Utilizing Automatic Query Reformulations as Genetic Operations to Improve Feature Location in Software Models

Francisca Pérez, Tewfik Ziadi, Carlos Cetina

Abstract—In the combination of Model-Driven Engineering (MDE) and Search-Based Software Engineering (SBSE), genetic operations are one of the key ingredients. Our work proposes a novel adaptation of automatic query reformulations as genetic operations that leverage the latent semantics of software models (the cornerstone artefact of MDE). We analyze the impact of these reformulation operations in a real-world industrial case study of feature location in models. As baselines, we use: 1) the widespread single-point crossover plus random mutation; and 2) mask crossover plus random mutation, which is the best performer for feature location in models. We also perform a statistical analysis to provide quantitative evidence of the impact of the results and to show that this impact is significant. Our reformulation operations improve the results of the best baseline by 37.73% in recall and 14.08% in precision. These results are relevant for the task of feature location in models (one of the main activities performed during software maintenance and evolution). Furthermore, given that the only requirement to apply our approach is term availability in models, our work opens a new research direction to improve more tasks in MDE such as bug location or requirements traceability.

Index Terms—Model-Driven Engineering, Search-Based Software Engineering, Automatic Query Reformulations

1 Introduction

Increasingly, the MDE community is paying more attention to the techniques offered by the SBSE community. This has led to the combination of MDE and SBSE techniques in a new field of study known as Search-Based Model-Driven Engineering (SBMDE) [1], [2] where search-based techniques are applied to MDE related tasks, such as optimizing models [3], automatically generating test procedures [4], maintaining consistency between models and metamodels [5], and feature location in models [6].

In the case of Feature Location (FL) in models, where the term 'feature' refers to a specific functionality or characteristic of a product, the goal is to identify the model fragment that is associated with that specific functionality. FL can arguably be seen as one of the most frequent maintenance tasks undertaken by developers [7], [8], [9], [10] since software maintenance and evolution involves adding new features to programs, improving existing functionalities, and removing unwanted functionalities. To this end, it is essential [7], [8] that developers find the elements of the system's features.

Fig. 1 shows an example of FL. The upper part of the figure shows inputs for FL: a feature description (produced by a domain engineer), and a set of product models that are made up of model elements. Of all these model elements, some model elements (model elements 1-3 of the figure) are relevant for the feature description and others are not (the rest of the model elements of the figure). Fig. 1 highlights in gray the result of FL, which is the set of model elements (i.e., model fragment) that is identified as relevant for the feature description.

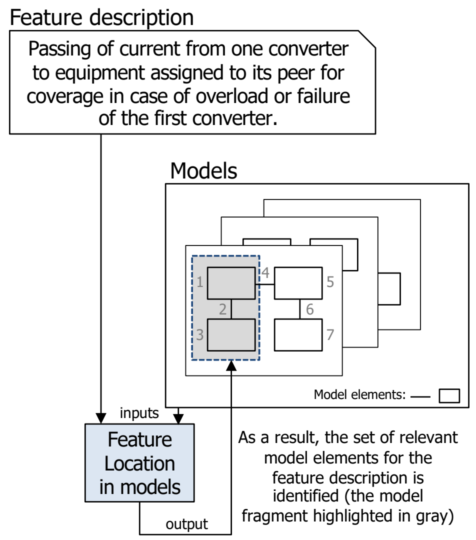

Feature description
Passing of current from one converter to equipment assigned to its peer for coverage in case of overload or failure of the first converter.
Models
1
2
3
4
5
6
7
Model elements:
inputs
Feature Location in models
output
As a result, the set of relevant model elements for the feature description is identified (the model fragment highlighted in gray)

Fig. 1. Example of Feature Location in models

Only three key ingredients are needed to apply SBSE: 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 evaluated (by the fitness function) in an iterative process until optimal (or near-optimal) solutions to the problem are found.

• *F. Pérez and C. Cetina are with the SVIT Research Group of Universidad San Jorge, Zaragoza, Spain. E-mail: {mfperez, ccetina}@usj.es. T. Ziadi is with the LiP6 of Sorbonne Université, Paris, France. E-mail: tewfik.ziadi@lip6.fr*

Manuscript received September X, 2019; revised .

In the case of FL in models, the candidate solutions are encoded as positions of a bit string where the positions of the relevant model fragment are set to 1. The fitness function is defined as the similarity between the text of each model fragment and the feature description. Finally, the operators are crossover and mutation genetic operations.

When SBSE is applied to MDE-related tasks, both the encoding and the fitness function strongly depend on the task at hand. However, when it comes to the operators, genetic operations (single-point crossover and random mutation) are the prominent choice of SBMDE works [1]. These operations randomly mix information from existing candidate solutions to produce new candidate solutions.

In this work, we argue that, regarding models, genetic operations could leverage the latent semantics that models hold instead of randomly generating new candidate solutions. To achieve this goal, we propose a novel adaptation of automatic query reformulations as genetic operations. Specifically, we utilize the reformulation of the terms of model elements to determine the model elements that will form new candidate solutions.

We analyze the impact of utilizing reformulations as genetic operations in a real-world industrial case study of feature location in models. Our industrial partner, Construcciones y Auxiliar de Ferrocarriles (CAF) [1], is a worldwide leader in train manufacturing that uses models to generate the firmware that controls their trains. For the reformulations, we not only evaluate *expansion* by Rochio (arguably the most popular reformulation technique), but we also evaluate *replacement* by domain specificity, *reduction*, and *selection* by TextRank. Our baselines include single-point crossover plus random mutation (the most popular choice in SBMDE) and mask crossover plus random mutation (the genetic operations that achieve the best results in feature location in models).

To perform these analyses, the case study of our industrial partner includes both the models of software that control and manage the trains and the oracle (the realization of features validated by our industrial partner). We compare the results of the four variants (expansion, replacement, reduction, and selection) with the baselines in terms of recall, precision, and the F-measure. Finally, we perform a statistical analysis (following the guidelines by Arcuri and Briand [11]) in order to provide quantitative evidence of the impact of the results and to show that this impact is significant.

The results show that utilizing automatic query reformulations as genetic operations pays off in the location of features in models. Our replacement variant improves the results of the best baseline by 37.73% in recall and 14.08% in precision. We cannot claim that reformulations will completely replace the widespread use of genetic operations in SBMDE, but this work provides an alternative in the operators department to significantly improve results in SBMDE. Since the only requirement for applying our approach is the availability of terms in model elements, it has the potential to influence most SBMDE works. However, not all models have the same number of terms in their elements. For instance, models that are used to generate code have more terms than those that are only used for sketching a system. Our future work includes new hybrid variants (crossover plus mutation plus reformulation) to address models with a low number of terms.

In summary, our paper claims that automatic query reformulations can be utilized as genetic operations in the context of SBMDE. Automatic query reformulations (expansion, replacement, reduction, and selection) are already discussed in the literature in other studies ([12], [13], [14], [15]). However, it is important to clarify that to date automatic query reformulations have not been used as genetic operations within an evolutionary algorithm. Neither the surveys of SBMDE [1] since 1998 nor the surveys of genetic operations [16], [17], [18] have reported genetic operations utilizing automatic query reformulations. Specifically, we claim that:

1) Utilizing automatic query reformulations as genetic operations improves the results in the task of FL in software models compared to the most popular choices of genetic operations in SBMDE (single-point crossover and random mutation). This is relevant for the SBMDE community since FL is an essential task for software maintenance and evolution [7], [8].
2) There are significant differences in performance in terms of the solution quality of the automatic query reformulations when they are utilized as genetic operations. We provide recommendations on when to use each operator.
3) Our adaptation of automatic query reformulations to genetic operations is in terms of software models. In other words, our adaptation has been designed to be generic and applicable not only to feature location and to the domain of our industrial partner but also to other tasks and domains. Since software models are the main artifacts of SBMDE, works from the SBMDE community can benefit from our adaptation.
4) Although our work focuses on models, our results can also motivate SBSE researchers to evaluate whether automatic query reformulations as genetic operations can improve the quality of the solutions when other artifacts (such as code, requirements, or tests) are used.

The rest of the paper is organized as follows: Section 2 reviews the related work. Section 3 introduces the domain of our industrial partner and genetic operations. Section 4 presents our proposed evolutionary algorithm, which uses the model fragment reformulation. Section 5 presents model fragment reformulation variants. Section 6 describes the evaluation, and Section 7 shows the results. Section 8 discusses the results. Section 9 describes the threats to validity. Finally, Section 10 concludes the paper.

2 Related work

There is a recent survey [1] that includes related SBMDE works since 1998. For each of these works, the survey includes information about encoding, search technique, and fitness. However, the survey does not include information

1. www.caf.net/en

about the genetic operations (crossover and mutation) used in each SBMDE work. Therefore, we have reviewed all of the SBMDE works in the survey in order to provide information about the genetic operations as well as to determine if the work has been evaluated on an industrial scale.

Table 1 shows the related SBMDE works (Column 1), the genetic operations (Column 2 for crossover and Column 3 for mutation), and the industrial scale (Column 4). With regard to the genetic operations, we use a hyphen to indicate that the work does not provide detailed information about the operation, and we use a cross mark to indicate that the work explicitly mentions that it does not use the operation. With regard to the industrial scale, we use either a check mark to indicate that the work explicitly mentions that the evaluation performed involves industry or a cross mark otherwise.

The upper part of the table shows the works following the same order used in the survey. As the table shows, 81.08% of the works that provide information about the crossover operation randomly select a single-point for crossover, and 98.11% of the works that provide information about the mutation operation indicate that it is done randomly. Therefore, the genetic operations of single-point crossover and random mutation are the most popular choices in these SBMDE works.

The lower part of Table 1 shows the works that deal with feature location in models within the SBMDE community. Feature location in models is the task that we are using in this work to evaluate automatic query reformulations as genetic operations. These are previous works of our SVIT research group, where feature location in models is a central topic. In [84], a combination of Formal Concept Analysis and Latent Semantic Analysis is evaluated to guide the evolutionary algorithm. In [6], the fitness function (the similarity to the feature description) is fixed and five search strategies are evaluated (Evolutionary Algorithm, Random Search, steepest Hill Climbing, Iterated Local Search with restarts, and a hybrid between Evolutionary Algorithm and Hill Climbing). In [85], a learning-to-rank approach is proposed to improve the fitness function to locate features in models, whereas the candidate model fragments are randomly generated. In [86], models at run-time are proposed to be used for increasing the information for feature location. This work also randomly generates new candidate solutions that are used to locate features in models. In [87], the study provides real measurements of location problems as a help to other researchers in the design of synthetic location problems. Their location strategy relies on a linguistic rule-based approach and Latent Semantic Indexing.

As Table 1 shows, none of the above SBMDE works propose novel genetic operations that leverage models. In contrast, our work leverages the latent semantics of software models utilizing query reformulations as genetic operations (as the final row of the table shows). Furthermore, our work is evaluated at an industrial scale (CAF models from real trains with more than 1000 model elements each), whereas only 25% of the above works are evaluated at an industrial scale.

Outside the SBMDE community, there are many crossover and mutation operations proposed [16], [17], [18]. These surveys identify more than 50 crossover operations

TABLE 1
Genetic operations of related SBMDE works

SBMDE work	Genetic operations		Industrial scale
	Crossover	Mutation	
Kessentini et al. [19]	-	Random	✗
Kessentini et al. [20]	-	Random	✗
Kessentini et al. [21]	-	Random	✓
Faunes et al. [22], [23]	N point	Random	✗
Baki et al. [24]	Single-point	Random	✗
Saada et al. [5]	Single-point	Random	✗
Kessentini et al. [25]	-	Random	✓
Mkaouer et al. [26]	Single-point	Random	✗
Gyapay et al. [27]	-	-	✗
Denil et al. [3]	-	Random	✓
Abdeen et al. [28]	Single-point	Random	✗
Fleck et al. [29]	Single-point	Random	✗
Nisbet et al. [30]	-	-	✗
Cooper et al. [31]	Single-point	Random	✗
Fatiregun et al. [32]	-	-	✗
Fatiregun et al. [33]	Single-point	Random	✗
Kulkarni et al. [34]	Single-point	Random	✗
Fatiregun et al. [35]	Single-point	Random	✓
Fleurey et al. [4]	-	Random	✗
Cadavid et al. [36]	-	Random	✓
Wang et al. [37]	-	Random	✓
Shelburg et al. [38]	-	Random	✗
Jilani et al. [39]	-	Random	✗
Sahin et al. [40]	Single-point	Random	✗
Rose and Poulding [41]	-	Random	✗
Batot [42]	Single-point	Random	✗
Kessentini et al. [43], [44]	Single-point	Random	✗
Faunes et al. [45]	Single-point	Random	✗
Kessentini et al. [46]	Single-point	Random	✓
Mansoor et al. [47]	Single-point	Random	✗
Debreceni et al. [48]	-	-	✗
Ghannem et al. [49]	Single-point	Random	✗
Ghannem et al. [50]	Single-point	Random	✗
Amoui et al. [51]	Single-point	Random	✗
Jensen and Cheng [52]	-	-	✓
ben Fadhel et al. [53]	Single-point	Random	✓
Moghadam and Cinneide [54]	-	-	✗
Ipate and Lefticaru [55]	Single-point	Random	✗
Li and Lam [56]	-	-	✗
Doungsa-ard et al. [57]	Double point	Random	✗
Lefticaru and Ipate [58]	Heuristic	Random	✗
Shirole et al. [59]	Single-point	Random	✗
Ali et al. [60], [61]	Single-point	Random	✓
Iqbal et al. [62]	Single-point	Random	✓
Hänsel [63]	-	-	✗
Kalaji et al. [64]	Single-point	Random	✗
Yano et al. [65]	✗	Random	✗
Zhang et al. [66]	-	-	✗
Li and Lam [67]	-	-	✗
Xu et al. [68]	-	-	✗
Shirole et al. [69]	-	-	✗
Shirole and Kumar [70]	Single-point	Random	✗
Prasanna and Chandran [71]	Single-point	Injection	✗
Farooq and Lam [72]	Double point	Random	✓
Hemmati et al. [73]	Single-point	Random	✓
Sabharwal et al. [74]	Single-point	Random	✗
Goldsby and Cheng [75], [76]	-	Random	✗
Goldsby et al. [77]	-	Random	✗
Ramirez et al. [78]	Double point	Random	✗
McKinley et al. [79]	-	Random	✗
Cheng et al. [80]	-	-	✗
Williams et al. [81]	-	Random	✗
Andrade and Macêdo [82]	-	-	✗
Harman et al. [83]	-	-	✗
Font et al. [84]	Mask	Random	✓
Font et al. [6]	Mask	Random	✓
Marcén et al. [85]	-	Random	✓
Arcega et al. [86]	-	Random	✓
Ballarín et al. [87]	-	-	✓
Pérez et al. [88]	Mask	Random	✓
Pérez et al. [89]	Mask	Random	✓
Our work	Query Reformulations		✓

and more than 20 mutation operations. However, none of these operations leverages the latent semantics of MDE models as our work does.

Moreover, works from the code retrieval community [14], [90], [91] reformulate an initial query with the goal of improving the performance of retrieving code. In contrast, our work uses query reformulation techniques for a different purpose. In our work, the query reformulation techniques are used as genetic operations of an evolutionary algorithm. Thus, the genetic operations could leverage the latent semantics instead of randomly generating new candidate solutions.

Next, we discuss the overlapping between this work and our previous works [88], [89]. Our previous works [88], [89] and this work address the task of Feature Location. Moreover, the three works rely on an Evolutionary Algorithm to identify the model fragment that is associated with an input search query. In addition, the three works use Natural Language Processing since text in natural language is processed to locate features. Fig. 2 shows an overview of the overlap among the three works (the Natural Language Processing and the Evolutionary Algorithm).

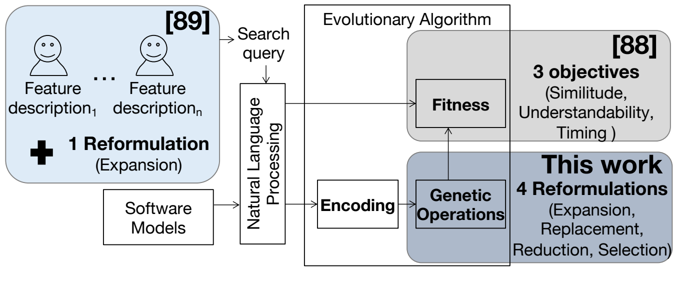


Fig. 2. Overlap and differences with our previous works

However, these three works address three completely different aspects as the shaded parts of Fig. 2 show. The aspect that is addressed in [89] focuses on supporting collaboration among software engineers to locate features. Collaboration is a useful and often necessary component in industrial contexts where a vast amount of software is accumulated over the years and this software has been developed and maintained by different individuals. In [89], we do not claim collaboration should be systematically applied to every case. Collaboration is necessary when the feature significantly transcends the knowledge of a single software engineer.

In [89], an automatic query reformulation technique (Expansion) is used to produce a search query that combines different feature descriptions. In this work, we use the same reformulation technique (Expansion), but its use has a completely different purpose, which is not related to collaboration (as the bottom right part of Fig. 2 shows). In addition, it is important to highlight that not only does this work use Expansion for a different purpose, but it also uses three other automatic query reformulation techniques (Replacement, Reduction, and Selection).

As the upper right part of Fig. 2 shows, the aspect that is explored in [88] focuses on analyzing the best fitness objective (Similitude, Understandability, and Timing) in common, relevant tasks in the Software Engineering field (Requirement Traceability, Bug Localization, and Feature Location). In this work, we use one of the fitness objectives (Similitude) that is explored in [89], but the contribution of this work does not focus on the fitness objective. This work focuses on the genetic operations of the evolutionary algorithm (as the bottom right part of Fig. 2 shows). The genetic operations that are used in the evolutionary algorithm of the previous works [88], [89] randomly mix information from existing candidate solutions, which is the most popular choice of genetic operations in the SBMDE community [1]. In contrast, the novel genetic operations that are proposed here do not randomly mix information. These novel genetic operations leverage the latent semantics of software models using an automatic query reformulation technique.

3 Background

This section introduces the railway domain of our industrial partner as well as a running example of the software model, the model fragment that realizes a feature, and the model fragment encoding with bit strings. In addition, this section introduces single-point crossover and random mutation, which is the most popular choice of genetic operations in the SBMDE community.

3.1 Railway domain and running example

Models, which are high-level specifications of systems, have progressively gained importance as the primary artefacts for generating products for major players in the software engineering field (i.e., tool vendors, researchers, and enterprise software developers). This is triggered by the demand for more abstract approaches than mere coding [92]. Models raise the abstraction level using terms that are much less bound to the underlying implementation and technology and that are much closer to the problem domain [92]. Abstract approaches provide benefits for improving productivity, while ensuring quality and performance [92] in industrial contexts.

Fig. 3 depicts a product model excerpt that is taken from a real-world train. The product model is specified using the Domain-Specific Language (DSL) that formalizes the train control and management of the products manufactured by our industrial partner. The DSL has the expressiveness required to describe both the interaction between the main pieces of installed equipment and the non-functional aspects related to regulation. It will be used throughout the rest of the paper to present a running example. For the sake of understandability and legibility, we present a simplified equipment-focused subset of the DSL (due to intellectual property rights concerns), and we do not show the terms that are included in the relationships between model elements.

Specifically, the example in the figure presents a converter assistance scenario where two High Voltage Equipment devices (*Pantograph Front* and *Pantograph Rear*) collect energy from the overhead wires and send it to their respective Contactors (*Breaker* and *Circuit*), which in turn send it to their independent Voltage Converters (*Converter* and *Peer Coverage*). The converters then power their assigned Consumer Equipment: the HVAC (air conditioning system) on the left, and the PA (public address system) and CCTV (cameras system) on the right.

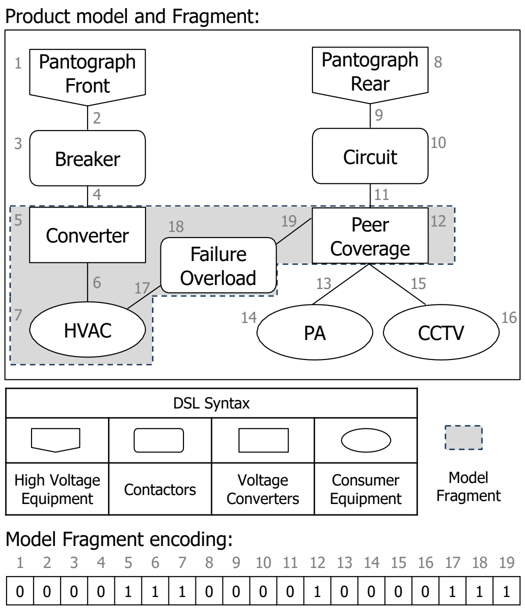


Fig. 3. Example of a product model, model fragment, and encoding

In [7], feature location is defined as follows: ”Feature location is the task of finding the source code in a system that implements a feature”. Since our work targets software models instead of source code, we adapt the above definition as: Feature location is the task of finding the model elements (i.e., a model fragment) in a system that implements a feature. The model elements of Fig. 3 highlighted in gray comprise an example model fragment, which includes one contactor (*Failure overload*) that connects the voltage converter (*Peer Coverage*) to a Consumer Equipment (*HVAC*) that is assigned to *Converter*. This model fragment implements the ’converter assistance’ feature, which allows the passing of current from one converter to equipment that is assigned to its peer for coverage in case of the overload or failure of the first converter.

A model fragment is encoded in a bit string to be easily manipulated. The bit string contains as many positions as elements in the parent product model. Each position in the string has two possible values: 0, if the element does not appear in the fragment; or 1, if the element does appear in the fragment. In Fig. 3, elements 5-7, 12, and 17-19 comprise the model fragment, so the corresponding values are set to ’1’ in its binary string representation.

Although this example of feature location in a model fragment in a model-driven industrial context may look easy, an industrial context tends to have a myriad of products that require large and complex models behind it, which makes manual feature location very complex. For example, the models of our industrial partner that specify each train unit include several thousand elements. In addition, manual feature location becomes even more complex because the models are created and maintained by different software engineers over long periods of time since the maintenance contracts of trains last 25 years. Under these complex conditions, an algorithm that automatically retrieves the model fragment that realizes the feature description provided as input is greatly needed.

3.2 Genetic operations in an evolutionary algorithm

To address feature location in models, evolutionary algorithms have obtained good results [6], [84], [88], [89]. The execution of an evolutionary algorithm involves iterating over a population (e.g., a population of model fragments) to generate new individuals (e.g., a model fragment) using genetic operations (e.g., crossover and mutation).

The crossover operation imitates the sexual reproduction that is followed by some living beings in nature to breed new individuals. In other words, two individuals mix their genomic information to give birth to a new individual that holds some genetic information from one parent and some from the other one. This could make the new individual adapt better (or worse) to its living environment, depending on the genetic information inherited from its parents. Following this idea, the crossover operation consists of randomly selecting a point on both parents and swapping the encoding from the selected point. As a result, two individuals are generated.

The upper part of Fig. 4 depicts an example of the single-point crossover operation applied to bit strings that represent the encoding of two model fragments (*Parent A* and *Parent B*). The selected point is 2 (half of the size of the parent model fragments). As the figure shows, two model fragments (MF_1 and MF_2) are generated as a result. MF_1 holds the first half of its encoding from Parent A, whereas the second half of its encoding is from Parent B. In contrast, MF_2 holds the first half of its encoding from Parent B, whereas the second half of its encoding is from Parent A.

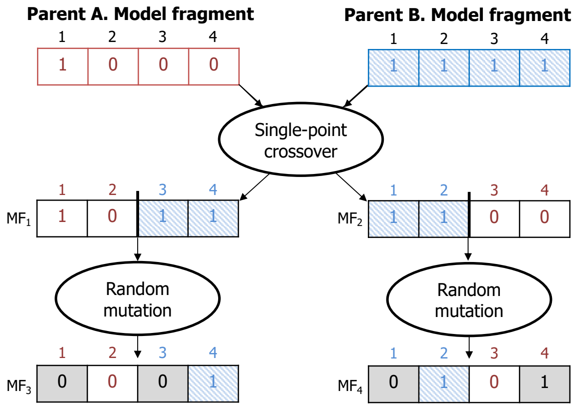


Fig. 4. Widespread Genetic Operations using encoding at the bit level

The mutation operation is used to imitate the mutations that randomly occur in nature when new individuals are born. In other words, a new individual holds a small difference with regard to its parent that could make it adapt better (or worse) to its living environment. Following this idea, the random mutation operation involves a probability that serves to determine whether a bit in a string is changed from its original state. The random mutation operation also involves the generation of a random value for each bit in the string. This random value indicates whether or not a specific bit will be modified.

The lower part of Fig. 4 shows two examples of the random mutation operation. On the one hand, the example in the left part of the figure shows that the encoding of MF_1 is taken as parent and MF_3 is generated. The bits that are randomly changed in MF_3 with regard to MF_1 are highlighted in gray (bits at Positions 1 and 3). On the other hand, the example in the right part of the figure shows that the encoding of MF_2 is taken as parent and MF_4 is generated. The bits that are randomly changed in MF_4 with regard to MF_2 are highlighted in gray (bits at Positions 1 and 4).

To assess the relevance of each individual of the population (e.g., model fragment) in relation to the provided query (e.g., feature description), a fitness function based on Information Retrieval (IR) techniques is applied. This is because each individual can be viewed as a textual document, which is made up of a set of terms. For each document, textual similarities are computed with regard to the query. As a result, a ranking of the individuals of the population is obtained, which is sorted from the highest to the lowest fitness value.

4 The Evolutionary Algorithm for Feature Location using the Model Fragment Reformulation

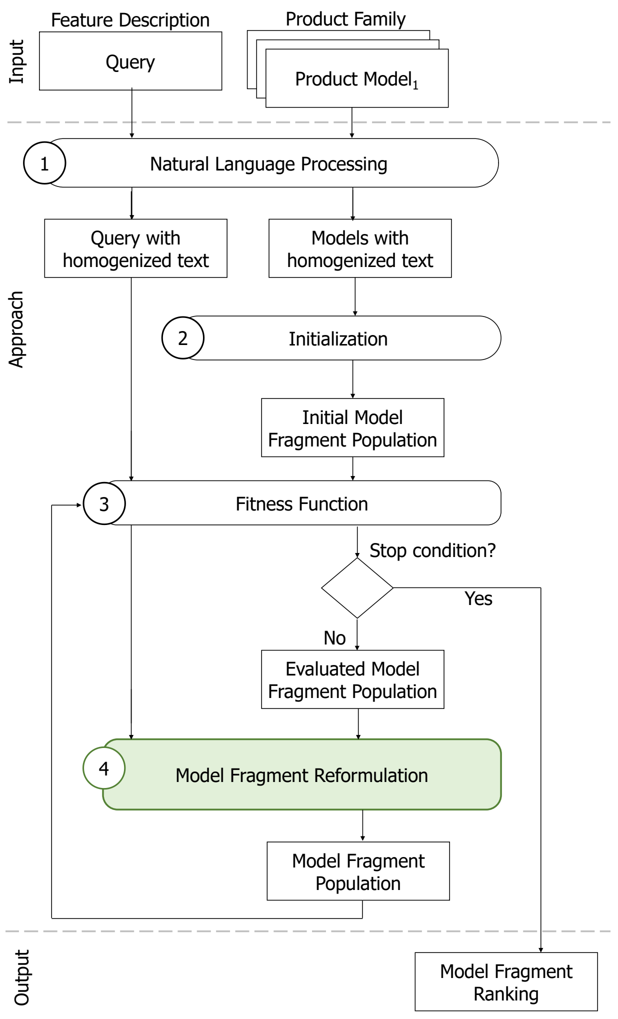
Our approach utilizes automatic query reformulations as the genetic operations of an evolutionary algorithm for feature location in models. Fig. 5 shows an overview of our approach. The upper part of the figure shows the inputs: the query, which is the feature description, and the product models where the model fragment that realizes the feature description must be located. The middle part of Fig. 5 shows the four steps of the evolutionary algorithm: Natural Language Processing (to homogenize the texts of the inputs); Initialization (to create an initial model fragment population); Fitness Function (to assess the similitude of a model fragment to the input feature description); and Model Fragment Reformulation (to generate new model fragments that are added to the population). The bottom part of the figure shows the output of the algorithm, which is a ranking of model fragments that is in descending order based on the similitude to the input feature description.

In the following subsections, we describe each step of the evolutionary algorithm.

4.1 Natural Language Processing

This step homogenizes the text that is extracted from the inputs of the evolutionary algorithm (the feature description and the models) through Natural Language Processing (NLP) techniques. Text homogenization using NLP techniques is often regarded as beneficial and is a frequent practice [93].

Text homogenization is performed as follows. 1) The text is tokenized (i.e., divided into words). As a result, the text is divided using a white space since it is a tokenizer that can usually be applied. More complex tokenizers, such as CamelCase naming, need to be applied for some sources. 2) The Parts-of-Speech (POS) tagging technique analyzes the words grammatically and infers the role of each word in the input text. The result is that each word is tagged in a category and some categories that do not provide relevant information can be removed (e.g., prepositions). 3) Stemming techniques unify the language that is used in the text by reducing each word to its root. This serves to group together different words that refer to similar concepts. For instance, plurals are turned into singulars (*circuits* to *circuit*). 4) The Domain Term Extraction and Stopword Removal techniques are applied to automatically filter terms in or out. We selected these NLP techniques to homogenize the text of the inputs of the evolutionary algorithm since they obtained the best results in a previous work [94].



Feature Description
Product Family
Input
Query
Product Model$_1$
1
Natural Language Processing
Query with homogenized text
Models with homogenized text
Approach
2
Initialization
Initial Model Fragment Population
3
Fitness Function
Stop condition?
Yes
No
Evaluated Model Fragment Population
4
Model Fragment Reformulation
Model Fragment Population
Output
Model Fragment Ranking

Fig. 5. Overview of the Evolutionary Algorithm for Feature Location using the model fragment reformulation

For example, the following feature description of our industrial partner *"Passing of current from one converter to the HVAC that is assigned to its peer converter for coverage in case of overload or failure of the first converter"* is homogenized into the following terms: *current, convert, hvac, peer, convert, coverag, overload, failur*, and *convert*.

In a model, the text homogenization is performed for the text of each model element using the same techniques as those used to homogenize a feature description. For example, the homogenized terms of Model element 18 in Fig. 3 are: *failur, overload*.

4.2 Initialization

This step generates an initial population of model fragments from the input set of product models. To generate the initial population of model fragments, model elements of a product model are randomly extracted and added to a collection of model fragments. This random technique is the one commonly used in the SBMDE community and in evolutionary algorithms in general [6].

Each generated model fragment belongs to a product model. Once this step calculates the initial population of model fragments, it is used as input in the next step.

4.3 Fitness Function

This step assesses each of the produced candidate model fragments by ranking them according to a fitness function. The fitness function assesses the relevance of each model fragment in relation to the provided query. To do this, methods based on Information Retrieval (IR) techniques are applied [88]. Specifically, we applied Latent Semantic Indexing (LSI) [95], [96] to analyze the relationships between the model fragments in the population and the query. We selected LSI since this technique obtained the best results for FL tasks [97].

LSI constructs vector representations of a query and a corpus of text documents by encoding them as a term-by-document co-occurrence matrix. Each row in the matrix corresponds to *terms* and each column corresponds to *documents*, followed by the *query* in the last column. Each cell of the matrix holds the number of occurrences of a *term* inside a *document* or the *query*. In our approach, *terms* are all individual words from the homogenized text of model fragments and the *query*, whereas the *documents* are the Natural Language (NL) representations of model fragments and the *query*.

Once the matrix is built, it is normalized and decomposed into a set of vectors using a matrix factorization technique called Singular Value Decomposition (SVD) [95]. With SVD, one vector that represents the latent semantics of the texts is obtained for each *document* and for the *query*. Finally, the similarities between each *document* and the *query* are calculated as the cosine between both of their vectors, obtaining values between 0 and 1. Cosine values that are closer to 1 denote a higher degree of similarity, whereas cosine values that are closer to 0 denote a lower degree of similarity. Similarity increases as vectors point in the same general direction (as more *terms* are shared between *documents*).

Once the similitude scores are obtained, if the stop condition is met, the algorithm will stop returning the model fragment ranking. Usually, the stop condition can be a time slot, a fixed number of generations, or a trigger value of the fitness that makes the process terminate when reached. In addition, it is also possible to monitor the fitness values and determine when they are converging and no further improvements are being made by new generations. The stop condition greatly depends on the domain and the problem being solved; therefore, it is adjusted based on the results being output by the process.

Once the model fragment ranking is returned, new inputs (feature description and product models where the feature must be located) will be necessary to execute the algorithm again. As occurs in other works that retrieve text from an initial query, the results depend on the quality of the queries [98], [99]. The more properties and values of the elements are explicitly mentioned in the query, the closer the result will be to the search objective. Therefore, if irrelevant model fragments are obtained in the ranking, the model fragments of the ranking can be considered as a starting point from where solutions can be manually refined, or the query may be refined to automatically obtain different solutions.

If the stop condition is not yet met, the evolutionary algorithm will continue its execution towards the next step of the algorithm by providing the evaluated model fragment population as input.

4.4 Model fragment reformulation

This step generates new model fragments using a novel application of query reformulation as genetic operations of the evolutionary algorithm. The goal of using query reformulation strategies as genetic operations of the evolutionary algorithm is to take into account the latent semantics of the inputs of the reformulation. In addition, the fitness score of each model fragment of the population can be used to select the best candidates from the population, which can drive the model fragment reformulation.

Researchers in the field of FL have proposed a large variety of query reformulation strategies for an initial query [100] in order to improve the quality of the retrieved information (usually, text or code). Since not all of these techniques can be applied in our work (i.e., model-based corpus), we selected the query reformulation strategies using the following criteria: 1) we did not consider strategies that relied on sources of information that are external to the corpus, such as the web or ontologies, since our goal is to support FL tasks in industrial domains that are specific and have intellectual property rights concerns; and 2) we did not consider non-practical techniques that were based on algorithms with high computational complexity since our goal is to support users' daily FL tasks.

Since different reformulation strategies can be used to perform the model fragment reformulation, the quality of the resulting model fragments could differ. Our proposed model fragment reformulation variants are presented in the next section.

5 Model fragment reformulation variants

To compare how different model fragment reformulations affect the quality of the model fragment, we propose four variants. The variants are based on reformulation strategies that are found in the literature to reformulate the terms of an initial search query (expansion [12], replacement [13], reduction [14], and selection [15]).

5.1 Variant 1: Expansion

This variant applies a query reformulation strategy that is based on Rocchio's method [12], which is perhaps the most commonly used method for query reformulation [98]. In Rocchio's method, the order of the terms in the top K relevant documents is based on the sum of the importance

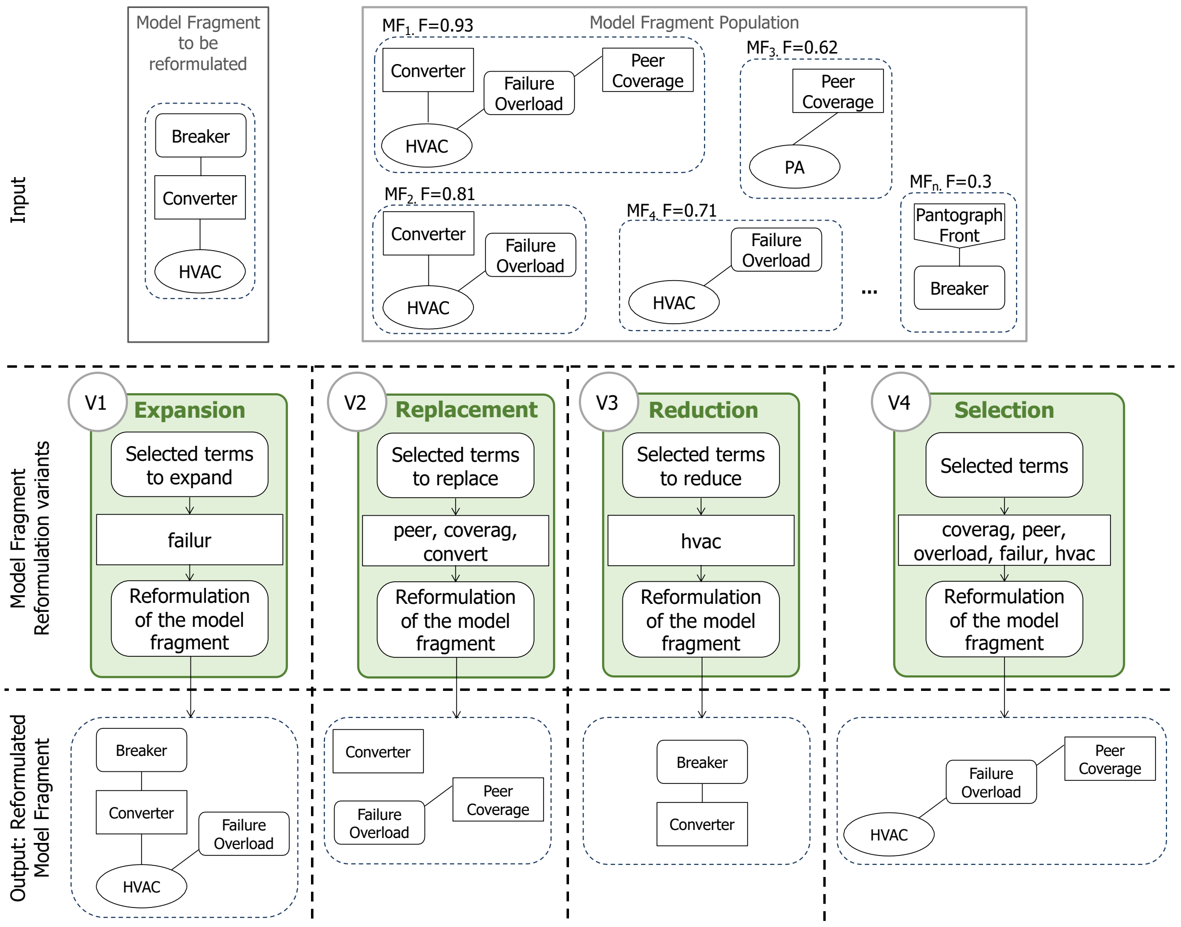


Fig. 6. Model fragment reformulation using four variants: (V1) expansion, (V2) replacement, (V3) reduction, and (V4) selection

of each term of the K documents relative to the corpus by using the following equation:

$$Rocchio = \sum_{d \in R} TfIdf(t, d)$$

where R is the set of top K relevant documents in the list of retrieved results, d is a document in R, and t is a term in d. The first component of the measure is the Term Frequency (Tf), which is the number of times that the term appears in a document and is an indicator of the importance of the term in the document compared to the rest of the terms in that document. The second component is the Inverse Document Frequency (Idf), which is the inverse of the number of documents in the corpus containing that term and indicates the specificity of that term for a document containing it. Once the terms in these K relevant documents are in order, the top N terms are selected to expand the input query.

To reformulate a model fragment by means of Rocchio expansion, this variant considers the following inputs: the input query (which is the model fragment to be reformulated), and the corpus (which is the model fragment population). First, the terms of the input query and the documents of the corpus are extracted. Second, the relevant and non-relevant documents are selected. The relevant documents are the top K model fragments that have the highest fitness score, whereas the non-relevant documents are the rest of the model fragments of the population. Third, the terms of the relevant documents are ordered using Rocchio's method. Fourth, the top N terms are selected. Finally, the model elements from the product model that match the selected terms are identified. These model elements are used to extend the model fragment to be reformulated.

Fig. 6-(V1) shows an example of the core of this variant, which sets the top three model fragments with the highest fitness score as relevant documents and reformulates the model fragment with the top term. The upper part of Fig. 6-(V1) shows the inputs: the model fragment to be reformulated (i.e., initial query) and a set of the model fragment population. Then, this variant selects the terms in order to expand the model fragment to be reformulated as follows:

1) It extracts the terms of the initial query *(breaker, convert, hvac)* and the terms of each model fragment of the population, (e.g., MF_1: *convert, hvac, failur, overload, peer, coverag*; MF_2: *convert, hvac, failur, overload*; and MF_4: *hvac, failur, overload*).
2) It selects the top three model fragments as relevant documents (MF_1 fitness score: 0.93; MF_2: 0.81; and MF_4: 0.71) and the rest of the model fragments of the population as non-relevant documents.
3) It orders the homogenized terms of the relevant documents using Rocchio's method: *convert, hvac, failur, overload, coverag, peer*.
4) It selects the top term that is not already included in the initial query (*failur*).

Once *failur* is selected as the term to expand, it is used to identify the model element of the product model that matches *failur* (shown in Fig. 3). As a result, the model element *Failure Overload* is identified in order to extend the model to be reformulated. The bottom part of Fig. 6-(V1) depicts the resulting reformulated model fragment.

5.2 Variant 2: Replacement

This variant is based on a query reformulation strategy that has been applied in source code retrieval [13]. The strategy selects a set of candidate terms to replace the initial query with a new set of query terms. These terms are domain-specific since domain specificity appears to be the dominant factor in improving the results [13].

Domain specificity (DS) measures the extent to which a term is specific to the domain document. In [13], domain specificity $DS(t,d)$ of term t in document d is estimated by comparing the relative frequency of the term within a domain-specific document versus its relative frequency in a general corpus of documents [101]. It is calculated as follows:

$$DS(t,d) = ln\left[\frac{freq(t,d)}{\sum_{t\in d} freq(t,d)} \Big/ \frac{freq(t,G)}{\sum_{t\in G} freq(t,G)}\right]$$

where $freq(t,d)$ is the total number of occurrences of term t in a given document d, the first component $freq(t,d)/\sum_{t\in d} freq(t,d)$ is the normalized number of occurrences of term t in the domain-specific document d, and the second component is the normalized number of occurrences of t in the general corpus of documents. DS is then calculated as the average value of all domain specificity values from each document of the collection:

$$DS(t) = \frac{1}{|D|}\sum_{d\in D} DS(t,d)$$

To reformulate a model fragment by means of replacement, this variant considers the terms of the model fragment to be reformulated as initial query . To select the terms that replace the N terms of the initial query, a DS value is calculated for each term of the relevant documents (i.e., terms from the top K model fragments that have the highest fitness score). Then, the top N terms with the highest DS value are selected. The selected terms are used to identify the model elements from the product model that comprise the reformulated model fragment.

Fig. 6-(V2) shows an example of the core of this variant, which takes both the model fragment to be reformulated (i.e., initial query) and the model fragment population as input. The terms that replace the three homogenized terms of the initial query (*breaker, convert, hvac*) are selected by calculating the DS value for each term of the relevant documents. In this example, the relevant documents are the top three model fragments with the highest fitness score: MF_1, MF_2, and MF_4. Afterwards, since the initial query has three terms, the top three terms with the highest DS value (*peer, coverag, convert*) are selected to identify the model elements from the product model to be included in the reformulated model fragment. The bottom part of Fig. 6-(V2) depicts the reformulated model fragment obtained as a result.

5.3 Variant 3: Reduction

This variant is based on a conservative automatic query reduction approach that has previously been used in software engineering [14], [102]. This approach consists of eliminating non-discriminating terms, which are those terms that appear in more than 25% of the documents in the corpus.

Recent research has shown that removing noisy terms from the query leads to substantial retrieval improvement [103]. Longer queries are typically used to express more sophisticated information needs. Nevertheless, longer queries are used to include important information as well as noise (i.e., terms that do more to confuse the search engine than support it in its task). Since the performance of most commercial and academic search engines deteriorates while handling longer queries [104], the aim of query reduction is to reduce long queries to shorter ones.

To reformulate a model fragment by means of reduction, its terms are extracted as the base query. Then, the terms of each document in the corpus (i.e., the terms of each model fragment of the population) are extracted and used to calculate the terms of the base query that are reduced (if they appear in more than 25% of the documents in the corpus). Afterwards, the terms to be reduced are used to identify the model elements to be removed from the model fragment to be reformulated.

Fig. 6-(V3) shows an example of the core of this variant in which the inputs are shown in the upper part of the figure (the model fragment to be reformulated and a set of the model fragment population). This variant performs the selection of terms to be reduced from the homogenized terms of the model fragment to be reformulated (*breaker, convert, hvac*). In this example, the term *hvac* appears in more than 25% of the documents in the corpus, so it is selected to be reduced. Then, this term is used to identify the model element (*hvac*) from the model fragment to be reformulated, and it is then removed. As a result, the reformulated model fragment shown at the bottom part of Fig. 6-(V3) is obtained.

5.4 Variant 4: Selection

This variant is based on a query reformulation strategy that has been applied in source code retrieval [15]. The strategy automatically selects a set of top N terms from the text of a change request (title and description). To do this, the technique starts with the transformation of the text into a text graph where terms are denoted as vertices, and meaningful relations among those terms are represented as the edges [105]. This relation can be statistical, syntactic, or semantic in nature [106]. Once the graph is created, TextRank and POSRank calculations are applied to determine the importance of a term in the graph.

This variant considers statistical relations to transform the input text into the text graph and TextRank calculation as applied in [15] since the combination of statistical and syntactic graphs as well as the combination of TextRank and POSRank calculations provide a marginal improvement [15].

To capture statistical relations, co-occurrence of the words within a fixed window (e.g., window size = 2) is considered across all of the sentences from the request (as recommended by Mihalcea and Tarau [105]). For example, the relationships obtained from the homogenized terms of MF_1 are: *convert* $\leftrightarrow$ *hvac*, *hvac* $\leftrightarrow$ *failur*, *failur* $\leftrightarrow$ *overload*, *overload* $\leftrightarrow$ *peer*, and *peer* $\leftrightarrow$ *coverag*; Then, these relationships are encoded as bi-directional into the connecting edges between the corresponding vertices (i.e., terms) in the text graph.

Once the text graph is encoded, it is considered to be a regular connected network, and a popular graph-based algorithm (PageRank [107]) for ranking its vertices (i.e., terms) is applied. PageRank was originally proposed by Brin and Page for web link analysis, and the algorithm exploits the topological properties of a graph to estimate the weight (i.e., importance) of each of the vertices. TextRank is an adaptation of PageRank for text graph [15]. It analyzes the connectivity (i.e., connected neighbors and their weights) of each term v_i in the graph recursively and then calculates the term's weight, $TR(v_i)$, as follows:

$$TR(v_i) = (1-\phi) + \phi \sum_{j \in V_i} \frac{TR(v_j)}{|V(v_j)|} \quad (0 \leq \phi \leq 1)$$

where $V(v_j)$ and ϕ denote the list of vertices connected to v_i and the dumping factor, respectively. In the context of web surfing, the dumping factor, ϕ, is considered to be the probability of randomly choosing a web page by the surfer, and 1 - ϕ is the probability of jumping off that page. Mihalcea and Tarau [105] use a heuristic value of $\phi = 0.85$ for natural language texts in the context of keyword extraction, and we also use the same value for our TextRank calculation as recommended in [15]. We initialize each of the terms in the graph with a default value of 0.25 [15] and run an iterative version of the algorithm [107]. The computation iterates until it reaches the maximum iteration limit (i.e., 100 as suggested by Blanco and Lioma [106]). Once the computation is over, each of the terms in the graph has a final score, which is considered to be the weight or importance of that term within the texts provided as input. Then, the top N terms are selected.

To reformulate a model fragment by means of statistical relations and TextRank, this variant considers as a text the terms of each of the relevant documents (i.e., the top k model fragments with the highest fitness). Each text is considered as a sentence that will be encoded in the text graph. Once the text graph is encoded and the computation of TextRank is completed, the top N terms will be used to identify the model elements that will comprise the reformulated model fragment.

Fig. 6-(V4) shows an example of the core of this variant in which the terms to be used as input in the reformulation of the model fragment are the top three model fragments of the population that have the highest fitness score (MF_1, MF_2, MF_4). The text graph of the previous example, which includes the homogenized terms and relationships of MF_1, is extended to encode the terms and relationships of the other relevant documents. Hence, the relationships obtained from the homogenized terms of MF_2 are *convert* $\leftrightarrow$ *hvac*, *hvac* $\leftrightarrow$ *failur*, and *failur* $\leftrightarrow$ *overload*; and the relationships from MF_4 are *hvac* $\leftrightarrow$ *failur* and *failur* $\leftrightarrow$ *overload*. Once the computation of TextRank is completed and if the top five homogenized terms are selected (as recommended in [15]), the result is: *failur, overload, coverag, peer*, and *hvac*. Afterwards, these terms are used to identify the model elements from the product model that comprise the reformulated model fragment. The lower part of Fig. 6-(V4) depicts the reformulated model fragment obtained as a result.

6 Evaluation

This section explains the evaluation of our work, the research questions we set out to answer, and the evaluation process, measures, and statistical analysis that we used to answer these questions.

6.1 Research questions

We seek to answer the following three research questions:

RQ_1: *What is the performance in terms of the solution quality of the baselines and the four model fragment reformulation variants*?

RQ_2: *Is the difference in performance between the variants and the baseline that obtains the best results significant?*

RQ_3: *How much is the quality of the solution influenced using each variant compared to the baseline that obtains the best results?*

6.2 Planning and execution

Fig. 7 presents an overview of the process that is planned to answer each research question. To put the performance of our work in perspective and to study the impact on the results, we set two baselines. Baseline 1 is the evolutionary algorithm that uses the most popular choice of genetic operations of the SBMDE community (see Table 1), whereas Baseline 2 is an evolutionary algorithm that is similar to Baseline 1 but that replaces the single-point crossover operation of Baseline 1 with a randomly generated mask. The mask determines how the combination of two parents is done, indicating for each element of the model fragments whether the new individual should inherit from one parent or the other. Then, the element may or may not be included depending on whether or not it is present in the parent selected. We set Baseline 2 because the mask crossover operation achieves the best results when it comes to locating features in models [6], [84], [88], [89].

The evaluation process takes the data set provided by our industrial partner as input, as the left part of Fig. 7 shows. The data is made up of 23 trains where, on average, each product model is composed of more than 1200 model elements. Each model element has about 15 properties that include terms, which are used to differentiate among model elements. Specifically, CAF provided the following documentation of their railway solutions: 121 feature descriptions; the 23 product models where the model fragments should be located; and the approved feature realization (i.e., the model fragment that corresponds to each feature) that will be considered to be the ground truth (oracle). The model fragments that correspond to each feature have between 5 and 20 model elements, with an average of 13.55 model elements and a median of 14 model elements.

In addition, CAF provided us with lists of domain terms and stopwords to process the NL. The list of domain terms has around 300 domain terms, and the stopwords list has around 60 words. The list of domain terms and

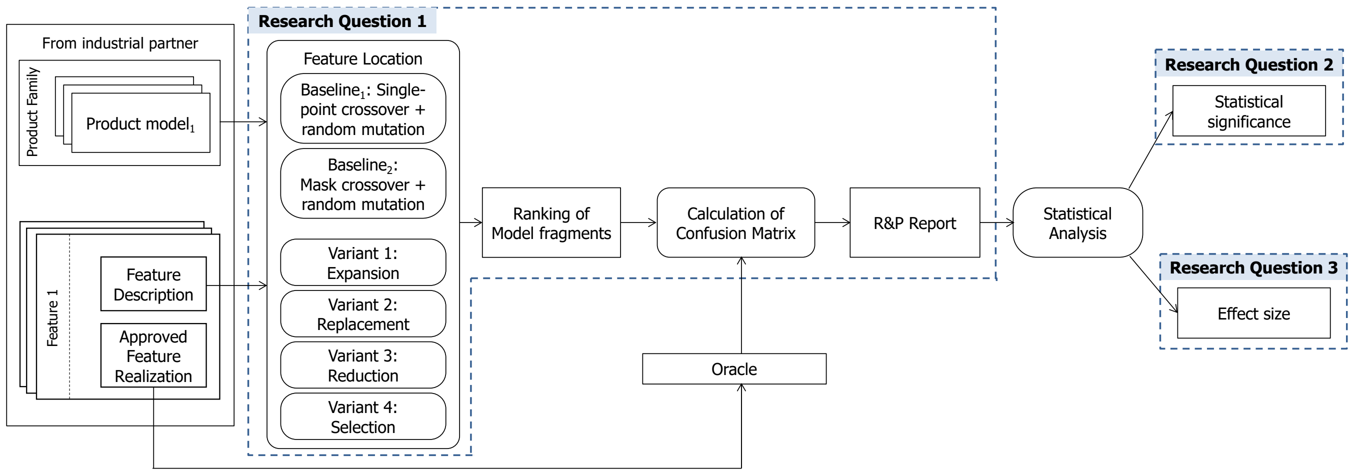


Fig. 7. Evaluation process to answer each research question

stopwords was obtained from the existing documentation in CAF. Specifically, these lists were obtained from the existing documentation that is used for training new employees. We believe that companies similar to CAF will also have documentation for training new employees in their domain, so this documentation can be used to generate the two lists that our approach needs to process the NL.

6.2.1 *Answering Research Question 1*

To assess the performance in terms of solution quality of the two baselines and the four model fragment reformulation variants (expansion, replacement, reduction, and selection), we executed 30 independent runs (as suggested by Arcuri and Fraser [108]) for each feature, baseline, and variant, i.e., 121 (features) x 6 (two baselines and four variants) x 30 repetitions = 21780 independent runs.

To assess the quality of each retrieved model fragment, a confusion matrix was calculated by comparing the retrieved model fragment and the oracle. The confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data (the best solutions) for which the true values are known (from the oracle). In our evaluation, each solution is a retrieved model fragment that is made up of a subset of the model elements that are part of the product model. Since the granularity is at the level of model elements, the presence or absence of each model element is considered to be a classification. The confusion matrix distinguishes between the predicted values and the real values, classifying them into four categories: 1) True Positive (TP): values that are predicted as true (in the solution) and are true in the real scenario (the oracle); 2) False Positive (FP): values that are predicted as true (in the solution) but are false in the real scenario (the oracle); 3) True Negative (TN): values that are predicted as false (in the solution) and are false in the real scenario (the oracle); and 4) False Negative (FN): values that are predicted as false (in the solution) but are true in the real scenario (the oracle).

In a report, from the comparison of each feature in the baselines and the four variants, we recorded three performance measures that are widely accepted in the software engineering research community [109]: Recall = $\frac{TP}{TP+FN}$ (measures the number of elements of the oracle that are correctly retrieved); Precision = $\frac{TP}{TP+FP}$ (measures the number of elements from the solution that are correct according to the oracle); and F-measure = $2 * \frac{Precision * Recall}{Precision + Recall}$ (corresponds to the harmonic mean of precision and recall). Recall and precision values can range between 0% to 100%. A value of 100% precision and 100% recall implies that both the solution and the oracle are the same.

6.2.2 *Answering Research Question 2*

To determine if the difference in performance between the variants and the baseline is significant, the results should be properly compared. To do this, all of the data resulting from the empirical analysis was analyzed using statistical methods following the guidelines in [11]. The goal of our statistical analysis is to provide formal and quantitative evidence (statistical significance) that the variants and the baseline (which obtains the best results) do in fact have an impact on the comparison metrics (i.e., the differences were not obtained by mere chance).

In order to enable statistical analysis, the baseline that obtains the best results and all variants should be run a large enough number of times (independently) to collect information on the probability distribution. A statistical test should then be run to assess whether there is enough empirical evidence to claim that there is a difference among the baseline and the variants. To do this, two hypotheses are defined: 1) the null hypothesis H_0 is typically defined to state that there is no difference when the baseline and the variants are compared; and 2) the alternative hypothesis H_1 states that there is a difference if at least one of the variants or the baseline is different. A statistical test aims to verify whether H_0 should be rejected.

The statistical tests provide a probability value, $p\text{-}value$, which obtains values between 0 and 1. The lower the $p\text{-}value$ of a test, the more likely that the null hypothesis H_0 (defined to state that there is no difference among the baseline and the variants) is false. It is accepted by the research community that a $p\text{-}value$ under 0.05 is statistically significant [11], and so H_0 can be considered false.

The test to be used depends on the properties of the data. Since our data does not follow a normal distribution, our analysis requires the use of non-parametric techniques.

There are several tests for analyzing this kind of data; however, the Quade test is more powerful when working with real data [110]. In addition, the Quade test has shown better results than the others when the number of algorithms is low [111].

To determine whether a variant has a significant impact on the quality of the solution with regard to the baseline, the quality of the solution of each variant should be statistically compared against the baseline. In order to do this, we performed an additional post-hoc analysis (pair-wise comparison between a variant and the baseline).

6.2.3 *Answering Research Question 3*

To determine how much the quality of the solution is influenced using each variant compared to the baseline, it is important to assess (through *effect size* measures) whether a criterion is statistically better than the baseline, and if so, the magnitude of the improvement.

For a non-parametric effect size measure, we used two non-parametric effect size measures: Vargha and Delaney's $\hat{A}_{12}$ [112], [113] and Cliff's delta [114], [115]. $\hat{A}_{12}$ measures the probability that running one variant yields higher values than running the baseline. If the variant and the baseline are equivalent, then $\hat{A}_{12}$ will be 0.5. For example, $\hat{A}_{12} = 0.7$ between Variant 1 and the baseline means that Variant 1 would obtain better results in 70% of the runs, and $\hat{A}_{12} = 0.3$ means that the baseline would obtain better results in 70% of the runs. We recorded an $\hat{A}_{12}$ value for each pair-wise comparison between a variant and the baseline.

Cliff's delta is an ordinal statistic that describes the frequency with which an observation from one group is higher than an observation from another group compared to the reverse situation. It can be interpreted as the degree to which two distributions overlap with values ranging from -1 to 1. For instance, when comparing distribution x and distribution y: a value of 0 means there is no difference between two distributions; a value of -1 means that all samples in distribution x are lower than all samples in distribution b; a value of 1 means the opposite (all samples in x are higher than all samples in y). In addition, threshold values were defined [116] for the interpretation of Cliff's delta effect size ($|d| < 0.147 \rightarrow$ "negligible"; $|d| < 0.33 \rightarrow$ "small"; $|d| < 0.474 \rightarrow$ "medium", $|d| \geq 0.474 \rightarrow$ "large").

6.3 Implementation details

For a fair comparison between the baselines and the variants, we have chose the parameters shown in Table 2. These parameters (such as population size and number of parents) correspond to those settings that are commonly used in the literature [6], [88], [89], [100], [117].

To implement the baselines and the model fragment reformulation variants, we used the Eclipse Modeling Framework [118] to manipulate the models. The techniques used to process the NL were implemented using OpenNLP [119] for the POS-Tagger and using the English (Porter2) stemming algorithm [120] for the stemming algorithm (originally created using snowball and then compiled to Java). The LSI was implemented using the Efficient Java Matrix Library (EJML [121]). The genetic operations are built upon the Watchmaker Framework for Evolutionary Computation [122]. An independent run of one of the baselines comprises more than 1406 lines of code, whereas an independent run of one of the variants comprises more than 1770 lines of code. Just the implementation of the operations that perform the model fragment reformulation comprises an average of 409 lines of code for each variant.

TABLE 2
Parameter settings

Genetic Operations	Parameter description	Value
Evolutionary algorithm	$Size$: Population Size	100
	r: Solutions replaced at population size	2
Crossover and mutation	μ: Number of Parents	2
	λ: Number of offspring from μ parents	2
	$p_{crossover}$: Crossover probability	0.9
	$p_{mutation}$: Mutation probability	0.1
Reformulations (V1,V2,V4)	Number of relevant documents	5
Reformulation (V1)	Terms to expand	10
Reformulation (V3)	Reduction value	25

Overall, there are two atomic performance measures for evolutionary algorithms with regard to solution quality and algorithm speed (or search effort). In this paper, we focus on the solution quality (i.e., obtaining a solution that is more similar to the one from the oracle in terms of precision and recall). After running some prior tests for each baseline and variant to determine the time to converge (and adding a margin to ensure convergence), we allocated a fixed amount of wall clock time (80 seconds) to stop the execution. During that time, the baselines are capable of executing an average of 347685 generations, whereas Variants 1-4 are capable of executing an average of 258, 384, 10176, and 480 generations, respectively. We performed the execution using a Mac Pro computer with an Intel Xeon E5-2697 V2 processor (clock speeds 2.7 GHz and 12 cores) and 64 GB of RAM. The computer was running macOS Mojave (10.14.5) as the hosting Operative System and the Java(TM) SE Runtime Environment (build 1.8.0_77).

The data set and part of the implementation are limited by confidentiality agreements that we have with the industrial partner. The trains of the data set are currently operating and under maintenance contracts or will be released in the near future. The CSV files used as input in the statistical analysis as well as an open-source implementation of the baselines and variants are available here: https://www.bitbucket.org/svitusj/mfr

7 Results

This section presents the results obtained for RQ$_s$ 1-3.

7.1 Research Question 1

Table 3 shows the mean values and standard deviations of recall, precision, and F-measure for the two baselines and the four model fragment reformulation variants.

RQ$_1$ answer. The results reveal that Baseline 2 outperforms Baseline 1 in the three performance measures (recall, precision, and F-measure). With regard to the variants, Variant 1 outperforms the baseline in recall (76.60%). Moreover, Variant 2 (replacement) obtains the best results in terms of recall, precision, and F-measure (95.67%, 66.50%, and 77.71%, respectively), whereas Variant 3 obtains the worst

TABLE 3
Mean values and standard deviations for recall, precision, and the F-measure for the baselines and variants

	Recall $\pm$ (σ)	Precision $\pm$ (σ)	F-measure $\pm$ (σ)
Baseline 1	34.34 $\pm$ 15.13	33.06 $\pm$ 12.47	30.67 $\pm$ 10.82
Baseline 2	57.94 $\pm$ 14.14	52.42 $\pm$ 15.19	52.70 $\pm$ 10.17
Variant 1	76.60 $\pm$ 14.02	37.47 $\pm$ 15.28	47.81 $\pm$ 14.75
Variant 2	95.67 $\pm$ 2.43	66.50 $\pm$ 13.30	77.71 $\pm$ 9.57
Variant 3	28.49 $\pm$ 13.12	10.67 $\pm$ 9.15	12.77 $\pm$ 7.99
Variant 4	67.43 $\pm$ 15.89	34.84 $\pm$ 13.29	43.89 $\pm$ 12.98

results (28.49% in recall, 10.67% in precision, and 12.77% in F-measure).

7.2 Research Question 2

The Quade test p-value is $\ll 2.2x10^{-16}$ for all performance indicators (recall, precision, and F-measure).

In order to report the results of the post-hoc analysis (pair-wise comparison between a variant and the baseline), we compare the results of the variants with the results of Baseline 2 since it outperforms Baseline 1. In addition, we compare the results of the variants. Table 4 shows the Holm's post-hoc $p-Values$.

TABLE 4
Holm's post-hoc $p-Values$ for comparing each variant with Baseline 2 and for comparing the variants

	Recall	Precision	F-measure
V1 vs Baseline	$\ll 2x10^{-16}$	$5.6x10^{-13}$	0.0025
V2 vs Baseline	$\ll 2x10^{-16}$	$6.4x10^{-14}$	$\ll 2x10^{-16}$
V3 vs Baseline	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
V4 vs Baseline	$1.7x10^{-5}$	$2.8x10^{-16}$	$7.6x10^{-8}$
V1 vs V2	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
V1 vs V3	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
V1 vs V4	$5.7x10^{-5}$	0.2	0.035
V2 vs V3	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
V2 vs V4	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
V3 vs V4	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$

RQ$_2$ answer. Since the Quade test p-Values and Holm's post-hoc $p-Values$ shown in Table 4 are smaller than 0.05 in all cases when a variant is compared with the baseline, we reject the null hypothesis. Consequently, we can state that there are significant differences for every pair-wise comparison between the variants (V1-V4) and the baseline for all of the performance indicators (recall, precision, and F-measure). With regard to the pair-wise comparisons between the variants, the performance indicators (recall, precision, and F-measure) are smaller than 0.05 in all cases except when precision of V1 is compared with V4. Consequently, we can state that there are significant differences for every pair-wise comparison between the variants (V1-V4) for all of the performance indicators (recall, precision, and F-measure) except for precision when V1 and V4 are compared.

7.3 Research Question 3

Table 5 shows the values of the effect size statistics between pair-wise comparisons of a Variant and the baseline (Baseline 2). In addition, Table 5 shows the values of the effect size statistics between pair-wise comparisons of two variants. Specifically, the upper part of the table shows the $\hat{A}_{12}$ values, whereas the lower part of the table shows the Cliff's Delta values for recall, precision, and F-measure.

TABLE 5
Effect size measures for comparing each variant with Baseline 2 and for comparing the variants

	$\hat{A}_{12}$		
	Recall	Precision	F-measure
V1 vs Baseline	0.8178	0.2466	0.4055
V2 vs Baseline	0.9980	0.7483	0.9659
V3 vs Baseline	0.0636	0.0133	0.0022
V4 vs Baseline	0.6665	0.1983	0.3015
V1 vs V2	0.0939	0.0765	0.0332
V1 vs V3	0.9962	0.9270	0.9665
V1 vs V4	0.6599	0.5490	0.5956
V2 vs V3	1	0.9994	1
V2 vs V4	0.9632	0.9578	0.9881
V3 vs V4	0.0273	0.0741	0.0283
	Cliff's Delta		
	Recall	Precision	F-measure
V1 vs Baseline	0.6356	-0.5068	-0.1890
V2 vs Baseline	0.9961	0.4967	0.9318
V3 vs Baseline	-0.8729	-0.9732	-0.9955
V4 vs Baseline	0.3331	-0.6033	-0.3968
V1 vs V2	-0.8121	-0.8469	-0.9335
V1 vs V3	0.9923	0.8541	0.9331
V1 vs V4	0.3199	0.0980	0.1912
V2 vs V3	1	0.9988	1
V2 vs V4	0.9264	0.9157	0.9762
V3 vs V4	-0.9453	-0.8518	-0.9433

RQ$_3$ answer. From the results, we can conclude how much the quality of the solution is influenced using each variant compared to the baseline. The magnitude of improvement using Variant 2 (replacement) instead of the baseline can be interpreted as being large according to the magnitude scales [116] of the Cliff's Delta values. According to the $\hat{A}_{12}$ value, Variant 2 obtains better results in the F-measure than the baseline in 96.59% of the runs. Hence, Variant 2 (replacement) has an actual impact on performance. In contrast, the baseline has an actual impact on performance with regard to Variant 1 (expansion), Variant 3 (reduction), and Variant 4 (selection). The largest difference is when Variant 3 and the baseline are compared, whereas the differences with Variant 4 and Variant 1 are medium and small, respectively. With regard to the comparisons of two variants, the largest differences are obtained when Variant 2 (replacement) and Variant 3 (reduction) are compared. Variant 2 obtains better results in the F-measure than Variant 3 in 100% of the runs.

8 Discussion

Our empirical results have confirmed that utilizing automatic query reformulations as genetic operations pays off in the location of features in models. However, it turns out that the results do not include all of the model elements of the features (100% of recall and precision). Our analysis of the results reveals that this happens because the fitness step that guides the evolutionary algorithm is not giving the highest fitness values to the approved feature realizations due to incomplete feature descriptions and vocabulary mismatch.

On the one hand, incomplete feature descriptions do not completely describe the model fragment to be located. On the other hand, vocabulary mismatch is a phenomenon that occurs when different words are used to refer to the same concept in the feature description and models. Although we use NLP to unify the terms, vocabulary mismatch remains an issue since in-house terms are often not recognized as eligible synonyms and are therefore excluded from NLP. This leads to vocabulary mismatch. For example, the terms PLC and system may be recognized as synonyms, but the terms PLC and COSMOS are definitely not known to be synonyms because COSMOS is an in-house term that is used exclusively by our industrial partner to refer to the term PLC. To minimize the vocabulary mismatch issue between the feature description and models, NLP should be extended in order to include a list of in-house synonyms.

By analyzing the results of each variant, we detected that there are significant differences in terms of the solution quality in comparison to the most popular choices of genetic operations in SBMDE (Baseline 2). In addition, we detected the following:

V1 (Expansion): This variant only has the ability to add elements to a model fragment. When a model fragment of the initial population is created with model elements that do not belong to the solution of the oracle, V1 is not able to remove those elements. V1 obtains the best results in the model fragments of the initial model fragment population that have been created with fewer model elements. However, V1 is not able to reach the solution of the oracle and even the model fragment to be reformulated has a small size (even when it only has one single model element).

V2 (Replacement): This variant has the ability to remove elements from the model fragment. In fact, this variant replaces all of the terms of the query as is stated in the literature (in our case, all of the elements of the model fragment to be reformulated). This could lead to believing that replacement does not preserve model elements after the reformulation. However, after inspecting the results, we found that the terms selected to perform the replacement correspond to some model elements that were originally in the model element. Hence, this variant has the capacity to conserve, to eliminate, and to add elements.

V3 (Reduction): This variant only eliminates elements of the model fragment because of its definition in the literature of reformulations. Analogously to what happens with V1, it may seem that model fragments with more elements are the ones that should give better results. However, this is not the case when the results are analyzed. The criterion used by this variant for reducing the most popular terms (i.e., the terms that appear in more than 25% of the documents) does not lead to good results. This is because domain experts produce queries that include terms that are popular in the relevant documents. Since this variant eliminates those popular terms, the reformulated model fragment differs from the oracle since it does not include model elements that are related to the popular terms.

V4 (Selection): This variant has the same capabilities to explore the solution space as V2 (Replacement) by conserving, eliminating, and adding elements to the model fragment to be reformulated. However, V4 follows a different strategy than V2. The main idea of V4 is to identify the elements that comprise the reformulated model fragment by selecting N terms according to their importance (given by the connected terms of the relevant model fragments and their weights). The main idea of V2 is to replace the elements of the model fragment to be reformulated with elements that correspond to the terms with the highest domain specificity. Our results show that the strategy of V2 is better suited for the task of feature location in models such as the ones of our industrial partner.

V2 (Replacement) obtains the best results for FL in models. One may think that this operator should always be the preferred option. However, by analyzing the results, we detected that V4 (Selection) is a better choice when incomplete feature descriptions are the input of feature location. In other words, if the software engineer in charge of feature location has confidence in the completeness of the feature description, then V2 is the recommended operator. The idea of domain specify of V2 yields better quality results in that scenario. On the other hand, if the software engineer thinks that the feature description is significantly incomplete, then V4 is the recommended operator. The idea of V4 of giving more importance to the connected terms is better suited to compensate for incomplete feature descriptions.

In the context of feature location in models, V3 (Reduction) is not recommended in any case. In all of the cases, the criterion of reduction worsens the quality of the results by eliminating terms that are relevant to the feature. Finally, when SBMDE researchers evaluate these operators in other software engineering tasks, we recommend that in the case of V1 (Expansion), the initial population should only compromise individuals of one single model element. This is the best scenario for this operator, and it will help to better understand the potential of this operator in other tasks.

Our results are relevant to the SBMDE community since researchers should consider reformulations as genetic operations to improve their approaches. All reformulation variants should still be considered since different MDE-related tasks might require different reformulation variants (or even combinations of them). In fact, a catalog of reformulations for MDE tasks is part of our future work.

It is important to highlight that the models taken as input in this work are used in a Model-Driven Development (MDD) context, so these models are used to obtain the code to completely control a train. For this reason, these MDD models are richer in terms than other kinds of models such as sketches for analysis. In the case of these MDD models, which are rich in terms, the use of reformulations to exploit the latent semantics pays off. However, in non-MDD models, we cannot affirm that the results will be similar. Our intuition makes us think that latent semantics can help in non-MDD models, but it might not be enough. Therefore, our future work also includes non-MDD models and hybrid variants that combine widespread genetic operations and reformulations.

There are other software maintenance and evolution activities where it is also essential that software elements be found, such as bug location and requirements traceability. Bug location aims to identify the location in the software that is pertinent to a software fault. Removing bugs is analogous to removing unwanted functionality [123]. In this sense, the bug description of a bug report could be used as the (unwanted) feature description. The output of the (unwanted) feature location is the ranking of model fragments that are relevant to the bug.

In the case of requirements traceability, it is important to highlight the differences between a feature and a requirement. They are written in a different style, in a different phase of development, and with a different goal in mind. Requirements are written before development, are client-influenced, and are for contracts. In contrast, features are written when products already exist, are internal, and are for reuse. Nevertheless, requirements are commonly specified in textual form by means of natural language, as is the case of feature descriptions. Therefore, a requirement could be used as the input query, whereas the output would hold model fragments that are relevant to the requirement.

Our adaptation of automatic query reformulations to genetic operations has been designed to be generic and applicable not only to feature location and to the domain of our industrial partner, but also to other tasks (e.g., bug location and requirements traceability) and domains. Hence, our work opens a new research direction for the SBMDE community (where models are the cornerstone artifact) to improve other software maintenance and evolution tasks.

Although our work focuses on models, our results can also motivate SBSE researchers, who use other artifacts such as code, requirements, or tests, to evaluate whether the quality of the solution is improved using genetic operations that leverage the latent semantics instead of randomly generating new candidate solutions.

9 Threats to validity

We use the classification to threats of validity suggested by De Oliveira et al. [124] to acknowledge the threats to the validity of our work.

Conclusion validity threats: We considered 30 independent runs for each variant and feature to address the first threat of this type, which is not accounting for random variation. To avoid the second threat of this type, which is the lack of a formal hypothesis and statistical tests, we employed standard statistical analysis following accepted guidelines [108]. To address the threat of the lack of a good descriptive analysis, we have used the recall, precision, and F-measure measurements to analyze the confusion matrix obtained; however, other measurements could be applied. Some works argue that the use of the Vargha and Delaney $\hat{A}_{12}$ measurement can be misrepresentative [108] and that data should be pre-transformed before applying it. We did not find any use cases for data pre-transformation that applied to our case study.

Internal validity threats: We used values from the literature for the algorithms to address the first identified threat of this type, which is the threat of poor parameter settings. As suggested by Arcuri and Fraser [108], default values are good enough to measure the performance of location techniques. These values have been tested in similar algorithms for Feature Location [125]. With regard to the parameter required to produce a solution (stop condition), we used 80 seconds since it was the time needed by our approach in the real-world industrial context. Nevertheless, we cannot yet claim how this time scales in other real-world industrial contexts with a larger search space size or a larger solution size. Another threat of this type is the lack of real problem instances. To address this threat, the evaluation of this paper was applied to an industrial case study.

Construct validity threats: To address the identified threat of the lack of assessing the validity of cost measures, we performed a fair comparison among the variants. In addition, our evaluation was performed using three measures (recall, precision, and F-measure), which are widely accepted in the software engineering research community [109].

External validity threats: To mitigate the threat of the lack of a clear object selection strategy, our approach uses an industrial case study, whose instances are collected from real-world problems.

With regard to what extent it is possible to generalize the results, the results depend on the quality of the queries, as occurs in other works [98], [99]. Poor queries assign a high rank to irrelevant model fragments. It is also worth noting that the language used for the model elements and the feature descriptions provided must be the same. This language is specific to each domain; however, as long as both elements are built using the same terminology, the LSI will work. To narrow the gap between the two elements, different NLP techniques such as tokenizers, stemming, or POS tagging techniques can eventually be applied. For instance, the naming conventions used by companies for model elements can follow different formats, but the approach can be tailored to handle them.

Moreover, to mitigate the generalization threat, our approach has been designed to be generic and applicable not only to the domain of our industrial partner but also to other different domains. Our approach can be applied to any model that conforms to MOF (the OMG metalanguage for defining modeling languages), and the text elements that are associated to the models are extracted automatically using the reflective methods provided by the Eclipse Modeling Framework. The requisites to apply our approach are that the set of models must conform to MOF, and the feature description must be provided in NL. However, our approach should be applied to other domains before assuring its generalization.

10 Conclusion

SBSE ideas are increasing in the MDE community. Nevetheless, the resulting new field of study known as SBMDE is still in its early stages. SBMDE researchers are reusing *as is* techniques from SBSE without leveraging the particularities of software models (the cornerstone artefact of MDE). This is the case of one of the main ingredients of SBMDE: genetic operations.

Our work proposes a novel adaptation of automatic query reformulations as genetic operations that leverage the latent semantics of software models. We analyze the impact of these operations in a real-world industrial case study of feature location in models. As baselines, we use single-point crossover plus random mutation (the most popular choice in SBMDE) and mask crossover plus random mutation (the genetic operations that achieve the best results in feature location in models).

Our reformulation operations have improved the results of the best baseline (by 37.73% in recall and 14.08% in precision). This is relevant for the task of feature location in models, which is one of the main activities performed during software maintenance and evolution [126]. Other main activities that share the goal of localizing relevant model elements in models (such as bug location and requirements traceability) are also candidates that can benefit from our reformulation operations.

Furthermore, the only requirement for applying our approach is the availability of terms in model elements. Therefore, our reformulation operations have the potential to influence most SBMDE works. We are aware that not all kinds of models have the same number of terms. For instance, a model for sketching a system is likely to have fewer terms than a model for generating the entire code of a system. Thus, exploring new variants of our reformulation operations to suit more kinds of models is part of our future work.

In addition, since reformulation operations are quite promising and suggest an optimistic future for improving feature location in software models, future works should explore the performance of feature location in code using reformulations as genetic operations instead of using operations that randomly mix information.

Acknowledgments

The work of Francisca Pérez and Carlos Cetina was mainly done during their stay at the LIP6 of Sorbonne University. 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 also thank the ITEA3 15010 REVaMP2 Project.

References

[1] I. Boussaïd, P. Siarry, and M. Ahmed-Nacer, "A survey on search-based model-driven engineering," *Autom. Softw. Eng.*, vol. 24, no. 2, pp. 233–294, 2017.

[2] M. Kessentini, P. Langer, and M. Wimmer, "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)*, May 2013, pp. 51–54.

[3] J. Denil, M. Jukss, C. Verbrugge, and H. Vangheluwe, "Search-based model optimization using model transformations," in *System Analysis and Modeling: Models and Reusability*, D. Amyot, P. Fonseca i Casas, and G. Mussbacher, Eds. Cham: Springer International Publishing, 2014, pp. 80–95.

[4] F. Fleurey, J. Steel, and B. Baudry, "Validation in model-driven engineering: testing model transformations," in *Proceedings. 2004 First International Workshop on Model, Design and Validation, 2004.*, Nov 2004, pp. 29–40.

[5] H. Saada, M. Huchard, C. Nebut, and H. A. Sahraoui, "Recovering model transformation traces using multi-objective optimization," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, 2013, pp. 688–693.

[6] J. Font, L. Arcega, Ø. Haugen, and C. Cetina, "Achieving feature location in families of models through the use of search-based software engineering," *IEEE Transactions on Evolutionary Computation*, vol. PP, no. 99, pp. 1–1, 2017.

[7] J. Krüger, T. Berger, and T. Leich, "Features and how to find them: A survey of manual feature location," in *Software Engineering for Variability Intensive Systems - Foundations and Applications*. Taylor & Francis Group, 2019, pp. 153–172.

[8] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.

[9] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 420–432, Jun. 2007.

[10] J. Wang, X. Peng, Z. Xing, and W. Zhao, "An exploratory study of feature location process: Distinct phases, recurring patterns, and elementary actions," in *Proceedings of the 27th Conference on Software Maintenance*. IEEE, 2011, pp. 213–222.

[11] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Softw. Test. Verif. Reliab.*, vol. 24, no. 3, pp. 219–250, May 2014.

[12] G. Salton, *The SMART Retrieval System—Experiments in Automatic Document Processing*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1971.

[13] M. Gibiec, A. Czauderna, and J. Cleland-Huang, "Towards mining replacement queries for hard-to-retrieve traces," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '10. New York, NY, USA: ACM, 2010, pp. 245–254.

[14] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13, 2013, pp. 842–851.

[15] M. M. Rahman and C. K. Roy, "STRICT: information retrieval based search term identification for concept location," *CoRR*, vol. abs/1807.04475, 2018.

[16] G. Pavai and T. V. Geetha, "A survey on crossover operators," *ACM Comput. Surv.*, vol. 49, no. 4, pp. 72:1–72:43, Dec. 2016.

[17] A. Umbarkar1 and P. Sheth, "Crossover operators in genetic algorithms: A review," *ICTACT Journal On Soft Computing*, vol. 6, no. 1, pp. 1083–1092, 2015.

[18] M. N. S. A. M. Siew Mooi Lim, Abu Bakar Md. Sultan and K. Y. Leong, "Crossover and mutation operators of genetic algorithms," *International Journal of Machine Learning and Computing*, vol. 7, no. 1, pp. 9–12, 2017.

[19] M. Kessentini, H. Sahraoui, and M. Boukadoum, "Model transformation as an optimization problem," in *Model Driven Engineering Languages and Systems*, K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 159–173.

[20] M. Kessentini, M. Wimmer, H. Sahraoui, and M. Boukadoum, "Generating transformation rules from examples for behavioral models," in *Proceedings of the Second International Workshop on Behaviour Modelling: Foundation and Applications*, ser. BM-FA '10. ACM, 2010, pp. 2:1–2:7.

[21] M. Kessentini, H. Sahraoui, M. Boukadoum, and O. B. Omar, "Search-based model transformation by example," *Software & Systems Modeling*, vol. 11, no. 2, pp. 209–226, May 2012.

[22] M. Faunes, H. Sahraoui, and M. Boukadoum, "Generating model transformation rules from examples using an evolutionary algorithm," in *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, Sep. 2012, pp. 250–253.

[23] M. Faunes, H. Sahraoui, and M. Boukadoum, "Genetic-programming approach to learn model transformation rules from examples," in *Theory and Practice of Model Transformations*, K. Duddy and G. Kappel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 17–32.

[24] I. Baki, H. Sahraoui, Q. Cobbaert, P. Masson, and M. Faunes, "Learning implicit and explicit control in model transformations by example," in *Model-Driven Engineering Languages and Systems*. Cham: Springer International Publishing, 2014, pp. 636–652.

[25] M. Kessentini, A. Ouni, P. Langer, M. Wimmer, and S. Bechikh,

"Search-based metamodel matching with structural and syntactic measures," *Journal of Systems and Software*, vol. 97, pp. 1 – 14, 2014.

[26] M. W. Mkaouer, M. Kessentini, S. Bechikh, and D. R. Tauritz, "Preference-based multi-objective software modelling," in *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering*, ser. CMSBSE '13. IEEE Press, 2013, pp. 61–66.

[27] S. Gyapay, A. Schmidt, and D. Varró, "Joint optimization and reachability analysis in graph transformation systems with time," *Electronic Notes in Theoretical Computer Science*, vol. 109, pp. 137 – 147, 2004, proceedings of the Workshop on Graph Transformation and Visual Modelling Techniques (GT-VMT 2004).

[28] H. Abdeen, D. Varró, H. Sahraoui, A. S. Nagy, C. Debreceni, A. Hegedüs, and A. Horváth, "Multi-objective optimization in rule-based design space exploration," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '14. ACM, 2014, pp. 289–300.

[29] M. Fleck, J. Troya, and M. Wimmer, "Marrying search-based optimization and model transformation technology," 2015.

[30] A. Nisbet, "Gaps: A compiler framework for genetic algorithm (ga) optimised parallelisation," in *HPCN Europe*, 1998.

[31] K. D. Cooper, P. J. Schielke, and D. Subramanian, "Optimizing for reduced code space using genetic algorithms," *SIGPLAN Not.*, vol. 34, no. 7, pp. 1–9, May 1999.

[32] D. Fatiregun, M. Harman, and R. Hierons, "Search based transformations," in *Genetic and Evolutionary Computation — GECCO 2003*, E. Cantú-Paz, J. A. Foster, K. Deb, L. D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, and J. Miller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 2511–2512.

[33] D. Fatiregun, M. Harman, and R. M. Hierons, "Evolving transformation sequences using genetic algorithms," in *Source Code Analysis and Manipulation, Fourth IEEE International Workshop on*, 2004, pp. 65–74.

[34] P. Kulkarni, S. Hines, J. Hiser, D. Whalley, J. Davidson, and D. Jones, "Fast searches for effective optimization phase sequences," in *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation*, ser. PLDI '04, 2004, pp. 171–182.

[35] D. Fatiregun, M. Harman, and R. M. Hierons, "Search-based amorphous slicing," in *12th Working Conference on Reverse Engineering (WCRE'05)*, 2005, pp. 10–12.

[36] J. J. Cadavid, B. Baudry, and H. Sahraoui, "Searching the boundaries of a modeling space to test metamodels," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, April 2012, pp. 131–140.

[37] W. Wang, M. Kessentini, and W. Jiang, "Test cases generation for model transformations from structural information," in *MDEBE@MoDELS*, 2013.

[38] J. Shelburg, M. Kessentini, and D. R. Tauritz, "Regression testing for model transformations: A multi-objective approach," in *Search Based Software Engineering - 5th International Symposium, SSBSE 2013, St. Petersburg, Russia, August 24-26, 2013. Proceedings*, 2013, pp. 209–223.

[39] A. A. Jilani, M. Z. Iqbal, and M. U. Khan, "A search based test data generation approach for model transformations," in *Theory and Practice of Model Transformations*, D. Di Ruscio and D. Varró, Eds. Cham: Springer International Publishing, 2014, pp. 17–24.

[40] D. Sahin, M. Kessentini, M. Wimmer, and K. Deb, "Model transformation testing: a bi-level search-based software engineering approach," *Journal of Software: Evolution and Process*, vol. 27, no. 11, pp. 821–837, 2015.

[41] L. M. Rose and S. Poulding, "Efficient probabilistic testing of model transformations using search," in *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering*, ser. CMSBSE '13. IEEE Press, 2013, pp. 16–21.

[42] E. Batot, "Generating examples for knowledge abstraction in MDE: a multi-objective framework," in *Proceedings of the ACM Student Research Competition at MODELS 2015 co-located with the ACM/IEEE 18th International Conference MODELS 2015, Ottawa, Canada, September 29, 2015.*, 2015, pp. 1–6.

[43] M. Kessentini, H. A. Sahraoui, and M. Boukadoum, "Sequence diagram to colored petri nets transformation testing: an immune system metaphor," in *Proceedings of the 2010 conference of the Centre for Advanced Studies on Collaborative Research, November 1-4, 2010, Toronto, Ontario, Canada*, 2010, pp. 72–85.

[44] M. Kessentini, H. Sahraoui, and M. Boukadoum, "Example-based model-transformation testing," *Automated Software Engineering*, vol. 18, no. 2, pp. 199–224, Jun 2011.

[45] M. Faunes, J. Cadavid, B. Baudry, H. Sahraoui, and B. Combemale, "Automatically searching for metamodel well-formedness rules in examples and counter-examples," in *Model-Driven Engineering Languages and Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 187–202.

[46] M. Kessentini, W. Werda, P. Langer, and M. Wimmer, "Search-based model merging," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '13. ACM, 2013, pp. 1453–1460.

[47] U. Mansoor, M. Kessentini, P. Langer, M. Wimmer, S. Bechikh, and K. Deb, "Momm: Multi-objective model merging," *Journal of Systems and Software*, vol. 103, pp. 423 – 439, 2015.

[48] C. Debreceni, I. Ráth, D. Varró, X. D. Carlos, X. Mendialdua, and S. Trujillo, "Automated model merge by design space exploration," in *FASE*, 2016.

[49] A. Ghannem, M. Kessentini, and G. El Boussaidi, "Detecting model refactoring opportunities using heuristic search," in *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*, ser. CASCON '11. IBM Corp., 2011, pp. 175–187.

[50] A. Ghannem, G. El Boussaidi, and M. Kessentini, "Model refactoring using interactive genetic algorithm," in *Search Based Software Engineering*, G. Ruhe and Y. Zhang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 96–110.

[51] M. Amoui, S. Mirarab, S. Ansari, and C. Lucas, "A genetic algorithm approach to design evolution using design pattern transformation," *International Journal of Information Technology and Intelligent Computing*, vol. 1, no. 2, pp. 235–244, 2006.

[52] A. C. Jensen and B. H. Cheng, "On the use of genetic programming for automated refactoring and the introduction of design patterns," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '10. ACM, 2010, pp. 1341–1348.

[53] A. ben Fadhel, M. Kessentini, P. Langer, and M. Wimmer, "Search-based detection of high-level model changes," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, Sep. 2012, pp. 212–221.

[54] I. H. Moghadam and M. O. Cinneide, "Automated refactoring using design differencing," in *Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering*, ser. CSMR '12. IEEE Computer Society, 2012, pp. 43–52.

[55] Ipate F and Lefticaru R, "Genetic model based testing: A framework and a case study," *Romanian Journal of Information Science and Technology*, vol. 11, pp. 209–227, 1 2008.

[56] Huaizhong Li and C. P. Lam, "An ant colony optimization approach to test sequence generation for state based software testing," in *Fifth International Conference on Quality Software (QSIC'05)*, Sep. 2005, pp. 255–262.

[57] C. Doungsa-ard, K. Dahal, A. Hossain, and T. Suwannasart, "Test data generation from UML state machine diagrams using gas," in *International Conference on Software Engineering Advances (ICSEA 2007)*, Aug 2007, pp. 47–47.

[58] R. Lefticaru and F. Ipate, "Functional search-based testing from state machines," in *2008 1st International Conference on Software Testing, Verification, and Validation*, April 2008, pp. 525–528.

[59] M. Shirole, A. Suthar, and R. Kumar, "Generation of improved test cases from UML state diagram using genetic algorithm," in *Proceedings of the 4th India Software Engineering Conference*, ser. ISEC '11. ACM, 2011, pp. 125–134.

[60] S. Ali, M. Z. Iqbal, A. Arcuri, and L. Briand, "A search-based ocl constraint solver for model-based test data generation," in *2011 11th International Conference on Quality Software*, July 2011, pp. 41–50.

[61] S. Ali, M. Zohaib Iqbal, A. Arcuri, and L. C. Briand, "Generating test data from OCL constraints with search techniques," *IEEE Transactions on Software Engineering*, vol. 39, no. 10, pp. 1376–1402, Oct 2013.

[62] M. Z. Iqbal, A. Arcuri, and L. Briand, "Empirical investigation of search algorithms for environment model-based testing of real-time embedded software," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ser. ISSTA 2012. ACM, 2012, pp. 199–209.

[63] J. Hänsel, "Model based test case generation with metaheuristics for networks of timed automata," in *Proceedings of the 7th International Workshop on Search-Based Software Testing*, ser. SBST 2014. New York, NY, USA: ACM, 2014, pp. 31–34.

[64] A. S. Kalaji, R. M. Hierons, and S. Swift, "An integrated search-based approach for automatic testing from extended finite state machine (efsm) models," *Information and Software Technology*, vol. 53, no. 12, pp. 1297 – 1318, 2011.

[65] T. Yano, E. Martins, and F. L. de Sousa, "Most: A multi-objective search-based testing from efsm," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, March 2011, pp. 164–173.

[66] J. Zhang, R. Yang, Z. Chen, Z. Zhao, and B. Xu, "Automated efsm-based test case generation with scatter search," in *Proceedings of the 7th International Workshop on Automation of Software Test*, ser. AST '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 76–82.

[67] H. Li and C. P. Lam, "Using anti-ant-like agents to generate test threads from the UML diagrams," in *Testing of Communicating Systems*, F. Khendek and R. Dssouli, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 69–80.

[68] Dong Xu, H. Li, and C. P. Lam, "Using adaptive agents to automatically generate test scenarios from the UML activity diagrams," in *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, Dec 2005, pp. 15–17.

[69] M. Shirole, M. Kommuri, and R. Kumar, "Transition sequence exploration of uml activity diagram using evolutionary algorithm," in *Proceedings of the 5th India Software Engineering Conference*, ser. ISEC '12. New York, NY, USA: ACM, 2012, pp. 97–100. [Online]. Available: http://doi.acm.org/10.1145/2134254.2134271

[70] M. Shirole and R. Kumar, "A hybrid genetic algorithm based test case generation using sequence diagrams," in *Contemporary Computing*, S. Ranka, A. Banerjee, K. K. Biswas, S. Dua, P. Mishra, R. Moona, S.-H. Poon, and C.-L. Wang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 53–63.

[71] M. Prasanna and K. R. Chandran, "Automatic test case generation for UML object diagrams using genetic algorithm," *International Journal on Advance Soft Computing Application, July*, 2009.

[72] U. Farooq and C. P. Lam, "Evolving the quality of a model based test suite," in *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops*, ser. ICSTW '09. IEEE Computer Society, 2009, pp. 141–149.

[73] H. Hemmati, L. Briand, A. Arcuri, and S. Ali, "An enhanced test case selection approach for model-based testing: an industrial case study," G.-C. Roman and A. van der Hoek, Eds. ACM, 2010, 1.

[74] S. Sabharwal, R. Sibal, and C. Sharma, "Applying genetic algorithm for prioritization of test case scenarios derived from UML diagrams," *IJCSI International Journal of Computer Science Issues*, vol. 8, 2011.

[75] H. J. Goldsby and B. H. C. Cheng, "Automatically generating behavioral models of adaptive systems to address uncertainty," in *Model Driven Engineering Languages and Systems*, K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 568–583.

[76] H. J. Goldsby and B. H. Cheng, "Avida-MDE: A digital evolution approach to generating models of adaptive software behavior," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '08. ACM, 2008, pp. 1751–1758.

[77] H. J. Goldsby, B. H. C. Cheng, P. K. McKinley, D. B. Knoester, and C. A. Ofria, "Digital evolution of behavioral models for autonomic systems," in *2008 International Conference on Autonomic Computing*, June 2008, pp. 87–96.

[78] A. J. Ramirez, D. B. Knoester, B. H. Cheng, and P. K. McKinley, "Applying genetic algorithms to decision making in autonomic computing systems," in *Proceedings of the 6th International Conference on Autonomic Computing*, ser. ICAC '09. New York, NY, USA: ACM, 2009, pp. 97–106.

[79] P. K. McKinley, B. H. C. Cheng, A. J. Ramirez, and A. C. Jensen, "Applying evolutionary computation to mitigate uncertainty in dynamically-adaptive, high-assurance middleware," *Journal of Internet Services and Applications*, vol. 3, no. 1, pp. 51–58, dec 2011.

[80] B. H. C. Cheng, A. Ramirez, and P. K. McKinley, "Harnessing evolutionary computation to enable dynamically adaptive systems to manage uncertainty," in *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering*, ser. CMSBSE '13. IEEE Press, 2013, pp. 1–6.

[81] J. R. Williams, S. M. Poulding, R. F. Paige, and F. Polack, "Exploring the use of metaheuristic search to infer models of dynamic system behaviour," in *Proceedings of the 8th Workshop on Models @ Run.time co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013), Miami, FL, USA, September 29, 2013.*, 2013, pp. 76–88.

[82] S. S. Andrade and R. J. d. A. Macêdo, "Toward systematic conveying of architecture design knowledge for self-adaptive systems," in *2013 IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops*, Sep. 2013, pp. 23–24.

[83] M. Harman, Y. Jia, W. B. Langdon, J. Petke, I. H. Moghadam, S. Yoo, and F. Wu, "Genetic improvement for adaptive software engineering (keynote)," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS 2014. ACM, 2014, pp. 1–4.

[84] J. Font, L. Arcega, O. Haugen, and C. Cetina, "Feature location in models through a genetic algorithm driven by information retrieval techniques," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '16. ACM, 2016, pp. 272–282.

[85] A. C. Marcén, J. Font, O. Pastor, and C. Cetina, "Towards feature location in models through a learning to rank approach," in *Proceedings of the 21st International Systems and Software Product Line Conference - Volume B*, ser. SPLC 17, 2017, p. 5764.

[86] L. Arcega, J. Font, Ø. Haugen, and C. Cetina, "Leveraging models at run-time to retrieve information for feature location," in *Proceedings of the 10th International Workshop on Models@run.time co-located with the 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015), Ottawa, Canada, September 29, 2015*, 2015, pp. 51–60.

[87] M. Ballarín, A. C. Marcén, V. Pelechano, and C. Cetina, "Measures to report the location problem of model fragment location," in *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018, Copenhagen, Denmark, October 14-19, 2018*, 2018, pp. 189–199.

[88] F. Pérez, R. Lapeña, J. Font, and C. Cetina, "Fragment retrieval on models for model maintenance: Applying a multi-objective perspective to an industrial case study," *Information & Software Technology*, vol. 103, pp. 188–201, 2018.

[89] F. Pérez, J. Font, L. Arcega, and C. Cetina, "Collaborative feature location in models through automatic query expansion," *Automated Software Engineering*, vol. 26, no. 1, pp. 161–202, 2019.

[90] Meili Lu, X. Sun, S. Wang, D. Lo, and Yucong Duan, "Query expansion via wordnet for effective code search," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, March 2015, pp. 545–549.

[91] A. Marcus, A. Sergeyev, V. Rajlich, and J. Maletic, "An information retrieval approach to concept location in source code," in *Proceedings of the 11th Working Conference on Reverse Engineering*, Nov 2004, pp. 214–223.

[92] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, 1st ed. Morgan & Claypool Publishers, 2012.

[93] A. Hulth, "Improved automatic keyword extraction given more linguistic knowledge," in *Proceedings of the 2003 conference on Empirical methods in natural language processing*, 2003, pp. 216–223.

[94] R. Lapeña, J. Font, O. Pastor, and C. Cetina, "Analyzing the impact of natural language processing over feature location in models," in *GPCE 2017 - 16th International Conference on Generative Programming: Concepts & Experience*, 2017.

[95] T. K. Landauer, P. W. Foltz, and D. Laham, "An Introduction to Latent Semantic Analysis," *Discourse processes*, vol. 25, 1998.

[96] T. Hofmann, "Probabilistic Latent Semantic Indexing," in *Proceedings of the 22nd Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, 1999.

[97] S. Winkler and J. Pilgrim, "A Survey of Traceability in Requirements Engineering and Model-Driven Development," *Software and Systems Modeling (SoSyM)*, vol. 9, no. 4, pp. 529–565, 2010.

[98] B. Sisman and A. C. Kak, "Assisting code search with automatic query reformulation for bug localization," in *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR*, 2013, pp. 309–318.

[99] E. Hill, L. Pollock, and K. Vijay-Shanker, "Automatically capturing source code context of nl-queries for software maintenance and reuse," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09, 2009, pp. 232–242.

[100] C. Carpineto and G. Romano, “A survey of automatic query expansion in information retrieval,” *ACM Comput. Surv.*, vol. 44, no. 1, pp. 1:1–1:50, Jan. 2012.

[101] X. Zou, R. Settimi, and J. Cleland-Huang, “Improving automated requirements trace retrieval: a study of term-based enhancement methods,” *Empirical Software Engineering*, vol. 15, no. 2, pp. 119–146, 2010.

[102] G. Gay, S. Haiduc, A. Marcus, and T. Menzies, “On the use of relevance feedback in ir-based concept location.” in *ICSM*. IEEE Computer Society, 2009, pp. 351–360.

[103] O. Chaparro, J. M. Florez, and A. Marcus, “Using observed behavior to reformulate queries during text retrieval-based bug localization,” in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, vol. 00, Sept. 2018, pp. 376–387.

[104] G. Kumaran and V. R. Carvalho, “Reducing long queries using query quality predictors,” in *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR ’09. ACM, 2009, pp. 564–571.

[105] R. Mihalcea and P. Tarau, “TextRank: Bringing order into texts,” in *Proceedings of EMNLP-04and the 2004 Conference on Empirical Methods in Natural Language Processing*, July 2004.

[106] R. Blanco and C. Lioma, “Graph-based term weighting for information retrieval,” *Inf. Retr.*, vol. 15, no. 1, pp. 54–92, Feb. 2012.

[107] S. Brin and L. Page, “The anatomy of a large-scale hypertextual web search engine,” *Comput. Netw. ISDN Syst.*, vol. 30, no. 1-7, pp. 107–117, Apr. 1998.

[108] A. Arcuri and G. Fraser, “Parameter tuning or default values? an empirical investigation in search-based software engineering,” *Empirical Software Engineering*, vol. 18, no. 3, pp. 594–623, 2013.

[109] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986.

[110] S. García, A. Fernández, J. Luengo, and F. Herrera, “Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power,” *Inf. Sci.*, vol. 180, no. 10, pp. 2044–2064, May 2010.

[111] W. Conover, *Practical nonparametric statistics*, 3rd ed., ser. Wiley series in probability and statistics. New York, NY [u.a.]: Wiley, 1999.

[112] A. Vargha and H. D. Delaney, “A critique and improvement of the cl common language effect size statistics of mcgraw and wong,” *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[113] R. J. Grissom and J. J. Kim, *”Effect sizes for research: A broad practical approach*. Mahwah, NJ: Earlbaum, 2005.

[114] N. Cliff, “Dominance statistics: Ordinal analyses to answer ordinal questions.” *Psychological Bulletin*, vol. 114, no. 3, p. 494, 1993.

[115] ——, “Ordinal methods for behavioral data analysis.” 1996.

[116] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, “Appropriate statistics for ordinal level data: Should we really be using t-test and cohensd for evaluating group differences on the nsse and other surveys,” in *annual meeting of the Florida Association of Institutional Research*, 2006, pp. 1–33.

[117] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar, “Scalable product line configuration: A straw to break the camel’s back,” in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, Nov 2013, pp. 465–474.

[118] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.

[119] “Apache opennlp: Toolkit for the processing of natural language text,” https://opennlp.apache.org/, 2019.

[120] “English (porter2) stemming algorithm,” http://snowball.tartarus.org/algorithms/english/stemmer.html, 2019.

[121] “Efficient java matrix library,” http://ejml.org/, 2019.

[122] D. Dyer, “The watchmaker framework for evolutionary computation (evolutionary/genetic algorithms for java),” http://watchmaker.uncommons.org/, 2016.

[123] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, “Feature Location in Source Code: A Taxonomy and Survey,” in *Journal of Software Maintenance and Evolution: Research and Practice*, 2011.

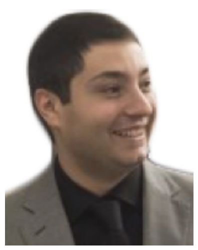
[124] M. de Oliveira Barros and A. C. D. Neto, “Threats to validity in search-based software engineering empirical studies,” Tech. Rep. 0006/2011, 2011.

[125] R. E. Lopez-Herrejon, L. Linsbauer, J. A. Galindo, J. A. Parejo, D. Benavides, S. Segura, and A. Egyed, “An assessment of search-based techniques for reverse engineering feature models,” *J. Syst. Softw.*, vol. 103, no. C, pp. 353–369, May 2015.

[126] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, “Feature location in source code: a taxonomy and survey.” *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.

Francisca Pérez is an assistant professor (ternure track) in the SVIT Research Group (https://svit.usj.es) at San Jorge University. She received a PhD in Computer Science from the Technical University of Valencia. Her research interests include Model-Driven Development, Collaborative Information Retrieval, Search-Based Software Engineering, and Variability Modeling. She publishes her research results and participates in high-level international software engineering conferences and journals, such as the Automated Software Engineering (AUSE) journal, the Information & Software Technology (IST) journal, and the Journal of Systems and Software (JSS). More about Pérez and her work is available online at http://franciscaperez.com.

Tewfik Ziadi is an associate professor at the Sorbonne University and the co-leader of the MoVe team. He received a PhD in Computer Science from Université Rennes 1. His research interests include Software Product Lines, Variability Modeling, and Model-Driven Development. His research has been published at several top-tier software engineering venues and journals, such as the IST journal and the IEEE/ACM International Conference on Automated Software Engineering (ASE). He has also been involved in program committees of international conferences, and as general chair in international conferences such as the 23rd International Systems and Software Product Line Conference (SPLC 2019). More about Tewfik and his work can be found at his website: https://pages.lip6.fr/tewfik.ziadi/.



Carlos Cetina is an associate professor with San Jorge University and the Head of the SVIT Research Group. He received a PhD in computer science from the Technical University of Valencia. His research focuses on software product lines and model-driven development. His research results have reshaped software development in world-leading industries from heterogeneous domains ranging from induction hob firmware to train control and management systems. More information about his background can be found at his website: http://carloscetina.com.