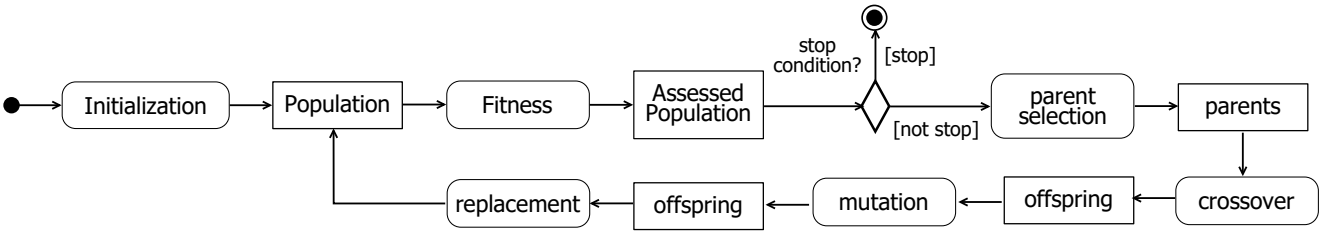

Fig. 2 Activity Diagram for the Evolutionary Algorithm

search (and the existence of an oracle is not always the case on real scenarios). An oracle (or golden set) is a set of problems and the solutions to those problems. In the case of FLiM, we have two oracles that were extracted from industry that include a set of problems of Feature Location in industrial models. Each of them includes a parent model where the feature should be located and a model fragment that realizes the feature. By using this oracle as the fitness function, we can remove the noise that is produced by fitness functions based on textual similarity, and focus only on the different strategies for handling nonconforming individuals and their impact on the search.

$$fitness(m) = \sum_{i=0}^{n} \frac{g(i)}{n}$$
$$g(i) = \begin{cases} 1 & \text{if } value(m,i) = value(o,i) \\ -1 & \text{otherwise} \end{cases} \quad (1)$$

where: m = any given model fragment
n = size of the model fragment
o = model fragment from the oracle.

Equation (1) shows how to compute the fitness of a model fragment (m). The fitness is the result of adding the $g(i)$ values for all of the genes present in the model fragment (from 0 to n) and dividing the sum by the size of the fragment. The $g(i)$ is 1 if the gene value is the same in the model fragment and in the oracle ($value(m,i) = value(o,i)$) or -1, otherwise.

Fig. 3 shows an example of the fitness calculation for Model Fragment 1 (left). First, the binary string of the individual is compared with the solution that was extracted from the oracle (right). If the value of the gene is the same in both model fragments, $\frac{1}{10}$ (it is divided by ten as there are ten genes) is added to the result. If the value of the gene is not the same in both model fragments, $\frac{1}{10}$ is subtracted from the result. Finally, the resulting $fitness(mf1)$ is equal to $\frac{6}{10}$.

The resulting fitness value of the model fragment ranges from the worst value, -1 (if all of the genes are the opposite of the oracle), to the best value, 1 (if all of the genes are the same as the oracle). In the case of a randomly generated individual, the fitness value should be close to 0 since the probability of correctly guessing a gene is the same as incorrectly guessing it.

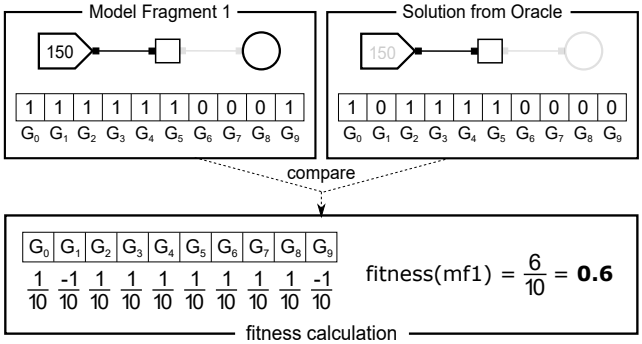

Fig. 3 Example of the Fitness calculation

The **Textual Similarity Fitness** that we apply in this work relies on Latent Semantic Indexing (LSI) [49] to determine how similar is each of the individuals of the population compared to a textual query that describes the feature being located. Before comparing the textual query and the texts obtained from the individuals of the population, texts need to be homogenized through the use of Natural Language Processing techniques [55].

First, the text is tokenized into words, different tokenizers can be applied based on the type of text being processed (i.e. white space for regular text, camelCase or underscore for source code). Secondly, the Parts-Of-Speech (POS) tagging technique can be applied to identify the grammatical role of each word, allowing to filter out some categories that do not contain relevant information and may introduce noise in the search process (i.e. prepositions). Thirdly, some words may not contain semantic information when used in particular domains (given their widespread), so they can be automatically removed if a list of stop words or domain terms is provided (e.g., in the induction hob domain, the word 'hob' will appear too many times, being no useful at all). Fourthly, stemming or lemmatization techniques can be applied to reduce the words to its root or lemma, enabling grouping and comparison of terms from the same family (e.g., 'induction' and 'inductors' would be reduced to 'induct').

LSI builds a vector representation of the query and a set of text documents, arranging them as a term-by document co-occurrence matrix. The rows of the matrix include all the terms present across the documents, the columns represent each of the individuals of the population and the query as last column, each cell indicates the number of occurrences of a particular term in an individual (or the query). Then, the matrix is decomposed applying the Singular Value Decomposition [55], resulting in a set of vectors that represents the latent semantic (one vector for each individual of the population and one vector for the query). Then, to compare the vectors we apply the cosine similarity between each of the vectors representing an individual and the vector of the query, resulting in the fitness value of each individual. The values range from -1 (no similarity at all) to 1 (both vectors are the same).

3.2.3 Genetic Operators

There are four basic operators that are generally applied in EAs (as depicted in Fig. 2):

Parent selection: This operator selects the parents that will be used as the basis for the new individuals of the population. In this case, we use the Roulette Wheel Selection operator. This selection strategy assigns a probability of being selected to each individual in the population proportional to their fitness score. As a result, the fittest individuals are selected more often than individuals that are unfitted.

Crossover: The aim of the crossover operator is to combine the genetic material from some individuals into new ones. In our case, we use a crossover operation that is based on a mask [38] that combines two parent individuals into two new offspring individuals.

Mutation: The aim of the mutation operator is to emulate the spontaneous mutations that occur in nature. In this case, we use an evenly distributed mutation where each gene of each individual has the same probability of undergoing a mutation.

Replacement: The aim of the replacement operator is to modify the population, adding the new offspring generated by the evolution and replacing some of the old individuals of the population. In this case, we apply widespread replacement of the least fit individuals by the new offspring.

3.3 Model and Metamodel conformance

A model conforms to a metamodel if it is expressed by the terms that are encoded in the metamodel. In other words, the metamodel specifies all of the concepts used by the model, and the model uses those concepts following the rules specified by the metamodel. Conformance between a model and the metamodel can be described as a set of constraints between the two [68, 73]. For example, one of the constraints could be that all multiplicities specified in references must be fulfilled.

In addition, as presented in [31], current metamodel techniques tend to define the metamodel as having two parts: a domain structure that captures the context and relationships used to build the models (typically expressed as class diagrams), and well-formedness rules that impose further constraints that must be satisfied by the models (typically expressed as logical formulas). In this work, we focus on the constraints imposed by the domain structure. The additional constraints imposed by the well-formedness rules are out of the scope of this work, and some works on the topic are available elsewhere [7, 8].

MDE is built around the concept of modeling, and several tasks can be automated through the use of models and specific tools (e.g., the generation of graphical editors and tools [81, 2] or model-to-text transformation [25]). However, those tools implicitly require that models conform to a metamodel in order to be used. Model and metamodel conformance is a topic that is widely studied in the field of software evolution [40, 73].

Model and metamodel conformance is a desired property of models, which is implicitly required by the MDE tools and approaches. However, when working with model fragments, the constraints that ensure conformance are not so clear (as we are not dealing with whole models, but only with model fragments). In this work, we explore nine strategies that are built around the conformance of model fragments and metamodels in order to boost the search process of model fragments that realize a specific feature (subsets of a parent model that conforms to a metamodel).

In what follows, we will work with a conformance between a model fragment and the metamodel where some constraints should be preserved:

Valid reference: A reference is considered to be valid if both the source and target model classes pointed by the reference are present in the model fragment. For instance, Fig. 1 shows Model Fragment 1, where the reference encoded by G2 is valid (the source of G2 is G3 and the target is G1, and both are present in the model fragment). In contrast, in Model Fragment 2, the reference encoded by G2 is not considered to be valid (the source of the reference is G0, which is not present in the model fragment).

Valid property: A property is considered to be valid if its parent class is present in the model fragment.

For instance, Fig. 1 shows Model Fragment 1, where the property encoded by G1 is valid (the parent class, G0, is present in the model fragment). In contrast, in Model Fragment 2, the property encoded by G1 is not valid (the parent class G0 is not present in the model fragment).

With these conformance constraints, model fragments can be classified into conforming individuals if they fulfill the constraints for all of the elements present in the model fragment (Model Fragment 1) or into nonconforming individuals, where any of the constraints are violated by any of the elements present in the model fragment (Model Fragment 2).

3.4 Search Space

The search space is the space where the EA performs the search, i.e., the set of all possible individuals that an EA is able to reach by applying the different operations. Depending on the encoding and operations being used by the EA, different search spaces will result.

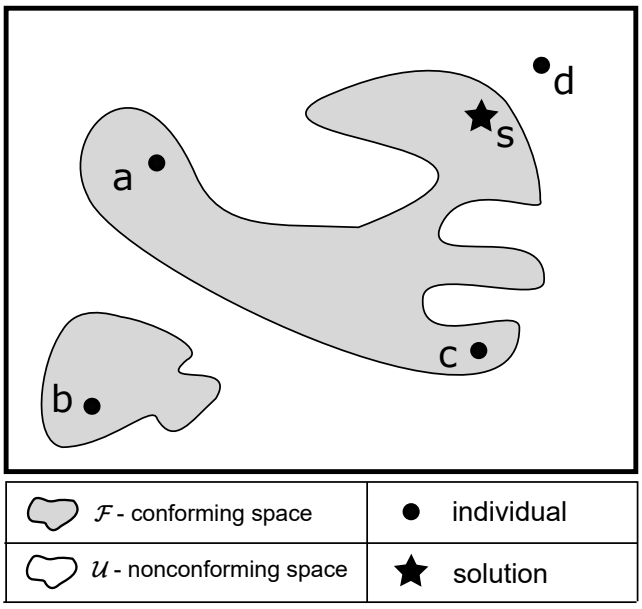

Fig. 4 Example of a search space representation that includes the conforming and nonconforming spaces

In general, a search space consists of two disjoint subsets of *feasible* and *unfeasible* subspaces ($\mathcal{F}$ and $\mathcal{U}$, respectively) [60]. In this work, we use the term *conforming* subspace instead of the term *feasible* subspace and *nonconforming* subspace instead of *unfeasible* subspace. We make this distinction in order to focus on the conformance between models and metamodels since it is what determines if an individual resides in one subspace or in the other. The individuals in the $\mathcal{F}$ subspace satisfy the constraints for the problem (conforming model fragments), while the individuals in the $\mathcal{U}$ subspace do not satisfy the constraints (nonconforming model fragments).

Fig. 4 shows a representation of an example of a search space. The grey areas correspond to the conforming subspace, and white areas correspond to the nonconforming subspace. Each point corresponds to a specific individual, while the star corresponds to the solution of the problem, which is the individual that gets the best fitness value. When applying a Multi Objective Evolutionary Algorithm such as NSGA-II [26], the search is guided by a fitness with multiple objectives and the solution is output as non-dominated set of solutions where all the objectives are optimal. In this work we apply a single objective fitness function, so we only depict a solution in Fig. 4, but we plan to study the application of these strategies in combination with Multi-Objective Evolutionary Algorithms in the future.

For instance, the individual tagged with an 'a' is a conforming individual, such as the one depicted in Model Fragment 1. The point tagged with a 'd' is a nonconforming individual, such as the one depicted in Model Fragment 2. The fittest individual that fulfills the constraints is considered the best-solution to the problem and resides in the conforming subspace. The fittest individual in Fig. 4 is depicted by the star tagged with an 's'.

In the case of SBSE applied to MDE, we want the EA to produce a conforming individual as a solution to the problem. Nevertheless, exploring nonconforming individuals could also lead to the solution faster and benefit the search. Therefore, we will study different methods to cope with nonconforming individuals. The next section presents our strategies for handling nonconforming individuals and how they can be applied to individuals encoding MOF models.

4 Handling nonconforming individuals in SBSE encoding model artifacts

This section presents the main strategies that are available in the literature for handling nonconforming individuals and how they can be applied to work when individuals encode model fragments. The main strategies studied are penalty functions, strong representations, closed operators, and repair operators.

4.1 Penalty functions

Penalty functions [13,89,52,76] are functions applied to nonconforming individuals that are designed to hinder their spread during the evolution. There are different variants of the penalty function method, ranging from static penalty functions and dynamic penalty functions, to the death penalty function, which is the most extreme one.

4.1.1 Static penalty

A static penalty applies a reduction to the fitness value of nonconforming individuals. In static penalties, the value can be a static constant or it can be proportional to the degree of violation of the constraints. To apply a static penalty in EAs, we need to identify nonconforming individuals and then modify their fitness value by subtracting the penalty value. This is done as an extra step after calculating the fitness value of the individuals.

Equation 2 shows the definition of *sta*, which is a static penalty function that applies a constant penalty value (λ_s) to the fitness of an individual (I) if it is a nonconforming individual. The value of the penalty applied (λ_s) needs to be adjusted depending on the domain.

$$sta(I) = \begin{cases} fitness(I) & \text{if } I \in \mathcal{F} \\ fitness(I) - \lambda_s & \text{if } I \in \mathcal{U} \end{cases} \quad (2)$$

Equation 3 shows the definition of *staDeg*, which is a static penalty function that applies a penalty value (λ_{sd}) to the fitness of a nonconforming individual proportional to the degree of violation of the constraints ($deg(I)$) of the given individual. The degree of violation of an individual ($deg(I)$) is calculated as the sum of the violation degree of each gene ($vio(i)$), where all violations of a constraint are weighted the same. A gene that is not violating any constraint is not taken into account for the calculations.

$$staDeg(I) = \begin{cases} fitness(I) & \text{if } I \in \mathcal{F} \\ fitness(I) - \lambda_{sd} * deg(I) & \text{if } I \in \mathcal{U} \end{cases} \quad (3)$$

$$deg(I) = \sum_{i=0}^{n} vio(i) \ where:$$

$$vio(i) = \begin{cases} 1 & \text{if } G_i \text{ is a property missing parent} \\ 1 & \text{if } G_i \text{ is a reference missing source} \\ 1 & \text{if } G_i \text{ is a reference missing target} \\ 2 & \text{if } G_i \text{ is a reference missing source and target} \\ 0 & \text{otherwise} \end{cases}$$

Static penalty functions can be easily applied to EAs that are used to find model fragments, with the trickiest parts being the adjustment of the constant (λ_{sd}) and the selection of the method used to assess the degree of violation of the constraints ($deg(I)$). As part of this work, we try different values and use the ones that provide the best results.

4.1.2 Dynamic penalty

Dynamic penalty functions are similar to static penalty functions in that they apply a reduction to the fitness value of nonconforming individuals. The difference with static penalty functions is that the penalty value applied is proportional to the current generation, making it more difficult for nonconforming individuals to survive as the evolution goes on. This penalty is well suited for the problem since we do not want nonconforming individuals as solutions; however, nonconforming individuals can lead to better results early in the process and removing them prematurely can affect the search negatively.

Depending on the representation used for the problem, some of the optimal individuals (those with the highest fitness scores) will be close to the boundaries between the $\mathcal{U}$ and $\mathcal{F}$ subspaces. Therefore, evolving a nonconforming individual into a conforming and optimal individual may be less expensive (in computational costs) than reaching the same optimal conforming individual through the evolution of another conforming individual (e.g., in Fig. 4, evolving 'd' to 's' may be less expensive than evolving 'a' to 's').

Equation 4 shows the definition of *dyn*, which is a dynamic penalty function that applies a penalty value (λd) to the fitness of a nonconforming individual proportional to the number of the current generation (g).

$$dyn(I) = \begin{cases} fitness(I) & \text{if } I \in \mathcal{F} \\ fitness(I) - \lambda_d * g & \text{if } I \in \mathcal{U} \end{cases} \quad (4)$$

Similarly, Equation 5 shows the definition of *dynDeg*, which is a dynamic penalty function that applies a penalty value (λ_{dd}) to the fitness of a nonconforming individual proportional to the degree of violation of the constraints ($deg(I)$) of the given individual and the number of the current generation (g).

$$dynDeg(I) = \begin{cases} fitness(I) & \text{if } I \in \mathcal{F} \\ fitness(I) - \lambda_{dd} * deg(I) * g & \text{if } I \in \mathcal{U} \end{cases} \quad (5)$$

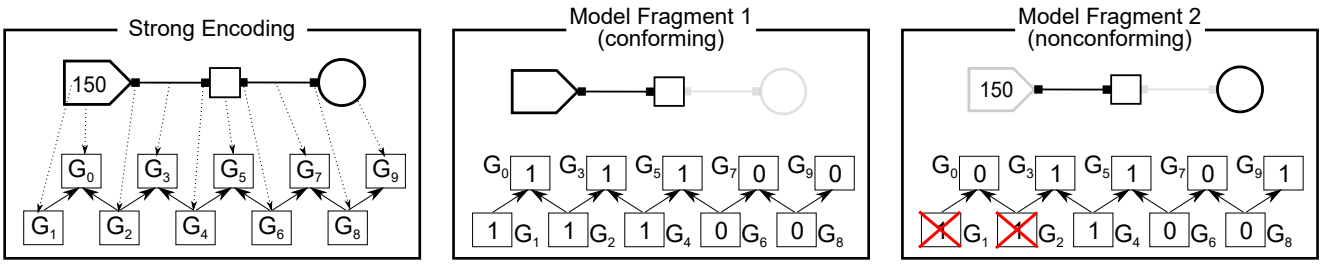

Fig. 5 Example of the Strong Encoding

4.1.3 Death penalty

The death penalty is the most extreme case of penalty. When new offspring are created through the combination of the genetic operators, the individuals are evaluated to check whether they belong to the conforming or the nonconforming subspace. If they belong to the nonconforming subspace, they are removed from the offspring, so they do not end up in the population of the next generation. If they belong to the conforming subspace, the EA continues normally, adding them to the population through the replace operator. When using this strategy, the population will never contain a nonconforming element, guaranteeing that the solution is a conforming individual.

4.2 Strong Encoding

The second strategy for handling nonconforming individuals is the use of a strong representation (or encoding) for the problem. Changing the encoding may also involve a change in some of the genetic operators being applied since the operations are designed to work on a specific representation. The main idea is to devise a strong encoding that guarantees by construction that any individual encoded using this representation is a conforming individual. Having this type of encoding makes the search space change reducing the $\mathcal{U}$ subspace to the empty set, thus simplifying the search space.

This solution has been successfully applied to problems that can be represented as a permutation of a set of values. For instance, the Travelling Salesman Problem poses the next question: Given a set of cities and their distances from each other, what is the shortest path to visits all of the cities? A typically strong encoding to solve this problems is a list that includes all of the cities. Each city appears once in each individual in the order it is visited, ensuring that all of the candidates fulfill the constraint (since all of the cities are visited).

In the case of model fragments some restrictions must be introduced in the encoding to guarantee that all individuals fulfill the constraints (valid references and valid properties) in order to be considered a conforming individual. Our strong encoding consists in introducing a hierarchy of requirements among the genes; that is, some genes require other genes and can only be set to true if the required genes are also true.

Fig. 5 (left) shows an example of our proposed strong encoding for models in EAs that use model fragments as individuals. It shows the encoding for a parent model, including the correspondence between each gene and the model elements (dashed arrows) and the requirements among the genes (regular arrows). For instance, the gene G_0 indicates the presence or absence of the *inverter* class element, while the gene G_1 corresponds to the pow property of the *inverter* class. The gene G_1 requires the gene G_0, so G_1 can only be true if G_0 is also true.

To ensure the 'valid reference' constraint, all of the reference elements require that their source and target class element are present in the model fragment. Therefore, a reference element can only be set to true if both the source and target class elements are also true. For instance, in Fig. 5 (left) the gene G_2 corresponds to the *from* reference of the *provider channel* class element. G_2 can only be true if the source of the reference (G_3) and the target of the reference (G_0) are also true in the model fragment.

To ensure the 'valid property' constraint, all of the property elements require their parent class element. Therefore, a gene representing a property can only be set to true if the parent element is also true.

By doing this, the representation ensures that both constraints are fulfilled by all of the individuals that are encoded using this strong encoding. Therefore, all of the individuals will be conforming individuals and the nonconforming space is reduced to the empty set.

Fig. 5 (center) shows the representation of a conforming individual, Model Fragment 1, encoded following the strong encoding instead of the regular encoding (as in Fig. 1). All of the genes that require other genes (G_1, G_2, G_4, G_6, G_8) can only be set to 1 if the required genes (G_0, G_3, G_5, G_7, G_9) are also set to 1.

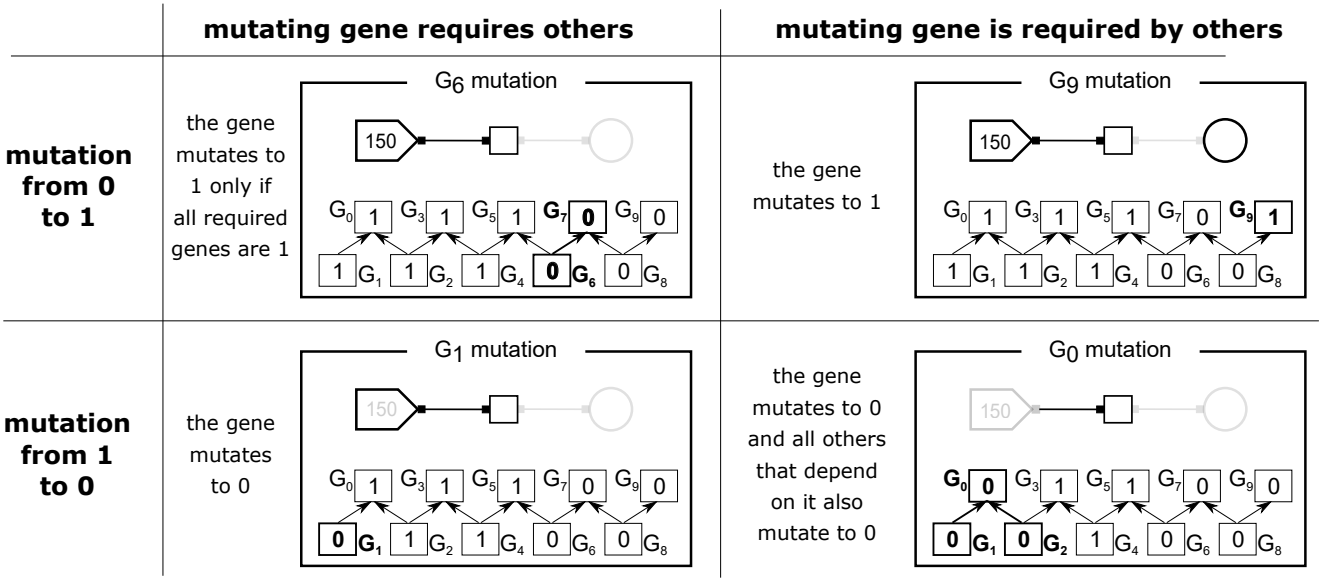

Fig. 6 Example of Mutation operator for Strong Encoding

Fig. 5 (right) shows a wrong and invalid representation of a nonconforming individual, Model Fragment 2 (the same model fragment as in Fig. 1). This is a nonconforming individual and is not allowed by the strong encoding. It is only depicted for clarification purposes (as the encoding will not allow it to exist). Gene G_1 requires G_0 and since G_0 is set to 0, G_1 cannot be set to 1. Similarly, G_2 requires G_0 and cannot be set to 1 either. Model Fragment 2 is nonconforming, so it cannot be built using the strong encoding.

The new strong encoding just introduced also needs genetic operations that are designed to work properly for this representation. The selection and replacement operations used by the regular encoding can also be applied directly to the strong encoding. However, the new strong encoding requires new mutation and crossover operations.

4.2.1 Mutation Operation for Strong Encoding

The **Mutation operation** that is used with the strong encoding is similar to the operation used with the regular encoding. Each gene will have a probability of mutation, changing its value (from 1 to 0 or from 0 to 1). However, the operator will act differently in some cases due to the dependencies. Fig. 6 shows a summary of how the mutation behaves when a gene affected by requirements is going to mutate. It also includes examples of mutations applied to Model Fragment 1.

The first row of Fig. 6 shows the behaviour when the gene that is going to mutate has a value of 0 and is going to mutate to 1. The second row shows the behaviour when the gene mutates from 1 to 0. The first column shows the behaviour when the gene that is going to mutate requires other genes. The second column shows the behaviour when the gene that is going to mutate is required by other genes.

For instance, in a mutation of a gene from 0 to 1 when the mutating gene requires other genes (G_6 mutation), the gene will only mutate if all of the genes required are set to 1 (otherwise, the strong encoding does not allow setting it to 1). Since G_7 is set to 0, the mutation will not take place, and G_6 will remain unchanged with a value of 0.

In a mutation of a gene from 1 to 0 when the mutating gene is required by other genes (G_0 mutation), the gene can mutate from 1 to 0, but then all of the genes that require it also need to mutate to 0 (as the strong encoding requires). Since G_1 and G_2 require G_0, they will also mutate to 0 (if their previous value was already 0, nothing changes).

In a mutation of a gene from 0 to 1 when the mutating gene is required by others genes (G_9 mutation), the mutation does not need any special action, so it proceeds as usual (e.g., mutating gene G_9 from 0 to 1). Similarly, in a mutation of a gene from 1 to 0 when the mutating gene requires other genes (G_1 mutation), the mutation proceeds without further action (e.g., mutating gene G_1 from 1 to 0).

4.2.2 Crossover Operation for Strong Encoding

The **crossover operation** that is used with the strong encoding needs to take into account the hierarchy of the representation. To do this, it follows three steps: 1) generate a random mask; 2) check the validity of the

random mask; 3) generate offspring. Fig. 7 shows this three-step process along with an example.

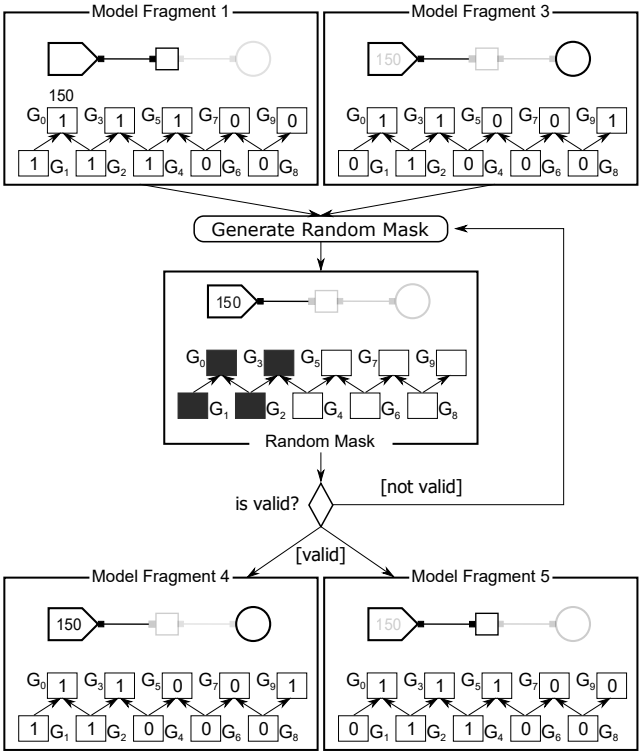

Fig. 7 Example of Crossover operator for Strong Encoding

Generate a Random Mask: The random mask is randomly generated each time a crossover operation is performed. The idea is to divide the set of genes that are present in the representation of an individual into two subsets (G_A and G_B) and then use them to determine which elements come from one parent and which from the other when performing the crossover. First, a random point in the encoding is selected (a random number from 0 to the size of the individual). Then, all of the elements below that index will be the first subset (G_A), while the rest will be the second subset (G_B). Fig. 7 (center) shows an example of a mask. In this case, the randomly selected index is 3, so genes G_0, G_1, G_2, and G_3 are the subset G_A (the encoding is shaded in dark grey); the rest of the genes are the second subset G_B (the encoding is empty and the elements of the individual are faded out).

Check Validity: The next step is to check the validity of the mask. Some masks could lead to nonconforming individuals (which is not possible in the strong encoding), so they cannot be applied. The purpose of this step is to detect those situations and generate a new mask when the current one is not valid. First, the boundaries between the two subsets are identified. In other words, any element from subset S_A that requires or is required by an element from subset S_B is considered a boundary. Each boundary has two parts, a requiring gene and a required gene each of which is in a different subset, S_A or S_B. Then, each boundary is classified into one of the following categories depending on the values of the boundary in each of the parents:

The requiring gene is 0 in both parents: In this case, the value of the required gene does not matter since the requiring gene is not going to be part of any of the two combinations generated as offspring. The mask is not invalidated.

The required gene is 1 in both parents: In this case, the value of the requiring gene does not matter since the required gene will always be part of the two combinations generated as offspring. The mask is not invalidated.

Otherwise: In the rest of the cases, the value of the requiring and required genes is different in each of the parents. This leads to a situation where one of the combinations generated as offspring is nonconforming. The mask is invalidated (making it necessary to generate a new mask)

Generate offspring: Finally, the crossover is applied following the valid mask, and two new individuals are generated. The first individual obtains the value for the genes contained in subset S_A from the Parent 1 and the value for the genes contained in subset S_B from the Parent 2. The second individual is the opposite and takes the values for genes in subset S_A from Parent 2 and the values for genes in subset S_B from Parent 1.

As a result of the crossover operation, two new conforming individuals that inherit genes from both parents are generated. By using these two new operations, the resulting individual will always be in the conforming subspace.

4.3 Closed operators

Another method for coping with nonconforming individuals in EAs is the development of closed operators. Closed operators have their roots in mathematics. Specifically, a set has closure under an operation if the application of that operation to elements of the set always produces an element of the set. For instance, the set $\mathbb{N}$ of positive numbers (some definitions also include 0) has closure under the addition operation (+); the addition of any two numbers from $\mathbb{N}$ will produce a number in $\mathbb{N}$. Or more formally:

	Initial Situation			Add Repair			Remove Repair		
	Source	Reference	Target	Source	Reference	Target	Source	Reference	Target
Missing Source		✓	✓	✓	✓	✓			✓
Missing Target	✓	✓		✓	✓	✓	✓		

Table 1 Repair scenarios and repair operators for the 'valid reference' constraint

	Initial Situation		Add Repair		Remove Repair	
	Property	Parent	Property	Parent	Property	Parent
Missing Parent	✓		✓	✓		

Table 2 Repair scenarios and repair operators for the 'valid property' constraint

$$\forall a, b \in \mathbb{N} \mid a + b = c \Rightarrow c \in \mathbb{N} \quad (6)$$

By extending this concept of closure, we can create operators that guarantee that if the individuals used as input are in the conforming subspace, the resulting individual produced by the operator will also be in the conforming space. Closed operators are similar to the operators used with strong encoding because they also ensure that resulting individuals reside in the conforming subspace. In addition to the definition of closed operations, the EA must be initiated with a set of conforming elements. By doing so, the population will always be part of the conforming space, guaranteeing that the solution will be a conforming individual.

In this work, we use two closed operators, which are adapted from the ones presented for the strong encoding, to apply them directly to the regular encoding. In order to obtain the initial population we generate two types of seeds: 1) the empty model fragment (a model fragment where all of the genes are set to 0), which is a conforming individual since no constraint is violated; and 2) the whole model fragment (a model fragment where all of the genes are set to 1), which is also a conforming individual since all of the constraints are satisfied. The evolution of those individuals (through mutations and crossovers) will eventually lead to the solution model fragment.

4.4 Repair operators

Repair operators [21,67,66] are those capable of turning a nonconforming individual into a conforming one. The repair operator is an operator that is applied after the evolution has taken place (selection, crossover, mutation) but before the individuals are included in the population (replacement).

Repair operators are usually bound to the domain since knowledge about how to repair an individual is needed. However, in this work, we have identified different generic scenarios where the repair operators can be applied. First, when taking into account the *valid reference* constraint, two scenarios may arise: missing Source and missing Target (Table 1). Taking into account the *valid property* constraint, a new scenario may arise: missing Parent (Table 2):

Missing Source: This scenario occurs when the individual includes the reference element and the target element of the reference but not the source element of the reference (See Initial situation of the first row in Table 1).

Missing Target: This scenario occurs when the individual includes the reference element and the source element of the reference but not the target element of the reference (See Initial situation of the second row in Table 1).

Missing Parent: This scenario occurs when a property element is present in the individual, but the parent element of the property is not present (See Initial situation of the first row in Table 2).

To repair these scenarios, we propose two different repair operators based on the addition or removal of elements

4.4.1 Add Repair

Add Repair will be applied to the repair scenarios described above and repair them by adding the missing elements:

Missing Source: The repair operator will add the source element of the reference to the individual (See Add Repair of the first row in Table 1).

Missing Target: The repair operator will add the target element of the reference to the individual (See Add Repair of the second row in Table 1).

Missing Parent: The repair operator will add the parent element of the property to the individual (See Add Repair of the first row in Table 2).

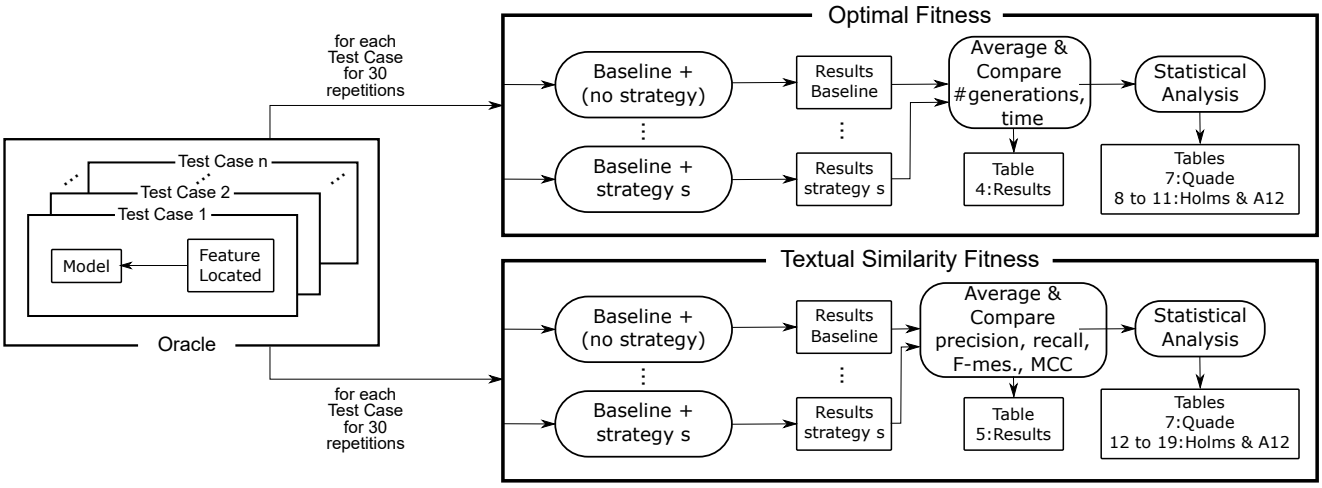

Fig. 8 Overview of the evaluation

4.4.2 Remove Repair

Remove Repair will be applied to the repair scenarios described above and repair them by removing the elements causing the individual to be nonconforming:

Missing Source: The repair operator will remove the reference element of the reference to the individual (See Remove Repair of the first row in Table 1).

Missing Target: The repair operator will remove the reference element of the reference to the individual (See Remove Repair of the second row in Table 1).

Missing Parent: The repair operator will remove the property element of the individual (See Remove Repair of the first row in Table 2).

After applying the operators, the nonconforming individual will turn into a conforming one (either by adding or removing elements). One problem that may arise with the Remove Operator is that it hinders the evolution of the model fragment because the operator does not allow new elements to emerge if they are not valid.

5 Evaluation

This section presents the evaluation performed to address the following research questions.

RQ1: Can the strategies for handling nonconforming individuals presented so far (penalty functions, strong encoding, closed operations or repair operators) improve the results of SBSE on models in terms of the number of generations and/or wall-clock time needed to reach the solution?

RQ2: If so, which strategies produce better results?

RQ3: Can any of the strategies produce solutions of better quality, in terms of precision, recall, F-Measure and MCC, than those produced by the baseline when combined with a state-of-the-art fitness function as the textual similarity fitness presented?

To address these research questions, the following subsections present a description of the experimental setup, the set of performance metrics used, a description of the two case studies where the strategies were applied, the results obtained, and the statistical analysis performed on the results.

5.1 Experimental Setup:

To evaluate the different strategies, we apply them as part of the EA explained in Section 3.2 following the process depicted in Figure 8.

The Oracles (left) contain a set of product models and several features contained in those product models. The oracles were obtained from industry and contain the realization of each feature in the form of a model fragment. In other words, the oracle can be considered a set of 'problems' and the 'answer' to each one. We use it to evaluate the impact of each of the strategies proposed in the search process. Each oracle is organized as a set of test cases where each test case contains a model (where the feature must be located) a feature that is already located, and a feature description (elaborated by the engineers of our industrial partners).

Most of the execution time of an EA is spent on the evaluation of the fitness function. Specifically, in the case of FLiM using a fitness function based on textual similarity [38], we have reported that around 85% of the execution time is spent on the fitness function. Therefore, to evaluate the impact of the search strategies in the search process, we will perform two experiments, using a different fitness function each time. First, to

avoid the impact of the fitness function on the results, we use the optimal fitness function (see Section 3.2.2), which indicates how far from or how close to the solution each of the individuals is. This setup will allow to answer RQ1 and RQ2, although is not possible to apply it to solve real problems (as it needs the existence of an oracle containing the answers to the questions beforehand). Secondly, to answer RQ3 and test the impact of the strategies on a real scenario, we repeat the experiment using a state-of-the-art fitness, the textual similarity fitness function (see Section 3.2.2).

For each test case (n) and each of the strategies (s), we executed 30 independent runs [11] (to avoid the effect of chance due to the stochastic nature of EAs) for each of the experiments. The set of strategies tested are the ones presented in Section 4. The EA with no strategy for handling nonconforming individuals is used as the baseline. The resulting data of the first experiment was averaged and compared in Table 4 and statistically analyzed to guarantee the validity of the results obtained (Tables 7 - 11, available in the Appendix). Similarly, the data obtained from the second experiment was averaged and used to build the confusion matrix of the result of each test case. Then, the performance measures (precision, recall, F-Measure and MCC) were derived from the confusion matrix, compared in Table 5 and statistically analyzed to guarantee the validity of the results obtained (Tables 7 and 12 - 19, available in the Appendix).

5.2 Performance Metrics

To measure the performance of the strategies on the search process we make use of standard metrics from literature, so comparisons among different studies can be performed. In general, there are two types of performance measures that are relevant for EAs: solution quality and search effort. The experiment using the optimal fitness is designed to measure the impact of the strategies on the search effort of the algorithm. To do so, we use the number of generations and the wall clock time. The experiment using the textual similarity fitness is designed to measure the solution quality. To do so, we use a confusion matrix to extract four metrics, precision, recall, F-Measure and Mathew Correlation Coefficient (MCC).

The performance of each of the strategies is directly related to the number of times that the fitness function needs to be executed (i.e., the number of generations). Therefore, for the experiment using the optimal fitness we measure the performance of each strategy as the number of generations needed to find the optimal solution (extracted from the oracle), as suggested in the literature [51]. The fitness function is executed once for each individual in the population for each generation. Using the number of generations as metric allows us to compare the impact of the different strategies and the baseline (no strategy) fairly.

In addition we use the wall clock time as metric to measure the performance of each strategy. However, the time spent by the EA to find the solution does not depend only on the strategy being applied, the computing power of the computer used to run the experiments will influence the results. Similarly, the differences in performance of the implementation of each of the strategies can also introduce noise into the results. Therefore, the number of generations should be used to compare the performance of different strategies and the wall clock time can be used as an indicator on the time needed by each strategy but should not be used to compare performance among strategies.

For the experiment using the textual similarity fitness, the EA will run for a fixed amount of time and then the best candidate obtained so far will be compared to the optimal solution from the oracle. To perform that comparison we make use of a confusion matrix, a table typically used to describe the performance of a classification model (the EA + strategy) on a set of test data (each of the test cases) for which the true values are known (the optimal solution from the oracle). The confusion matrix distinguishes between the predicted values (solution of the EA + strategy) and the real values (optimal solution from oracle) and arranges the elements (each of the genes of each individual) into four categories:

- True Positive (TP): values that are true in the real scenario and have been predicted as true.
- True Negatives (TN): values that are false in the real scenario and have been predicted as false.
- False Positive (FP): values that are false in the real scenario but have been predicted as true.
- False Negative (FN): values that are true in the real scenario but have been predicted as false.

Then, performance metrics can be derived from the confusion matrix, in this experiment we use precision, recall, F-Measure and MCC.

Precision (see Equation 7) measures the number of elements from the solution that are correct according to the optimal solution from the oracle. Precision values can range from 0% (no single element present in the solution is also present in the optimal solution from the oracle) to 100% (all the elements present in the solution are also present in the optimal solution from the oracle).

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

Recall (see Equation 8) measures the number of elements of the optimal solution that have been correctly retrieved in the solution. Recall values range from 0% (none of the elements that are true in the oracle solutions is present in the solution) to 100% (all the elements that are true in the optimal solution are also present in the solution).

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

However, achieving a high value in precision or recall alone is not enough. The empty model fragment (where all the genes are set to false) would achieve 100% in precision (but 0% in recall). Similarly, the complete model fragment (where all the genes are set to true) would achieve 100% recall (but 0% in precision). Therefore, there is a need for overall measures that take into account all the figures present in the confusion matrix.

F-Measure (see Equation 9) is the harmonic mean between precision and recall, and provides a good overview of the overall performance of a strategy. Values can range from 0% (either precision or recall is 0%) to 100% (both, precision and recall are 100%).

$$F\text{-}Measure = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{9}$$

Finally, MCC (see Equation 10) has recently proven to be more informative than F-Measure as metric of the overall performance [20], as it takes into account all the values from the confusion matrix (including the TN, which is not used by the F-Measure). The values range from -1 (worst value possible) to 1 (best value possible).

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}} \tag{10}$$

5.3 Case Studies: BSH & CAF

The present work has been evaluated in two industrial case studies. The first case study used for the evaluation was **BSH**, the leading manufacturer of home appliances in Europe. Their induction division has been producing induction hobs (under the brands of Bosch and Siemens among others) for more than 15 years. The second case study used for the evaluation was **CAF**, a worldwide provider of railway solutions. They have been developing a family of PLC software to control their trains for more than 25 years.

The BSH case study has already been used as a running example throughout the paper. Their newest induction hobs include full cooking surfaces where dynamic heating areas are dynamically generated and activated or deactivated depending on the cookware placed on top of them. In addition, the new hobs have increased the amount and type of feedback provided to the user while cooking, providing data such as the temperature of the food being cooked or real-time measures of the power consumption of the hob. These changes have been made possible by increasing the complexity of the software that drives the induction hob.

The DSL used by our industrial partner to specify the induction hobs is composed of 46 meta-classes, 74 references with each other, and more than 180 properties. The running example presented in 3.2 shows a simplification of their DSL (to increase legibility and due to intellectual property rights concerns).

Their oracle is composed of 46 product models (induction hob), where each product contains (on average) around 500 model elements. The oracle is composed of 96 features that may or may not be part of a specific product model. Those features correspond to products that are currently on the market or will be released to the market in the near future. Each of the 96 features can be part of several product models, making a total of 608 occurrences of any of the features in any of the product models. Therefore, there are 608 test cases, each of which includes the product model where the feature should be located and the model fragment itself that realizes the feature (which is used as fitness).

The CAF case study is based on the family of software products used to manage their trains in different forms (regular train, subway, light rail, monorail,etc.) all over the world. Each train unit is equipped with different pieces of hardware installed on their vehicles and cabins. Those pieces of equipment are often provided by different companies, and their aim is to carry out specific tasks for the train such as traction, compression for the hydraulic brakes, harvesting of power from the overhead wires, etc. The control software is responsible for making the cooperation among all the equipment possible in order to achieve the functionality desired for a specific train and guaranteeing compliance with the specific regulations of each country.

The DSL used to specify the products from CAF has expressiveness to describe the interaction among the equipment pieces. In addition, the DSL also provides expressiveness to specify non-functional aspects that are related to specific regulations (such as the quality of the signals or the different level of redundancies needed).

The CAF oracle is composed of 23 product models (train units), where each product contains (on average) 1200 elements. The products are built from 121

different features that may or may not be part of a specific product model. Again, some features are present in more than one product model, making a total of 140 occurrences. For each occurrence there is a test case that includes the product model and the model fragment that realizes the feature (which is used as fitness).

For the evaluation with the BSH oracle, we performed 608 (test cases) * 30 (repetitions) * 10 (baseline + strategies) * 2 (fitness functions) = 364,800 independent runs. For the evaluation with the CAF oracle, we performed 140 (test cases) * 30 (repetitions) * 10 (baseline + strategies) * 2 (fitness functions) = 84,000 independent runs.

To prepare the oracles, our industrial partners provided us with the product models and the model fragments that were used to build those product models. Therefore, the information about which elements realize each of the features comes directly from industry. For each test case, we had previously checked that the model fragment exists in the provided product model and that there are no inconsistencies (such as the empty model fragment or the complete model fragment).

5.4 Implementation Details

The presented strategies were implemented within the Eclipse environment and the source code has been released to the public [35] as part of this work. We have used the Eclipse Modeling Framework [82] to manipulate the models from our industrial partner. The EA is based on the watchmaker framework [29] for evolutionary computation, creating custom genetic operators and representations to implement the strategies. The IR techniques that were used to process the language were implemented using OpenNLP [3] for the POS-Tagger and the English (Porter2) [4] as stemming algorithm. Finally, the LSI fitness was implemented using the Efficient Java Matrix Library (EJML [1]). We performed the execution of the EA with the strategies using an array of computers with processors ranging from 4 to 8 cores, clock speeds between 2.2 and 4 GHz, and 4–16 GB of RAM. All of them were running Windows 10 Pro N 64 bits as the hosting operative system and the Java SE runtime environment (build 1.8.0_73-b02).

5.5 Parameters and Budget

There are some parameters in EAs that need to be configured prior to running them. We use default parameter values extracted from the literature [12] (and previously tested for this EA [39]) when available. However, the new penalty functions proposed in Section 4.1 also need values for some parameters. To give those values we have performed a tuning to determine which parameters work better for this problem. In other words, we have tried different combinations of parameters to determine the ones that result in a faster search. The parameters that we use are shown in Table 3.

Parameter	Description	Value
size	Size of population	100
p_c	Probability of crossover	0,75
p_m	Probability of mutation, where n is the length of the individual being mutated	1/n
λ_s	constant for static penalty	$2.5x10^{-2}$
λ_{sd}	constant for static penalty with degree of violation	$1.8x10^{-3}$
λ_d	constant for dynamic penalty	$2x10^{-4}$
λ_{dd}	constant for dynamic penalty with degree of violation	$1.0x10^{-5}$

Table 3 Parameters for the evolutionary algorithm and for the penalty strategies

Regarding the stop condition of the EA for the first experiment (optimal fitness), since we want to compare the different strategies against the baseline, we allocate a budget that is larger than three times the number of generations needed by the baseline. If the strategies find the solution in the allocated number of generations, we obtain the number of generations needed and compare it against the baseline; if the strategies do not help the algorithm to find the solution within the allocated number of generations we indicate that in the results table. Since the baseline results were about 6,400 generations for the BSH case study and 9,700 for the CAF case study, we allocated a budget of 30,000 generations.

Regarding the stop condition of the EA for the second experiment (textual similarity fitness), since we want to determine if the use of the strategies has an impact on the solution quality, we allocated a fixed amount of time for each test case (10 seconds for BSH and 20 seconds for CAF), based on the size of the model being explored and the times needed in a pilot test. After that time, we will stop the execution of the EA, get the best candidate obtained so far and compare it against the optimal solution obtained from the oracle using the metrics presented in Section 5.2. This will result in measures of precision, recall, F-Measure and MCC for the EA when using each of the strategies and the baseline.

EA + Optimal Fitness + Strategy	BSH		CAF	
	Generations ± σ	Time ± σ (s.)	Generations ± σ	Time ± σ (s.)
Baseline	6405 ± 2484	0.164 ± 0.063	9759 ± 5248	0.441 ± 0.227
Static Penalty	6740 ± 2671	2.726 ± 1.085	12085 ± 7702	8.540 ± 10.759
Static Degree Penalty	10281 ± 4112	3.893 ± 1.593	11560 ± 6897	10.094 ± 5.397
Dynamic Penalty	9452 ± 5021	3.852 ± 2.115	19591 ± 9121	13.231 ± 5.572
Dynamic Degree Penalty	7945 ± 3651	3.288 ± 1.462	17504 ± 11265	13.393 ± 7.561
Death Penalty	30000* ± 0	13.061 ± 1.292	30000* ± 0	14.023 ± 1.237
Strong Encoding	456 ± 509	0.010 ± 0.012	2448 ± 1267	0.058 ± 0.033
Closed Operations	1011 ± 1034	0.022 ± 0.022	6372 ± 4164	0.225 ± 0.180
Add Repair	29431 ± 2611	13.802 ± 1.281	23453 ± 9201	16.269 ± 6.054
Remove Repair	4049 ± 1936	1.304 ± 0.619	6789 ± 3363	3.620 ± 1.616

Table 4 Results of the Optimal Fitness for BSH and CAF, including the number of generations and the wall-clock time for each strategy and the baseline.

EA + Textual Similarity Fitness + Strategy	BSH				CAF			
	Precision ± σ	Recall ± σ	F-Meas. ± σ	MCC ± σ	Precision ± σ	Recall ± σ	F-Meas. ± σ	MCC ± σ
Baseline	33.6 ± 28.7	58.5 ± 28.0	41.2 ± 26.3	0.38 ± 0.28	36.0 ± 22.2	58.1 ± 20.9	39.2 ± 20.0	0.35 ± 0.28
Static Penalty	20.1 ± 20.3	55.7 ± 26.6	28.0 ± 19.7	0.26 ± 0.20	22.4 ± 15.5	53.1 ± 22.7	26.6 ± 15.3	0.19 ± 0.27
Static Degree Penalty	14.6 ± 16.3	55.8 ± 26.0	21.8 ± 17.7	0.20 ± 0.18	19.8 ± 13.3	58.1 ± 24.4	25.3 ± 14.9	0.15 ± 0.32
Dynamic Penalty	11.6 ± 12.8	54.9 ± 24.0	18.0 ± 14.2	0.16 ± 0.14	16.6 ± 9.6	50.2 ± 21.0	22.1 ± 11.3	0.13 ± 0.24
Dynamic Degree Penalty	13.5 ± 12.9	59.0 ± 24.9	20.7 ± 15.7	0.20 ± 0.16	25.9 ± 18.4	57.4 ± 23.5	30.5 ± 17.7	0.23 ± 0.30
Death Penalty	4.3 ± 4.5	99.9 ± 0.4	7.9 ± 7.9	0.01 ± 0.02	6.0 ± 4.0	90.8 ± 4.7	10.9 ± 6.9	-0.01 ± 0.24
Strong Encoding	48.8 ± 27.6	93.7 ± 19.9	61.6 ± 23.1	0.62 ± 0.23	53.8 ± 24.7	84.9 ± 16.4	61.2 ± 21.0	0.61 ± 0.24
Closed Operations	51.1 ± 30.9	85.6 ± 26.7	58.8 ± 26.3	0.60 ± 0.27	50.4 ± 28.7	78.0 ± 21.0	55.1 ±2 4.6	0.53 ± 0.29
Add Repair	6.7 ± 7.2	65.9 ± 19.6	11.4 ± 10.7	0.10 ± 0.08	7.1 ± 4.2	67.9 ± 17.0	12.3 ± 6.6	-0.05 ± 0.31
Remove Repair	38.5 ± 21.1	22.8 ± 9.5	25.2 ± 10.8	0.25 ± 0.12	38.2 ± 18.4	29.2 ± 11.7	29.3 ± 10.7	0.26 ± 0.18

Table 5 Results of the Textual Similarity Fitness for BSH and CAF, including the Precision, Recall, F-Measure and MCC achieved by each strategy and the baseline.

5.6 Results

Table 4 shows the results of the application of the different strategies presented (and the baseline) to the two case studies presented, using the optimal fitness. For each of the strategies (rows), the table shows the average number of generations needed to find the solution model fragment (and the standard deviation $\pm\sigma$) and the mean time in seconds needed to locate each test case (and the standard deviation $\pm\sigma$). The first two columns show the averaged results for the 608 test cases from BSH, and the next two columns shows the averaged results for the 140 test cases from CAF. The first row shows the results for the baseline, without applying any strategy; the second to sixth rows show the different penalty functions presented; the seventh row shows the results for strong encoding; the eighth row shows the results for closed operations; and the last two rows show the results for the repair strategies. The strategies that needed fewer generations and less time than the baseline to find the solution are highlighted in the table.

For BSH, the baseline was 6,405 generations and 0.164 s, so the results for strong encoding (456 generations and 0.010 s), closed operations (1,011 generations and 0.022 s) were below the baseline for both metrics. For CAF the baseline took 9,759 generations and 0.441 s to find the solution, while the strong encoding (2,448 generations and 0.058 s) and the closed operations (6,372 generations and 0.225 s) were able to find the solution in fewer generations and less time than the baseline. When applying the remove repair, the number of generations needed to reach the solution in the BSH case study (4,049 generations) was fewer than the number of generations needed by the baseline, but the mean time needed by the remove repair (1.304 s) was bigger than the time needed by the baseline. The same happens when applying the remove repair operator to the CAF case study. When applying the death penalty, the EA was unable to find the solution in the number of generations allocated (30,000), so it is marked with an asterisk (*). Thus, in answer to RQ1, there are strategies that are capable of helping the EA to find the solution in fewer generations and less time than the baseline.

Table 5 shows the results of the application of the different strategies (and the baseline) to the two case studies presented, using the textual similarity fitness. For each of the strategies (rows), the table shows the mean precision values, the mean recall values, the mean F-Measure values, and the mean MCC values obtained. All the metrics are presented along with its standard deviation ($\pm\sigma$). The first four columns show the averaged results for the 608 test cases from BSH, and the next four columns shows the averaged results for the 140 test cases from CAF. The first row shows the results for the baseline, without applying any strategy; the second to sixth rows show the different penalty functions presented; the seventh row shows the results for strong encoding; the eighth row shows the results for closed operations; and the last two rows show the results for the repair strategies. The strategies that obtained better values than the baseline on the four metrics are highlighted in the table.

For BSH, the baseline achieved an average value of 33.6% in precision, 58.5% in recall, 41.2% in F-Measure and 0.38 in MCC, so the results for strong encoding (48.8% in precision, 93.7% in recall, 61.6% in F-Measure and 0.62 in MCC) and closed operations (51.1% in precision, 85.6% in recall, 58.8% in F-Measure and 0.60 in MCC) were above the baseline for the four metrics. For CAF, the baseline achieved a value of 36.0% in precision, 58.1% in recall, 39.2 in F-Measure and 0.35 in MCC, while the results for the strong encoding (53.8% in precision, 84.9% in recall, 61.2% in F-Measure and 0.61 in MCC), and the results for the closed operations (50.4% in precision, 78.0% in recall, 55.1 in F-Measure and 0.53 in MCC) were above the baseline. When applying the death penalty, the results in recall were 99.9% for BSH and 90.8 for CAF (way beyond the results achieved by the baseline), but the results in precision (4.3% for BSH and 6.0% in CAF) were too low when compared to the values obtained by the baseline, resulting in a worse value of the more general metrics (F-Measure and MCC). Thus, in answer to RQ3, there are strategies that are capable of helping the EA to find the solution when applied in combination to a state-of-the-art fitness function, outperforming the baseline in terms of precision, recall, F-Measure and MCC.

The values for standard deviations achieved by the different metrics are due to the differences in size and complexity of each test case. Bigger test cases require more generations to be solved, while smaller test cases require less generations (and thus time). Similarly, the values of precision. recall, F-Measure and MCC that can be achieved in a fixed time vary depending on the size of the model being explored and the size of the solution [14]. When the experiment is performed, the execution of each combination of strategy and test case is performed 30 times (as suggested in literature [11]) to mitigate the stochastic nature of the EA and ensure that the result is not due to mere chance. Those 30 values are averaged and the standard deviation in that case was below 1% for all the test cases, showing the robustness of the search and ensuring that solutions of similar quality are produced each time that the search is performed.

5.7 Statistical Analysis

To compare the results from the different strategies, all of the data resulting from the runs of the algorithms were analyzed following the statistical methods and guidelines described in [11].

To provide an answer for RQ2, we performed a statistical analysis to: 1) provide formal and quantitative evidence (statistical significance) that the different strategies have an impact on the search (ensuring that the differences in results are not obtained by mere chance); and 2) show that those differences are significant in practice (effect size). The analysis is performed for each experiment separately.

5.7.1 Statistical Significance

First, all of the strategies should be run a large enough number of times (30 independent runs) to collect information about the probability distribution for each strategy. Then, a statistical test is run to assess where there is enough empirical evidence to claim (with a high enough level of confidence) that there are differences among the results of the strategies (and thus be able to claim that one strategy is better than another). To achieve this, two hypothesis are defined. H_0: is the null hypothesis, stating that there is no difference among the strategies; H_1 is the alternative hypothesis, stating that at least one strategy differs from another. Finally, a statistical test is run to determine whether or not the null hypothesis (H_0) can be rejected.

A statistical test returns a probability value ($p - value$) that ranges between 0 and 1. The lower the $p - value$ the higher the probability of the null hypothesis being false (and, therefore, there are differences among the strategies). In this field of study, a $p - value$ under 0.05 is considered to be statistically significant [11], enabling the null hypothesis to be considered false.

The statistical test used to determine this significance depends on the properties of the data. The data obtained in this evaluation does not follow a normal distribution, which requires the use of non-parametric

tests. Of the non-parametric tests available, we applied the Quade test, which has shown to be more powerful than the rest when working with real data [42].

Table 7 shows the results for the Quade tests applied to the result sets of the experiments. The Quade test is applied to the result sets of each experiment, metric and case study separately. The $p - value$ for each metric and case study is smaller than 0.05, which is considered to be statistically significant [11]. Therefore, we can conclude that there are statistically significant differences among the results of, at least, a pair of strategies. However, the Quade test is not able to answer the question: *Which strategy gives the best performance (in terms of the metrics analyzed in each experiment)?* To answer that question, the results from each strategy should be pairwise compared, determining whether or not there are statistically significant differences among the strategies. Therefore, we applied an additional post hoc analysis after the Quade test that performed these pairwise comparisons. In this evaluation, we applied the Holm's post hoc procedure, as suggested by [42].

Values above the diagonal of Tables 8-19 (Available on the Appendix) show the Holm's post hoc results for each metric and case study. Each cell shows the $p - value$ obtained when comparing the pair of strategies from the row and the column using the Holm's method. Again, values below 0.05 are considered to be statistically significant. In all tables, the values that are over 0.05 have been highlighted. For example, in Table 8, the first column, Static Penalty, first row, Baseline, shows a value of 0.066, indicating that the differences between the results of the two strategies in terms of number of generations do not differ enough to be considered statistically significant. In contrast, the differences between the remove repair and all of the other strategies (the last column) are always below 0.05.

For the optimal fitness (Tables 8-11), most of the results are significant when compared pairwise. The differences in number of generations between the static penalty and the baseline are not significant enough. In fact, if we compare the results in Table 4, the difference between the two is minimal (6,405 generations for the baseline versus 6740 generations for the static penalty). The effect of the static penalty strategy is not big enough to be noticeable. Similarly, the difference in number of generations between the death penalty and the add repair for the BSH case study (Table 8 death penalty row, add repair column) also provides a value (0.701) over the threshold, indicating that the differences are minimal (Table 4 shows that the difference is low). For the CAF case study, we confirm that the differences between the baseline and the static penalty are not significant (Table 9). In addition, the static degree penalty does not have significant differences with the baseline or with the static penalty, either. In this case, the differences between the closed operations and the remove repair are also not significant, which differs from the BSH case study. When comparing the time needed to reach the solution, the differences between Dynamic Penalty and Dynamic Degree Penalty are not significant enough for the BSH case study. For the CAF case study the differences both dynamic penalties and the Death Penalty is not significant either. This indicates that the strategies behaved differently in each of the case studies and, therefore, all of them could be relevant for a specific case study (depending on the nature of the models).

For the textual similarity fitness (Tables 12-19), we can observe that the differences between the baseline and the static and dynamic penalties are not significant for some of the metrics and case studies (such as the precision in BSH and the recall in both case studies). This effect is spread also to the more general metrics of F-Measure and MCC. Similarly, the remove strategy does not provide significant differences when compared to other strategies in terms of precision, F-Measure and MCC for the CAF base study. This is due to the low differences achieved by those strategies on the performance metrics (Table 5).

5.7.2 Effect Size

After we have determined that there are differences among the results of the strategies (using the Quade and Holm's analysis), we need to determine how big those differences are. Even after obtaining statistically significant differences, they can be too small and have no practical value [11] (especially when dealing with a large enough number of runs). Therefore, it is important to assess the magnitude of the difference, using an effect size analysis.

For a non-parametric effect-size measure [45], we used Vargha and Delaney's $\hat{A}_{12}$ [84]. $\hat{A}_{12}$ is applied to two groups of data (e.g., the results of two strategies S_1 and S_2) and is related to the probability that an observation in one group will be greater than an observation in the other group. In other words, a $\hat{A}_{12} = 0.5$ indicates that the two strategies are equivalent and will need a similar number of generations (or any other metric being compared) in any case. However, $\hat{A}_{12} = 0.7$ would mean that the number of generations needed by S_1 will be higher than the number of generations needed by s_2 70% of the times (similarly, we can state that S_2 will need less generations 30% of the times). When comparing number of generations or wall clock time, the lower the value the better. However, When comparing preci-

sion, recall, F-Measure or MCC, the greater the value the better. Therefore, a $\hat{A}_{12} = 0.2$ applied to the precision values of two strategies indicates that the first strategy will achieve a greater value of precision than the second strategy 20% of the times (so first strategy provides better performance than the second strategy 20% of the times).

Values below the diagonal of Tables 8-19 (Available on the Appendix) show the results of the $\hat{A}_{12}$ for each metric and case study. The values of each cell indicate the number of times (in percentage) that the strategy in that row will yield a higher value than the strategy in the column for the metric being analyzed. For example, in Table 8 the last row (Remove Repair), second column (Static Penalty) shows 19.71%, indicating that remove repair needs more generations than static penalty around 20% of the times. Values of the strategies that perform best than the baseline has been highlighted in the tables, so a quick overview allows to observe the essence of the results.

For the optimal fitness (Tables 8-11, a low value indicates that the strategy from the row will outperform the strategy of the column. For instance, the results for strong encoding and the closed operations are much better than the rest of the strategies for both case studies. In particular, strong encoding outperforms the baseline always when the number of generations is compared and 99.53% of the times when wall clock time is compared for the BSH case study. A similar behaviour is observed for the CAF case study (strong encoding outperforms the baseline 96.3% of the times when comparing the number of generations and 95.67% of the times when wall clock time is compared. The closed operation performs worse than the strong encoding but also able to outperform the baseline in terms of number of generations (99.22% of the times for BSH and 70.27% of the times for CAF) and time (98.82% of the times for BSH and 77.5% of the times for CAF) required to find the solution. The remove repair is also able to outperform most of the other strategies in terms of number of generations. However, when comparing the time needed to reach the solution, the remove repair will be outperformed by the baseline most of the times (96.64% of the times for BSH and 97.43% of the times for the CAF case study).

When comparing the two best strategies for the BSH case study, it can be observed that the strong encoding outperforms the closed operations 65.64% of the times when number of generations is compared and 68.79% of the times when the wall clock time is compared. Similarly, for the CAF case study, the strong encoding outperforms the closed operations most of the times for both metrics (87.21% of the times when comparing number of generations and 82.32% of the times when comparing the time needed to reach the solution). Therefore, as an answer to RQ2, we can conclude that the best strategy to be applied is strong encoding, followed by the closed operations, and both are able to outperform the baseline in terms of number of generations and time required to find the solution.

For the textual similarity fitness (Tables 12-19), a high value indicates that the strategy from the row outperforms the strategy from the column. Again, the results of strong encoding are better than the baseline for all the metrics analysed, outperforming the baseline in precision (68.40% of the times for BSH and 70.18% of the times for CAF), recall (91.55% of the times for BSH and 84.62% of the times for CAF), F-Measure (70.61% of the times for BSH and 77.94% of the times for CAF) and MCC (77.37% of the times for BSH and 79.79% of the times for CAF). The same tendency can be observed for the closed operations, although the results are worse than the results of the strong encoding. Some strategies outperform the baseline for one of the metrics (as we indicated when analyzing the results) but they are not able to outperform the baseline for the overall metrics of F-Measure and MCC. As a response to RQ3, we can conclude that the strong encoding and the closed operations strategies are able to outperform the baseline in terms of solution quality when applied in combination with a state-of-the-art fitness function as the textual similarity fitness .

6 Discussion

This section provides a discussion about the results obtained in the evaluation, giving some explanations to the results of each of the strategies. To determine the rationale behind the results obtained, the EA has been observed at runtime, checking how the individuals were evolving while each of the strategies was applied. To illustrate the findings, we use the parent model used as running example (see bottom-left of Fig. 1). Table 6 shows a subset of the models fragments that can be generated using that parent model. Each row shows the encoding of one of the possible individuals (randomly selected out of the 1,024 individuals present in the search space). The first column shows a name to allow the identification of each model fragment (the name assigned to each of them corresponds to the binary string of the encoding in decimal representation). The rows with grey background correspond to conforming individuals; the rows with white background correspond to nonconforming individuals. The columns named from G_0 to G_9 show the value for each specific gene of the individual. The last column shows the value obtained

Individual	G0	G1	G2	G3	G4	G5	G6	G7	G8	G9	Fitness
MF-127	1	1	1	1	1	1	1	0	0	0	0.8
MF-15	1	1	1	1	0	0	0	0	0	0	0.6
MF-38	0	1	1	0	0	1	0	0	0	0	0.4
MF-42	0	1	0	1	0	1	0	0	0	0	0.4
MF-379	1	1	0	1	1	1	1	0	1	0	0.4
MF-14	0	1	1	1	0	0	0	0	0	0	0.4
MF-243	1	1	0	0	1	1	1	1	0	0	0.2
MF-139	1	1	0	1	0	0	0	1	0	0	0.2
MF-885	1	0	1	0	1	1	1	0	1	1	0
MF-329	1	0	0	1	0	0	1	0	1	0	-0.2
MF-0	0	0	0	0	0	0	0	0	0	0	-0.2
MF-784	0	0	0	0	1	0	0	0	1	1	-0.4
MF-578	0	1	0	0	0	0	1	0	0	1	-0.4
MF-834	0	1	0	0	0	0	1	0	1	1	-0.6
MF-898	0	1	0	0	0	0	0	1	1	1	-0.6

Table 6 Subset of model fragments yield by the running example model presented in Fig. 1. Rows with grey background correspond to conforming individuals while rows with white background correspond to nonconforming individuals

by each individual for the optimal fitness function (see subsection 3.2.2).

When the baseline EA with no strategies is executed, the only guide of the search is performed by the fitness function. The individuals that can be present in the population can belong to any subspace (conforming or nonconforming), so any of the individuals present on Table 6 could be part of the population. Each time a new generation is produced, the individuals with higher fitness values have better opportunities to survive an eventually the EA will find the solution.

The results of the penalty strategies when combined with the optimal fitness are low in general and are not able to outperform the baseline in any of the versions (static and dynamic, with or without the violation degree) for either of the case studies. This is due to the penalty strategies reduce the fitness value of some individuals (that have good fitness values) since they belong to the nonconforming subspace. This reduces the possibilities of the algorithm to improve the population generation after generation; the genetic operations need to produce individuals that have high fitness values and also belong to the conforming space.

For example, consider the selection in a population composed of MF-299, MF-885 and MF-0 (see Table 6), whose fitness are 0.4, 0 and -0.2 respectively. If there are no penalty strategies being applied, the probabilities of MF-299 being selected as parent for the next generation are twice the probabilities of MF-885 being selected and four times the probabilities of MF-0 being selected; when the penalties are applied, the fitness of MF-299 and MF-885 is reduced as they belong to the nonconforming subspace and thus their probabilities of being selected as parents for next generation. This kind of situations slows down the search, as the EA needs that the offspring generated not be only fitter than current generation, but also belong to the conforming subspace.

This is not always the case for penalties, and their effect can be positive in some situations (as we have found when observing the strategies at runtime). For example, if the population is composed by the individuals from the five last rows of Table 6, the penalty will penalize individuals whose fitness value is lower than the only conforming one (MF-0), and thus will boost the search. In overall, results show that there are more situations where the penalties are affecting negatively than those where they are affecting positively.

When the penalty strategies are applied in conjunction with the textual similarity fitness, the results are similar, not being able to outperform the baseline in terms of precision, F-Measure or MCC. However, the values obtained in recall are almost as good as those obtained by the baseline. This is due to the fact that model fragments with more genes set to true tend to achieve better fitness values than those with less genes set to true. This results in the EA exploring more solutions around those conforming model fragments that have more genes set to true (such as MF-15 in Table 6), that contain more genes that are also present in the solution and yield greater recall values. However, those fragments also contain more genes set to true that are not present in the solution and thus the precision is reduced and resulting in lower values than those from the baseline in the overall metrics (F-Measure and MCC).

The case for the death penalty is even more extreme, not allowing the existence of any individual outside of the conforming space. Consider an offspring obtained in a generation composed by the five last rows of Table 6. Four individuals will be removed from the population and the only survivor will be MF-0 since it belongs to the conforming space. The problem is that the removal of the nonconforming elements can drastically reduce the ability to explore the search space of the EA [24]. If this situation is repeated across generations, and no new areas of the search space are reached, the EA can be unable to find the solution.

In the evaluation of the death penalty using the optimal fitness, all of the attempts to evolve individuals towards the solution eventually result in nonconforming individuals that are removed by the death penalty. This creates an endless loop that lasts for the allocated generations (30,000), resulting in the strategy not being able to find the solution. The population resides on an island of the conforming space (as depicted in Fig. 4) and reaching the solution is not possible without traversing the nonconforming space. Strategies as the death penalty will keep the individuals from getting outside of the conforming island and will therefore not be able to find the solution.

When the death penalty is applied in conjunction with the textual similarity fitness, a similar situation can be observed. However, this time the search gets stuck around a conforming model fragment that has most of its genes set to true. This results in a value of recall close to 100% for the BSH case study and 90.8% for the CAF case study. However, as happened with the other penalty strategies, there is a drastic reduction in the precision metric and low values for the F-Measure and MCC metrics.

The strong encoding strategy uses a different encoding to solve the problem, and, therefore, the search space is different. The search space is a single conforming subspace, which enables the emergence of faster evolution paths between the individuals and the solution. This results in a lower number of generations and time needed to reach the solution, as the results show. This type of encoding reorders the conforming search space into a space that is easier to navigate. For example, in the case of the running example (Table 6) the search space (1,024 individuals) is reduced to the conforming space (144 individuals), so the EA has better chances of reaching the solution.

This effect is also happening when applying the textual similarity fitness, resulting in performance values higher than those from the baseline for both case studies. In particular the F-Mesure is 20% points better when using the strong encoding compared to the baseline for the BSH case study and around 22% points better for the CAF case study. The improvement is also noticeable for the MCC metric, obtaining values around 0.25 units (out of 1) higher than the baseline for both case studies.

The closed operations strategy guarantees that the results of genetic operations remain in the conforming space. This is done through a wise combination of the individuals, resulting in larger steps each time an operator is applied. Some of the mutations and crossovers will result in bigger changes to the individual, and if those changes result in higher fitness values and preservation across generations, there will be a lower number of generations and less time needed to reach the solution. When an individual is generated by the use of closed operations, it will remain in the conforming subspace. For example, if MF-0 is evolved (through mutations or crossover), it will not produce any nonconforming individual (like MF-329 or MF-885) but a conforming one (like MF-139). This results in a smaller search space (as is the case of the strong encoding) and thus in a lower number of generations needed to reach the solution.

The same behaviour can be observed when the closed operations are used in combination with the textual similarity fitness. Again, values in precision, recall, F-Measure and MCC are above the values obtained by the baseline for both case studies. The performance of the closed operations in terms of solution quality is slightly below the performance of the strong encoding.

Finally, the two types of repair show totally opposite behaviour when used in conjunction with the optimal fitness: while the remove repair improves the search process and results in a lower number of generations, the add repair hinders the process, resulting in a much higher number of generations. Both are repairing the individuals to ensure that they remain in the conforming subspace after applying the operator. However, the modifications done by the add repair are counterproductive since they include elements that are not part of the solution, while the modifications done by the remove repair are more productive, making the individual more similar to the solution. Anyhow, even if the remove repair needs less generations than the baseline to reach the solution, the time spent is above the baseline (due to the complexity of the operation performed). This can be illustrated by model fragments from Table 6. For example, the individual MF-14 might be repaired into MF-15 (by the add repair) or repaired into MF-0 (by the remove repair). In some situations the repair operator is boosting the search (MF-15 has higher fitness value than the MF-14), while in other situations

the repair operator is obstructing the search (MF-0 has lower fitness value than MF-14).

In fact, we have checked the individuals while they are evolving, and the add repair hinders the evolution since it is adding new genes to the individuals that need to be removed later by the mutation and crossover operations (since those genes do not form part of the solution). This can lead to a loop that makes the solution unreachable in the number of generations allocated (as is the case for the BSH case study, where the number of generations are close to the 30,000 limit).

When the repair operators are used in conjunction with the textual similarity fitness, a similar behaviour can be observed. The add repair is performing much worse than the baseline, while the remove repair obtains values better than the add repair (but still worse than the values obtained by the baseline). It is interesting to see how the add repair has lower values in precision than the baseline, as it is adding elements (that may not be present in the solution and thus reduce the precision) to repair the individuals. By contrast, the recall values are higher (outperforming the baseline) as some of the elements added during the repair may be part of the solution. Similarly, the remove repair achieves better values of precision but worse values of recall. Most of the elements removed during the repair are not part of the solution (so the precision rises) but some of the elements removed were correctly placed in the model fragment as they are part of the solution (so the recall is reduced). However, the repair strategies are not able to outperform the baseline in terms of F-Measure or MCC.

7 Threats to validity

In this section, we present some of the possible threats to validity of the evaluation performed and how we have addressed or mitigated them. We follow the guidelines suggested by De Oliveira et. al [64] to identify those that apply to this work. The threats are divided into four groups:

Conclusion validity threats: These are concerned with the relationship between the treatment of the data and the outcome. The design must ensure the statistical relationship between the parts. We have identified four threats of this type:

- Not accounting for random variation: To address this threat, we considered 30 independent runs for each execution of each of the nine strategies and the baseline.
- Lack of good descriptive statistics: In this work, we have used several metrics to compare the different approaches, including the number of generations (as suggested in the literature [51]) and wall clock time needed to find the solution when using the optimal fitness, and precision, recall, F-Measure and MCC when using the textual similarity fitness. In addition, some works [63] argue that the use of the Vargha and Delaney's $\hat{A}_{12}$ metric may be unrepresentative and that the data should be treated before applying it. We did not find any use case for data pre-transformation that applies to our case studies.
- Lack of a meaningful comparison baseline: To address this threat, we compare the nine different strategies against a baseline, the same EA without any strategy for handling nonconforming individuals.
- Lack of formal hypothesis and statistical tests: To address this threat we have performed a standard statistical analysis, following accepted guidelines [12].

Internal validity threats: If a relationship between treatment and outcome is observed, the experimental design must guarantee that it is a causal relationship. We have identified four threats of this type:

- Poor parameter settings: In this work, we use standard values for the evolutionary algorithm that have been tested in similar conditions for feature location [56,39]. For the parameters that have no values reported yet (penalties in models), we have performed a parameter tuning (based on procedures described in the literature [12]) to find the ones that provide the best results. For each penalty parameter we tested different sets of values, selected the best-performing one and repeated with values above and below the best-performing so far, until there was no further improvement. Further evaluation could be needed to find the best values (as we plan to do in the future) although there is no guarantee that those values will perform similarly when the approach is applied to other problems or domains.
- Lack of discussion on code instrumentation: To avoid the inclusion of tweaks or instrumentation to favor certain algorithms, we have made public the source code [35] of an open-source implementation of the nine strategies presented, as suggested in the literature [51].
- Lack of clarity of data collection tools and procedures: The set of 748 test cases used in the evaluation has been provided by domain experts from our industrial partners (BSH and CAF). The test cases provided are representative of their respective domains, and the only pre-processing performed was to identify malformed test cases (where the solution

was the whole model fragment or the empty model fragment).
- Lack of real problem instances: The evaluation of this paper was applied to industrial case studies, (BSH and CAF), with the problem instance being obtained directly from industry.

Construct validity threats: These are concerned with the relations between theory and observation. We have identified one threat of this type:

- Lack of assessing the validity of cost measures: To address this threat, we have performed a fair comparison between the different strategies and the baseline by using the number of generations as the cost measure [51]. In addition, the solution quality measures used (precision, recall, F-Measure and MCC) are widely used in the field of information retrieval [77,58,20].

External validity threats: Concerned with the generalization of observed results to a larger population outside of the experiment. We have identified three threats of this type:

- Lack of clear definition of target instances: To address this threat, the test cases are explained, giving as much detail as possible (such as the number and type of items of the models from the test cases and the languages used to build them). The non-disclosure agreements signed with our industrial partners prevent us from providing the test cases themselves as they correspond to products that are currently on the market.
- Lack of a clear object selection strategy: We have detected three situations where this threat could prevent the application of the presented approach (as is) to different scenarios:
  - Clear test cases selection strategy: the strategy has been described in the internal threat about data collection tools and procedures. The domain experts from our industrial partners provided us with a set of test cases that are representative of their domains (covering the full range of products) and we performed only a sanity check to remove malformed test cases.
  - Problem selected for evaluation (feature location): the nine techniques presented in this work are generic and they do not include any particularity of the problem being addressed (feature location) or the domain specific language used; the constraints included are derived from the conformance between the model and the metamodel. They can be applied directly to other problems where the EA is using model fragments as individuals. We expect that the strategies will behave similarly when applied to other problems; However, we cannot guarantee that the results will be the same and further evaluation could be needed to determine if the results vary when applied to other problems or when adapted to work with a different encoding.
  - Fitness selected for the evaluation: the techniques presented have been evaluated with two different fitness functions, optimal fitness and textual similarity fitness. The optimal fitness is based on an oracle to reduce the noise that could be introduced by the fitness function, making it impossible to apply to a real scenario. The textual similarity fitness is a state-of-the-art fitness and shows how the strategies behave on a real scenario. There are multiple fitness functions being applied to solve different MDE related problems available in the literature [17] and the results could vary depending on the specific details of each fitness. Both of the fitness used in this work provided results that are consistent; however, further evaluations with different fitness functions are needed to determine if the behaviour is the same with any fitness function.
- Lack of evaluation instances of growing size and complexity: To mitigate this threat, we have applied the strategies to two case studies varying in size and complexity.

8 Conclusion

EAs can be applied to find solutions to several problems related to MDE practices. Reducing the number of generations needed by the EA to find those solutions can be the difference between (i) a search process that is not able to find the solution in a reasonable time, (ii) the same search being applied as an offline process, (iii) the same search being applied at run-time, providing the results while the user is interacting with the system, and (iv) the usage of more complex fitness functions that can now be applied given the reduced number of generations required by the search.

In this work, we have presented nine different strategies that can be applied to handle nonconforming individuals when applying EAs encoding model fragments. The strategies presented are generic and include only constraints that are derived directly from MOF, making them independent from the domain of application (Induction hobs and train control systems in this work). Bigger improvements in performance could be achieved if the strategies were tailored with domain knowledge and adapted to specific problems.

The nine strategies have been applied in combination with two different fitness functions to solve feature location problems from two different industrial domains, providing statistically significant results and comparison among them. The evaluation using the optimal fitness shows that some of the strategies were able to boost the search process, resulting in a lower number of generations needed to reach the solution (ten times fewer generations in the most extreme case) and less time spent. The evaluation using the textual similarity fitness confirms the results, showing that the strong encoding and closed operation strategies were able to outperform the baseline in terms of solution quality for both case studies.

From the related work analysis performed, we have discovered that strategies for handling nonconforming individuals are being applied by some researchers using SBSE solutions to solve MDE problems, but its spread is not generalized yet. To help in the spread of this kind of strategies, an open-source implementation of the nine generic strategies has been made publicly available in order to facilitate its adoption by the community. In addition, we provide insights of the behaviour of the different strategies when solving the feature location problem that could benefit other practitioners when choosing which strategy should be applied when solving his MDE problem. We believe that this work could lead to the application of strategies for handling nonconforming individuals that yield to the results faster by more researchers of the SDMDE community. Similarly, we expect more research to evaluate if the results are similar when applying the strategies to other SBMDE problems and when adapting them to work with other encoding when required by the problem. As a result, new generic strategies may emerge, resulting in a catalogue of strategies that can be reused and improved by the community.

Acknowledgements This work has been partially supported by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds under the Project ALPS (RTI2018-096411-B-I00).

We thank William B. Langdon and Justyna Petke because their feedback while Carlos stayed at University College London inspired this work.

References

1. Efficient java matrix library. `http://ejml.org/`. [Online; accessed 7-April-2016]
2. Eclipse Development Using the Graphical Editing Framework and the Eclipse Modeling Framework. IBM Corp., USA (2004)
3. Apache opennlp: Toolkit for the processing of natural language text. `https://opennlp.apache.org/` (2016). [Online; accessed 7-April-2016]
4. The english (porter2) stemming algorithm. `http://snowball.tartarus.org/algorithms/english/stemmer.html` (2016). [Online; accessed 7-April-2016]
5. Abdeen, H., Varró, D., Sahraoui, H., Nagy, A.S., Debreceni, C., Hegedüs, Á., Horváth, Á.: Multi-objective optimization in rule-based design space exploration. In: Proc. of the 29th ACM/IEEE international conf. on Automated software engineering, pp. 289–300. ACM (2014)
6. Afzal, W., Torkar, R., Feldt, R.: A systematic review of search-based testing for non-functional system properties. Inf. and Soft. Technology **51**(6), 957 – 976 (2009)
7. Ali, S., Iqbal, M.Z., Arcuri, A., Briand, L.: A search-based ocl constraint solver for model-based test data generation. In: 2011 11th International Conference on Quality Software, pp. 41–50. IEEE (2011)
8. Ali, S., Iqbal, M.Z., Arcuri, A., Briand, L.C.: Generating test data from ocl constraints with search techniques. IEEE Transactions on Software Engineering **39**(10), 1376–1402 (2013)
9. Alshahwan, N., Harman, M.: Automated web application testing using search based software engineering. In: 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 3–12 (2011)
10. Arcega, L., Font, J., Haugen, Ø., Cetina, C.: An approach for bug localization in models using two levels: model and metamodel. Softw. Syst. Model. **18**(6), 3551–3576 (2019). DOI 10.1007/s10270-019-00727-y
11. Arcuri, A., Briand, L.: A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Software Testing, Verification and Reliability **24**(3), 219–250 (2014)
12. Arcuri, A., Fraser, G.: Parameter tuning or default values? an empirical investigation in search-based software engineering. Empirical Software Engineering **18**(3), 594–623 (2013)
13. Bäck, T., Schütz, M., Khuri, S.: A comparative study of a penalty function, a repair heuristic, and stochastic operators with the set-covering problem. In: European conf. on Artificial Evolution, pp. 320–332. Springer (1995)
14. Ballarín, M., Marcén, A.C., Pelechano, V., Cetina, C.: Measures to report the location problem of model fragment location. In: 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS '18, pp. 189–199. ACM, New York, NY, USA (2018)
15. Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA journal on computing **6**(2), 154–160 (1994)
16. Bill, R., Fleck, M., Troya, J., Mayerhofer, T., Wimmer, M.: A local and global tour on momot. Softw. Syst. Model. **18**(2), 1017–1046 (2019)
17. Boussaïd, I., Siarry, P., Ahmed-Nacer, M.: A survey on search-based model-driven engineering. Automated Software Engineering **24**(2), 233–294 (2017)
18. Burdusel, A., Zschaler, S., John, S.: Automatic generation of atomic consistency preserving search operators for search-based model engineering. In: ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 106–116 (2019)
19. Cetina, C., Font, J., Arcega, L., Pérez, F.: Improving feature location in long-living model-based product families designed with sustainability goals. J. Syst. Softw. **134**, 261–278 (2017)

20. Chicco, D., Jurman, G.: The advantages of the matthews correlation coefficient over f1 score and accuracy in binary classification evaluation. BMC genomics **21**(1), 6 (2020)
21. Chootinan, P., Chen, A.: Constraint handling in genetic algorithms using a gradient-based repair method. Computers & operations research **33**(8), 2263–2281 (2006)
22. Coello, C.A.C.: Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Comp. Methods in Applied Mechanics and Engineering **191**(11), 1245 – 1287 (2002)
23. Colanzi, T.E., Vergilio, S.R.: Representation of software product line architectures for search-based design. In: 2013 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE), pp. 28–33 (2013)
24. Črepinšek, M., Liu, S.H., Mernik, M.: Exploration and exploitation in evolutionary algorithms: A survey. ACM computing surveys (CSUR) **45**(3), 1–33 (2013)
25. Czarnecki, K., Helsen, S.: Classification of model transformation approaches. In: 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, vol. 45, pp. 1–17. USA (2003)
26. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE transactions on evolutionary computation **6**(2), 182–197 (2002)
27. Denil, J., Jukss, M., Verbrugge, C., Vangheluwe, H.: Search-based model optimization using model transformations. In: D. Amyot, P. Fonseca i Casas, G. Mussbacher (eds.) System Analysis and Modeling: Models and Reusability, pp. 80–95. Springer International Publishing, Cham (2014)
28. Dit, B., Revelle, M., Gethers, M., Poshyvanyk, D.: Feature location in source code: a taxonomy and survey. J. of Soft.: Evolution and Process **25**(1), 53–95 (2013)
29. Dyer, D.: The watchmaker framework for evolutionary computation. `http://watchmaker.uncommons.org/` (2016). [Online; accessed 7-April-2016]
30. Faunes, M., Cadavid, J., Baudry, B., Sahraoui, H., Combemale, B.: Automatically searching for metamodel well-formedness rules in examples and counter-examples. In: International Conference on Model Driven Engineering Languages and Systems, pp. 187–202. Springer (2013)
31. Faunes, M., Cadavid, J., Baudry, B., Sahraoui, H., Combemale, B.: Automatically searching for metamodel well-formedness rules in examples and counter-examples. In: A. Moreira, B. Schätz, J. Gray, A. Vallecillo, P. Clarke (eds.) Model-Driven Engineering Languages and Systems, pp. 187–202. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
32. Fleck, M., Troya, J., Kessentini, M., Wimmer, M., Alkhazi, B.: Model transformation modularization as a many-objective optimization problem. IEEE Trans. Software Eng. **43**(11), 1009–1032 (2017)
33. Fleck, M., Troya, J., Wimmer, M.: Search-based model transformations with momot. In: P. Van Gorp, G. Engels (eds.) Theory and Practice of Model Transformations, pp. 79–87. Springer International Publishing, Cham (2016)
34. Font, J.: Location of features as model fragments and their co-evolution. Ph.D. thesis, U. of Oslo, Nor. (2017)
35. Font, J.: Source Code for Feature Location in Models through an Evolutionary Algorithm - Handling nonConforming Individuals (2020). `https://bitbucket.org/svitusj/flimea-hci`
36. Font, J., Arcega, L., Haugen, Ø., Cetina, C.: Building software product lines from conceptualized model patterns. In: 19th International Conference on Software Product Line, SPLC '15, pp. 46–55 (2015)
37. Font, J., Arcega, L., Haugen, Ø., Cetina, C.: Feature location in model-based software product lines through a genetic algorithm. In: 15th International Conference on Software Reuse: Bridging with Social-Awareness - Volume 9679, ICSR 2016, pp. 39–54 (2016)
38. Font, J., Arcega, L., Haugen, Ø., Cetina, C.: Feature location in models through a genetic algorithm driven by information retrieval techniques. In: ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, MODELS '16, pp. 272–282 (2016)
39. Font, J., Arcega, L., Haugen, Ø., Cetina, C.: Achieving feature location in families of models through the use of search-based software engineering. IEEE Transactions on Evolutionary Computation **PP**(99), 1–1 (2017)
40. Font, J., Arcega, L., Haugen, Ø., Cetina, C.: Leveraging variability modeling to address metamodel revisions in model-based software product lines. Computer Languages, Systems & Structures **48**, 20–38 (2017)
41. Font, J., Ballarín, M., Haugen, Ø., Cetina, C.: Automating the variability formalization of a model family by means of common variability language. In: 19th International Conference on Software Product Line, SPLC '15, pp. 411–418 (2015)
42. García, S., Fernández, A., Luengo, J., Herrera, F.: Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. Information Sciences **180**(10), 2044–2064 (2010)
43. Goldberg, D.E., Lingle, R., et al.: Alleles, loci, and the traveling salesman problem. In: Int. conference on genetic algorithms and their applications, vol. 154, pp. 154–159. Carnegie-Mellon University Pittsburgh, PA (1985)
44. Gomez, J.J.C., Baudry, B., Sahraoui, H.: Searching the boundaries of a modeling space to test metamodels. In: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, pp. 131–140 (2012)
45. Grissom, R.J., Kim, J.J.: "Effect sizes for research: A broad practical approach. Mahwah, NJ: Earlbaum (2005)
46. Harman, M., Jia, Y., Krinke, J., Langdon, W.B., Petke, J., Zhang, Y.: Search based software engineering for software product line engineering: A survey and directions for future work. In: 18th International Software Product Line Conference - Volume 1, SPLC '14, pp. 5–18 (2014)
47. Harman, M., Jones, B.F.: Search-based software engineering. Inf. and soft.Technology **43**(14), 833–839 (2001)
48. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: Trends, techniques and applications. ACM Comput. Surv. **45**(1), 11:1–11:61 (2012)
49. Hofmann, T.: Probabilistic Latent Semantic Indexing. In: 22nd Annual International ACM/SIGIR Conf. on Research and Development in Information Retrieval (1999)
50. Holthusen, S., Wille, D., Legat, C., Beddig, S., Schaefer, I., Vogel-Heuser, B.: Family model mining for function block diagrams in automation software. In: 18th International Software Product Line Conference: Volume 2, pp. 36–43 (2014)
51. Johnson, D.S.: A theoretician's guide to the experimental analysis of algorithms. Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges **59**, 215–250 (2002)

52. Joines, J.A., Houck, C.R.: On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with ga's. In: Evolutionary Computation. First IEEE World Congress on Computational Intelligence., pp. 579–584. IEEE (1994)
53. Kent, S.: Model driven engineering. In: Integrated Formal Methods, pp. 286–298. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
54. Kessentini, M., Langer, P., Wimmer, M.: Searching models, modeling search: On the synergies of sbse and mde. In: 2013 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE), pp. 51–54 (2013)
55. Landauer, T.K., Foltz, P.W., Laham, D.: An introduction to latent semantic analysis. Discourse processes **25**(2-3), 259–284 (1998)
56. Lopez-Herrejon, R.E., Linsbauer, L., Galindo, J.A., Parejo, J.A., Benavides, D., Segura, S., Egyed, A.: An assessment of search-based techniques for reverse engineering feature models. Journal of Systems and Software **103**, 353 – 369 (2015)
57. Mandow, L., Montenegro, J.A., Zschaler, S.: Mejora de una representación genética genérica para modelos. In: Actas de la XVII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA) (in press) (2016)
58. Marcus, A., Sergeyev, A., Rajlich, V., Maletic, J.: An information retrieval approach to concept location in source code. In: 11th Working Conference on Reverse Engineering, pp. 214–223 (2004)
59. Martinez, J., Ziadi, T., Bissyandé, T.F., Klein, J., Traon, Y.L.: Bottom-up adoption of software product lines: a generic and extensible approach. In: 19th Int. Conf. on Software Product Line (SPLC), pp. 101–110 (2015)
60. Michalewicz, Z.: Do not kill unfeasible individuals. In: Fourth Intelligent Information Systems Workshop, pp. 110–123 (1995)
61. Michalewicz, Z.: A survey of constraint handling techniques in evolutionary computation methods. In: 4th Annual Conference on Evolutionary Programming, pp. 135–155. MIT Press (1995)
62. Michalewicz, Z., Nazhiyath, G.: Genocop iii: A coevolutionary algorithm for numerical optimization problems with nonlinear constraints. In: Evolutionary Computation, 1995., IEEE International Conference on, vol. 2, pp. 647–651. IEEE (1995)
63. Neumann, G., Harman, M., Poulding, S.: Transformed Vargha-Delaney Effect Size, pp. 318–324 (2015)
64. de Oliveira Barros, M., Dias-Neto, A.C.: 0006/2011-threats to validity in search-based software engineering empirical studies. RelaTe-DIA **5**(1) (2011)
65. (OMG), O.M.G.: Meta object facility (mof) version 2.4.1 (2013). Http://www.omg.org/spec/MOF/2.4.1/
66. Orvosh, D., Davis, L.: Shall we repair? genetic algorithms combinatorial optimization and feasibility constraints. In: 5th Int. Conf.on Genetic Algorithms, p. 650 (1993)
67. Orvosh, D., Davis, L.: Using a genetic algorithm to optimize problems with feasibility constraints. In: Evolutionary Computation, 1994. First IEEE World Congress on Computational Intelligence., pp. 548–553 (1994)
68. Paige, R.F., Brooke, P.J., Ostroff, J.S.: Metamodel-based model conformance and multiview consistency checking. ACM Transactions on Software Engineering and Methodology (TOSEM) **16**(3), 11 (2007)
69. Pérez, F., Font, J., Arcega, L., Cetina, C.: Automatic query reformulations for feature location in a model-based family of software products. Data Knowl. Eng. **116**, 159–176 (2018)
70. Pérez, F., Font, J., Arcega, L., Cetina, C.: Collaborative feature location in models through automatic query expansion. Autom. Softw. Eng. **26**(1), 161–202 (2019)
71. Pérez, F., Lapeña, R., Font, J., Cetina, C.: Fragment retrieval on models for model maintenance: Applying a multi-objective perspective to an industrial case study. Inf. Softw. Technol. **103**, 188–201 (2018)
72. Pérez, F., Ziadi, T., Cetina, C.: Utilizing automatic query reformulations as genetic operations to improve feature location in software models. IEEE Transactions on Software Engineering pp. 1–1 (2020)
73. Rose, L.M., Kolovos, D.S., Paige, R.F., Polack, F.A.C.: Model migration with epsilon flock. In: L. Tratt, M. Gogolla (eds.) Theory and Practice of Model Transformations, pp. 184–198 (2010)
74. Rothlauf, F.: Representations for genetic and evolutionary algorithms. In: Representations for Genetic and Evolutionary Algorithms, pp. 9–32. Springer (2006)
75. Rubin, J., Chechik, M.: A survey of feature location techniques. In: Domain Engineering, pp. 29–58. Springer Berlin Heidelberg (2013)
76. Runarsson, T.P., Yao, X.: Stochastic ranking for constrained evolutionary optimization. IEEE Transactions on evolutionary computation **4**(3), 284–294 (2000)
77. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill, Inc., NY, USA (1986)
78. Segura, S., Parejo, J.A., Hierons, R.M., Benavides, D., Ruiz-Cortés, A.: Automated generation of computationally hard feature models using evolutionary algorithms. Ex. Sys. with Applications **41**(8), 3975–3992 (2014)
79. Semeráth, O., Barta, A., Horváth, A., Szatmári, Z., Varró, D.: Formal validation of domain-specific languages with derived features and well-formedness constraints. Software and Systems Modeling **16**(2), 357 – 392 (2017)
80. Semeráth, O., Nagy, A.S., Varró, D.: A graph solver for the automated generation of consistent domain-specific models. In: 40th International Conference on Software Engineering, ICSE '18, p. 969–980 (2018)
81. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: eclipse modeling framework. Pearson Edu. (2008)
82. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0, 2nd edn. Addison-Wesley Professional (2009)
83. Svendsen, A., Zhang, X., Lind-Tviberg, R., Fleurey, F., Haugen, Ø., Møller-Pedersen, B., Olsen, G.K.: Developing a software product line for train control: a case study of cvl. In: 14th international conference on Software product lines (SPLC) (2010)
84. Vargha, A., Delaney, H.D.: A critique and improvement of the cl common language effect size statistics of mcgraw and wong. Journal of Educational and Behavioral Statistics **25**(2), 101–132 (2000)
85. Wille, D., Holthusen, S., Schulze, S., Schaefer, I.: Interface variability in family model mining. In: 17th International Software Product Line Conference: Co-located Workshops, pp. 44–51 (2013)
86. Williams, J.R.: A novel representation for search-based model-driven engineering. Ph.D. thesis, U. of York (2013)
87. Williams, J.R., Paige, R.F., Kolovos, D.S., Polack, F.A.: Search-based model driven engineering. Tech. rep., Citeseer (2012)
88. Williams, J.R., Poulding, S., Rose, L.M., Paige, R.F., Polack, F.A.: Identifying desirable game character behaviours through the application of evolutionary algorithms to model-driven engineering metamodels. In: International Symposium on Search Based Software Engineering, pp. 112–126 (2011)

89. Yeniay, Ö.: Penalty function methods for constrained optimization with genetic algorithms. Mathematical and computational Applications **10**(1), 45–56 (2005)
90. Zhang, X., Haugen, Ø., Moller-Pedersen, B.: Model comparison to synthesize a model-driven software product line. In: 2011 15th International Software Product Line Conference (SPLC), pp. 90–99 (2011)
91. Zhang, X., Haugen, Ø., Møller-Pedersen, B.: Augmenting product lines. In: 19th Asia-Pacific Software Engineering Conference (APSEC), vol. 1, pp. 766–771 (2012)

A Statistical Analysis Results

	BSH						CAF					
	Generations	Time	Precision	Recall	F-Measure	MCC	Generations	Time	Precision	Recall	F-Measure	MCC
$p-value$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Statistic	1251.9	1182	515.26	457.13	402.27	367.43	244.21	571.36	83.311	79.148	78.927	71.36

Table 7 The Quade Test statistic and $p-value$ for the BSH and CAF case studies

	Baseline	SP	SDP	DP	DDP	Death	Strong	Closed	Add	Remove
Baseline	-	0.066	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$7.8x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
SP	53.19%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$2.3x10^{-06}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
SDP	79.81%	77.19%	-	$2.8x10^{-07}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
DP	71.34%	68.57%	41.09%	-	$5.1x10^{-09}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
DDP	61.85%	58.89%	31.84%	40.42%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Death	100%	100%	99.84%	98.52%	100%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	0.701	$\ll 2x10^{-16}$
Strong	0%	0.01%	0%	0%	0%	0%	-	$4.3x10^{-05}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Closed	0.78%	0.59%	0.01%	0.07%	0.29%	0%	65.64%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Add	99.96%	99.94%	99.44%	98.19%	99.80%	46.55%	100%	100%	-	$\ll 2x10^{-16}$
Remove	21.87%	19.71%	6.05%	9.93%	14.94%	0%	99.79%	93.60%	0.01%	-

Table 8 Results of the statistical analysis for the number of generations from BSH case study. Values above the diagonal show the Holm's post hoc for each pair of strategies (row and column), with values below 0.05 being statistically significant. Values not significant are highlighted in grey. Values below the diagonal show the $AHat_{12}$ value, indicating the number of times that the first strategy (row) performs worse than the second strategy (column). Values outperforming the baseline are highlighted in grey.

	Baseline	SP	SDP	DP	DDP	Death	Strong	Closed	Add	Remove
Baseline	-	0.24620	0.24620	$\ll 2x10^{-16}$	$9.8x10^{-08}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.0x10^{-15}$
SP	56.92%	-	0.89463	$\ll 2x10^{-16}$	0.00038	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
SDP	56.29%	48.77%	-	$\ll 2x10^{-16}$	0.00025	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
DP	80.17%	73.27%	74.87%	-	$7.5x10^{-06}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	0.0006	$\ll 2x10^{-16}$
DDP	66.42%	62.06%	63.22%	45.18%	-	$4.2x10^{-11}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Death	100%	97.87%	99.29%	86.52%	80.85%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	0.00928	$\ll 2x10^{-16}$
Strong	3.70%	3.40%	3.61%	0.94%	3.48%	0%	-	$1.1x10^{-07}$	$\ll 2x10^{-16}$	$2.4x10^{-13}$
Closed	29.73%	26.19%	26.54%	10.16%	22.39%	0%	87.21%	-	$\ll 2x10^{-16}$	0.18534
Add	86.46%	82.16%	83.18%	64.05%	65.66%	30.14%	99.83%	93.80%	-	$\ll 2x10^{-16}$
Remove	33.67%	29.62%	29.24%	11.03%	25.13%	0%	90.92%	56.57%	7.79%	-

Table 9 Results of the statistical analysis for the number of generations from CAF case study. Values above the diagonal show the Holm's post hoc for each pair of strategies (row and column), with values below 0.05 being statistically significant. Values not significant are highlighted in grey. Values below the diagonal show the $AHat_{12}$ value, indicating the number of times that the first strategy (row) performs worse than the second strategy (column). Values outperforming the baseline are highlighted in grey.

	Baseline	SP	SDP	DP	DDP	Death	Strong	Closed	Add	Remove
Baseline	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
SP	99.09%	-	$\ll 2x10^{-16}$	$2.5x10^{-08}$	$6.8x10^{-05}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
SDP	98.55%	26.68%	-	0.0122	$5.7x10^{-05}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
DP	95.51%	67.95%	51.05%	-	0.1192	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
DDP	98.80%	61.47%	38.16%	41.35%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Death	100%	100%	99.99%	100%	100%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	0.0056	$\ll 2x10^{-16}$
Strong	0.47%	0.66%	1.06%	4.08%	1.15%	0%	-	0.0122	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Closed	1.18%	0.67%	1.10%	4.10%	1.14%	0%	68.79%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Add	100%	100%	100%	100%	100%	65.69%	100%	100%	-	$\ll 2x10^{-16}$
Remove	96.64%	12.54%	6.28%	12.19%	10.14%	0%	97.91%	97.88%	0%	-

Table 10 Results of the statistical analysis for wall clock time metric from BSH case study. Values above the diagonal show the Holm's post hoc for each pair of strategies (row and column), with values below 0.05 being statistically significant. Values not significant are highlighted in grey. Values below the diagonal show the $AHat_{12}$ value, indicating the number of times that the first strategy (row) performs worse than the second strategy (column). Values outperforming the baseline are highlighted in grey.

	Baseline	SP	SDP	DP	DDP	Death	Strong	Closed	Add	Remove
Baseline	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$3.0x10^{-06}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
SP	77.39%	-	$7.4x10^{-08}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.1x10^{-09}$
SDP	95.49%	54.35%	-	$4.2x10^{-15}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
DP	99.04%	64.67%	65.61%	-	0.97	0.82	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.6x10^{-10}$	$\ll 2x10^{-16}$
DDP	95.38%	64.36%	64.64%	51.39%	-	0.97	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.9x10^{-09}$	$\ll 2x10^{-16}$
Death	100%	70.00%	77.10%	56.24%	50.90%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$8.4x10^{-08}$	$\ll 2x10^{-16}$
Strong	4.33%	21.88%	4.10%	0.68%	4.28%	0%	-	$3.5x10^{-08}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Closed	22.50%	22.13%	4.22%	0.89%	4.39%	0%	82.32%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Add	99.34%	73.48%	77.96%	64.59%	61.48%	64.58%	99.34%	99.34%	-	$\ll 2x10^{-16}$
Remove	97.43%	31.93%	12.29%	4.66%	10.73%	0%	99.16%	98.46%	2.22%	-

Table 11 Results of the statistical analysis for wall clock time metric from CAF case study. Values above the diagonal show the Holm's post hoc for each pair of strategies (row and column), with values below 0.05 being statistically significant. Values not significant are highlighted in grey. Values below the diagonal show the $AHat_{12}$ value, indicating the number of times that the first strategy (row) performs worse than the second strategy (column). Values outperforming the baseline are highlighted in grey.

	Baseline	SP	SDP	DP	DDP	Death	Strong	Closed	Add	Remove
Baseline	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	0.0693
SP	34.84%	-	$1.3x10^{-11}$	$\ll 2x10^{-16}$	$2.4x10^{-12}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
SDP	26.95%	40.71%	-	0.0012	1	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
DP	22.49%	35.63%	45.27%	-	0.0024	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
DDP	26.06%	40.13%	49.49%	54.37%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Death	10.83%	17.80%	23.41%	25.72%	23.08%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$6.7x10^{-11}$	$\ll 2x10^{-16}$
Strong	68.40%	82.66%	88.12%	91.34%	89.45%	96.59%	-	1	$\ll 2x10^{-16}$	$7.8x10^{-10}$
Closed	68.44%	82.35%	88.15%	91.46%	89.39%	97.37%	50.49%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Add	14.79%	24.44%	31.87%	35.19%	32.00%	61.40%	4.94%	4.38%	-	$\ll 2x10^{-16}$
Remove	59.91%	76.18%	83.37%	87.44%	84.62%	96.83%	41.02%	40.39%	93.81%	-

Table 12 Results of the statistical analysis for the precision metric from BSH case study. Values above the diagonal show the Holm's post hoc for each pair of strategies (row and column), with values below 0.05 being statistically significant. Values not significant are highlighted in grey. Values below the diagonal show the $AHat_{12}$ value, indicating the number of times that the first strategy (row) performs better than the second strategy (column) in terms of precision. Values outperforming the baseline are highlighted in grey.

	Baseline	SP	SDP	DP	DDP	Death	Strong	Closed	Add	Remove
Baseline	-	$9.7x10^{-06}$	$2.9x10^{-07}$	$1.2x10^{-11}$	0.06435	$\ll 2x10^{-16}$	$5.5x10^{-07}$	0.00029	$\ll 2x10^{-16}$	0.54982
SP	32.03%	-	0.60322	0.13274	0.12861	$7.9x10^{-15}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.6x10^{-11}$	$2.3x10^{-09}$
SDP	28.32%	45.84%	-	0.52783	0.02475	$1.3x10^{-12}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.4x10^{-09}$	$2.9x10^{-11}$
DP	23.86%	40.47%	44.14%	-	$5.1x10^{-05}$	$4.8x10^{-08}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.2x10^{-05}$	$\ll 2x10^{-16}$
DDP	36.66%	54.82%	58.84%	64.17%	-	$\ll 2x10^{-16}$	$1.4x10^{-14}$	$1.7x10^{-10}$	$\ll 2x10^{-16}$	0.00044
Death	9.19%	15.55%	17.49%	16.54%	13.57%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	0.60322	$\ll 2x10^{-16}$
Strong	70.18%	85.04%	87.58%	91.02%	81.07%	97.28%	-	0.57261	$\ll 2x10^{-16}$	0.00053
Closed	64.61%	78.97%	81.62%	85.05%	75.08%	94.14%	46.20%	-	$\ll 2x10^{-16}$	0.05140
Add	10.84%	18.61%	20.48%	20.04%	16.18%	58.15%	3.06%	6.93%	-	$\ll 2x10^{-16}$
Remove	53.38%	74.67%	79.23%	85.22%	69.25%	96.95%	31.02%	37.43%	96.06%	-

Table 13 Results of the statistical analysis for the precision metric from CAF case study. Values above the diagonal show the Holm's post hoc for each pair of strategies (row and column), with values below 0.05 being statistically significant. Values not significant are highlighted in grey. Values below the diagonal show the $AHat_{12}$ value, indicating the number of times that the first strategy (row) performs better than the second strategy (column) in terms of precision. Values outperforming the baseline are highlighted in grey.

	Baseline	SP	SDP	DP	DDP	Death	Strong	Closed	Add	Remove
Baseline	-	0.30647	0.30647	0.42903	1	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.1x10^{-14}$	$\ll 2x10^{-16}$
SP	46.82%	-	1	1	0.15561	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
SDP	46.73%	50.03%	-	1	0.18279	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
DP	45.97%	49.07%	48.68%	-	0.30647	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
DDP	49.95%	53.48%	53.45%	54.93%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.6x10^{-13}$	$\ll 2x10^{-16}$
Death	98.54%	99.06%	99.07%	98.73%	98.90%	-	0.00067	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Strong	91.55%	92.39%	92.31%	92.12%	91.71%	44.71%	-	$4.4x10^{-06}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Closed	82.02%	83.05%	82.99%	82.82%	81.85%	37.01%	42.35%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Add	56.88%	60.97%	60.93%	63.49%	57.46	2.27%	10.21%	21.39%	-	$\ll 2x10^{-16}$
Remove	12.50%	12.90%	13.21%	11.19%	9.96%	0%	2.74%	4.78%	3.89%	-

Table 14 Results of the statistical analysis for the recall metric from BSH case study. Values above the diagonal show the Holm's post hoc for each pair of strategies (row and column), with values below 0.05 being statistically significant. Values not significant are highlighted in grey. Values below the diagonal show the $AHat_{12}$ value, indicating the number of times that the first strategy (row) performs better than the second strategy (column) in terms of recall. Values outperforming the baseline are highlighted in grey.

	Baseline	SP	SDP	DP	DDP	Death	Strong	Closed	Add	Remove
Baseline	-	1	1	0.06287	1	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.1x10^{-12}$	0.00028	$\ll 2x10^{-16}$
SP	42.97%	-	1	1	1	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.9x10^{-07}$	$1.3x10^{-11}$
SDP	50.00%	56.00%	-	0.12801	1	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.2x10^{-13}$	$7.6x10^{-05}$	$2.2x10^{-15}$
DP	38.70%	46.14%	40.24%	-	0.17941	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$4.8x10^{-11}$	$6.8x10^{-08}$
DDP	49.04%	55.01%	49.32%	59.09%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$3.5x10^{-14}$	$3.6x10^{-05}$	$8.2x10^{-15}$
Death	93.83%	93.95%	88.65%	96.17%	91.57%	-	1	0.00331	$7.7x10^{-11}$	$\ll 2x10^{-16}$
Strong	84.62%	86.78%	81.53%	89.26%	82.81%	46.19%	-	0.12801	$1.1x10^{-07}$	$\ll 2x10^{-16}$
Closed	75.34%	78.83%	73.22%	81.83%	74.13%	37.24%	41.05%	-	0.01154	$\ll 2x10^{-16}$
Add	63.82%	69.89%	62.27%	74.91%	63.36%	10.18%	22.35%	33.68%	-	$\ll 2x10^{-16}$
Remove	12.11%	18.05%	14.90%	19.02%	15.21%	0%	0.85%	3.43%	3.45%	-

Table 15 Results of the statistical analysis for the precision metric from CAF case study. Values above the diagonal show the Holm's post hoc for each pair of strategies (row and column), with values below 0.05 being statistically significant. Values not significant are highlighted in grey. Values below the diagonal show the $AHat_{12}$ value, indicating the number of times that the first strategy (row) performs better than the second strategy (column) in terms of recall. Values outperforming the baseline are highlighted in grey.

	Baseline	SP	SDP	DP	DDP	Death	Strong	Closed	Add	Remove
Baseline	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.1x10^{-14}$	$1.9x10^{-07}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
SP	37.08%	-	$2.9x10^{-10}$	$\ll 2x10^{-16}$	$1.6x10^{-11}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$3.2x10^{-05}$
SDP	28.11%	39.59%	-	0.0002	0.6577	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	0.0762
DP	22.94%	34.03%	44.88%	-	0.0010	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$4.5x10^{-09}$
DDP	26.01%	37.82%	48.15%	53.28%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	0.0474
Death	6.33%	12.46%	19.45%	22.13%	20.68%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.6x10^{-11}$	$\ll 2x10^{-16}$
Strong	70.61%	81.77%	87.33%	90.36%	89.93%	99.04%	-	0.0474	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Closed	67.21%	78.38%	84.69%	88.42%	87.21%	98.48%	45.19%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Add	10.73%	19.01%	27.89%	31.29%	29.36%	61.04%	2.51%	3.53%	-	$\ll 2x10^{-16}$
Remove	30.43%	46.79%	59.23%	66.78%	61.03%	90.23%	7.45%	11.26%	83.33%	-

Table 16 Results of the statistical analysis for F-Measure metric from BSH case study. Values above the diagonal show the Holm's post hoc for each pair of strategies (row and column), with values below 0.05 being statistically significant. Values not significant are highlighted in grey. Values below the diagonal show the $AHat_{12}$ value, indicating the number of times that the first strategy (row) performs better than the second strategy (column) in terms of F-Measure. Values outperforming the baseline are highlighted in grey.

	Baseline	SP	SDP	DP	DDP	Death	Strong	Closed	Add	Remove
Baseline	-	0.00015	$7.9x10^{-06}$	$1.0x10^{-09}$	0.09738	$\ll 2x10^{-16}$	$2.3x10^{-10}$	$5.6x10^{-06}$	$\ll 2x10^{-16}$	0.07066
SP	31.29%	-	1	0.23328	0.48220	$1.2x10^{-13}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.3x10^{-11}$	0.51503
SDP	29.39%	47.88%	-	0.51503	0.15086	$9.9x10^{-12}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$8.1x10^{-10}$	0.19615
DP	24.07%	41.98%	43.82%	-	0.00092	$2.0x10^{-07}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$6.7x10^{-06}$	0.00150
DDP	37.32%	56.14%	58.13%	64.10%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$4.2x10^{-13}$	$\ll 2x10^{-16}$	1
Death	10.52%	19.06%	21.03%	21.04%	16.22%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	1	$\ll 2x10^{-16}$
Strong	77.94%	89.92%	91.07%	93.81%	86.55%	97.25%	-	0.51503	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Closed	69.43%	82.64%	83.92%	87.20%	78.83%	93.90%	42.93%	-	$\ll 2x10^{-16}$	$1.5x10^{-13}$
Add	11.96%	21.78%	23.68%	24.28%	18.45%	56.50%	3.02%	6.83%	-	$\ll 2x10^{-16}$
Remove	33.87%	56.55%	58.71%	67.76%	48.95%	91.76%	9.52%	17.98%	90.13%	-

Table 17 Results of the statistical analysis for the F-Measure metric from BSH case study. Values above the diagonal show the Holm's post hoc for each pair of strategies (row and column), with values below 0.05 being statistically significant. Values not significant are highlighted in grey. Values below the diagonal show the $AHat_{12}$ value, indicating the number of times that the first strategy (row) performs better than the second strategy (column) in terms of F-Measure. Values outperforming the baseline are highlighted in grey.

	Baseline	SP	SDP	DP	DDP	Death	Strong	Closed	Add	Remove
Baseline	-	$2.4x10^{-13}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
SP	37.11%	-	$2.3x10^{-08}$	$8.1x10^{-13}$	$9.9x10^{-09}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	1
SDP	29.23%	40.54%	-	0.59	1	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$3.1x10^{-06}$
DP	22.92%	33.15%	42.91%	-	0.64	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$5.0x10^{-10}$
DDP	29.72%	41.44%	51.08%	58.32%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.6x10^{-06}$
Death	25.53%	26.91%	28.47%	28.90%	27.32%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$6.2x10^{-12}$	$\ll 2x10^{-16}$
Strong	77.37%	88.71%	91.88%	94.47%	92.84%	7726%	-	1	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Closed	74.71%	85.72%	89.71%	92.89%	90.38%	77.74%	47.27%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Add	14.03%	21.68%	30.40%	35.97%	29.04%	68.57%	3.20%	3.73%	-	$\ll 2x10^{-16}$
Remove	38.10%	52.91%	63.78%	72.43%	62.71%	77.81%	7.02%	10.98%	85.14%	-

Table 18 Results of the statistical analysis for the MCC metric from BSH case study. Values above the diagonal show the Holm's post hoc for each pair of strategies (row and column), with values below 0.05 being statistically significant. Values not significant are highlighted in grey. Values below the diagonal show the $AHat_{12}$ value, indicating the number of times that the first strategy (row) performs better than the second strategy (column) in terms of MCC. Values outperforming the baseline are highlighted in grey.

	Baseline	SP	SDP	DP	DDP	Death	Strong	Closed	Add	Remove
Baseline	-	$3.4x10^{-05}$	$1.2x10^{-07}$	$8.2x10^{-09}$	0.02147	$\ll 2x10^{-16}$	$1.5x10^{-10}$	$3.4x10^{-05}$	$\ll 2x10^{-16}$	0.13849
SP	31.13%	-	0.82880	0.62474	0.55499	$5.8x10^{-07}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.5x10^{-10}$	0.14695
SDP	31.48%	49.73%	-	0.99636	0.05639	0.00012	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.3x10^{-07}$	0.00752
DP	23.69%	41.81%	42.21%	-	0.01583	0.00083	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.5x10^{-06}$	0.00144
DDP	37.59%	56.35%	56.57%	64.39%	-	$1.8x10^{-11}$	$\ll 2x10^{-16}$	$1.8x10^{-13}$	$5.7x10^{-16}$	0.99636
Death	14.64%	27.76%	29.68%	32.70%	23.78%	-	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	0.69584	$1.3x10^{-13}$
Strong	79.79%	91.54%	90.62%	94.94%	87.74%	96.92%	-	0.21764	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Closed	70.78%	83.92%	83.17%	88.29%	79.65%	93.03%	42.32%	-	$\ll 2x10^{-16}$	$2.4x10^{-11}$
Add	14.23%	26.65%	28.07%	31.66%	22.80%	47.20%	2.84%	6.89%	-	$\ll 2x10^{-16}$
Remove	34.31%	57.05%	57.14%	68.19%	48.82%	83.06%	7.74%	17.08%	83.85%	-

Table 19 Results of the statistical analysis for the MCC metric from BSH case study. Values above the diagonal show the Holm's post hoc for each pair of strategies (row and column), with values below 0.05 being statistically significant. Values not significant are highlighted in grey. Values below the diagonal show the $AHat_{12}$ value, indicating the number of times that the first strategy (row) performs better than the second strategy (column) in terms of MCC. Values outperforming the baseline are highlighted in grey.