Comparison of Search Strategies for Feature Location in Software Models

Jorge Echeverría*, Jaime Font, Francisca Pérez, Carlos Cetina

Universidad San Jorge. SVIT Research Group
Autovía A-23 Zaragoza-Huesca Km.299, 50830, Zaragoza, Spain

Abstract

Search-based model-driven engineering is the application of search-based techniques to specific problems that are related to software engineering that is driven using software models. In this work, we make use of measures from the literature to report feature location problems in models (size and volume of the model and density, multiplicity, and dispersion of the feature being located) and a set of search strategies (random search, iterated local search, hill climbing, an evolutionary algorithm, and a hybrid between an evolutionary algorithm and hill climbing). The goal is to analyze of the impact of different values that are used to describe the feature location problems and the performance obtained by the different search strategies. We apply the search strategies to 1895 feature location problems that are obtained from 40 industrial software product lines. This work shows that: 1) the best results overall are obtained by a hybrid between evolutionary algorithm and hill climbing; 2) the size of the search space has the greatest impact on the results obtained by the search strategies; and 3) the impact of each of the measures is not the same in the five search strategies. This work highlights the use of the search strategy that produces the best results. In addition, we provide recommendations on when to use each search strategy.

Keywords: Feature Location in Models, Search Strategies

1. Introduction

Since the term Search-Based Software Engineering (SBSE) was coined in 2001 [1], many research efforts have been carried out in this area [2]. Part of its success resides in the fact that many software engineering problems can be expressed in a form that can be managed by a search strategy, enabling the automation of some tasks that otherwise must be performed manually.

Model-Driven Engineering (MDE) aims to facilitate the development of complex systems by shifting the focus from code assets to models [3]. The use of models as first class entities enables an increase in the abstraction level, simplifying some of the software engineering tasks (such as the specification of a system's components) while imposing new challenges (such as maintaining the traceability and consistency between the models and the generated code [4]).

Recently, the combination of SBSE and MDE has lead to the emergence of a new community around the topic of Search-Based Model-Driven Engineering (SBMDE) [5], whose focus is the use of SBSE techniques to address problems that are related to the application of MDE for software development. Specifically, in this work, Feature Location (FL) is one of the most important and common activities performed during software maintenance and evolution [6], consisting of the identification of source code elements that implement a

*Corresponding author. Tel.: +34 976060100
Email addresses: jecheverria@usj.es (Jorge Echeverría), jfont@usj.es (Jaime Font), mfperez@usj.es (Francisca Pérez), ccetina@usj.es (Carlos Cetina)

specific functionality [7, 8, 9]. Similarly, Feature Location in Models (FLiM) deals with the task of identifying what elements of a model realize a particular feature of a system [10, 11, 12].

FL is a mature field that has received a lot of of attention from researchers [8, 7, 9]. However, there are not as many works that report experiences with FLiM [5]: surveys that focus on models [5] do not report FL among their principal application areas. This is a big concern given the evidence of the usage of models in industry (e.g., around 30% of the case studies in a recent catalogue of Software Product Lines [13] use models as their main artifacts).

FLiM is particularly useful for the maintenance and evolution of these models' transformations [14, 15, 16] or in systems whose models are interpreted at runtime [17]. It can be argued that the abstraction layer provided by the models ease the tasks of maintenance (as, for example, feature location) when compared to source code [18] and so this might be a beneficial approach for researchers and practitioners. For example software product lines based on models, generate the source code of the products directly from the models using a model-to-text transformation, resulting in a code that does not include as semantic information in contrast to the original models (where the domain knowledge is formalized). Hence it might be more effective to locate features in the model to ultimately find them in the code.

In addition, reports of empirical evaluations of FL techniques place the focus on the technique being applied [8] rather than focusing on the specific characteristics that make the problem challenging (such as the sizes of the models or the number of elements that need to be located).

To address these gaps, our study includes a comparison of five search strategies for FLiM [11] in the light of specific characteristics that make the problem challenging. This comparison is performed on 40 Software Product Lines (SPLs) from the home appliance industry and the railway industry, which are presented in the context of five feature characteristics from the literature (size and volume of the search space, and density, multiplicity and dispersion of the model fragment being located) [19].

The results obtained can benefit practitioners when choosing a FLiM technique. The search strategy that provides the best performance results is a hybrid between an evolutionary algorithm and hill climbing. In addition, the study shows how the characteristics that are used here to report feature location problems can yield better results when deciding on the search strategy to address a feature location problem. Specifically, we claim that:

1. Our work provides recommendations for practitioners when selecting which search strategy to apply based on their needs regarding precision and recall when performing FLiM.
2. Our work also provides recommendations for practitioners to help determine the best search strategy based on the information about the nature of the feature location problem [19] that they are facing (size and volume of the model and density, multiplicity, and dispersion of the feature being located).
3. In the absence of information regarding the above points, our work shows that the hybrid between an evolutionary algorithm and hill-climbing obtains the best overall results among the five search strategies studied.

To date, the most popular search strategy in the SBMDE community is the evolutionary algorithm approach [5]. Our work suggests that SBMDE researchers should possibly evaluate whether the use of a hybrid between evolutionary algorithm and hill-climbing can improve the quality of their solutions.

The rest of the paper is structured as follows: Section 2 presents an overview of the problem of feature location in models, the search strategies used to address it, and the measures used to characterize and describe the problems. Section 3 presents the design of the controlled experiment that will be carried out in this work. Section 4 shows the results of the experiment and the statistical analysis performed and Section 5 presents the findings of this work. Section 6 discusses the threats to the validity of the work. Section 7 presents a discussion of related work and concludes the paper.

2. Overview of the Problem

This section contains an overview of the problem. First, it describes feature location in models, and then it presents an overview of the search strategies that are applied here to address the problem. Finally, it summarizes the measures that are used to report and characterize FLiM problems.

2.1. Feature Location in Models FLiM

A feature is traditionally defined as "a logical unit of behavior specified by a set of functional and non-functional requirements" [20], although the definition of feature is itself subject of study [21]. Feature Location [9, 7] is known as the process of identifying the set of software artifacts that realize a specific feature. Most of the works from the literature focus on retrieving the portions of source code that are relevant for the feature; however the artifact identified can be anything that is relevant for the feature, such as models, documentation, technical reports, etc.

Model-Driven Engineering (MDE) [3] is a generalization of Model-Driven Architecture, which is a framework for software development proposed by the Object Management Group (OMG) where models are central in the development of software. In MDE, all models are built using a specific language that is formalized by a metamodel that contains the concepts and their relationships with each other for that specific domain. General purpose modeling languages (such as UML) can be used, but the common practice in industrial domains is the use of Domain-Specific Languages (DSL), which are mainly built using the Meta Object Facility (MOF) [22] metamodel for modeling languages.

When the artifacts where the features are being located are software models we are refering to Feature Location in Models (FLiM) [11, 10]. The goal is the same (to identify the parts of the model that are relevant for a specific feature), but the result will come in the form of a model fragment (a subset of the elements of a parent model) instead of a portion of source code.

In the context of SPLs and MDE, the common practice is to generate the source code of the products using model transformations [3]. This results in source code that generally contains less embedded domain knowledge (as the developers have formalized it in the models or the transformations, but not in the resulting code); therefore, feature location techniques applied to this code can produce worse results. Another common problem when following a generative approach is the desynchronization between the models and the generated code, where modifications are done at the resultant code level only. Instead, modifications should be done in the models or could be lost when the code is re-generated. Both these issues argue that the maintenance tasks (such as the feature location) should be done at the model level. Additionally, there are systems where the models are not turned into source code or this transformation is done at runtime [17], and therefore feature location cannot be performed at source code level and in those systems feature location has to be undertaken at the model level.

Figure 1 shows an example of the metamodel, the models, and the model fragments used by one of our industrial partners to develop the firmware of their induction hobs. For the sake of simplicity and to preserve intellectual property rights, the metamodels, models, and model fragments shown in the paper are simplified versions of the original ones. However, all of the work presented in this paper has been done using the original models from our industrial partners.

The top-left part of Figure 1 shows the metamodel of the Induction Hob Domain-Specific Language, which is one of the languages used by our industrial partners to build their products. It formalizes the concepts ('Induction Hob', 'Inverter', 'Provider Channel', 'Power Manager', 'Consumer Channel', 'Inductor') with their properties ('pow' in the 'Inverter') and their relationships with each other (an induction hob can aggregate 0 or more inverters through the 'inverters' relationship; a 'Provider Channel' is related to exactly one inverter through the 'from' relationship, etc).

The bottom-left part of Figure 1 shows a model that was built using the IHDSL metamodel. This is just one of the multiple induction hob models that can be built using the presented metamodel. In this example, the induction hob aggregates a single 150 power inverter that is connected (through a provider channel) to a single power manager that is connected (through a consumer channel) to a single inductor. The dotted lines show the relationship between the metamodel concepts and the model. Additionally, the bottom part shows the encoding of the model[1]; this encoding will be used to define model fragments on the model which is explained in Subsection 2.2.1.

The bottom-right part of Figure 1 shows a model fragment that was defined on the previous model. The presence or absence of each of the elements of the model (concept, properties, or relationships) is represented

[1]In this example, the root element (Induction Hob) and the containment relationships (inverters, pchannels, pmanagers, pchannels, inductors) have been omitted for simplicity)

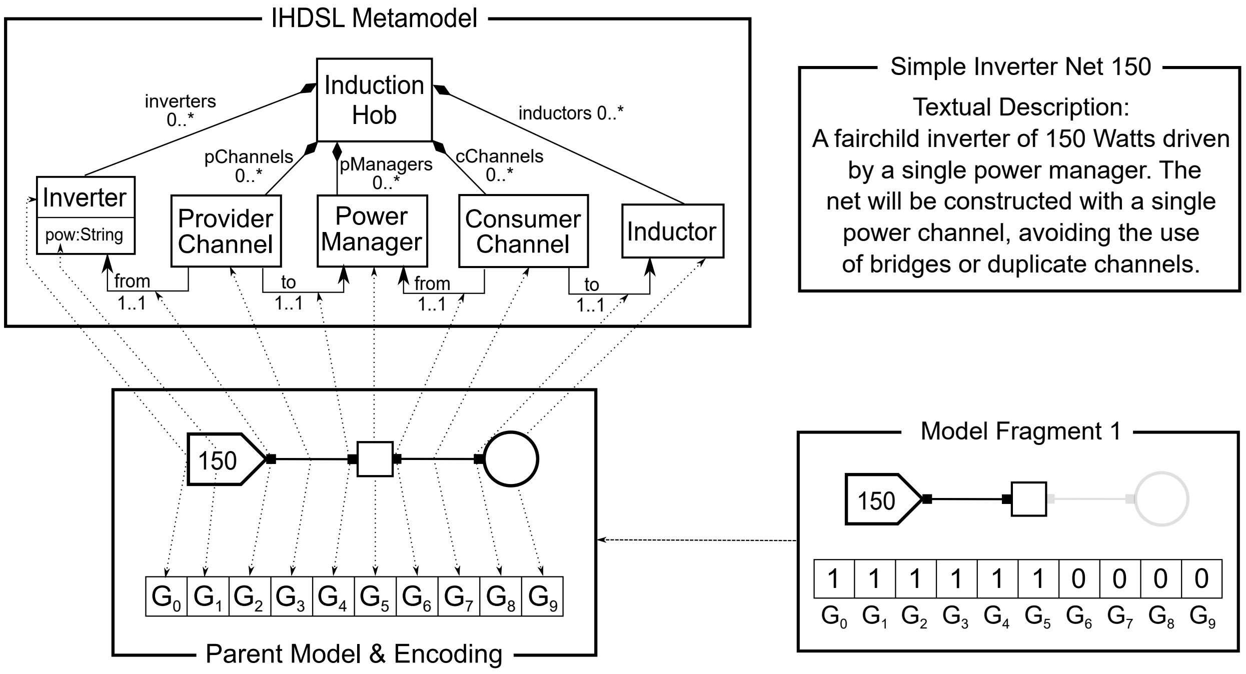


Figure 1: Example of a Model Fragment from the Induction Hob Domain-Specific Language

by a bit in the encoding array. All of the model fragments are defined in relation to a parent model (bottom-left) since the fragment is just a subset of the model elements of the original model. For instance, Model Fragment 1 is a model fragment that contains the inverter, the provider channel, and the power manager of the original model.

When performing feature location in models, we identify the model elements of a parent model that are relevant for a specific feature in the form of a model fragment. For instance, Model Fragment 1 (Figure 1, bottom-right) could represent the 'Simple Inverter Net 150' feature, whose description is shown in Figure 1, top-right. The process of feature location presented in this work will use descriptions such as the one shown to identify the model elements that realize that particular feature. These descriptions should be constructed by the owner of the models and will usually come from internal documentation of the products, comments or annotations in the models, bug reports, and oral descriptions from the engineers. As a result, the language used to write the description will be similar to the language used for the model elements and properties, including domain-specific terms.

When performing feature location in source code, different information retrieval techniques have been applied [23] and the granularity of the results vary from complete source code files to single statements of the source code. However, when performing FLiM, the result will come in the form of a model fragment (a subset of the elements of a parent model) instead of a portion of source code. Information retrieval techniques can be applied to a model fragment to determine its similarity with a textual query, but we need a method to generate different model fragments so they can be evaluated by the information retrieval technique.

Generating model fragments is not a trivial task, as there are restrictions present in the modeling language that determine how the models are built. In addition, the number of model fragments that can be generated from a single model fragment can grow too big (up to 10^{29} elements for a model with 500 elements), yielding a search space that is too big to be explored manually [24]. Search strategies are applied to help in traversing the search space and to determine what model fragments should be evaluated as realization of the feature being located. Next subsection presents these search strategies.

2.2. *Search-Based Model-Driven Engineering (SBMDE)*

Search-Based Software Engineering attempts to address software engineering problems through the application of already existing search strategies or metaheuristics [1]. Similarly, Search-Based Model Driven Engineering (SBMDE) deals with the application of searches or metaheuristics to specific MDE problems, such as the location of features in models. Part of the success of this emerging field resides in its versatility [1, 5] as many of the MDE problems can be expressed in a way that can be addressed by an already existing search strategy or metaheuristic. To do so, only three key ingredients are needed: 1) an encoding of the problem that can be used to represent the candidate solutions; 2) a fitness function that can be used to assess how good each candidate solution is as a solution to the problem; and 3) a set of genetic operations that can be applied to modify and evolve the set of candidate solutions.

There are many different search strategies and metaherustics, but all of them share similar ideas. Candidate solutions (that follow a specific encoding) are evaluated (with the fitness function) and evolved (by applying the genetic operators) until more optimal solutions are found. Different search strategies will use different operations to evolve the candidates, different sets of candidate solutions evolved in parallel or different ways of combining those ingredients, but all of them try to find an optimal solution to the problem.

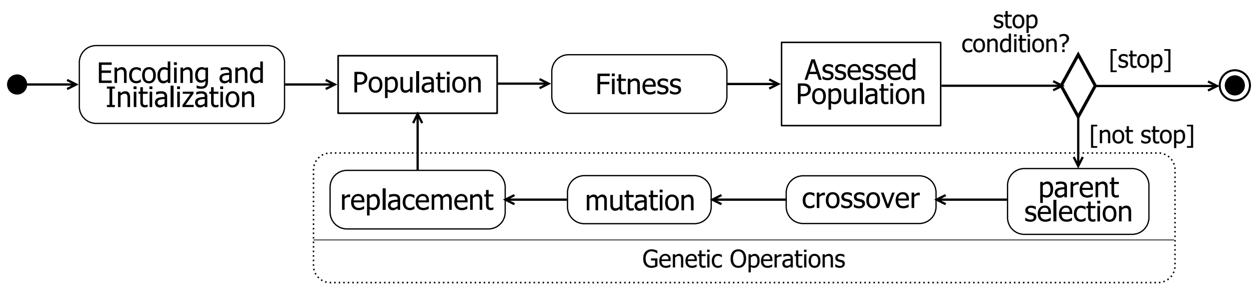


Figure 2: Overview of an Evolutionary Algorithm

Figure 2 shows an overview of a generic evolutionary algorithm (the search strategy that is most used by the SDMDE community) [5]). First, a population is initialized following a specific encoding (see Figure 2, left). Then, the loop starts and the fitness of each individual is calculated (see Figure 2, center). Individuals of the population are then evolved by applying a set of genetic operations, the most common being a selection of parents, followed by a crossover a mutation and a replacement, (see Figure 2, bottom)). This process of evaluation and evolution is repeated until the stop condition is met (see Figure 2, right) and then the search ends. Next subsections present the three key ingredients (encoding, fitness function and genetic operations) and the five search strategies used in this work.

2.2.1. *Encoding*

FLiM results in model fragments, and, therefore, the candidates will be model fragments. To represent and manipulate those candidates, we need an encoding that is capable of describing those model fragments. The encoding presented in Figure 1 is the one used by the five search strategies and, therefore, all of them will be able to create the same model fragments. This encoding allows fragments of a parent model to be defined by expressing the presence or absence of each of the model elements of the parent.This encoding allows the fast manipulation of the individuals (by the search strategies) and the generation of random individuals. In addition, since the encoding ensures that model fragments are subsets of the original model, the resulting model fragments are well-formed and valid (i.e., the model fragments keep the conformance to the metamodel) although they could be incomplete (the feasibility of the model fragments is not the focus in this work, detailed work on this topic can be found elsewhere [25].

2.2.2. *Fitness*

When applying SBMDE, a fitness function is used to evaluate how good each of the candidate solutions is as a solution to the problem. In other words, if we are locating features, we need to evaluate how good a model fragment is as the realization of the feature being located. In this case, features are represented by a feature description obtained from the engineers of our industrial partners (for an example, see Figure 1, top-right).

The feature description is provided in natural language and the similarity of a model fragment with the feature description is assessed through Information Retrieval techniques. Specifically, we apply Latent Semantic Analysis (LSA) [26] to determine the degree of similarity between the textual description provided and the solution candidates being generated by the search strategies. LSA has been used before as fitness for the problem of FL in source code and in models with good results [11].

LSA builds a vector representation of the query and a set of text documents in the form of a *term-by-document co-occurrence matrix*. This matrix is composed of several columns (each column corresponds to a document, and the last column corresponds to the query) and rows (each row corresponds to a relevant term). The cells of the matrix capture the number of occurrences of each of the relevant terms in each of the documents and in the query.

To apply LSA to model fragments, the text in each of the model fragments is extracted and turned into one of the documents used by LSA. In other words, the text is extracted from the model fragment processing each of the concepts, relationships, or attributes; depending on whether or not each element is part of the model fragment, it will be included on the resulting document. The text from the feature description is used to build the query used by LSA.

Additionally, all of the texts need to be homogenized by applying well-known natural language processing techniques [27]. This is done to ensure that the language used by the different documents is similar and can be properly compared. In addition, this serves as a filter to remove the irrelevant words that could add noise to the process. This process is performed with both the textual description and the text obtained from model fragments.

Tokenization: First, all texts are divided into separate words. This is usually done using a white space tokenizer that breaks strings whenever it finds a white space. However, depending on the source of the texts, a different tokenizer may be applied (e.g., names of model elements do not allow for white spaces, so they usually follow a camelCase naming).

Parts-Of-Speech: After breaking the documents into words, a grammatical analysis must be performed to tag each word based on its role in the document. This allows some grammatical categories whose relevance is low to be removed (e.g., conjunctions, articles, or prepositions). After this step, only the words with relevant information are part of the document.

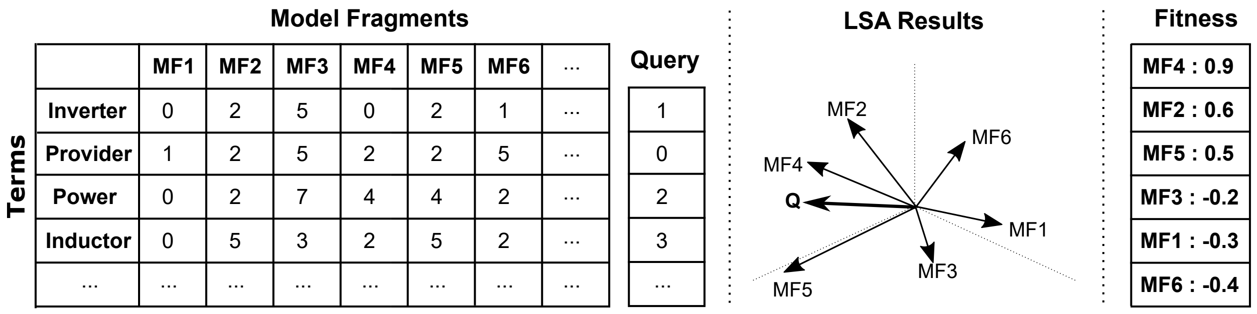
Stop-words: Some words may not contain semantic information when used in particular domains (given their widespread use), so they can be automatically removed if a list of stop-words or domain terms is provided (e.g., in the induction hob domain, the word 'hob' will appear too many times, resulting in not being useful at all).

Stemming: Finally, the language is unified through stemming techniques. In other words, each word is reduced to its root, allowing different words that refer to similar concepts to be considered as the same word (e.g., plurals are turned into singulars, or verb tenses are unified by using the root).

Figure 3 (left) shows an example of a co-occurrence matrix applied to model fragments. Each cell of the matrix shows the number of occurrences of a specific term (rows) in the text extracted from the model fragment (columns). The last column shows the occurrences for the query of the feature being located.

After the matrix is built, it is decomposed and normalized into a set of vectors using Singular Value Decomposition (SVD) [26]. SVD performs a reduction of the dimensional space, projecting the original matrix into a lower dimensional space. Then, similarities between the vectors representing each of the documents (or , in this case, model fragments) can be calculated. Figure 3 (center) shows a representation of the model fragments after the SVD has been applied.

$$fitness(mf) = \cos(\theta) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \tag{1}$$



Model Fragments

Terms	MF1	MF2	MF3	MF4	MF5	MF6	...
Inverter	0	2	5	0	2	1	...
Provider	1	2	5	2	2	5	...
Power	0	2	7	4	4	2	...
Inductor	0	5	3	2	5	2	...
...	...	...	...	...	...	...	...

Query
1
0
2
3
...

Fitness
MF4 : 0.9
MF2 : 0.6
MF5 : 0.5
MF3 : -0.2
MF1 : -0.3
MF6 : -0.4

Figure 3: Example of a co-occurrence term-by-document matrix (left), Singular Value Decomposition representation (center), and fitness values (right).

Similarities between the vectors are calculated as the cosine similarity, as shown in Equation 1 where the fitness of a model fragment (mf) is calculated as the cosine of the angle θ, which is the angle formed between the vector representing the latent semantic of the model fragment (A) and the vector representing the latent semantic of the query (B). This results in values between -1 (no similarity at all) and 1 (total similarity). The fitness results are shown in Figure 3 (right), where all of the model fragments have been ordered according to their similarity with the query.

2.2.3. Genetic Operations

The four genetic operations [28, 10, 11] applied to evolve the individuals are independent from the domain and can be applied to any model fragment following the presented encoding. The idea is to reuse the parents to generate new individuals as occurs in nature.

First, the **Parent Selection** operation, selects two individuals based on their fitness values, giving more probability of being selected to fitter individuals. Then, the **Crossover** operation mixes both parents into two new individuals that inherit their genes from both of the parents. A random binary string with the same length as the parents is generated to indicate which genes will be inherit from each parent. Then, the **Mutation** operator modifies the value of some of the genes of the new individual following a mutation probability, changing its value to the opposite. Finally, the **Replacement** operator introduces the new individuals generated into the population, some of the least fittest individuals of the population are replaced by the new individuals.

2.2.4. Search Strategies

This subsection describes the five different search strategies applied in this work to perform feature location over the models from our industrial partners. The different search strategies have been selected as they have been previously applied to similar problems in literature [29, 11, 30]. All search strategies will be using the same encoding for the individuals and guided using the same fitness function. Therefore, the differences reside on how the solution space is traversed looking for the best solution. The pseudo code of each of them is Available in the Appendix.

The first search algorithm used in this work is a traditional **Evolutionary Algorithm (EA)** that iterates a population of model fragments and evolves them using the genetic operations described above. As output, the algorithm provides the model fragment that best realizes the feature according to the fitness function. The generation of model fragments is done by applying genetic operators that have been adapted to work with our encoding of model fragments. Algorithm 2 (Available in the Appendix) shows the pseudo code of the EA being used.

The second search strategy that we use in this experiment is a standard **Random Search (RS)** that is used as a sanity check. We want to determine if the presented search strategies perform better than mere chance (represented by this Random Search algorithm). If the search strategies do not add value and only impose a burden over the underlying LSA driving the search, this algorithm should perform better than the rest. The pseudo code for the search strategy can be seen in Algorithm 3 (Available in the Appendix).

The third search strategy applied is a **Hill-Climbing (HC)** algorithm (specifically we use a steepest ascent hill-climbing with replacement [30]). The search starts with a random individual and then some neighbors (small modifications of the individual) are assessed to find the best one (in terms of the fitness function). Those steps are repeated until the stop condition is met. The quality of the solutions found by this algorithm greatly depends on the first individual generated, as the search is reduced to its neighbourhood. We applied the algorithm as outlined in Algorithm 4 (Available in the Appendix).

The fourth search strategy that is used is the **Iterated Local Search (ILS)** with random restarts. Since 1981, alternative versions of this algorithm have been used ([31, 32]). The strategy searches for a local optimum during a certain period of time. Then, it switches to a near hill (restart) and performs a new search for a local optimum. The main particularity of this search strategy is the selection of a new hill (which is performed by the *newHomeBase* function). We applied this search strategy as outlined in Algorithm 5 (Available in the Appendix).

The fifth search strategy compared in this experiment is a **hybrid between an Evolutionary algorithm and Hill-climbing (EHC)**. It leverages the hill-climbing capabilities of searching local optimum values, while it makes use of the crossover operation to move to different hills (and then repeat the local search again through the hill-climbing). We used this search strategy as outlined in Algorithm 6 (Available in the Appendix). First, a population of model fragments is randomly generated (P, line 2). Then, hill-climbing is performed to explore N_{size} neighbours (line 1) around each of the individuals in the population. If a better solution is found in this process, it is saved in *Best* (lines 6-11). Finally, when the local searches of the hill-climb have ended, new individuals are obtained through the *breedPopulation* operation (line 12), adding new hills that will be explored in the next iteration. The process is repeated until the *StopCondition* is met (lines 3-12).

2.3. Measures to Report Model Fragment Location Problems (MR-MFLP)

In [19], the authors tackle how Model Fragment Location Problems (MFLP) are reported in the literature. They realize that most of the work from the literature only reports the size of the the search space (the model where the search is being performed) and propose a set of five measures to better report and characterize model fragment location problems (MR-MFLP), as is the case for feature location in models. In this work, we apply those measures to the feature location problems from our industrial partners and look for relationships in the results obtained by the five search strategies and the values for the different MR-MFLP.

The first two measures are related to the search space, which in this case is the parent model where the feature is going to be located. The other three measures characterize the solution, which is the model fragment that realizes the feature being located. Below, we describe the five measures and how they are applied in this work:

Search Space Volume (SS-Volume): This measures the number of models where the location is taking place. Sometimes, the engineers that are performing the location of a feature are not able to limit the search to a single model and need to search in more than one model. In the example from Figure 1 the search is only being performed in a single model, so the SS-Volume would have a value of 1. When the SS-Volume is bigger than one, we repeat the feature location process, resulting in one model fragment located in each of the models provided. This results in the need of several model fragments to represent a single feature.

Search Space Size (SS-Size): This measures the number of elements that the model(s) where the feature is located has/have (i.e., the number of concepts, relationships and properties used to build the parent model). Following with the example in Figure 1 (bottom left), the SS-size would have a value of 10 (the same as the number of bits in the encoding).

Model Fragment Density (MF-Density): This measures the ratio between the size of the model fragment being located and the size of the parent model(s) (i.e., the ratio between the number of model elements that form part of the solution and the total number of model elements present in the search space). In the example of Figure 1, the MF-Density would be the number of elements present in the solution (6) divided by the total size of the parent (10), giving a value of 0.6 (6/10).

Model Fragment Multiplicity (MF-Multiplicity): This measures the number of times that the solution appears in the search space. Some of the features being located may appear several times in the search space. The example in Figure 1 would correspond to a MF-Multiplicity of 1.

Model Fragment Dispersion (MF-Dispersion): This measures the degree of connection between the different elements of the model fragment. To calculate it, we first identify the number of groups of connected elements in the model fragment. Then, MF-Dispersion is computed as the ratio between the number of groups and the number of model elements. In the example in Figure 1, there is only one group (as all of the elements are connected), so the MF-Dispersion would be the number of groups (1) divided by the total number of elements present in the solution (6), giving a value of 0.17.

3. Controlled Experiment Design

This section presents the design of the controlled experiment that is carried out in this work. Subsection 3.1 introduces the research questions that the experiment addresses. Then, Subsection 3.2 presents the objects of the experiment, which is the set of 1895 MFLPs obtained from industrial SPLs. Subsection 3.3 describes the experimental procedure, including a set of four performance metrics (Precision, Recall, F-Measure, and MCC) and the details of the implementation of the five search strategies.

3.1. Objective

The experiment has been designed to address the following research questions:

RQ1: Which search strategy (EA, EHC, HC, ILS, and RS) provides the best results in terms of performance for the MFLPs?

RQ2: What are the differences in performance (Precision, Recall, F-Measure, and MCC) between the HIGH and LOW values of the MR-MFLP (SS-Size, SS-Volume, MF-Density, MF-Multiplicity, and MF-Dispersion)?

RQ3: Are there any interactions between the MR-MFLP values (SS-Size, SS-Volume, MF-Density, MF-Multiplicity, and MF-Dispersion) and the search strategies (EA, EHC, HC, ILS, and RS) that affect the strategies' performance?

3.2. Objects

This study was performed on a total number of 1895 MFLP obtained from 40 industrial SPLs from two different industrial domains. The following subsections introduce both domains (induction hob firmware and train control software) and show how oracles provided by the companies are organized to be useful in evaluating the different search strategies applied to the problem of feature location in models.

3.2.1. Induction Hob Firmware

The first domain used in this study was obtained from BSH, the leading manufacturer of home appliances in Europe. Their induction division has been producing induction hobs under several brands that are sold in different countries around the world for the last 17 years. Each induction hob that is available on the market has different configurations of hotplates and characteristics that are made possible though the use of software.

An induction hob is composed of a set of inverters and hotplates that are interconnected and managed through a network of power managers and relays. Basic induction hobs consist of a set of inverters that are directly mapped to the inductors through simple channels. High-end induction hobs include more complex features such as several inverters that share the load to provide current to the same inductor, optional inductors that can be enabled or disabled based on the characteristics of the pot placed on top, or even a dynamic cooking surface where groups of inverters that provide current and inductors that receive that current are determined at runtime.

To develop the products, the company maintains different SPLs based on the needs of different markets (different brands, countries, or platforms of induction hobs). Specifically, for this study we use the models and features obtained from six different SPLs, with a total number of 608 features that can be part (or not) of the products being developed. The description of each of the features (obtained from the domain experts), the models of the products, and the model fragments corresponding to each of the features have been provided as an oracle and will be used to evaluate the different search strategies.

3.2.2. Train Control Software

The second domain used in this study was obtained from CAF, a worldwide provider of railway solutions. Their solutions can be seen all over the world in the form of regular trains, subways, light rails, monorails, etc. Although all of these solutions may seem different, all of them share a substantial, common core and are formalized using the same DSL.

A train unit is composed of several vehicles and cabins, each of which needs to be furnished with different pieces of equipment. Each equipment piece is designed and manufactured by a different provider and performs a specific task for the train, but all of the equipment needs to be coordinated to drive the train properly. Some examples of that equipment are the traction system, the compressors that feed the brakes, the pantograph that is used to get power from the overhead wires, or the sand delivery system that is used to improve grip when the rails are slippery.

The control software of the train is responsible for the management and cooperation of all of the equipment, achieving the train functionality while guaranteeing compliance with the specific regulations of each country. This control software is generated from a software model that represents the train, applying a model-to-text transformation.

Each railway solution is developed for a specific client, which requires different characteristics for the train (e.g., speed, passenger capacity, energy system). Each solution will be deployed in different countries, with different weather conditions (e.g., locomotives designed to provide service under extreme environmental conditions up to 55ºC for Saudi Arabia) and regulations that the train must adhere to (e.g., the different levels of installed redundancy). Additionally, some railway solutions can be reconfigured dynamically based on specific needs (e.g., use different configurations of train units based on the number of passengers).

All of this variability leads to the development of different families of products that are managed as different SPLs. Specifically in the case of CAF, there are 34 different SPLs composed of a total number of 1287 features that can be part (or not) of the products generated from those SPLs. The description of each of the features (formalized in the internal documentation of the company by the engineers that maintain the SPLs,) the models (where the features must be located), and the model fragments (the approved feature realization that is currently being used by the company) have been provided to be used as an oracle that will be used to evaluate the different search strategies.

	Search Space (SS)		Model Fragment (MF)		
measure	size	volume	density	multiplicity	dispersion
min	889	1	0.0081	1	0.0053
Q_1	1023	9	0.0208	2	0.0462
Q_2	1699	11	0.0251	4	0.0600
Q_3	1869	14	0.0307	9	0.0667
max	2241	88	0.1834	18	0.1111
mean	1553	14	0.0316	6	0.0604
sd	425.2792	16.63175	0.02705922	4.569013	0.02461526

Table 1: Report of the search problems used for the experiment

In total, there are 1895 different feature location problems that have been provided by our industrial partners and will be used as oracles.In other words, each feature location problem is composed of a feature description, models where the feature should be located, and the realization of the feature that is currently

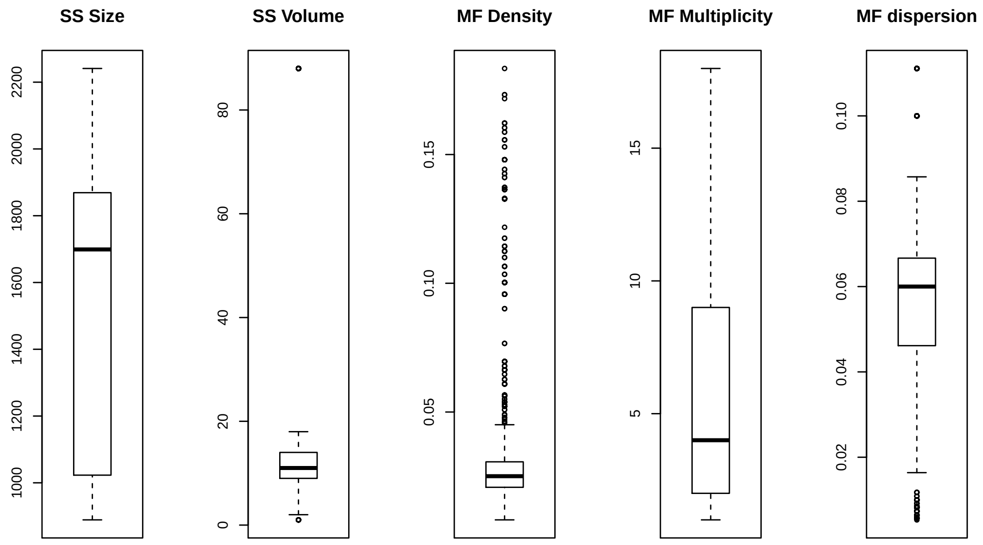


Figure 4: Boxplots with the value for each MR-MFLP obtained from the 1895 MFLPs

accepted by the companies (so it can be used as a golden set to measure the performance of the different search strategies). Using two different domains from real industrial environments increases the variety of feature location problems studied and their casuistry. As a consequence, the generalization of our experiment is improved. Table 7 (available in the Appendix) shows how many MFLPs are taken from each of the SPLs in the domains. Each SPL contains models for a different subsystem of the domain; therefore, the variations in number of MFLPs for each different SPL. In this work we use the complete set of MFLP present in the SPLs provided by our industrial partners and the only check that has been performed is to remove incomplete MFLP (e.g.: the description is not provided or the solution model fragment is not formalized)

In this work we use a median-based discretization of the MR-MFLP. For each MR-MFLP in the set of feature location problems we split the value at the sample median, thereby defining HIGH and LOW groups; this approach is referred to as a median split [33, 34]. Figure 4 and Table 1 show the values for each of the five MR-MFLP achieved by the MFLPs that are used in this study. For each MR-MFLP, values above the median will be regarded as HIGH while values below the median will be considered LOW. For instance, the median of the SS-Size of all the MFLP is 1699; all MFLPs whose SS-Size value are above 1699 are defined as HIGH for SS-Size; in contrast, all MFLPs whose SS-Size value are below 1699 are defined as LOW for SS-Size.

3.3. Experimental Procedure

Figure 5 shows an overview of the experimental procedure followed. The part on the left shows the Oracles obtained from the 40 SPLs of our industrial partners. Overall, the oracles contain a total of 1895 different MFLPs. Each MFLP is composed of: 1) a feature description provided by the software engineers that work with those SPLs, 2) the set of models where the feature can be present, and 3) a model fragment that realizes the feature. In other words, each MFLP in the oracle is composed of a feature location problem and the solution to that problem. The solution is only used as part of the experiment to assess the performance of each of the search strategies, but is not used by the search strategy to explore the search space and is not needed when the approach is applied.

The part in the center shows the five search strategies driven by LSA. Each strategy will be run to locate the feature present in each of the MFLPs 30 different times (to ensure an average value, given the stochastic nature of the search strategies) as recommended in the literature [35]. Each run of a search strategy will result in a model fragment that realizes the feature. Then, those model fragments will be

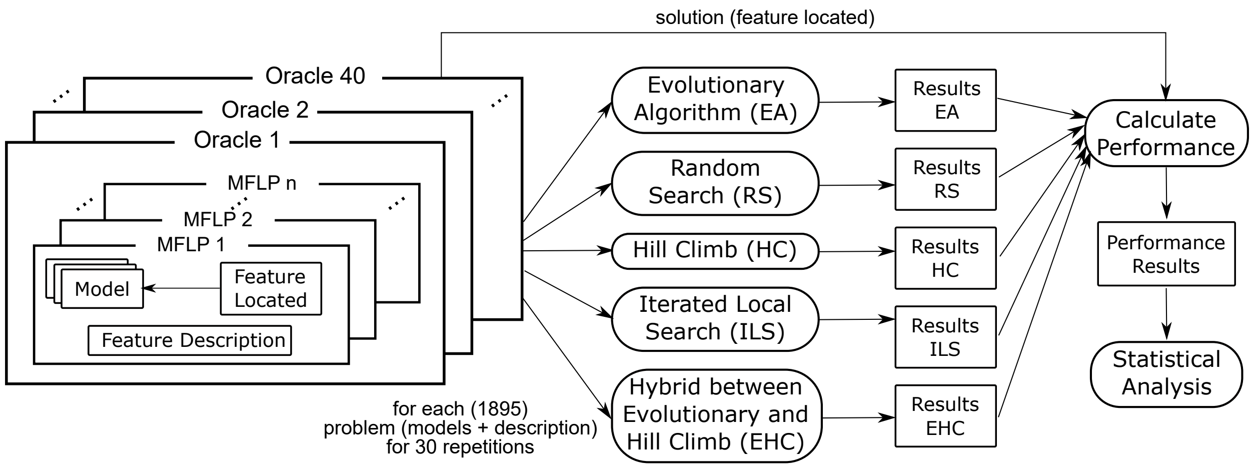


Figure 5: Overview of the experimental procedure. The 40 oracles containing the 1895 MFLP (left) are feed to the five search strategies that will locate the features using the LSA fitness and the genetic operations (middle). The results will be processed to calculate the four performance metrics and those results will be used for the statistical analysis (right).

compared against the solutions from the oracle to calculate the performance metrics (see Figure 5, top-right). The 30 different values of the performance metrics achieved by each search strategy and MFLP were averaged and the standard deviation in that case was below 1% for all them, showing the robustness of the search strategies and ensuring that solutions of similar quality are produced each time that the searches are performed. Finally, a statistical analysis will be performed on the performance results. The following subsections provide details about the implementation of the search strategies and present how the performance metrics are calculated.

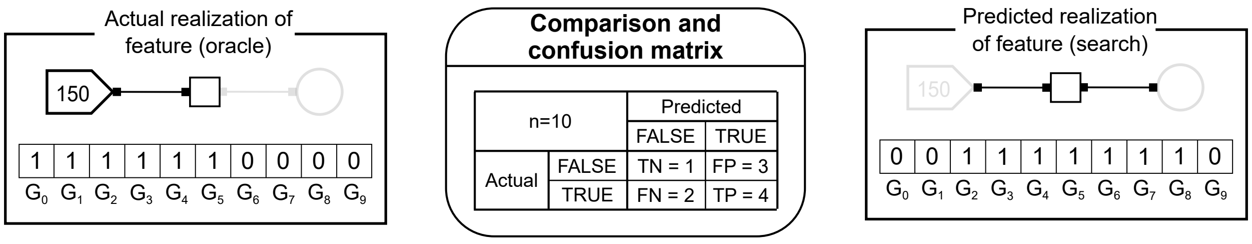
3.3.1. Implementation Details

The presented search strategies were implemented within the Eclipse environment. We have used the Eclipse Modeling Framework [36] to manipulate the models and model fragments from our industrial partner. The search strategies are based on the watchmaker framework for evolutionary computation [37]. Using it we have created a custom encoding and genetic operators to implement the search strategies. The information retrieval techniques that were used to process the language were implemented using OpenNLP [38] for the POS-Tagger and the English (Porter2) [39] as stemming algorithm. Finally, the LSA fitness has been implemented using the Efficient Java Matrix Library (EJML [40]). We performed the execution of the search strategies using ten identical laptops (model Asus GL502VM-FY040T) with i7-6700HQ processors (2.6 Ghz clock speed), and 16 GB of RAM. All of them were running Windows 10 Pro N 64 bits as the hosting operative system and the Java SE runtime environment (build 1.8.0_73-b02).

Parameter	Description	Value
Pop_{size}	Size of the population being evolved	100
P_c	Probability of crossover	0.75
P_m	Probability of mutation where n is the number of genes of the individual	$1/n$
$stopCondition$	time budget allocated for the execution of each MFLP	60s.
Rep_{size}	number of individuals replaced each generation	20

Table 2: Parameters for the search strategies

Table 2 shows the values of the parameters that need to be configured for the search strategies. We use default parameter values extracted from the literature [35] (and previously tested for this search strategies [11]). Regarding the genetic operations, the selection operation is a roulette wheel selection, the crossover is


Actual realization of feature (oracle)
150
1 1 1 1 1 1 0 0 0 0
G0 G1 G2 G3 G4 G5 G6 G7 G8 G9
Comparison and confusion matrix
Predicted realization of feature (search)
150
0 0 1 1 1 1 1 1 1 0
G0 G1 G2 G3 G4 G5 G6 G7 G8 G9

n=10		Predicted	
		FALSE	TRUE
Actual	FALSE	TN = 1	FP = 3
	TRUE	FN = 2	TP = 4

Figure 6: Example of the calculation of the confusion matrix and quality metrics (Precision, Recall, F-Measure, and MCC)

a multiple point crossover adapted to work with model fragments [10] and the replacement operator follows elitism, preserving the fittest individuals across generations. The *stopCondition* will represent the time budget allocated for each of the runs for each of the algorithms. Some prior test have been run to determine the time needed by each search strategy to reach the convergence point (from that point there are no further improvements in the solutions provided by the search). The same time (60 seconds) is used for all the algorithms, ensuring the fairness of the experiment.

3.3.2. Performance Metric Calculation

To evaluate how good each of the search strategy is, it is necessary to determine the accuracy of the resultant MFs. To do this, the model fragments will be compared to the solution that is present in the oracle in order to assess its quality in terms of Precision, Recall, F-measure and MCC.

Figure 6 shows an example of the assessment of the solution quality. Figure 6 (left) shows a feature realization obtained from the oracle, which is the actual feature realization provided by our industrial partner (the search strategy should ideally produce that model fragment as the output of the feature location). Figure 6 (right) shows the predicted realization of the feature, which is the output of the search strategy. Both the actual and predicted fragment models are compared using a confusion matrix (see Figure 6, center). Tables of this kind are used to describe the performance of classification models (the search strategies) on a set of known test data (the oracle). To build this matrix, each of the elements from both models fragments is compared in order to measure the number of:

True Positives (TP): elements predicted as true (in the solution) that are actually true (in the oracle), such as G_2 in Figure 6.

True Negatives (TN): elements predicted as false (in the solution) that are actually false (in the oracle), such as G_9 in Figure 6.

False Positives (FP): elements predicted as true (in the solution) that are actually false (in the oracle), such as G_7 in Figure 6.

False Negatives (FN): elements predicted as false (in the solution) that are actually true (in the oracle), such as G_0 in Figure 6.

Once the confusion matrix is built, it can be used to calculate performance metrics. In this case, we will calculate the precision, recall, F-Measure, and MCC. Precision indicates the number of elements from the solution that are correctly placed according to the oracle, using a range from 0 (no element is correctly placed) to 1 (all elements are correctly placed). This is calculated in Equation 2; the example in Figure 6 has a precision value of 0.57.

$$Precision = \frac{TP}{TP + FP} = \frac{4}{4+3} = 0.57 \tag{2}$$

Recall indicates the number of elements from the oracle that have been properly retrieved by the search strategy, using a range from 0 (no element is present) to 1 (all elements from the oracle are present in the prediction). This is calculated in Equation 3; the example in Figure 6 has a recall value of 0.67.

$$Recall = \frac{TP}{TP + FN} = \frac{4}{4+2} = 0.67 \tag{3}$$

The F-Measure is the harmonic mean between Precision and Recall. It is possible to obtain high values of precision or recall separately (the empty model fragment will obtain a 1 in precision, while the whole model fragment will always achieve a 1 in recall); therefore, there is a need for a performance indicator that combines both. This is calculated as Equation 4; the example in Figure 6 has an F-Measure value of 0.62.

$$F{-}Measure = \frac{2 \cdot precision \cdot recall}{precision + recall} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} = \frac{8}{8+3+2} = 0.62 \tag{4}$$

Finally, the Matthews Correlation Coefficient (MCC) is another alternative for describing the confusion matrix that has been widely used in the field of machine learning. It takes into account all of the values that are present in the confusion matrix, capturing the effect of the True Negative values (which are neglected by the other performance metrics). The MCC is reported in a range from 1 (perfect match) to -1 (total disagreement between the oracle and the prediction); a value of 0 is what is expected to be achieved by a random system. This is calculated in Equation 5; the example in Figure 6 has an MCC value of -0.09.

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP) \cdot (TP+FN) \cdot (TN+FP) \cdot (TN+FN)}} = \frac{4-6}{\sqrt{7 \cdot 6 \cdot 4 \cdot 3}} = -0.09 \tag{5}$$

4. Results

This section shows the results obtained following the experimental procedure described above. The analysis is divided into a descriptive analysis of the results followed by hypothesis testing.

4.1. Descriptive Statistics

4.1.1. Performance by Search Strategy

Table 3 shows the mean values and standard deviations that were calculated for each performance metric and each search strategy. Each row is the result of applying the search strategy (indicated in first column) to solve the 1895 feature location problems that are part of the case study. Similarly, Figure 7 shows the boxplots obtained from those results.

Search Strategy	Precision $\pm\ \sigma$	Recall $\pm\ \sigma$	F-Measure $\pm\ \sigma$	MCC $\pm\ \sigma$
EA	0.50 ± 0.27	0.52 ± 0.28	0.50 ± 0.27	0.49 ± 0.27
EHC	0.81 ± 0.17	0.88 ± 0.13	0.83 ± 0.15	0.83 ± 0.14
HC	0.21 ± 0.25	0.63 ± 0.21	0.27 ± 0.28	0.25 ± 0.30
ILS	0.57 ± 0.24	0.59 ± 0.24	0.58 ± 0.24	0.57 ± 0.24
RS	0.03 ± 0.03	0.51 ± 0.10	0.06 ± 0.04	0.00 ± 0.03

Table 3: Mean values and standard deviation for the Precision, Recall F-Measure and MCC values by each of the search strategies

There exists strong similarities among the mean values of Precision, F-Measure, and MCC for each search strategy. The best values were obtained by the Evolutionary Hill Climbing (EHC) search strategy, achieving mean values of 0.81 in Precision, 0.83 in F-Measure, and 0.83 in MCC. EHC is followed by Iterated Local Search (ILS), which achieved values of 0.57 in precision and MCC, and 0.58 in F-Measure. The third search strategy is the traditional Evolutionary Algorithm (EA), which achieved mean values of 0.5 in precision and F-Measure, and 0.49 in MCC. The fourth search strategy is the Hill Climbing (HC) achieving mean values of 0.21 in precision, 0.27 in F-Measure, and 0.25 MCC. Finally the Random Search (RS) obtained mean values of 0.03 in precision, 0.06 in F-Measure, and 0.0 in MCC.

However, for Recall, there are some differences compared to the other three performance metrics. The first three search strategies obtained mean values that are similar to the ones obtained in Precision, F-Measure, and MCC: 0.88 for the EHC, 0.59 for the ILS, and 0.52 for the EA. In contrast, HC and RS

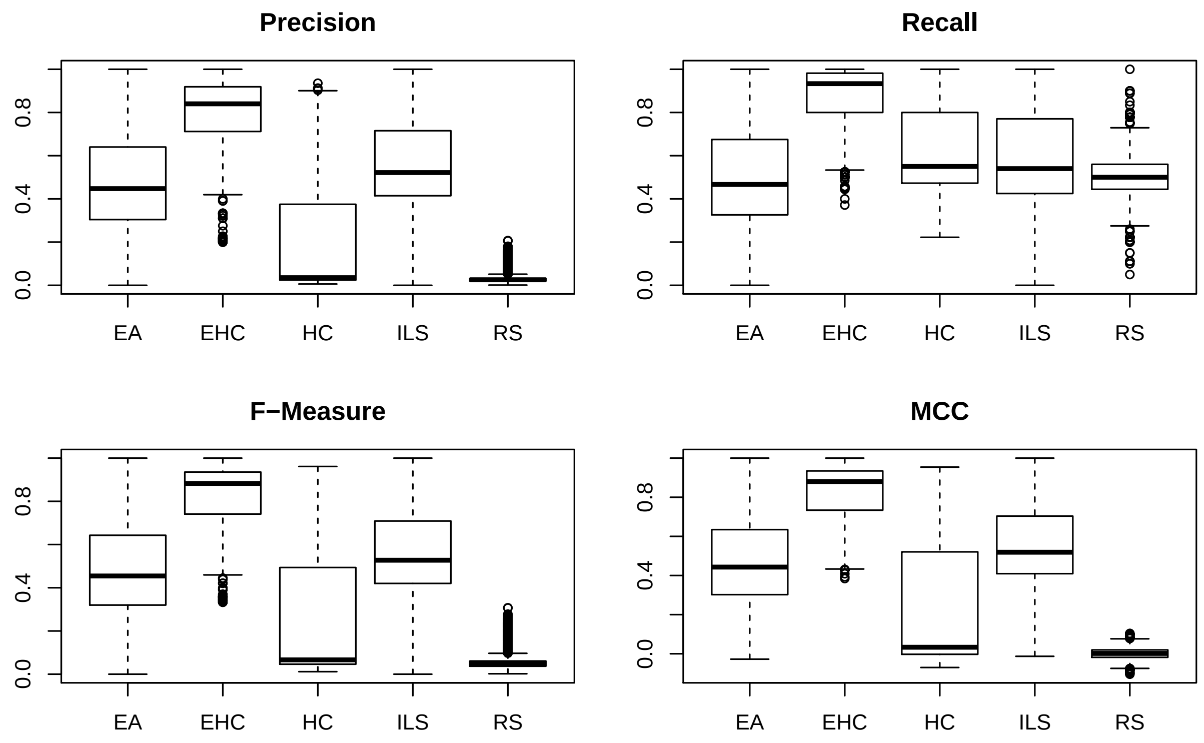


Figure 7: Boxplots of the Precision, Recall, F-Measure, and MCC values obtained by each of the search strategies

obtained Recall values that were much higher than the ones for Precision, F-Measure, and MCC: 0.63 for HC and 0.51 for RS.

The EHC provides the best results thanks to its balance between local search and global search. We have manually inspected some of the resulting model fragments and observed that the local search algorithms tend to properly identify portions of the feature when they are connected but tend to fail in identifying isolated elements (a single gene) or portions of the feature that are small (two or three bits).

The RS was included in the comparison as a sanity check, to determine if the search strategies where needed or the search space could be analyzed without the need of a search strategy. In the light of the results we can conclude that the search strategy is needed to properly traverse the search space; otherwise, the LSA is able to evaluate some model fragments but not enough to find a near optimal solution. Surprisingly, the RS was able to achieve recall values close to 50%. We have observed that medium sized model fragments (where around half of the elements of the model are part of the model fragment) tend to obtain good average fitness values and, therefore, result as the best model fragment found by the search strategy. As those model fragments contains around half of the elements of the model, they have a high chance of containing also elements that are part of the feature being located, resulting in those recall values. Those model fragments also contain several elements that are not part of the feature being located and thus their precision value is low.

From these results it is clear that there are some differences in the performance of each of the search strategies; however, we cannot see whether or not these differences are related to the different MR-MFLP.

4.1.2. Performance by Metric to Report Fragment Location Problem

Table 4 shows the mean values and standard deviations that were calculated for each performance metric by each MR-MFLP. Each row shows the results for each of the five MR-MFLP (SS-Size, SS-Volume, MF-Density, MF-Multiplicity, MF-Dispersion) for the set of feature location problems defined as LOW and

HIGH for each of the five MR-MFLP. Similarly, Figure 8 shows the boxplots obtained from those results.

Metric to report MFLP	Value	Precision $\pm \sigma$	Recall $\pm \sigma$	F-Measure $\pm \sigma$	MCC $\pm \sigma$
SS Size	LOW	0.50 ± 0.36	0.69 ± 0.26	0.53 ± 0.35	0.51 ± 0.36
	HIGH	0.34 ± 0.31	0.55 ± 0.20	0.36 ± 0.31	0.33 ± 0.33
SS Volume	LOW	0.42 ± 0.34	0.61 ± 0.24	0.44 ± 0.34	0.42 ± 0.36
	HIGH	0.43 ± 0.35	0.65 ± 0.25	0.46 ± 0.34	0.44 ± 0.36
MF Density	LOW	0.41 ± 0.35	0.64 ± 0.26	0.44 ± 0.35	0.43 ± 0.37
	HIGH	0.44 ± 0.33	0.61 ± 0.23	0.46 ± 0.33	0.43 ± 0.35
MF Multiplicity	LOW	0.49 ± 0.37	0.70 ± 0.26	0.52 ± 0.36	0.51 ± 0.38
	HIGH	0.35 ± 0.30	0.54 ± 0.20	0.37 ± 0.30	0.34 ± 0.31
MF Dispersion	LOW	0.46 ± 0.35	0.66 ± 0.24	0.49 ± 0.35	0.46 ± 0.37
	HIGH	0.38 ± 0.32	0.58 ± 0.24	0.40 ± 0.32	0.39 ± 0.34

Table 4: Mean values and standard deviation of the Precision, Recall, F-Measure, and MCC values obtained by each of the MR-MFLP

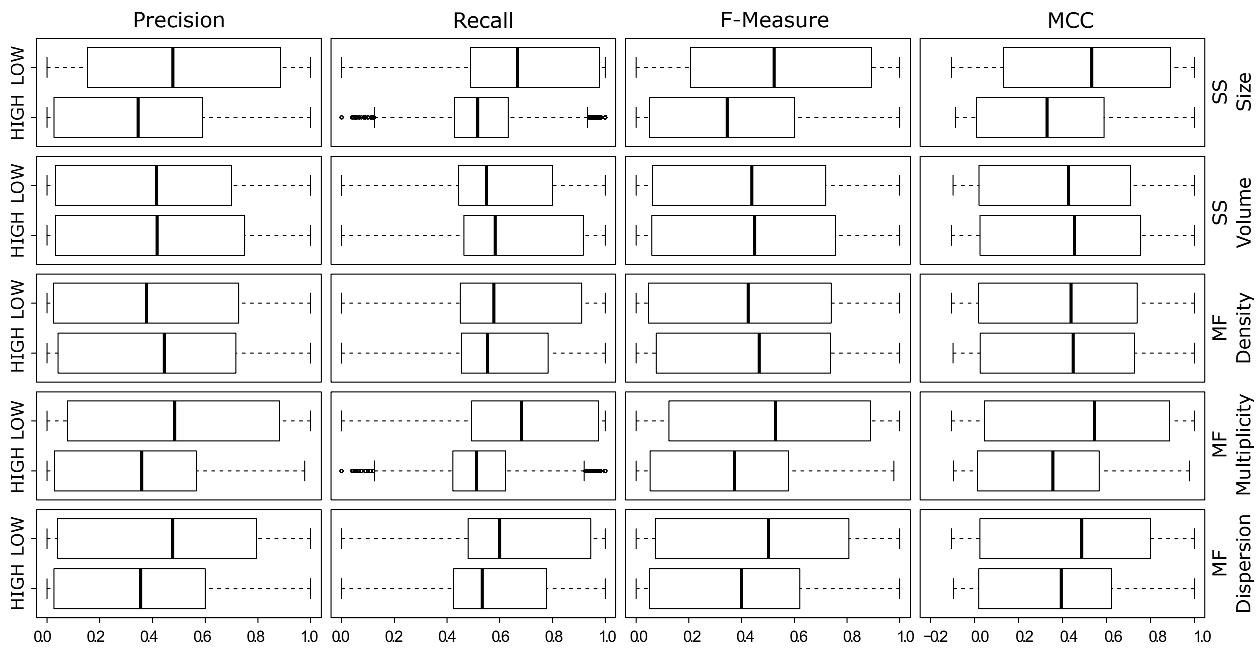


Figure 8: Boxplots of the Precision, Recall, F-Measure, and MCC values obtained by each of the five MR-MFLP. Each measure is reported as a two-value factor (HIGH and LOW).

The results show some differences when comparing the HIGH and LOW factor levels for the MR-MFLP. Specifically, the difference between a HIGH and a LOW SS-Size are above 0.15 points for the four performance metrics, obtaining better results when the SS-Size is LOW. Similarly, the differences between a HIGH and a LOW MF-Multiplicity are around 0.15 points for all of the performance metrics, achieving better performance when the MF-Multiplicity is LOW. The differences for MF-Dispersion are smaller, around 0.7 points for the four performance metrics, achieving better performance for LOW MF-Dispersion values. The differences between the LOW and HIGH factor level for the other two measures (SS-Volume and MF-Density) are much smaller, being below 0.4 in all cases.

4.1.3. Performance by Search Strategy and Metric to report Fragment Location Problem

Figure 9 shows the boxplots of the results for each of the four performance metrics grouped by the factor value of each of the MR-MFLP and search strategy. The figure is divided into four columns for the performance metrics (Precision, Recall, F-Measure, and MCC) and five rows for the MR-MFLP (SS-Size, SS-Volume, MF-Density, MF-Multiplicity, MF-Dispersion). For each row and column, there is a group of 10 boxplots showing the results of that particular performance metric (column) and MR-MFLP (row) for each of the five search strategies being analyzed (EA, EHC, HC, ILS, and RS) and each factor level of the MR-MFLP (LOW and HIGH). In addition, Table 8 (available in the Appendix) shows the mean values and standard deviations.

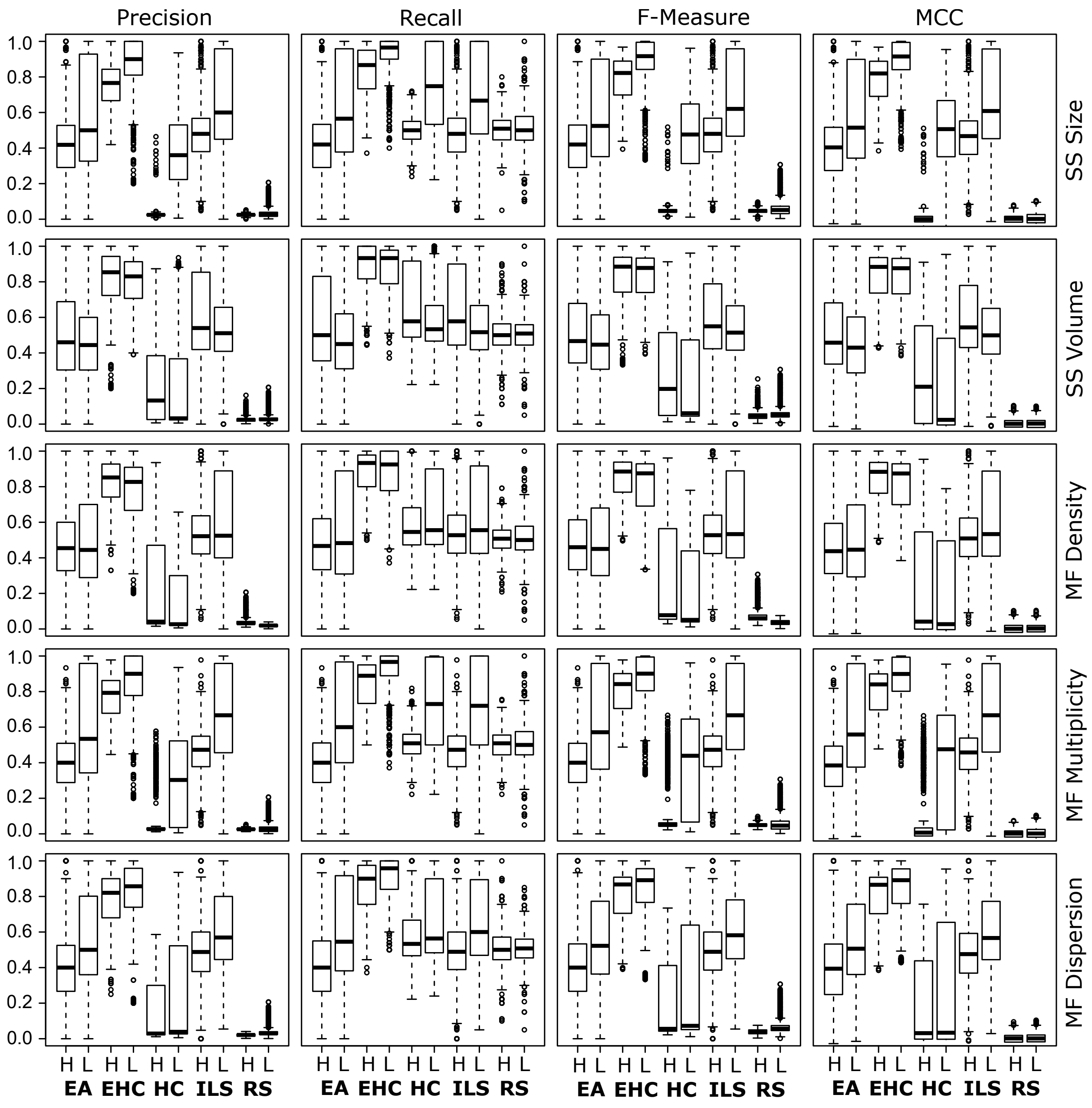


Figure 9: Boxplots of the Precision, Recall, F-Measure, and MCC values obtained by each of the five MR-MFLP and the search strategy

For the MR-MFLP that exhibits the biggest differences between high and low values (SS-Size, MF-Multiplicity, and MF-Dispersion), we can see the influence of the search strategy. For instance, the SS-Size showed a difference of around 0.15 points for the four performance metrics, and the difference is smaller for some search strategies (around 0.10 for EHC) and much bigger for others (around 0.30 for HC).

Similarly, the MF-Dispersion metric showed a mean difference of around 0.7 points; however when taking into account the different search strategies, the differences vary. For instance, EA differences are around 0.15 points, while differences for EHC are below 0.5 points.

It is clear that there are differences in the performance of the different search strategies depending on the factor values of the different MR-MFLP. In the following subsection, we perform an analysis of the variance and assess the significance of the differences.

4.2. Hypothesis Testing

We used the repeated measures General Linear Model (GLM) to analyze the collected data. Our design is a split-plot design where the search strategy is the within subject factor and every MR-MFLP is the between subject factor. There are two requirements for applying a GLM test: homogeneity of the covariance matrices and sphericity. The box's M test is used to check the condition of homogeneity of covariance matrices using as null hypothesis that the observed covariance matrices of the dependent variables are equal across groups [41]. If group sizes are over 30, then the method is robust against violations of homogeneity of variance-covariance matrices assumption. Furthermore, if sample sizes are equal, heterogeneity is not an issue [42]. Our group sizes are equal to and over 30, so it is possible to assume the homogeneity of the covariance matrices. On the other hand, Mauchly's test is used to check the sphericity condition. When our data does not conform to the sphericity assumption to decrease the risk of Type I error, we use the Greenhouse–Geisser adjustment [43]. This adjustment corrects the degrees of freedom for the repeated measures and provides better control for the Type I error rate. The GLM analysis allows us to determine if the differences among search strategies and among scenarios are significant enough. If they are, we will perform the post hoc analysis to determine the significance between pairs using the Bonferroni correction.

In the first analysis, we have used the GLM with the five search strategies (EA, EHC, HC, ILS, and RS) as the within subjects factor (or repeated measures), all of the possible scenarios as the between subjects factor, and the performance metrics (Precision, Recall, F-Measure, and MCC) as the dependent variables. Each of the scenarios belongs to a combination of values for each of the five MR-MFLP (Table 9 shows the complete list of scenarios and values). Figure 10 shows the estimated marginal means for each of the performance metrics and scenarios. Each search strategy is shown with a different color and symbol (green triangles for the HC, purple crosses for the ILS, blue circles for the EA, red squares for the EHC, and cyan asterisks for the RS).

EHC stands out because is the search strategy that provides the best results for the majority of scenarios and performance metrics. However, the differences with the rest of the search strategies is much smaller for some of the scenarios (such as 14, 16, 26, 28, 29, 30, and 32; most of those scenarios have a LOW SS-Size value).

In the case of the ILS and the EA, the differences are much smaller and vary depending on the specific scenario being analyzed for the four performance metrics studied. Similarly, the differences between HC and RS vary depending on each of the scenarios. The case for Recall is worth mentioning, as is the case where the differences between the HC, RS, ILS, and EA are smaller; the search strategy that produces the best results is not the same for all of the scenarios.

The significance test of the results provides values below 0.001 for all of the performance metrics, showing that the scenario, the search strategy, and the interaction between the two is significant for all of the performance metrics (Precision, Recall, F-Measure, and MCC).

The post hoc analysis to determine the significance between pairs shows that the differences among all of the search strategies are significant for each of the four performance metrics with the exception of one case (see Table 10 available on Appendix). The differences are considered significant for values below 0.05. The exceptional case is observed in the comparison between the EA and RS search strategies for the Recall metric, where the value obtained is 0.538 and therefore the difference is not significant.

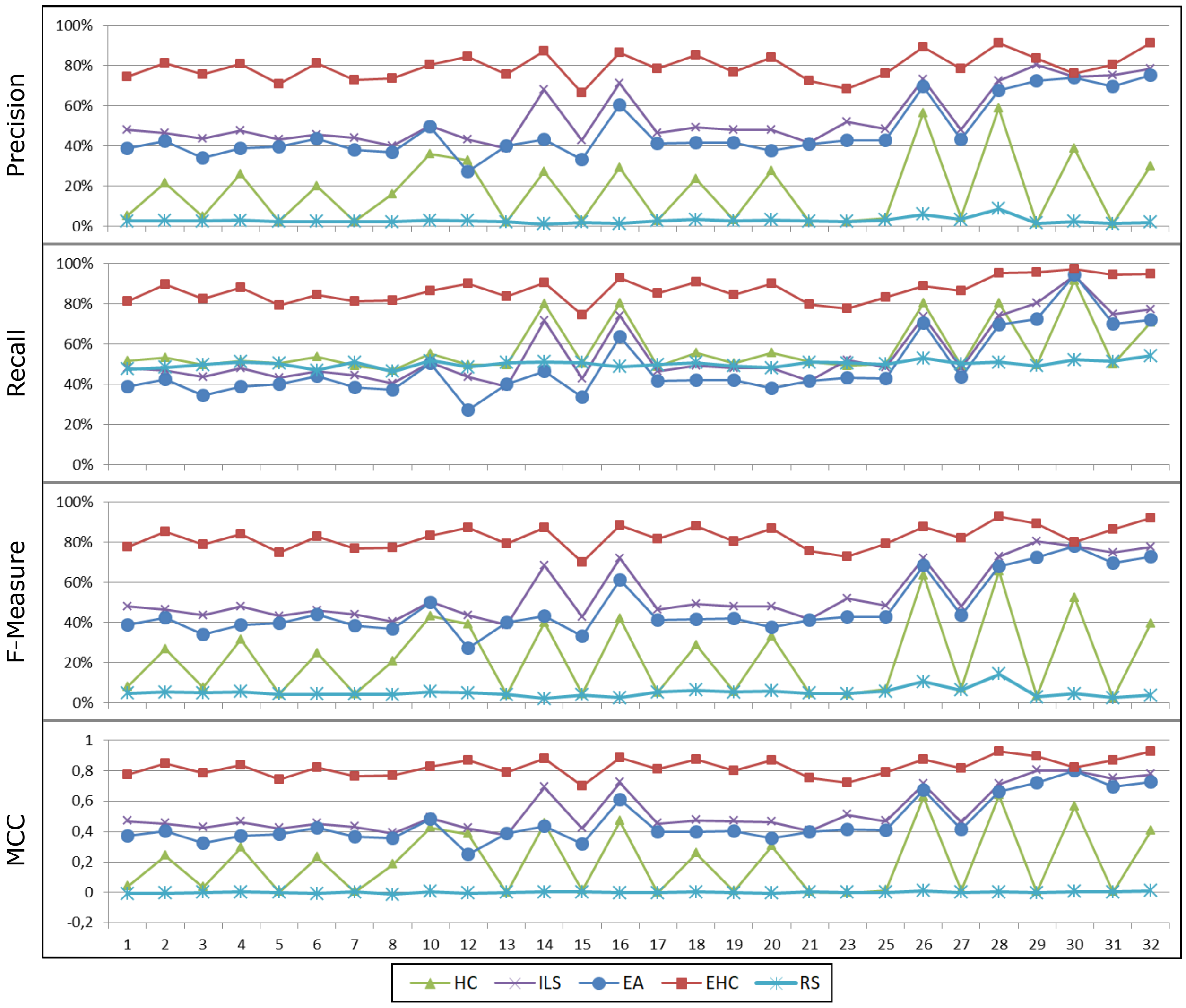


Figure 10: Measure of estimated marginal means for each of the performance metrics (y axis) and scenarios (x axis)

4.2.1. Interaction effects

In this subsection, we apply the GLM using the five search strategies (EA, EHC, HC, ILS, and RS) as the within subjects factor (or repeated measures), the performance metrics (Precision, Recall, F-Measure, and MCC) as the dependent variables and the MR-MFLP (SS-Size, SS-Volume, MF-Density, MF-Multiplicity, MF-Dispersion) as the between subjects factor. We analyze the interaction effects between each search strategy and each MR-MFLP. Figure 11 shows the estimated marginal means of the interaction between each search strategy and MR-MFLP for each of the four performance metrics reported. Each column shows one of the performance metrics, while each row shows one of the MR-MFLP. Each of the cells shows the interaction among each of the five search strategies and the two values (red squares for HIGH values and blue triangles for LOW values) of the corresponding factor (row) for the corresponding performance metric (column).

In addition, a summary table with the differences between the LOW value minus the HIGH value for each search strategy and interaction is shown in Table 5. Dark grey is used to highlight cells with large differences (above 0.250); medium grey is used to highlight cells with medium differences (from 0.150 to 0.250); light grey is used to highlight cells with low differences (from 0.050 to 0.150); finally, white is used b

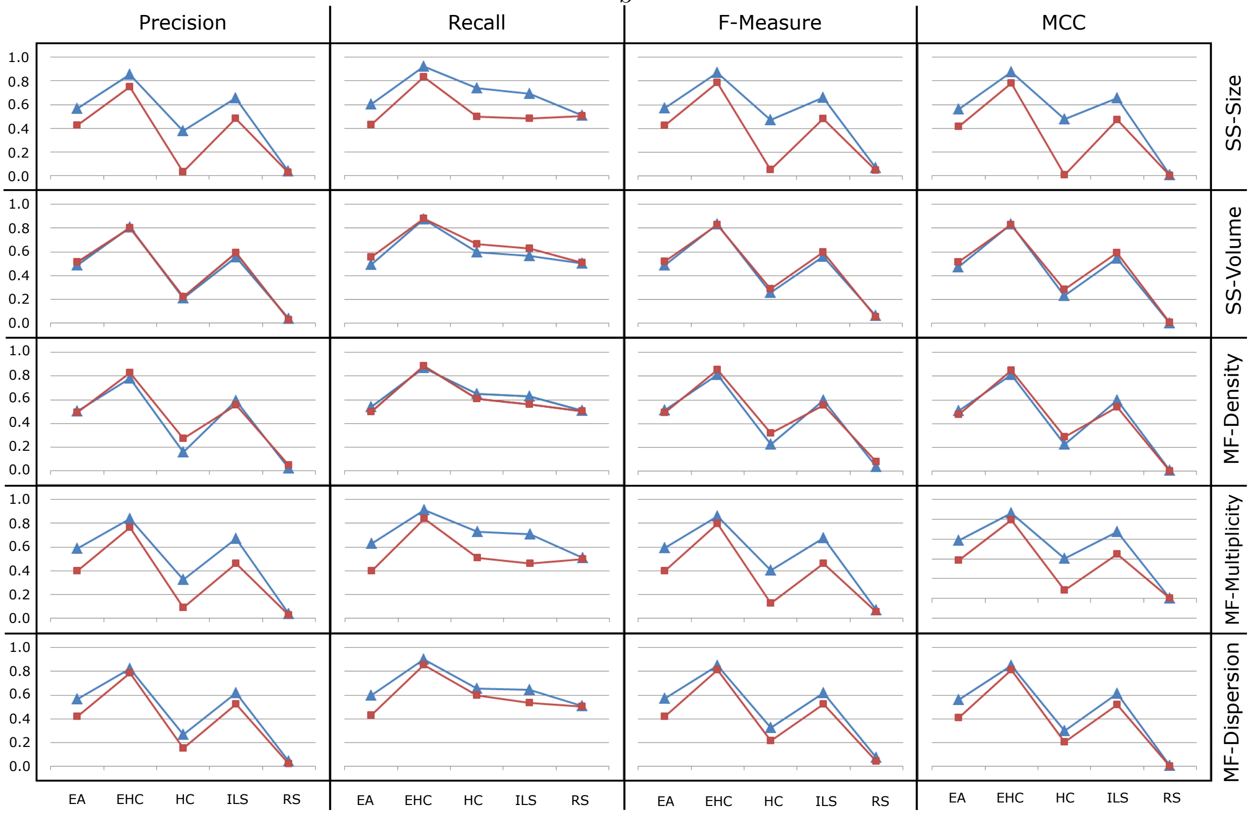


Figure 11: Estimated marginal means of the interaction between each search strategy and MR-MFLP for each of the four performance metrics reported. HIGH values are depicted as red squares and LOW values are depicted as blue triangles.

		Precision	Recall	F-Measure	MCC
SS-Size	EA	0.140	0.177	0.145	0.152
	EHC	0.106	0.089	0.085	0.092
	HC	0.350	0.243	0.418	0.470
	ILS	0.171	0.207	0.176	0.183
	RS	0.014	0.008	0.021	0.003
SS-Volume	EA	-0.028	-0.059	-0.031	-0.040
	EHC	0.001	-0.008	0.008	0.003
	HC	-0.012	-0.066	-0.031	-0.050
	ILS	-0.036	-0.067	-0.039	-0.047
	RS	0.009	-0.005	0.015	-0.001
MF-Density	EA	0.011	0.041	0.017	0.030
	EHC	-0.049	-0.014	-0.043	-0.036
	HC	-0.114	0.042	-0.094	-0.063
	ILS	0.037	0.068	0.043	0.055
	RS	-0.026	0.004	-0.043	0.001
MF-Multiplicity	EA	0.192	0.228	0.196	0.206
	EHC	0.077	0.075	0.062	0.070
	HC	0.232	0.223	0.282	0.325
	ILS	0.211	0.246	0.215	0.225
	RS	0.011	0.010	0.015	0.003
MF-Dispersion	EA	0.147	0.167	0.150	0.150
	EHC	0.037	0.048	0.036	0.038
	HC	0.118	0.056	0.113	0.097
	ILS	0.093	0.110	0.095	0.094
	RS	0.021	0.005	0.035	0.002

Table 5: Differences on the estimated marginal means of the interaction between each search strategy and MFLP for each of the four performance metrics reported. Each cell shows the difference between the LOW value minus the HIGH value of the MFLP. Dark grey means high interaction effect (above 0.250). Middle grey means middle interaction effect (from 0.150 to 0.250). Light grey means low interaction effect (from 0.050 to 0.150). White means almost no interaction effect (from 0 to 0.050)

to highlight cells with smallest differences (below 0.050). Notice that the sign of the value will indicate the direction of the effect; positive values correspond to LOW values obtaining better results than HIGH values.

First of all, it is important to remark that the EHC provides the best results, no matter the value of the MR-MFLP or the performance metric being analyzed for this case study.

The post hoc analysis has been applied to the pairwise comparison of all of the results of every search strategy, performance metric, and MR-MFLP. The results show significant differences for all of the comparisons except for the comparison between EA and RS for the Recall values for the five MR-MFLP.

The interaction between the **SS-Size** of the MFLP (the first row of Figure 11) and the performance metrics is as follows: The results for the four performance metrics are better for smaller SS-Size, no matter the search strategy used. The HC is the search strategy that is most affected by the SS-Size, obtaining high differences in Precision, F-Measure, and MCC and medium differences in Recall. The ILS is also heavily affected by the SS-Size, resulting in medium differences between high and low values for the four performance metrics. The EA is also affected by the SS-size, obtaining medium differences in Recall and MCC and low values in Precision and F-Measure. Similarly, the EHC obtains low difference values for the four performance metrics. The RS is hardly affected by the SS-Size. The HC search strategy obtains values in MCC and F-Measure as low as those from the RS when the SS-Size is HIGH. In addition, the values of Recall for the EA, the HC, and the ILS are worse than those from the RS when the SS-Size is HIGH.

The interaction between the **SS-Volume** of the MFLP (the second row of Figure 11) and the performance metrics is as follows: There are no significant differences for Precision; the effect is low in general for F-Measure and MCC (very small values) and slightly higher for Recall (small values). In general, better performance values are achieved for HIGH SS-Volume values with some exceptions (Precision and F-Measure of EHC and RS, and MCC of EHC), but the improvements are really low. The EA search strategy achieves values that are worse than the RS for LOW values of SS-Volume

The interaction between the **MF-Density** of the MFLP (the third row of Figure 11) and the performance metrics is as follows: The effects of the interaction are low in general (very small values) and slightly higher for the HC and ILS (some small difference values). The EA and the ILS achieve better results in the four performance metrics with LOW MF-Density values, while the EHC achieves better results with HIGH values. The HC achieves better results for HIGH MF-Density values for all of the performance metrics except for Recall (which provides better results with LOW MF-Density values). The RS achieves better results with LOW MF-Density values for Precision and F-Measure. The EA produces results as low as those from RS for HIGH values of MF-Density.

The interaction between the **MF-Multiplicity** of the MFLP (the fourth row of Figure 11) and the performance metrics is as follows: The results for the four performance metrics are better for LOW MF-Multiplicity values, no matter the search strategy used. The HC is the search strategy that is most affected by the MF-Multiplicty, obtaining high differences values (between HIGH and LOW values of MF-Multiplicity) for F-Measure and MCC, and medium for Precision and Recall. The ILS and the EA are also heavily affected, achieving medium difference values for the four performance metrics. The EHC is hardly affected, and the differences measured are small. The RS is the least affected search strategy, with values close to zero. The ILS and the EA search strategies obtain values that are below those obtained by the RS for Recall when the MF-Multiplicity is HIGH. The HC achieves results for F-Measure as low as those from RS when the MF-Multiplicity is HIGH.

The interaction between the **MF-Dispersion** of the MFLP (the fifth row of Figure 11) and the performance metrics is as follows: The results for the four performance metrics are better for LOW MF-Dispersion values, no matter the search strategy used. The influence of the MF-Dispersion is lower than the influence of the SS-Size or the MF-Multiplicity. The EA search strategy is the one that is most affected by the MF-Dispersion obtaining medium difference values (around 0.150) for the four performance metrics. The HC and ILS obtain small difference values for the four performance metrics. The EHC and RS show almost no affect by the MF-Dispersion, obtaining difference values below 0.050. The EA search strategy obtains values that are worse than those from RS for Recall when the MF-Dispersion is HIGH.

5. Findings

The results presented in the section 4 allows us to provide answers to the research questions:

RQ1: Which search strategy (EA, EHC, HC, ILS, RS) provides the best results in terms of performance (Precision, Recall, F-Measure, MCC) for the MFLPs?

The experiment presented in this work provides a detailed analysis of the values for four performance metrics obtained by the results of the application of five different search strategies to 1895 feature location problems extracted from 40 different industrial SPLs.

Measure	Acceptable	Good	Excellent
Precision	20% - 29%	30% - 49%	>50%
Recall	60% - 69%	70% - 79%	>80%

Table 6: Classification of the precision and recall results. Obtained from [44]

Those performance results are classified by Hayes et al. in [44] as acceptable, good, or excellent, as shown in Table 6. The EHC obtains overall excellent results for precision and recall; the EA and the ILS are able to obtain excellent precision results, but the recall obtained by both is not acceptable. In contrast, HC achieves acceptable precision and recall values. Finally, RS is not able to provide acceptable results (as expected). This information can be useful for practitioners when selecting which search strategy to apply based on their needs regard precision and recall.

It is important to note that achieving a good result on either precision or recall while getting a low value on the other is a straightforward task for any search strategy (the empty solution achieves 100% recall while obtaining a 0% precision and the complete solution does the opposite). Therefore, we add the F-Measure and MCC measurements because they are better indicators of the overall quality.

The results show that the best search strategy overall is the EHC. It provides the best results in Precision, Recall, F-Measure, and MCC no matter what the levels for the MR-MFLP are. However, even though it is the best search strategy for this type of problem, it is not the most commonly used strategy among the works in the field [5]. We believe that works like this one will help in raising its popularity among the community.

RQ2: What are the differences in performance (Precision, Recall, F-Measure, MCC) between the HIGH and LOW values of the MR-MFLP (SS-Size, SS-Volume, MF-Density, MF-Multiplicity, MF-Dispersion)?

There are differences in the performance metrics (Precision, Recall, F-Measure, and MCC) between HIGH and LOW values of the MR-MFLP. The results show that the highest differences of performance between HIGH and LOW values are obtained for SS-Size and MF-Multiplicity, the differences in performance for the HIGH and LOW values of MF-Dispersion are medium, and differences in performance for the HIGH and LOW values of SS-Volume and MF-Density are the smallest ones.

The directional difference obtained in performance due to the LOW and HIGH values of each of the MR-MFLPs (SS-Size, SS-Volume, MF-Density, MF-Multiplicity, MF-Dispersion) is homogeneous for all performance metrics. The SS-Size and the MF-Multiplicity achieve the best values in the performance metrics with the LOW values. The difference in the performance metrics generated by the LOW and HIGH values of SS-Volume are small and the best results are achieved with the HIGH value. For MF-Density, the best values of Precision and F-measure are obtained with the HIGH value; on the other hand, the best value of Recall is achieved with the LOW value, and for MCC there is not difference. Finally, the best performance metrics values for the MF-Dispersion are obtained with the LOW value.

The analysis presented in this work provides detailed quality measurements based on the values of the five MR-MFLP analyzed (see Table 3). In other words, depending on the nature of the feature location problems that will be addressed, the results provided by each of the search strategies will vary. This information will be valuable for practitioners so that they can decide the best search strategy based on the information about the nature of the feature location problem they are facing.

RQ3: Are there any interactions between the MR-MFLP values (SS-Size, SS-Volume, MF-Density, MF-Multiplicity, MF-Dispersion) and the search strategies (EA, EHC, HC, ILS, RS) that affect the strategies' performance (Precision, Recall, F-Measure, MCC)?

There are significant interactions between the MR-MFLP and the search strategies affecting the performance metrics in all possible pairwise combinations (see Figure 11). Considering all the MR-MFLPs, the highest interactions with the search strategies can be seen between different values of the SS-Size, followed by different values for the MF-Multiplicty; and to a less extent when the MF-Dispersion, MF-Density, or SS-Volume values vary.

Figure 10 shows that EHC obtains the best values for all performance metrics regardless of the MR-MFLP values. Only in one scenario (scenario 30, see Table 9 in Appendix) the EA and ILS search strategies obtain similar values to the values achieved by EHC for Precision, F-measure and MCC. In the same scenario, EA, ILS, HC ,and EHC achieve similar values of Recall. On the other hand, EHC and RS are the search strategies least affected by the difference of LOW and HIGH values of MR-MFLP.

In relation to the influence of the MR-MFLP on the performance of search strategies, we have found that the search strategies more influenced are HC, ILS and EA. We have shown how the different MR-MFLP can have an impact on the results of the feature location, but there is a need for further research on the impact of these MR-MFLP when using other search strategies and also more works aiming for new MR-MFLP that can be used to better characterize the feature location problems and that could yield to a better selection of the appropriate search strategy.

In addition, a better characterization and description of the case studies (as can be done with the use of the MR-MFLP) would enable fairer comparisons among different studies. Otherwise it is not easy to compare properly the performance results from approaches using different case studies as they can greatly depend on the characteristics of the feature location problem (as shown in this work).

6. Threats to Validity

To describe the threats to validity of our work, we use the classification of [45], which distinguishes four aspects of validity (conclusion validity, internal validity, construct validity, and external validity). This section presents the threats that we could not avoid.

Conclusion validity is concerned with issues that affect the ability to draw the correct conclusion about relations between the treatment and the outcome of an experiment. There are two threats of this type: low statistical power, and violated assumptions of statistical test. With *low statistical power*, there is a high risk that an erroneous conclusion can be drawn. To mitigate this threat, we have used a confidence interval where conclusions are 95% representative. To improve the statistical power, it would be interesting to replicate the experiment with more sets of feature location problems from industrial SPLs in order to increase the sample size. The experiment was affected by *violated assumptions of statistical test* that are related to normally distributed data. This threat was mitigated by the number of samples used in the experiment [46].

Internal validity threats appear when causal relations are examined, since there is a risk that the studied response variables may be affected by other factors that are not considered in the experiment. There are two threats of this kind: instrumentation and ambiguity about direction of casual influence. The *instrumentation* threat is the effect caused by the artifacts used for the execution of the experiment. We mitigated this threat by using the same laptops to perform the experiment ensuring that are strategies have the same computational power to locate the features. Furthermore, the experiment was performed with SPLs extracted from real contexts. The *ambiguity about direction of casual influence* threat occurs when the influence of the treatments is not clear. To mitigate this threat, we studied the influence of the different scenarios, the five dimensions, and the five treatments. We have mitigated this threat but it is possible that other values for the variables or other variables not considered in this work influence the results.

Construct validity concerns generalizing the result of the experiment to the concept or theory behind the experiment. There are two threats of this kind: mono-operation bias, and confounding constructs and levels of constructs. The *mono-operation bias* appears when the experiment includes a single independent variable, case, or treatment; as a result, the experiment may under-represent the construct and thus not give the full picture of the theory. Our experiment was affected by this threat, since we worked with only two domains. For this reason, generalization of results must be approached with caution. The *confounding constructs and levels of constructs* threat appears when in some relations it is not primarily the presence or absence of a construct, but the level of the construct which is important for the results. The experiment was affected by this threat because we considered only two levels for each studied dimension.

External validity threats are concerned with to what extent the findings are of relevance for industrial practice. There are two threats of this kind: interaction of setting and treatment, and the lack of a clear object selection strategy.

The *interaction of setting and treatment* threat appears when the experimental setting or material is not representative of industrial contexts. We tried to mitigate this threat by using MFLPs from a large number of SPLs (40). However, the experiment could be affected by this threat since MFLPs come from only two domains: induction hob firmware and train control.

The *lack of clear object selection strategy* could prevent the application of the findings of this study by two reasons:

- Search strategy and fitness selection strategy: Apart from the five search strategies included in this work, there could be other feature location approaches (that do not rely on SBSE) more similar to those used in code [23] that provide different results than those presented in this work. Similarly, other fitness functions that do not rely on LSA could perform better than those presented here (e.i. VSM-Lucene). To mitigate this threat the search strategies and the fitness function have been selected from the literature [5, 11, 30, 29] for being the ones providing the best results for the FLiM problem. However, more studies would be needed to generalize and determine if the results can be extrapolated to other feature location approaches and/or fitness functions and we plan to conduct this study in the future.

- Incomplete model fragments: When working with software models, there is a need to check the well-formedness of the models (or otherwise could not be automatically manipulated by some tools). Incomplete model fragments could not fit the needs of some practitioners when performing feature location in models. To mitigate this thread we have used an encoding that only allows the existence of model fragments in the form of subsets of the original model fragment. However, more restricting measures could be taken as the use of a strong encoding or repair operators [25].

7. Related Work

This section provides some discussion about works from literature that provide empirical evaluations of feature location approaches.

A case study of automated feature location in industry is presented in [47]. The authors tested two textual/static feature location approaches (Lucene and a combination of Lucene with the PageRank algorithm) against a large software system with a development lifespan of over 20 years. They compared both algorithms by surveying their industrial partner about the accuracy of the list of function points produced by each algorithm. The objective was to determine which feature location technique should be used in their industrial context of cost estimation. Their search consists in the identification of function points in code. The combination of Lucene with PageRank was able to produce results with a precision value of 75.24% and a recall value of 0.43%. When varying the minimum confidence score used by Lucene the recall raised up to more than 43% (but the precision drop to less than 29%). In [48], the authors evaluated three standard search-based techniques for reverse engineering feature models from the feature descriptions of each individual variant. These techniques (evolutionary algorithms, hill climbing, and random search) were evaluated on 74 SPLs. Their evaluation revealed a clear trade-off between recall and precision in two objective functions. Hill climbing obtained the best precision values for both objective functions (23% and 66% respectively) while The evolutionary algorithm obtained the best recall values (96% and 33% respectively). These works did not consider the five MR-MFLP as we do.

Paper [49] presents the results of a systematic study about SBSE techniques to solve SPL problems. The study identified a need to improve the empirical evaluation and some common pitfalls, and it provided a short guideline to address them. The study concludes that SBSE techniques are applied mainly for testing at the Domain Engineering level, and the most common technique used is genetic algorithms with an increasing interest in multi-objective optimization problems. The work in [2] presents a survey of work on SBSE for SPLs. The authors studied 58 papers on SBSE for SPL, where the authors highlighted the use of genetic programming in SBSE research. These might address searching based branch merge and Grow and Graft Genetic Improvement to automatically augment a SPL with new branches. In [8] , the authors present a systematic review of 170 articles about empirical evaluation in feature location. They concluded that it is difficult to answer questions like "What are the best Feature Location Techniques?". On the other hand, they showed a set of best practices for empirical work in the feature location field. These works do not perform an empirical evaluation. Paper [23] focuses on providing means for comparisons among different feature location in source code techniques. The authors assessed eight baseline techniques in twelve case studies to rank their performance. Results of the case studies suggest that different baseline techniques perform differently and two techniques (VSM-Lucene and LSI-Matlab) performed better than other implementations. Similar to this work, the results suggest that the performance of feature location techniques depends on system/benchmark characteristics, in addition to the techniques themselves. This idea is widely confirmed in a recently published paper [50] that proposes a suite of feature-characteristic metrics that are postulated to control Feature Location Techniques (FLT) performance. They perform a controlled study including four FLTs (VSM-Lucene, LSI-Matlab, LDA-R and a combination of PageRank and LSI), using four commonly used evaluation measures (precision, recall, mean average precision and mean reciprocal rank) and a set of 41 source-code metrics. Their evaluation explains the variance in performance of the different FLTs and illustrates the effect of feature characteristics on those FLT's performance. They also provide an online tool that allows researchers to obtain a feature set profile, helping them in choosing the best FLT for the characteristics of their data set. We believe that our work is similar in the sense that we try to guide practitioners when selecting a search strategy based on the characteristics of their data set (in our case software models instead of source code).

The work in [51] introduced a solution called LDA-GA, which uses Genetic Algorithms to determine a near-optimal configuration for a modelling technique in the context of three different software engineering tasks: traceability link recovery, feature location, and software artifact labeling. The authors performed an empirical study that includes feature location on two software systems (jEdit and ArgoUML). The paper applies the idea of data fusion to feature location, which is the process of identifying the source code that implements specific functionality in software. A data fusion model for feature location is presented which defines new feature location techniques based on combining information from textual, dynamic, and web-mining analyses applied to software. A novel contribution of the proposed model is the use of advanced web mining algorithms to analyze execution information during feature location. The results of an extensive evaluation indicate that the new feature location techniques based on web mining improve the effectiveness of existing approaches by as much as 62%.

Following, we discuss the overlap between this work and our previous works: In one of our previous works [11], we show the feasibility of applying SBSE techniques to address feature location problems in software models. The work included the execution of 217 MFLPs with five different search strategies and the statistical analysis of the results in terms of performance metrics. The work concluded that the search strategy that provides the best results was the hybrid between evolutionary algorithm and hill climbing. In another previous work [19], we proposed the five MR-MFLP, describing them and showing their applicability to report feature location problems. The work included a statistical analysis of the results applied to a case study, highlighting the most common cases found in that industrial study. However, there were no analyses of the interaction among the different measures and their impact on the different search techniques applied. In the present work, we confirm the results from the previous work, as the EHC is again the search strategy that provides the best performance results overall. However, in this work, we extend the set of MFLPs, obtaining them from 40 different industrial SPLs and reaching a total figure of 1895 MFLPs. In addition, we study the impact of the MR-MFLP on the performance results provided by the search strategies, through a repeated measures General Linear Model analysis. This leads to interesting findings; for instance, the differences on performance metrics are different for LOW or HIGH values of the MR-MFLP, and are not the same for every MR-MFLP or search strategy. We also conclude that the SS-Size is the measure that has the greatest impact on the results, followed by the MF-Multiplicity. All in all, this is the first work that provides recommendations when selecting which search strategy to apply based on 1) the needs regarding Precision and Recall; and 2) the nature of the feature location problem at hand (SS-Size, SS-Volume, MF-Density, MF-Multiplicity, MF-Dispersion).

8. Conclusion

This work presents the application of five different search strategies (EA, EHC, HC, ILS, RS) to 1895 feature location problems in models obtained from 40 industrial SPLs. It analyzes the impact of five different measures used to report model fragment location problems (SS-Size, SS-Volume, MF-Density, MF-Multiplicity, MF-Dispersion) that can be used to characterize the problems based on the results provided by the search strategies in the form of four measurements of performance (Precision, Recall, F-Measure, and MCC).

In our study EHC was provided the best results overall, being resilient to differences in the values of the five MR-MFLP and providing better results for the four performance metrics studied. Given these findings and the fact that is not the most used search strategy nowaday, we suggest that EHC should be further explored by the search-based model driven community .

The use of MR-MFLP can yield better results and ease the task of comparing the results of different approaches and studies. We believe that its use by the community should increase and new MR-MFLP should be created in order to allow for a better characterization for the problem. However, further experimentation is needed to confirm/generalize the results and perform experiments with other search strategies or fitness functions. In particular, as our future work, we want to extend this study with the state-of-the-art techniques used to locate features in source code [23, 7] to determine if the results are similar.

This study provides practical information for practitioners when deciding which search strategy should be applied to a specific problem of feature location in models. Depending on the nature of the problem, the search strategies will yield results with different levels of performance. Specifically, the SS-Size, MF-Multiplicity and MF-Dispersion have shown the greatest impact on the performance results and could provide relevant information for other practitioners when performing FLiM or selecting search strategies.

Even if the EHC is the approach that provides the best results in general, another approach may be better suited when different constraints apply (as could be the case for the budget time for the search when the system must run in real time). Therefore, we hope that this work encourages further research on the impact of the measures used to report feature location problems in different scenarios.

9. Appendix

Algorithm 1 Shared functions and parameters

```
1:  function initPopulation($C_{size}, Pop_{size}$)        ▷ Generates a random initial population
2:      $Population \leftarrow []$
3:      for $i = 1$ to $Pop_{size}$ do
4:          $Fragment \leftarrow$ randomFragment($C_{size}$)        ▷ Generate random fragment
5:          $Population \leftarrow Population + Fragment$        ▷ Add new random fragment
6:      end for
7:      return $Population$
8:  end function

9:  function breedPopulation($Population, Rep_{size}$)        ▷ Generates a $Rep_{size}$ offspring
10:     $Offspring \leftarrow []$
11:     for $i = 1$ $to$ $Rep_{size}$ do
12:         $Parents \leftarrow$ selectParents($Population$)
13:         $Crossed \leftarrow$ crossover($parents, p_c$)
14:         $Mutated \leftarrow$ mutation($crossed, p_m$)
15:         $Offspring \leftarrow Offspring + Mutated$        ▷ Add the new offspring
16:     end for
17:     return $Offspring$
18: end function

19: function doHillClimb($N_{size}, Original$)        ▷ Finds the best neighbour of Original
20:     $BestNeighbour \leftarrow Original$
21:     for $i = 1$ $to$ $N_{size}$ do
22:         $Candidate \leftarrow$ tweak($Original$)
23:         if (fitness($Candidate$) > fitness($BestNeighbour$)) then
24:             $BestNeighbour \leftarrow Candidate$
25:         end if
26:     end for
27:     return $BestNeigbour$
28: end function
```

Algorithm 2 Evolutionary Algorithm

```
1: $Population \leftarrow$ initPopulation($C_{size}$)
2: while (!StopCondition) do
3:     $Population \leftarrow$ fitness($Population$)
4:     $Offspring \leftarrow$ breedPopulation($Population$)
5:     $Population \leftarrow$ replace($Popuplation, Offspring$)
6: end while
7: return $Population[0]$
```

Algorithm 3 Random Search

```
1: $Best \leftarrow$ randomFragment($C_{size}$)
2: while (!$StopCondition$)) do
3:     $Candidate \leftarrow$ randomFragment($C_{size}$)
4:     if (fitness($Candidate$) > fitness($Best$)) then
5:         $Best \leftarrow Candidate$
6:     end if
7: end while
8: return $Best$
```

Algorithm 4 Steepest Ascent Hill Climbing with Replacement

```
1: $Candidate \leftarrow$ randomFragment($C_{size}$)
2: $N_{Size} \leftarrow 100$                          ▷ number of neighbors desired
3: $Best \leftarrow Candidate$
4: while (!$StopCondition$) do
5:     $Candidate \leftarrow$ doHillClimb($N_{size}, Candidate$)
6:     if (fitness($Candidate$) > fitness($Best$)) then
7:         $Best \leftarrow Candidate$
8:     end if
9: end while
10: return $Best$
```

Algorithm 5 Iterated Local Search (ILS) with Random Restarts

```
 1: $Current \leftarrow$ RANDOMFRAGMENT($C_{size}$)
 2: $Times \leftarrow$ distribution of time intervals
 3: $Home \leftarrow Current$
 4: $Best \leftarrow Current$
 5: while (!$StopCondition$) do
 6:     $time \leftarrow$ random time chosen from $Times$
 7:     while (!$StopCondition$ && $time \neq 0$) do
 8:         $Aux \leftarrow$ TWEAK($Current$)
 9:         if  (FITNESS($Aux$) > FITNESS($Current$) then
10:             $Current \leftarrow Aux$
11:         end if
12:     end while
13:     if  (FITNESS($Current$) > FITNESS($Best$) then
14:         $Best \leftarrow Current$
15:     end if
16:     $Home \leftarrow$ NEWHOMEBASE($Home, Current$)
17:     $Current \leftarrow$ PERTURB($Home$)
18: end while
19: return $Best$
```

Algorithm 6 Hybrid between Evolutionary and Hill-Climbing

```
 1: $N_{size} \leftarrow 100$                                    ▷ number of iterations to Hill-Climb
 2: $Population \leftarrow$ INITPOPULATION($C_{size}, Pop_{size}$)
 3: $Best \leftarrow$ RANDOMFRAGMENT($C_{size}$)
 4: while (!$StopCondition$) do
 5:     FITNESS($Population$)
 6:     for $P_i$ in $P$ do
 7:         $P_i \leftarrow$ DOHILLCLIMB($N_{size}, P_i$)
 8:         if (FITNESS($P_i$) > FITNESS($Best$)) then
 9:             $Best \leftarrow P_i$
10:         end if
11:     end for
12:     $P \leftarrow$ BREEDPOPULATION($Population$)
13: end while
14: return $Best$
```

SPL	number of MFLP
CAF 1	41
CAF 2	43
CAF 3	39
CAF 4	32
CAF 5	33
CAF 6	44
CAF 7	26
CAF 8	34
CAF 9	37
CAF 10	37
CAF 11	38
CAF 12	33
CAF 13	38
CAF 14	37
CAF 15	31
CAF 16	39
CAF 17	32
CAF 18	46
CAF 19	38
CAF 20	37
CAF 21	41
CAF 22	42
CAF 23	37
CAF 24	35
CAF 25	35
CAF 26	48
CAF 27	38
CAF 28	41
CAF 29	41
CAF 30	37
CAF 31	45
CAF 32	38
CAF 33	43
CAF 34	31
BSH 1	101
BSH 2	75
BSH 3	73
BSH 4	113
BSH 5	89
BSH 6	157

Table 7: Number of MFLP for each of the SPLs

MR-MFL	Search Strategy	Factor Level	Precision ± σ	Recall ± σ	F-Measure ± σ	MCC ± σ
SS Size	EA	HIGH	0.42 ± 0.19	0.43 ± 0.19	0.43 ± 0.19	0.41 ± 0.20
		LOW	0.56 ± 0.31	0.60 ± 0.31	0.57 ± 0.30	0.56 ± 0.31
	EHC	HIGH	0.75 ± 0.12	0.83 ± 0.13	0.79 ± 0.12	0.78 ± 0.13
		LOW	0.86 ± 0.18	0.92 ± 0.12	0.87 ± 0.15	0.88 ± 0.14
	HC	HIGH	0.03 ± 0.04	0.50 ± 0.07	0.05 ± 0.04	0.01 ± 0.05
		LOW	0.38 ± 0.23	0.74 ± 0.23	0.47 ± 0.25	0.48 ± 0.26
	ILS	HIGH	0.48 ± 0.16	0.48 ± 0.16	0.48 ± 0.16	0.47 ± 0.17
		LOW	0.65 ± 0.27	0.69 ± 0.26	0.66 ± 0.26	0.65 ± 0.26
	RS	HIGH	0.02 ± 0.01	0.50 ± 0.08	0.05 ± 0.01	0.00 ± 0.02
		LOW	0.04 ± 0.04	0.51 ± 0.12	0.07 ± 0.06	0.00 ± 0.03
SS Volume	EA	HIGH	0.51 ± 0.28	0.55 ± 0.29	0.52 ± 0.27	0.51 ± 0.28
		LOW	0.49 ± 0.26	0.49 ± 0.26	0.49 ± 0.26	0.47 ± 0.26
	EHC	HIGH	0.80 ± 0.19	0.88 ± 0.13	0.83 ± 0.16	0.83 ± 0.15
		LOW	0.81 ± 0.14	0.88 ± 0.13	0.84 ± 0.13	0.83 ± 0.13
	HC	HIGH	0.22 ± 0.23	0.66 ± 0.23	0.29 ± 0.27	0.28 ± 0.30
		LOW	0.21 ± 0.26	0.60 ± 0.19	0.26 ± 0.28	0.23 ± 0.31
	ILS	HIGH	0.59 ± 0.26	0.63 ± 0.26	0.60 ± 0.25	0.59 ± 0.25
		LOW	0.56 ± 0.23	0.56 ± 0.23	0.56 ± 0.23	0.55 ± 0.23
	RS	HIGH	0.03 ± 0.02	0.51 ± 0.11	0.05 ± 0.03	0.00 ± 0.03
		LOW	0.04 ± 0.03	0.50 ± 0.10	0.06 ± 0.05	0.00 ± 0.03
MF Density	EA	HIGH	0.49 ± 0.24	0.50 ± 0.24	0.49 ± 0.24	0.48 ± 0.24
		LOW	0.50 ± 0.30	0.54 ± 0.31	0.51 ± 0.29	0.51 ± 0.30
	EHC	HIGH	0.83 ± 0.13	0.89 ± 0.12	0.85 ± 0.12	0.85 ± 0.12
		LOW	0.78 ± 0.19	0.87 ± 0.15	0.81 ± 0.17	0.81 ± 0.16
	HC	HIGH	0.27 ± 0.29	0.61 ± 0.19	0.32 ± 0.31	0.29 ± 0.33
		LOW	0.16 ± 0.18	0.65 ± 0.23	0.23 ± 0.23	0.22 ± 0.27
	ILS	HIGH	0.56 ± 0.21	0.56 ± 0.20	0.56 ± 0.20	0.54 ± 0.21
		LOW	0.59 ± 0.27	0.63 ± 0.27	0.60 ± 0.26	0.60 ± 0.27
	RS	HIGH	0.04 ± 0.03	0.50 ± 0.08	0.08 ± 0.05	0.00 ± 0.03
		LOW	0.02 ± 0.01	0.51 ± 0.13	0.04 ± 0.01	0.00 ± 0.03
MF Multiplicity	EA	HIGH	0.40 ± 0.16	0.40 ± 0.16	0.40 ± 0.16	0.38 ± 0.16
		LOW	0.59 ± 0.31	0.63 ± 0.31	0.59 ± 0.30	0.59 ± 0.31
	EHC	HIGH	0.76 ± 0.12	0.84 ± 0.13	0.80 ± 0.12	0.80 ± 0.13
		LOW	0.84 ± 0.19	0.91 ± 0.13	0.86 ± 0.16	0.86 ± 0.15
	HC	HIGH	0.09 ± 0.14	0.51 ± 0.09	0.12 ± 0.16	0.08 ± 0.17
		LOW	0.32 ± 0.27	0.73 ± 0.23	0.40 ± 0.29	0.41 ± 0.31
	ILS	HIGH	0.46 ± 0.13	0.46 ± 0.13	0.46 ± 0.13	0.45 ± 0.14
		LOW	0.67 ± 0.27	0.71 ± 0.26	0.68 ± 0.26	0.67 ± 0.26
	RS	HIGH	0.03 ± 0.01	0.50 ± 0.08	0.05 ± 0.01	-0.00 ± 0.03
		LOW	0.04 ± 0.04	0.51 ± 0.12	0.06 ± 0.06	0.00 ± 0.03
MF Dispersion	EA	HIGH	0.42 ± 0.24	0.43 ± 0.25	0.42 ± 0.24	0.41 ± 0.24
		LOW	0.57 ± 0.28	0.60 ± 0.28	0.57 ± 0.27	0.56 ± 0.27
	EHC	HIGH	0.79 ± 0.16	0.85 ± 0.14	0.81 ± 0.15	0.81 ± 0.15
		LOW	0.82 ± 0.17	0.90 ± 0.12	0.85 ± 0.14	0.85 ± 0.14
	HC	HIGH	0.15 ± 0.15	0.60 ± 0.20	0.21 ± 0.20	0.20 ± 0.24
		LOW	0.27 ± 0.29	0.65 ± 0.22	0.32 ± 0.32	0.30 ± 0.34
	ILS	HIGH	0.52 ± 0.23	0.53 ± 0.24	0.53 ± 0.23	0.52 ± 0.23
		LOW	0.62 ± 0.24	0.64 ± 0.24	0.62 ± 0.23	0.61 ± 0.24
	RS	HIGH	0.02 ± 0.01	0.50 ± 0.12	0.04 ± 0.01	0.00 ± 0.03
		LOW	0.04 ± 0.03	0.51 ± 0.09	0.07 ± 0.05	0.00 ± 0.03

Table 8: Mean values and standard deviation for the Precision, Recall F-Measure and MCC values by each of the Search Strategies and Metrics to report FL problems

	SS Size	SS Volume	MF Density	MF Multiplicity	MF Dispersion
1	HIGH	HIGH	HIGH	HIGH	HIGH
2	LOW	HIGH	HIGH	HIGH	HIGH
3	HIGH	LOW	HIGH	HIGH	HIGH
4	LOW	LOW	HIGH	HIGH	HIGH
5	HIGH	HIGH	LOW	HIGH	HIGH
6	LOW	HIGH	LOW	HIGH	HIGH
7	HIGH	LOW	LOW	HIGH	HIGH
8	LOW	LOW	LOW	HIGH	HIGH
10	LOW	HIGH	HIGH	LOW	HIGH
12	LOW	LOW	HIGH	LOW	HIGH
13	HIGH	HIGH	LOW	LOW	HIGH
14	LOW	HIGH	LOW	LOW	HIGH
15	HIGH	LOW	LOW	LOW	HIGH
16	LOW	LOW	LOW	LOW	HIGH
17	HIGH	HIGH	HIGH	HIGH	LOW
18	LOW	HIGH	HIGH	HIGH	LOW
19	HIGH	LOW	HIGH	HIGH	LOW
20	LOW	LOW	HIGH	HIGH	LOW
21	HIGH	HIGH	LOW	HIGH	LOW
23	HIGH	LOW	LOW	HIGH	LOW
25	HIGH	HIGH	HIGH	LOW	LOW
26	LOW	HIGH	HIGH	LOW	LOW
27	HIGH	LOW	HIGH	LOW	LOW
28	LOW	LOW	HIGH	LOW	LOW
29	HIGH	HIGH	LOW	LOW	LOW
30	LOW	HIGH	LOW	LOW	LOW
31	HIGH	LOW	LOW	LOW	LOW
32	LOW	LOW	LOW	LOW	LOW

Table 9: Composition of the scenarios based on the metrics values. Scenarios 9, 11, 22 and 24 are missing due to the lack of test cases with those combinations of Metrics to report Feature Location problems.

		Precision	Recall	F-Measure	MCC
EA	EHC	0.000	0.000	0.000	0.000
	HC	0.000	0.000	0.000	0.000
	ILS	0.000	0.000	0.000	0.000
	RS	0.000	0.538	0.000	0.000
EHC	EA	0.000	0.000	0.000	0.000
	HC	0.000	0.000	0.000	0.000
	ILS	0.000	0.000	0.000	0.000
	RS	0.000	0.000	0.000	0.000
HC	EA	0.000	0.000	0.000	0.000
	EHC	0.000	0.000	0.000	0.000
	ILS	0.000	0.001	0.000	0.000
	RS	0.000	0.000	0.000	0.000
ILS	EA	0.000	0.000	0.000	0.000
	EHC	0.000	0.000	0.000	0.000
	HC	0.000	0.001	0.000	0.000
	RS	0.000	0.000	0.000	0.000
RS	EA	0.000	0.538	0.000	0.000
	EHC	0.000	0.000	0.000	0.000
	HC	0.000	0.000	0.000	0.000
	ILS	0.000	0.000	0.000	0.000

Table 10: Significance between pairs of search strategies (confidence interval 95%).

References

[1] M. Harman, B. F. Jones, Search-based software engineering, Information and software Technology 43 (14) (2001) 833–839.

[2] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, Y. Zhang, Search based software engineering for software product line engineering: A survey and directions for future work, in: Proceedings of the 18th International Software Product Line Conference - Volume 1, SPLC '14, ACM, New York, NY, USA, 2014, pp. 5–18. doi:10.1145/2648511.2648513.
URL `http://doi.acm.org/10.1145/2648511.2648513`

[3] S. Kent, Model driven engineering, in: M. Butler, L. Petre, K. Sere (Eds.), Integrated Formal Methods, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 286–298.

[4] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Leveraging variability modeling to address metamodel revisions in model-based software product lines, Computer Languages, Systems Structures 48 (2017) 20–38, special Issue on the 14th International Conference on Generative Programming: Concepts Experiences (GPCE'15). doi:http://dx.doi.org/10.1016/j.cl.2016.08.003.
URL `http://www.sciencedirect.com/science/article/pii/S147784241630001X`

[5] I. Boussaïd, P. Siarry, M. Ahmed-Nacer, A survey on search-based model-driven engineering, Automated Software Engineering 24 (2) (2017) 233–294. doi:10.1007/s10515-017-0215-4.
URL `https://doi.org/10.1007/s10515-017-0215-4`

[6] T. Eisenbarth, R. Koschke, D. Simon, Locating features in source code, IEEE Trans. Softw. Eng. 29 (3) (2003) 210–224. doi:10.1109/TSE.2003.1183929.
URL `https://doi.org/10.1109/TSE.2003.1183929`

[7] B. Dit, M. Revelle, M. Gethers, D. Poshyvanyk, Feature location in source code: a taxonomy and survey, Journal of Software: Evolution and Process 25 (1) (2013) 53–95. doi:10.1002/smr.567.

[8] A. Razzaq, A. Wasala, C. Exton, J. Buckley, The state of empirical evaluation in static feature location, ACM Trans. Softw. Eng. Methodol. 28 (1) (2018) 2:1–2:58. doi:10.1145/3280988.
URL `http://doi.acm.org/10.1145/3280988`

[9] J. Rubin, M. Chechik, A survey of feature location techniques, in: I. Reinhartz-Berger, A. Sturm, T. Clark, S. Cohen, J. Bettin (Eds.), Domain Engineering, Springer Berlin Heidelberg, 2013, pp. 29–58. doi:10.1007/978-3-642-36654-3_2.

[10] J. Font, L. Arcega, Ø. Haugen, 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, MODELS '16, ACM, New York, NY, USA, 2016, pp. 272–282. doi:10.1145/2976767.2976789.
URL `http://doi.acm.org/10.1145/2976767.2976789`

[11] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Achieving feature location in families of models through the use of search-based software engineering, IEEE Transactions on Evolutionary Computation PP (99) (2017) 1–1. doi:10.1109/TEVC.2017.2751100.

[12] J. Martinez, T. Ziadi, T. F. Bissyandé, J. Klein, Y. L. Traon, Bottom-up adoption of software product lines: a generic and extensible approach, in: 19th Int. Conf. on Software Product Line (SPLC), 2015, pp. 101–110.

[13] J. Martinez, W. K. G. Assunção, T. Ziadi, ESPLA: A catalog of extractive SPL adoption case studies, in: Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017, Volume B, Sevilla, Spain, September 25-29, 2017, 2017, pp. 38–41. doi:10.1145/3109729.3109748.
URL `https://doi.org/10.1145/3109729.3109748`

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

[15] M. Fleck, J. Troya, M. Kessentini, M. Wimmer, B. Alkhazi, Model transformation modularization as a many-objective optimization problem, IEEE Trans. Software Eng. 43 (11) (2017) 1009–1032.

[16] R. Bill, M. Fleck, J. Troya, T. Mayerhofer, M. Wimmer, A local and global tour on momot, Softw. Syst. Model. 18 (2) (2019) 1017–1046.

[17] D. Blasco, J. Font, M. Zamorano, C. Cetina, An evolutionary approach for generating software models: The case of kromaia in game software engineering, Journal of Systems and Software 171 (2021) 110804. doi:https://doi.org/10.1016/j.jss.2020.110804.
URL `http://www.sciencedirect.com/science/article/pii/S0164121220302089`

[18] Á. Domingo, J. Echeverria, O. Pastor, C. Cetina, Evaluating the benefits of model-driven development - empirical evaluation paper, in: S. Dustdar, E. Yu, C. Salinesi, D. Rieu, V. Pant (Eds.), Advanced Information Systems Engineering - 32nd International Conference, CAiSE 2020, Grenoble, France, June 8-12, 2020, Proceedings, Vol. 12127 of Lecture Notes in Computer Science, Springer, 2020, pp. 353–367. doi:10.1007/978-3-030-49435-3_22.
URL `https://doi.org/10.1007/978-3-030-49435-3_22`

[19] M. Ballarín, A. C. Marcén, V. Pelechano, 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 '18, ACM, New York, NY, USA, 2018, pp. 189–199. doi:10.1145/3239372.3239397.
URL `http://doi.acm.org/10.1145/3239372.3239397`

[20] J. Bosch, Design and use of software architectures: adopting and evolving a product-line approach, Pearson Education, 2000.

[21] T. Berger, D. Lettner, J. Rubin, P. Grünbacher, A. Silva, M. Becker, M. Chechik, K. Czarnecki, What is a feature? a qualitative study of features in industrial software product lines, in: Proceedings of the 19th International Conference on Software Product Line, SPLC '15, Association for Computing Machinery, New York, NY, USA, 2015, p. 16–25.

doi:10.1145/2791060.2791108.
URL `https://doi.org/10.1145/2791060.2791108`

[22] Meta object facility (mof) version 2.4.1, object Management Group (OMG) Specification (2013).

[23] A. Razzaq, A. L. Gear, C. Exton, J. Buckley, An empirical assessment of baseline feature location techniques, Empir. Softw. Eng. 25 (1) (2020) 266–321. doi:10.1007/s10664-019-09734-5.
URL `https://doi.org/10.1007/s10664-019-09734-5`

[24] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Building software product lines from conceptualized model patterns, in: Proceedings of the 19th International Conference on Software Product Line, SPLC '15, ACM, New York, NY, USA, 2015, pp. 46–55. doi:10.1145/2791060.2791085.
URL `http://doi.acm.org/10.1145/2791060.2791085`

[25] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Handling nonconforming individuals in search-based model-driven engineering: nine generic strategies for feature location in the modeling space of the meta-object facility, Software and Systems Modeling (Mar 2021). doi:10.1007/s10270-021-00870-5.
URL `https://doi.org/10.1007/s10270-021-00870-5`

[26] T. K. Landauer, P. W. Foltz, D. Laham, An introduction to latent semantic analysis, Discourse processes 25 (2-3) (1998) 259–284.

[27] C. D. Manning, H. Schütze, et al., Foundations of statistical natural language processing, Vol. 999, MIT Press, 1999.

[28] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Feature location in model-based software product lines through a genetic algorithm, in: Proceedings of the 15th International Conference on Software Reuse: Bridging with Social-Awareness - Volume 9679, ICSR 2016, Springer-Verlag New York, Inc., New York, NY, USA, 2016, pp. 39–54.

[29] R. E. Lopez-Herrejon, L. Linsbauer, J. A. Galindo, J. A. Parejo, D. Benavides, S. Segura, A. Egyed, An assessment of search-based techniques for reverse engineering feature models, Journal of Systems and Software 103 (2015) 353 – 369. doi:http://dx.doi.org/10.1016/j.jss.2014.10.037.

[30] S. Luke, Essentials of Metaheuristics, 2nd Edition, Lulu, 2013, available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[31] J. Baxter, Local optima avoidance in depot location, The Journal of the Operational Research Society 32 (9) (1981) 815–819.

[32] H. R. Lourenço, O. C. Martin, T. Stützle, Iterated Local Search, Springer US, Boston, MA, 2003, pp. 320–353.

[33] J. Nam, S. Kim, Clami: Defect prediction on unlabeled datasets (t), in: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015, pp. 452–463. doi:10.1109/ASE.2015.56.

[34] R. C. MacCallum, S. Zhang, K. J. Preacher, D. D. Rucker, On the practice of dichotomization of quantitative variables., Psychological methods 7 (1) (2002) 19.

[35] A. Arcuri, G. Fraser, Parameter tuning or default values? an empirical investigation in search-based software engineering, Empirical Software Engineering 18 (3) (2013) 594–623. doi:10.1007/s10664-013-9249-9.

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

[37] D. Dyer, The watchmaker framework for evolutionary computation, `http://watchmaker.uncommons.org/`, [Online; accessed 7-April-2016] (2016).

[38] Apache opennlp: Toolkit for the processing of natural language text, `https://opennlp.apache.org/`, [Online; accessed 7-April-2016] (2016).

[39] The english (porter2) stemming algorithm, `http://snowball.tartarus.org/algorithms/english/stemmer.html`, [Online; accessed 7-April-2016] (2016).

[40] Efficient java matrix library, `http://ejml.org/`, [Online; accessed 7-April-2016].

[41] L. Meyers, G. Gamst, A. Guarino, Applied Multivariate Research: Design and Interpretation, SAGE Publications, 2016.
URL `https://books.google.lu/books?id=LVPdjwEACAAJ`

[42] B. G. Tabachnick, L. S. Fidell, J. B. Ullman, Using multivariate statistics, Vol. 5, Pearson Boston, MA, 2007.

[43] K. F. Nimon, Statistical assumptions of substantive analyses across the general linear model: A mini-review, Frontiers in Psychology 3 (2012).

[44] J. H. Hayes, A. Dekhtyar, S. K. Sundaram, Advancing candidate link generation for requirements tracing: The study of methods, IEEE Transactions on Software Engineering 32 (1) (2006) 4–19.

[45] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, A. Wessln, Experimentation in Software Engineering, Springer Publishing Company, Incorporated, 2012.

[46] J. A. Rice, Mathematical statistics and data analysis, Cengage Learning, 2006.

[47] A. Armaly, J. Klaczynski, C. McMillan, A case study of automated feature location techniques for industrial cost estimation, in: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2016, pp. 553–562.

[48] R. E. Lopez-Herrejon, L. Linsbauer, J. A. Galindo, J. A. Parejo, D. Benavides, S. Segura, A. Egyed, An assessment of search-based techniques for reverse engineering feature models, Journal of Systems and Software 103 (2015) 353 – 369. doi:https://doi.org/10.1016/j.jss.2014.10.037.
URL `http://www.sciencedirect.com/science/article/pii/S0164121214002349`

[49] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, L. Linsbauer, A. Egyed, E. Alba, A hitchhiker's guide to search-based software engineering for software product lines, CoRR abs/1406.2823 (2014). arXiv:1406.2823.
URL `http://arxiv.org/abs/1406.2823`

[50] A. Razzaq, A. Ventresque, R. Koschke, A. D. Lucia, J. Buckley, The effect of feature characteristics on the performance of feature location techniques, IEEE Transactions on Software Engineering (01) (5555) 1–1. doi:10.1109/TSE.2021.3049735.

[51] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshynanyk, A. De Lucia, How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms, in: 2013 35th International Conference on Software Engineering (ICSE), 2013, pp. 522–531. doi:10.1109/ICSE.2013.6606598.