Empirical Software Engineering manuscript No.
(will be inserted by the editor)

Analysis of a Many-Objective Optimization Approach for Identifying Microservices from Legacy Systems

Wesley K. G. Assunção · Thelma Elita Colanzi · Luiz Carvalho · Alessandro Garcia · Juliana Alves Pereira · Maria Julia de Lima · Carlos Lucena

Received: date / Accepted: date

Abstract The expensive maintenance of legacy systems leads companies to migrate such systems to modern architectures. Microservice architectural style has become a trend to modernize monolithic legacy systems. A microservice architecture consists of small, autonomous, and highly-independent services communicating by using lightweight network protocols. To support the designing of microservice architectures, recent studies have proposed either *single* or *multi-objective approaches*. In order to improve the effectiveness of existing approaches, we introduced `toMicroservices` that is a *many-objective* search-based approach to aid the identification of boundaries among services. In previous studies, we have focused on a qualitative evaluation of the applicability and adoption of the proposed approach from a practical point of view, thus the optimization process itself has not been investigated in depth. In this paper, we extend our previous work by performing a more in-depth analysis of our *many-objective* approach for microservice identification. We compare our approach against a baseline approach based on a random search using a set of performance indicators widely used in the literature of many-objective optimization. Our results are validated through a real-world case study. The study findings reveal that (i) the criteria optimized by our approach are interdependent and conflicting; and (ii) all candidate solutions lead to better performance indicators in comparison to random search. Overall, the proposed many-objective approach for microservice identification yields promising results, which shed light on insights for further improvements.

Wesley Klewerton Guez Assunção ✉ · Luiz Carvalho · Alessandro Garcia · Juliana Alves Pereira · Carlos Lucena
Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brazil
E-mail: wesleyklewerton@gmail.com (✉ corresponding author)

Thelma Elita Colanzi
State University of Maringá (UEM), Maringá, Brazil

Maria Julia de Lima
Tecgraf Institute, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil.

Keywords Microservice architecture · Many-criteria optimization · Industrial case study · Performance indicators.

1 Introduction

Microservices are small and autonomous services that work together by using lightweight network protocols [49]. Fundamentally, microservices are distributed systems built as a collection of intercommunicating fine-granularity services with independent computational resources. Microservices architectural style has become a trend to develop highly scalable and available software systems [22]. A crucial step for a well-designed microservice architecture is the identification of proper microservice boundaries, which allow the adequate implementation of microservices. Some authors discuss that this identification should take into account the coupling and cohesion of the microservices [21,49]. In addition, other authors recommend the conception of microservices based on business capabilities [21]. That is, intrinsically related to the main features in each microservice. Thus, several criteria should be used to adequately identify microservices, where a criterion defines rules or properties on how to deal with existing information to support the decision-making process of defining microservices boundaries.

Recent studies show that practitioners have to simultaneously consider five typical criteria – cohesion, coupling, feature modularization, reuse and network overhead [9, 10, 11] – along the decision-making process. However, most of the state-of-the-art approaches are single or multi-objective, commonly optimizing only coupling and cohesion, without their validation in industrial systems [24, 25, 38, 39, 45, 62] (further described in Section 2). To fulfill this existing gap in practice, in a recent study we defined `toMicroservices`, a many-objective optimization approach considering the five criteria mentioned above [11, 12] (Section 3). In previous studies [3, 11], we performed an initial evaluation of the performance of `toMicroservices` and focused on the practical adoption of the microservices by maintainers of a legacy system. Those results pointed out that our many-objective approach is more promising than the single and multi-objective ones. However, the optimization process of our many-objective approach has not been investigated in depth. In addition, to the best of our knowledge, the relationship among the criteria adopted as objective functions has not been investigated yet, such as the interdependence among network overhead, feature modularization and reuse.

In this sense, the objective of this paper is to investigate the complexity of the microservice identification problem when optimizing five objectives. As an extension of our previous work in which we defined `toMicroservices` [11,12], in this paper, we report a detailed study (Section 4) with an in-depth analysis (Section 5) of the optimization process of our many-objective approach for microservice identification. We compare `toMicroservices`, based on *NSGA-III* with five objectives, against a *random search (RS)* using six performance indicators widely used in the literature, differently from our previous study that used only two performance indicators. Also, we extended our analysis with correlation analysis to investigate the interdependent and conflicting nature of the five criteria used as objective functions. The approach was validated in a real-world case study to answer the following research questions:

– *RQ1. To what extent are the five objective functions interdependent and conflicting in an industrial system?*
– *RQ2. How do NSGA-III and RS compare in terms of performance indicators when optimizing five criteria?*

To answer *RQ1*, we investigate the correlation between pairs of objective functions related to the criteria of microservice identification. By analyzing this correlation, we can understand how difficult the problem we are dealing with is. The correlation serves as an indicator of the complexity of the microservice identification problem justifying or not the use of a many-objective optimization approach. To address *RQ2*, we investigate the behavior of NSGA-III in comparison to a RS for solving the many-objective optimization problem of microservice identification. A search algorithm can always be compared against at least a random search to check that its success is not due to the simplicity of solving the posed problem [2]. In such evaluations we analyzed: (i) the results according to six performance indicators and three statistical tests widely used in the SBSE field [16, 17], and (ii) how the criteria are optimized by search algorithms.

The main contribution of our work is a detailed analysis on the complexity of the microservice identification problem when optimizing five objectives. Such an analysis was done in the context of the many-objective treatment for the microservice identification problem given by `toMicroservices`, as the related work [24,25,38,39,45,62] do not deal with more than 3 objectives. Our main findings indicate that (i) the referred problem deserves to be considered as a many-objective optimization problem since all criteria are important to the problem, (ii) the five criteria optimized during the microservice identification are in conflict and some of them are interdependent, and (iii) NSGA-III properly deals with the problem. This implies that further studies can deal with microservice identification as a many-objective problem without requiring this kind of investigation.

2 Background

This section presents a background of microservice architectures, migration from legacy systems, the activity of identifying microservices in the code, basic concepts on many-objective optimization, and an illustrative example of the identification of microservices from legacy systems.

Microservice architectures. Microservices are small and autonomous services that work together [49], where a service is a unit of software that is independently replaceable and upgradeable [21, 22]. The “small” aspect refers to the fact that a microservice should have fine granularity and address a single responsibility [21,22]. A microservice is also expected to be "autonomous": (i) it should consist of a service highly independent from others, and (ii) it should enable independent use of technologies. A microservice is not an entirely isolated architectural element. A microservice architecture usually relies on lightweight protocols. Each protocol provides reliable communication without responsibility for processing business rules [21]. For example, a common lightweight synchronous protocol communication is HTTP. The characteristics aforementioned involving a microservice and their relationships define what is a microservice architecture [49].

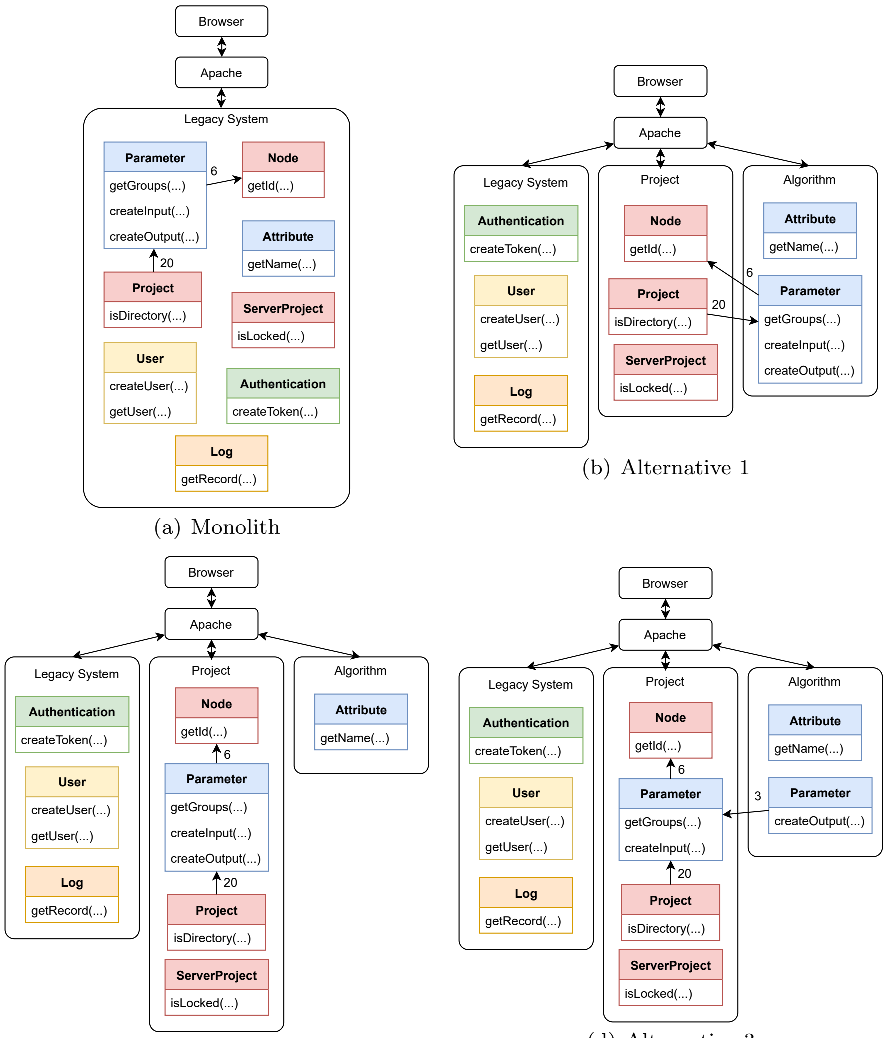
Migration to microservice architectures. There are several studies reporting the migration to microservice architectures, as reported in a mapping study [21]. The migration is usually motivated by many limitations, including difficult maintainability and inadequate resource usage in a cloud environment. One of the most challenging activities of the migration is the identification of microservices by defining their boundaries based on the legacy code [28, 42]. The manual identification of microservices in legacy code is a time-consuming, risky activity. Several manual and (semi-)automated approaches have been proposed to identify microservices.

Microservice identification in legacy code. The problem of microservice identification in legacy code is commonly seen as a software remodularization task [1], which is known to be an NP-hard problem. There is a huge number of possible combinations of source-code elements and its multi-criteria nature [46]. In addition to the huge combinations of source-code elements, there are also the desired properties to be achieved or criteria/constraints that should be taken into account when performing the remodularization [5]. Existing approaches adopt different criteria to identify microservices. Coupling is the most adopted criterion [24, 25, 38, 39, 45, 62]. Similarly to coupling, cohesion is also commonly used by automated approaches to promote better modularization [38,62]. Besides, some features can be used to derive execution cases of the legacy system [38, 39] to measure dynamic coupling or cohesion. However, there is no approach that uses features' information in objective functions. The same applies to reuse and communication overhead, which are also basic criteria for microservice identification in practice [9, 10]. Based on existing limitations, we defined an automated approach that uses five criteria as objective functions to provide an approach that suits better practitioners' expectations [11, 12]. In addition, our approach operates at the granularity of methods rather than only classes; in fact, features may be tangled and scattered in classes. This is one contribution of our work since existing approaches only operate at the class level, which leads to a coarse-grained analysis undesirable outputs. For instance, let us consider a possible threat in restricting to classes: legacy systems often consist of very large classes, which incorporates too many features. The result is that each class is artificially tagged with a single functionality, possibly the one with more class elements realizing it. Even worse, the other feature realized by the same class (but ignored by the tagging strategy) is likely to be mislocated in the resulting decomposition of the microservices.

Multi- and Many-Objective Optimization. Multi-objective optimization is an area of multiple criteria decision making that is concerned with mathematical optimization problems involving two or three objective functions to be optimized simultaneously [15]. In general, there is more than one solution for this kind of problems. Thus, several good solutions represent the trade-off between the defined objectives. Many-objective optimization refers to a class of optimization problems that have more than three objectives. Multi-objective evolutionary algorithms (MOEAs), such as the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [20] and the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [63], have received immense recognition due to their effectiveness and efficiency in tackling multi-objective optimization problems. Recently, numerous studies on MOEAs revealed that when handling many-objective optimization problems, MOEAs encounter challenges and the behavior of MOEAs resembles a random walk in search space as the proportion of non-dominated solutions increases subsequently [51]. The Non-Dominated Sorting Genetic Algorithm III (NSGA-III) [19] was designed to solve many-objective problems and has been successfully applied. In the SBSE field, NSGA-III was also applied to solve Software Engineering many-objective problems, such as software product line testing [37], software remodularization [48] and software refactoring [47].

Example of Many-Objective Analysis. Next we illustrate the need for simultaneously analyzing many criteria during microservice identification. Figure 1(a) depicts a monolithic architecture of a legacy system (related to our case study described in Section 4.1), which is the source of information for the identification of microservices. The goal is to identify the boundaries to extract the microservices that implement the features *Algorithm* and *Project*. Figures 1(b), 1(c), and 1(d) illustrate alternative microservice architectures, each one with two microservice candidates and the residual legacy system. Implementation elements responsible for the feature *Algorithm* are highlighted in blue whereas the elements are related to *Project* are in red. The numbers in the dependencies represent calls between implementation elements in the normal operation of the systems, which can be obtained with dynamic analysis. Several details were omitted in this figure to improve legibility, such as attributes, methods, relationships, as this is only an illustrative example.

Alternative 1, presented in Figure 1(b), is an architecture candidate that has one microservice with implementation exclusively of the feature *Project*. This architecture also has another microservice with implementation elements only related to *Algorithm*. In this architecture, the features *Project* and *Algorithm* are well modularized regarding their implementation elements, which might benefit a better visualization and source code organization. However, it does not represent a proper architecture if we consider the traditional metrics of coupling and cohesion, as we can observe dependencies between classes in different microservices. Maintenance in one feature might require modifying two microservices. The simplistic assumptions that features of the existing



(a) Monolith

(b) Alternative 1

(c) Alternative 2

(d) Alternative 3

Fig. 1 Alternative Architectures for the Legacy System

legacy system usually have well-modularized features in files do not hold in practice, as we further discuss in the results of our case study (Section 5). Furthermore, Alternative 1 does not take into account that a method allocated in the *Project* microservice candidate can massively call a method allocated within the *Algorithm*, and vice-versa, leading to prohibitive network overhead. In our illustrative example, we can observe that a dependency of 20 calls between the classes `Project` and `Parameter` become network communication.

The architecture candidate in Figure 1(c) allows an easier maintenance, since it groups elements that are structurally dependents. Alternative 2 also avoids prohibitive network overhead as dependent elements are in only one microservice. We can say that this solution is good in accordance with coupling, which is low. However, it reduces the cohesion of this architecture, as elements that realize the feature *Algorithm* are in another microservice. Existing approaches for microservice identification usually are based on the traditional metrics of coupling and cohesion to evaluate solutions. Consequently, we can observe that features are not well-modularized, since the implementation of *Algorithm* is scattered in two microservices, and more importantly, *Algorithm* is tangled with *Project* in the second microservice.

To reach architecture candidate that presents better cohesion, still avoiding the problem of massive calls between the two microservices, we can keep in *Project* only the method that this microservice is highly dependent on. Alternative 3, in Figure 1(d), presents an architecture in which one method of the class `Parameter` was decoupled to be kept in the microservice responsible for the feature it belongs to. The method `createOutput(...)` that is responsible for implementing *Algorithm* was moved to the proper microservice. This modification created a new dependency with 3 calls between microservices, increasing the coupling, which not lead to prohibitive network overhead. However, it still does not have an optimal feature modularization.

The alternative architecture candidates illustrate that more than two criteria are needed to achieve satisfactory microservice identification. In addition, software engineers notably can have different needs or preferences in the scenario they work on. For legacy systems, as the case study we deal with in this study, approaches should consider several criteria and optimize them to obtain a suitable microservice architecture. This also highlights the shortcoming of existing approaches that make simplistic assumptions about real systems from which microservices will be identified and extracted, *e.g.*, features of the existing system usually has well-modularized features in files, not being tangled and scattered through several methods of those files.

3 Many-Objective Identification of Microservices

In this work we use `toMicroservices` for the identification of microservices, which was introduced in a recent work [11, 12]. `toMicroservices` is an automated approach to identify microservice candidates to aid developers in designing microservice-based systems. Our approach relies on a many-objective optimization with five objective functions related to criteria classified by developers as useful to identify microservices [10]. The criteria are coupling, cohesion, feature modularization, network overhead, and reuse, which are described in Section 3.2. This approach requires three pieces of information as input: (i) an initial representation of the system to be implemented as microservices, *e.g.* a class diagram or the source code of a legacy system; (ii) a list of features with mapping to their implementation elements; and (iii) the number of microservices to be identified. It is important to note that our approach analyzes the input at the method level to achieve fine-grained microservices.

The output generated is a set of candidate solutions, named Pareto front (PF), where each solution is an alternative microservice architecture.

In the next subsections we describe details about `toMicroservices`, such as representation, objective functions, genetic operators and previous findings.

3.1 Representation

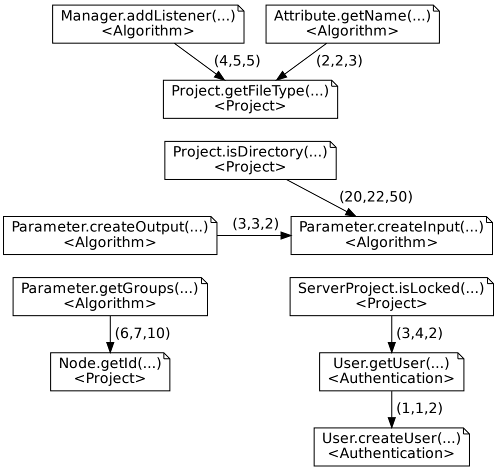
The proposed approach uses a graph-based representation. Each vertex represents a method, which is assigned to its respective feature. The edges are labeled with a triple e = (*sc*, *dc*, *ds*) in which *sc* contains information about static calls, *dc* the dynamic calls, and *ds* represents the estimated size of data used in the communication between methods. The optimization process is responsible for using these three pieces of information in the edges to group vertices of the graph that will be the microservices.

Figure 2 depicts an example of input and a possible output based on the case study described in Section 4.1. The input graph in Figure 2(a) has 11 methods/vertices from which there are seven calls among them. The features assigned to each vertex is presented between "<" and ">". For example, the method `Manager.addListener(...)` implements part of the feature *Algorithm*. The output in Figure 2(b) presents an example of a microservice architecture, where the methods are grouped into three microservices, corresponding to features *Algorithm*, *Project*, and *Authentication*. In this example of output, we can see the dependencies between microservices, which will become network communications. For example, three methods that belong to `Microservice_Algorithm` (blue in the figure) call two methods that belong to `Microservice_Project` (red in the figure).

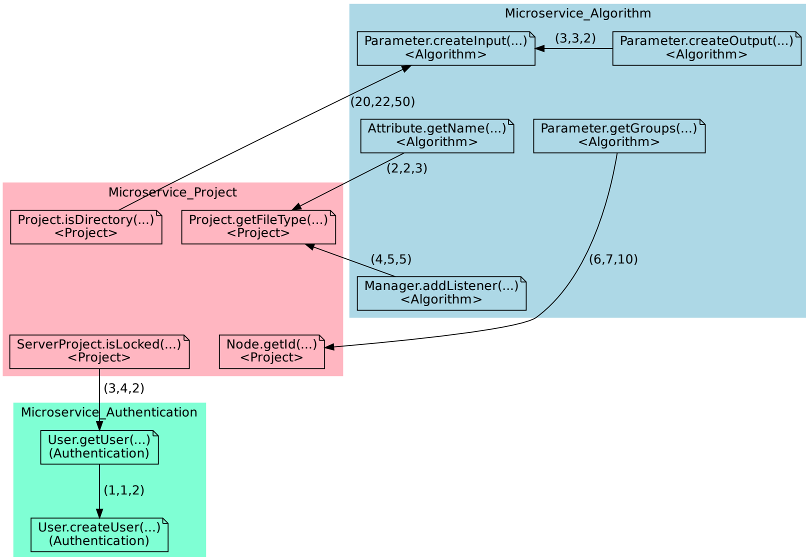
3.2 Objective Functions

In this section we describe the criteria adopted as objective functions. The criteria of coupling and cohesion were based on related work, but adapted in our approach to a fine-grained, *i.e.*, method level. The criteria of feature modularization and reuse have been also used in the context of traditional architectures, which inspired us on proposing and adapting their application to the context of microservices. Network overhead is a criterion designed specifically for microservice architecture. From now on, MSA (MicroServices Architecture) refers to the graph of the microservice candidates (vertices) and their communications (edges) generated by our approach, and MS_C (MicroService Candidate) refers to some vertices in the MSA. The criteria are defined next, with illustrative examples of their computation using the output presented in Figure 2(b).

1. **Coupling:** coupling for each microservice candidate MS_c is computed by the sum of the number of static calls from any method v_i that belongs to MS_c to any methods v_j that do not belong to MS_c (see Equation 1),



(a) Input graph with methods



(b) Output graph with microservices

Fig. 2 Example of representation used by our approach

similarly to [13]. A static call consists of a syntactic call to another v_j method in the body of the v_i method. The total coupling of a solution, *i.e.*, an individual, is the sum of the values of coupling associated with every MS_c in a MSA (see Equation 2).

$$\delta(MS_c) = \sum^{v_i \in MS_c \wedge v_j \notin MS_c} sc(v_i, v_j) \qquad (1)$$

$$Coupling(MSA) \;=\; \sum^{\forall MS_c \in MSA} \delta(MS_c) \qquad (2)$$

In the MSA presented in Figure 2(b), `Microservice_Algorithm` has three dependencies to another microservice, as for example `Parameter.getGroups(...)` depends on `Node.getId(...)`. This dependency has 6 static calls, as we can observe in the first item of the tuple (6,7,10). To compute the coupling of this first microservice, we sum the static calls for those three dependencies (Equation 1), as follows: δ(`Microservice_Algorithm`) = 6 + 4 + 2 = **12**. `Microservice_Project` has two dependency to other microservices, namely a dependency with 20 static calls to `Microservice_Algorithm` and a dependency with 3 static calls to (`Microservice_Authentication`). This lead to δ(`Microservice_Project`) = 20 + 3 = **23**. `Microservice_Authentication` has no dependency to other microservices, then δ(`Microservice_Authentication`) = **0**. Finally, the coupling for the candidate solution is $Coupling(MSA) = 12 + 23 + 0$ = **35**, which is the sum of coupling for every microservice (Equation 2).

2. **Cohesion:** cohesion is defined by dividing the number of the static calls between methods within the microservice boundary, *i.e.*, the set of methods assigned to the candidate, by all possible existing static calls, similarly to [13]. Hence, this measure indicates how strongly related the methods are within a microservice candidate. In order to compute cohesion, the ce function is defined in Equation 3 as a boolean function indicating the existence of at least a static call. The cohesion of a microservice candidate is presented in Equation 4, where $|MS_c|$ is the cardinality of a MS_c. Basically, Equation 4 divides the number of static calls by the number of all possible dependencies between methods of a microservice candidate. In this sense, the denominator of Equation 4 is the combination two-by-two of all methods within a MS_c. The total cohesion of a solution is the sum of the cohesion associated with every MS_c in a MSA (see Equation 5).

$$ce(v_i, v_j) = \begin{cases} 1, if sc(vi, vj) > 0 \\ 0,\ otherwise \end{cases} \qquad (3)$$

$$C(MS_c) \;=\; \frac{\sum^{\forall v_i \in MS_c \wedge v_j \in MS_c} ce(vi, vj)}{\frac{|MS_c|(|MS_c| - 1)}{2}} \qquad (4)$$

$$Cohesion(MSA) \;=\; \sum^{\forall MS_c \in MSA} C(MS_c) \qquad (5)$$

Let us consider the MSA presented in Figure 2(b) to illustrate the computation of Cohesion. According to the denominator of Equation 4, the number of all possible dependencies (independently of the direction) between the five methods withing

`Microservice_Algorithm` is obtained by $\frac{5(5-1)}{2} = 10$. However, among all these possible two-by-two combinations withing this microservice, there is only one existing dependency with $sc(vi, vj) > 0$ (Equation 3), namely between `Parameter.createOutput(...)` and `Parameter.createInput(...)`. This means that the sum of all existing dependencies with static call greater than 0 is equal to 1, representing the the numerator of Equation 4 . In this case, $C($`Microservice_Algorithm`$) = \frac{1}{10} = 0.1$. For `Microservice_Project`, the number of all possible dependencies is $\frac{4(4-1)}{2} = 6$, without existing any dependency, which lead to $C($`Microservice_Project`$) = \frac{0}{6} = 0$. For `Microservice_Authentication` the computation of possible dependencies is $\frac{2(2-1)}{2} = 1$ and it has one dependency, then $C($`Microservice_Authentication`$) = \frac{1}{1} = 1$. Finally, $Cohesion(MSA) = 0.1 + 0 + 1 = 1.1$.

3. **Feature Modularization:** a microservice architecture (MSA) can have microservice candidates (MS_c) composed of methods belonging to several features. We used the vertices labeled to recommend feature modularization in the microservice candidates with fine granularity and limited responsibility. The predominant feature number for a MS_c is the number of occurrences of the most common feature divided by the sum of all features occurrences within MS_c. Equation 6 defines the predominant feature (pf function) of a MS_c, where F_{MS_c} is a set of occurrences by features in a MS_c. The feature modularization of a microservice candidate is defined in Equation 7, that is, a measure of the number of occurrences of the most common features divided by the sum of all features occurrences within a microservice candidate. This equation avoids the fact that each microservice candidate has largely different features. The feature modularization of a solution MSA is the sum of the predominant features number in every MS_c added to the division of the number of distinct predominant features ($|FRCA|$) in the MSA by the number of microservice candidates ($|MSA|$), as shown in Equation 8. It is desired to have a degree of feature modularization as high as possible.

$$pf(MS_c) = max_{\ \forall k \in F_{MS_c}}\{k\} \tag{6}$$

$$f(MS_c) = \frac{pf(MS_c)}{\sum^{\forall k \in F_{MS_c}}\{k\}} \tag{7}$$

$$F(MSA) = \sum_{\forall MS_c \in MSA} f(MS_c) + \frac{|FRCA|}{|MSA|} \tag{8}$$

To illustrate the computation of feature modularization, we recall that MSA of our example implements three features, namely *Algorithm*, *Project*, and *Authentication*. Starting with `Microservice_Algorithm`, the first step is identify the maximum occurrence of the predominant feature (Equation 6). In this case, all methods of this microservice are from the feature *Algorithm*, i.e., $pf(MS_c) = 5$. This value is the numerator for Equation 7, which is divided by all occurrences of features withing the microservice that in this case is also five: $\sum^{\forall k \in F_{MS_c}} \{k\} = 5$. Then, $f($`Microservice_Algorithm`$) = \frac{5}{5} = 1$. The value 1 is the perfect one for a microservice, meaning it has only methods of a single feature. The other microservices will have the same value of feature modularization, since in our example, each microservice has methods belonging to only one feature. In other words, $f($`Microservice_Project`$) = \frac{4}{4} = 1$ and $f($`Microservice_Authentication`$) = \frac{2}{2} = 1$. Finally, the feature modularization for the whole architecture is $F(MSA) = (1 + 1 + 1) + \frac{3}{3} = 4$.

4. **Network Overhead:** Some non-functional requirements may be affected by the network overhead of the identified microservices. To minimize the overhead, we created a heuristic that uses dynamic information to predict the network overhead. The heuristic uses the size of the objects and primitive types passed as parameters between methods during the execution of the legacy system. In addition, the heuristic considers the network overhead caused by the adopted protocol to communicate with the future extracted microservices. For example, the HTTP protocol adds a header to each call and, therefore, the size of this header is considered in our estimation of network overhead. The network overhead measurement is presented in Equation 9 where the function $P(v_j)$ returns the set of arguments used in the execution of the method v_j. The function sizeOf(p, m) is the size of the p-th parameter in the m-th call from v_i to v_j. Data traffic function (dt) is computed as shown in Equation 10, where dc function is the total of calls from method v_i to method v_j in execution time. The network overhead of MS_c (see Equation 11) is the sum of all data traffic within their methods, and the network overhead of a proposed MSA is defined as the sum of the sizes of the network traffic data to each MS_c (see Equation 12).

$$overhead(v_i, v_j, m) = \sum^{\forall p \in P(v_j)} sizeOf(p, m) \tag{9}$$

$$dt(v_i, v_j) = max_{m=1}^{m=dc(v_i,v_j)}(overhead(v_i, v_j, m)) \tag{10}$$

$$O(MS_c) = \sum^{\forall v_i \in MS_c \wedge \forall v_j \notin MS_c} dt(v_i, v_j) \tag{11}$$

$$Overhead(MSA) = \sum^{\forall MS_c \in MSA} O(MS_c) \tag{12}$$

To compute the network overhead of a solution in Figure 2(b) we use the second and third information of the triple in the edge labels, namely dc the dynamic calls and the ds size of data used in the communication between method. The values of ds is basically the result of Equation 9, that is obtained from the execution of the legacy system. Equation 10 considers this information and the number of dc to identify the maximum potential data traffic between methods. For the sake of simplicity, in this illustrative example we consider as result of Equation 10 the values of ds presented in the edges as the maximum value. In Equation 11 we compute the total of data traffic between methods of different microservices. For example, three methods of `Microservice_Algorithm` communicate with methods of other microservice, namely $dt($`Attribute.getName(...)`, `Project.getFileType(...)`$) = 3$, $dt($`Manager.addListener(...)`, `Project.getFileType(...)`$) = 5$, and $dt($`Parameter.getGroups(...)`, $Node.getId(...)) = 10$. In this case, $O($`Microservice_Algorithm`$) = 3 + 5 + 10 = 18$. Similarly, $O($`Microservice_Project`$) = 50 + 2 = 52$ and $O($`Microservice_Authentication` $= 0$, as this latter does not communicate other microservices. Finally, $Overhead(MSA) = 18 + 52 + 0 = 70$.

5. **Reuse:** The reuse of a microservice candidate is measured by the relationships between the microservice candidate and the user of the legacy system (e.g, calling the API or user interface). To do so, static and dynamic analysis are combined to observe the level of reuse of a microservice within the microservice architecture. In the dynamic analysis, each microservice candidate is reusable when it is directly called by a user. This concept is captured by the mdu function (**m**icroservice **d**irectly called by the **u**ser). mdu function considers the system executions that allow identifying dynamic calls between vertices, including start points by the user.
Equation 13 measures the reuse associated with each microservice candidate. Such an equation captures whether each microservice is useful for other microservices in the architecture or directly by the user. Whenever a microservice candidate is reused at least twice, the microservice candidate indicates an adequate reuse level. Ideally, a microservice should be reused as much as possible or at least twice [8]. Thus, whenever a microservice candidate is reused at least twice, the result of Equation 13 indicates an adequate reuse level of the microservice (*i.e.*, $r(MS_c) = 1$).
The reuse of a microservice architecture is defined in Equation 14, where $|MSA|$ is the number of microservices. *Reuse* assumes the value 1 when all microservices are used at least twice by another microservice or the user. Its value ranges from 0 to 1. The goal is to maximize the reuse of the microservices encompassed by a solution.

$$r(MS_c) \;=\; \begin{cases} 1, if \;\sum^{v_i \in MS_c \wedge v_j \notin MS_c} sc(v_j, v_i) + mdu(MS_c) > 1 \\ 0, \;otherwise \end{cases} \tag{13}$$

$$Reuse(MSA) \;=\; \frac{\sum^{\forall MS_c \in MSA} r(MS_c)}{|MSA|} \tag{14}$$

For our example in Figure 2(b), we use Equation 13 to compute the sum of the first and second information of the triple of all external call to methods withing a microservice. To compute the reuse of `Microservice_Authentication` we observe that its method `User.getUser(...)` is called by `ServerProject.isLocked(...)` that belongs to another microservice. In this case, the sum of $sc = 3$ and $dc = 4$ is greater than 1, so $r(\texttt{Microservice\_Authentication}) = 1$. The other two microservices of this architecture candidate also have their methods called at least twice, which also lead to $r(\texttt{Microservice\_Algorithm}) = 1$ and $r(\texttt{Microservice\_Project}) = 1$. Finally, $Reuse(MSA) = \dfrac{1+1+1}{3} = 1$.

3.3 Genetic Operators

In the current version, our approach focuses on the use of a mutation operator, which is based on related studies that adopted genetic algorithms [38,39]. The reason for focusing on such a mutation is to avoid architectural violations, guarantee accuracy and consistency, which is pointed by Harman and Tratt as very complex when applying crossover operators for the optimization of software designs [33]. The mutation of one individual consists of moving methods from one microservice candidate to another one, *i.e.*, regrouping vertices in the graphs composing different microservices. In a simplified form, we can see the mutation operator as an analogy of the move method refactoring [27].

3.4 Previous Findings

In the studies that are the basis of this extension [11, 12], we have observed some interesting findings. In a first evaluation of the use of many-objective optimization for identifying microservices [12], the results pointed out that the criteria of feature modularization, network overhead, and reuse introduced a new perspective in the optimization of the solutions. Also, we observed that these additional criteria are not subsumed by coupling and cohesion. Furthermore, the obtained solutions allow restructuring features to be smoothly migrated to a microservice architecture. On the other hand, it was noticed that human interaction during the evolutionary process would obtain solutions to better satisfy the developer's expectations.

In another study [11], we evaluated the performance of NSGA-III in comparison with NSGA-II, which is the algorithm used in related work [38], using the traditional criteria of coupling and cohesion to optimize the microservice identification. Based on results of two performance indicators, namely HV and IGD (Section 4.4), and statistical analysis we observed that NSGA-III outperformed NSGA-II even when optimizing only two objectives. This confirmed that the NSGA-III was the right choice for the optimization algorithm of `toMicroservices`. In addition to the analysis based on two objectives, we performed a preliminary comparison of `toMicroservices` with a RS considering the five criteria as objectives [11]. Using only two performance indicators, we observed that the identification of microservices is a complex problem, requiring an optimization strategy. This becomes clear as `toMicroservices` always outperform the RS.

In addition to the preliminary quantitative analysis, we interviewed eight experienced developers of the legacy system to collect their opinion about the potential adoption of solutions obtained by `toMicroservices` when optimizing five objectives [11]. As an overall result of this qualitative analysis we observed that developers found the solutions generated by `toMicroservices` adoptable in practice. Four participants would adopt all microservices identified by our approach, three participants would adopt some microservices identified by `toMicroservices`, and only one participant would not adopt any microservice.

As the main reason why some participants would not adopt all microservices generated by `toMicroservices` is the granularity level of the obtained microservices, in another study [3], we asked the same participants to evaluate three solutions generated by `toMicroservices` using different granularity levels during another interview. The results show that `toMicroservices` is flexible and able to generate solutions according to the developers' needs and preferences. Empirical evidence reveals that the solutions generated by our approach allow restructuring features to be smoothly migrated to a microservice architecture. The main findings of our study [3] include discovering that (i) the features were modularized as microservices aligned with the business capabilities of the legacy system; (ii) the developers of the legacy system consider mainly the criteria included in our strategy; and (iii) reuse and customization opportunities arouse from the redesign of features as microservices.

It is important to highlight that the goal of `toMicroservices` is not to provide a ready-to-use solution, but providing near-optimal solutions that can be used as a starting point by software engineers, who must to analyze some generated solutions and make the necessary changes before the migration process.

The preliminary findings of our study show that `toMicroservices` is a promising approach to aid the microservice identification during the process of modernization of legacy systems with microservices. The fact of using five criteria that were observed as useful in practice is beneficial for the optimization process. However, the relationship among these criteria, their conflicting or dependent nature, has not been investigated yet. For instance, there is no study in the literature investigating the relationship among the network overhead, feature modularization, and reuse criteria. Also, a more strong quantitative analysis with more performance indicators and statistical analysis can bring more evidence on the advantages of using `toMicroservices` in industrial settings.

4 Study Design

To answer the RQs raised in Section 1, we designed the study described next.

4.1 Industrial Subject System

Our case study relies on a monolithic legacy system, in the domain of the oil and gas industry, currently subject to a modernization process by extracting features as microservices. This legacy system has been maintained for more than 15 years and is predominantly developed in Java. The developers reported that the maintenance of this system is very complex and time-consuming. In addition, either the inclusion of new features or the adoption of new technologies is a cumbersome task.

System under analysis. The monolithic architecture of the legacy system shares software libraries and holds highly coupled components with overlapping responsibilities. Our partner has experienced severe challenges with this architecture when trying to rapidly change or develop new features: (i) the legacy system holds a huge amount of features, resulting in unnecessary complexity and confusion; (ii) there are many calls and dependencies cross-crossing the system, resulting in hard to identify errors; and (iii) developers are limited to the technologies of the legacy system. To overcome these problems, we aim to rely on the extractive approach to substitute the large components of the monolithic architecture by ten independent microservices. The benefit of adopting an extractive approach is directly to group related responsibilities of the system. Thus, evolution and maintenance opportunities can be easily discovered. Through interviews with our partner, we obtained three main features they prefer to extract as microservices in the first moment: *algorithm*, *project*, and *authentication*.

- *Algorithm*: provides algorithms information by a REST API, including parameters, binary, documents, and connection points with other algorithms. In addition, this information can be stored using different resources.
- *Project*: responsible for the concept of a collaborative environment between the system's users. This collaborative environment includes shared projects and their metadata between different users or types of users.
- *Authentication*: authenticates and provides information of system users. This includes the creation and validation of tokens, verification of login and password, update of password, and related simple information about users. Source codes related to this feature are used extensively in the entire system for checks and information retrieval.

Information use. As discussed in Section 3, our approach uses a mapping of features to their corresponding methods and a dependency graph between methods (see Fig. 2(a)). As an example, the method `Manager.addListener(...)` implements part of the feature *algorithm* and it makes a call to the method `Project.getFileType(...)` that implements the feature *project*. After the complete mapping, we can use this information to identify microservice solutions. For example, we have as output three groupings: the left blue grouping of methods labeled *algorithm*, the right red grouping of methods labeled *project*, and the right down green grouping of methods labeled *authentication* (see Fig. 2(b)). These groups are used by the many-objective algorithm to optimize the criteria of coupling, cohesion, network overhead, feature modularization, and reuse. In the following sections, we will explain the algorithm in detail.

4.2 Implementation Aspects and Parameter Settings

State-of-the-art approaches to deal with the identification of microservices are based on NSGA-II [38, 39, 62], which is the most common evolutionary algorithm to deal with multi-objective problems. However, NSGA-II faces some challenges and difficulties for problems with more than three objectives [19]. Since our approach relies on five criteria, which are the objective functions, we adopt NSGA-III as the search-based algorithm. NSGA-III is designed to face up with many objectives at the same time [19]. Despite NSGA-III has a different strategy to compose the set of non-dominated solutions, its algorithmic complexity is quite similar to NSGA-II [18]. This algorithm was implemented on top of jMetal[1] that is a Java-based framework that includes modern state-of-the-art algorithms [23]. We also used jMetal to implement a RS algorithm, which is considered as the baseline in our study.

To represent solutions in our implementation, we create a class `Vertex` that represents each method. Then, a class named `Microservice` has a list of `List<Vertex> vertices`, which stores the methods that belongs to a microservice. Finally, a solution is an object of a class named `MicroservicesSolution` that implements the jMetal interface `Solution`. This class has a field `List<Microservice> microservices`, described above, that stores all microservices of the architecture.

In the implementation, we decided to treat the microservice identification as a minimization problem. Hence, the objective functions related to cohesion, feature modularization and reuse have their values inverted during the evolutionary process. In addition, a constraint related to the minimum and maximum numbers of methods per microservice was established in order to balance the granularity and preserve the reasonability of each microservice. Solutions that violate this constraint are discarded.

Settings. For the experiments, NSGA-III was configured as follows. The population size of 100 individuals. The maximum number of fitness evaluations equal to 10,000, which is also the stopping criterion. In addition to these traditional parameters, there are three more parameters related to our problem: (i) the fraction of methods exchanged by the mutation operator, that was set to the minimum of 1 to the maximum of 50% of all microservice methods;

[1] `http://jmetal.sourceforge.net/`

(ii) the number of microservice candidates was set to 10; and (iii) number of methods allocated in each microservice was set to between 3% and 16% of the total number of methods. For the RS, the random selection of vertex per each microservices follows the size of the constraints previously presented. For statistical significance, we executed 30 independent runs for each algorithm.

Solution sets. For analysis, we rely on three sets of solutions: (i) PF_{approx} is the Pareto front of non-dominated solutions obtained in each run of an algorithm. Since each algorithm is run 30 times, we have 30 PF_{approx} sets for each algorithm. (ii) PF_{known} is the set of non-dominated solutions found by an algorithm, considering the union of all solutions obtained in all its runs, eliminating the dominated ones. (iii) PF_{true} is conceptually known as the set with ideal solutions for a problem. As the PF_{true} of our problem is not known in advance, we adopted a common way to estimate this Pareto front that is using the non-dominated solutions found by all algorithms in all runs [65].

Feature label in the vertices. We developed an extractor to perform a pre-processing traceability step. This step was performed before the optimization process to label each vertex with the feature that it implements. In other words, our optimization process is independent of this extractor. We present how the extractor works next. In spite of that, for other legacy systems that use different technologies, the data can be extracted differently.

The input for the extractor was provided by a developer with experience in the legacy system. The input was the source code, the list of features to redesign with corresponding entry points to their source code, and the functional test cases related to these features. For the feature labeling, the expert on the legacy system informed the entry points in the trace execution of the three features selected to become microservices, and their subfeatures. Each entry point defines one of the possible entries to the feature boundaries. Then, all entry points are used in the trace execution to label the vertices of the graph representation (Section 3.1). This definition of the entry points were the only manual analysis required during the analysis of the legacy system.

Our extractor is based on execution of functional test cases. Here, the entry points are used to associate features with execution traces. In summary, an entry point is a relationship between a regular expression and a feature. These expressions are compared with patterns in the names of packages, classes, or methods in the execution trace. When there is a match between the entry point with a feature label, and the method in the execution trace, it is labeled in graph vertex with the related feature. All methods in the execution trace that are not entry points are then labeled with the feature of the last entry point (lower depth number).

To illustrate this process, Listing 1 shows an excerpt of trace execution and Listing 2 presents entry points to *Algorithm* and *Project*. When performing the comparison, `Algorithm.getAdminIds` is matched with the regular expression associated with the feature *Algorithm*, since this method is an entry point and labeled with this feature. Also, the following methods with higher depth are also labeled with the feature *Algorithm* until reach the method `ProjectService.getAllProjects`. At this point, this method matches the regular expression *Project*, then it is labeled with the feature *Project* as well as `ProjectInfoService.getInfo` because it has a higher depth. Next, the method `Algorithm.algorithmsToVector` is labeled with *Algorithm*, due the lower depth as an entry point is *Algorithm* and not *Project*.

Listing 1 Execution trace example

```
Name:Algorithm.getAdminIds#Depth:12
Name:Algorithm.getAllAlgorithms#Depth:13
Name:Algorithm.loadLocalAlgorithmCache#Depth:14
Name:Algorithm.getPermission#Depth:13
Name:AlgorithmPermission.getAllPermissionIds#Depth:14
Name:AlgorithmPermission.getPermissionIds#Depth:15
Name:ProjectService.getAllProjects#Depth:16
Name:ProjectInfoService.getInfo#Depth:17
Name:Algorithm.algorithmsToVector#Depth:14
```

Listing 2 Entry points of the features Algorithm and Project

```
Algorithm<Algorithms.getAdminIds>
Project<ProjectService.*>
```

4.3 Correlation Test

To analyze the correlation between the five objectives (RQ1), we firstly applied the Shapiro-Wilk normality test [54] to verify the distribution of data. In the cases the test points out that all sets have non-normal distribution, the Spearman correlation test [55] is applied to check for any correlation (positive or negative) between the pairs of objective functions. Spearman's rank correlation coefficient is a non-parametric measure of rank correlation (statistical dependence between the rankings of two variables). Both tests were applied with confidence level of 95% (significance level 5% - p-value$<$ 0.05). These tests are widely used in Software Engineering studies [2, 16, 17]. To better understand the correlation coefficient, we used the following scale [35]:

- 0.9 to 1.0 (or -0.9 to -1.0): very high positive (or negative) correlation;
- 0.7 to 0.9 (or -0.7 to -0.9): high positive (or negative) correlation;
- 0.5 to 0.7 (or -0.5 to -0.7): moderate positive (or negative) correlation;
- 0.3 to 0.5 (or -0.3 to -0.5): low positive (or negative) correlation;
- 0.0 to 0.3 or 0.0 to -0.3: negligible correlation.

4.4 Performance Indicators

The use of performance indicators is the most common way to compare multi/many-objective optimization algorithms [60]. In this way, we can observe their use in many SBSE studies [16, 17]. Performance indicators enable us to assign scores to Pareto fronts found by multi/many-objective optimization algorithms [60]. In addition, these indicators enable the decision-maker to visualize the consequences of his/her choices regarding the performance of a criterion at the expense of one or other criteria, supporting appropriate decisions [60].

As a single performance indicator alone cannot provide a comprehensive measure for multi/many-objective optimization algorithms [60], we choose to use six performance indicators with different purposes, namely the evaluation of convergence, distribution, coverage and cardinality of solutions [65]. Generational Distance (GD) [56] and Inverted Generational Distance (IGD) [53] measure the closeness of the solutions to the theoretical Pareto front. Hypervolume (HV) [65] considers both closeness and diversity at the same time. Coverage (C) [64] compares a pair of algorithms in terms of the dominance of the solutions found. Error Ratio (ER) [56] counts the number of Pareto optimal solutions in the set found by an algorithm. Furthermore, we used the Euclidean Distance to the Ideal Solution (ED) [14] as an indicator to identify the solution with the best trade-off among the objectives as the decision-makers usually prefer to select this solution from the set of alternative solutions. These indicators are described in details in the following.

HV measures the area of the objective space from a reference point to a front of solutions [65]. This indicator enables to analyze both closeness and diversity of a Pareto front [60]. In this study, we use the HV computed by a recursive and dimension-sweep algorithm [26]. To compute HV we normalized each PF_{approx} between 0 and 1, and adopted a reference point with the value of 1.1 for all five objectives. Pareto fronts with high values of HV are the best since their solutions are far from the reference point.

GD and IGD measure the convergence/closeness between PF_{approx} and PF_{true}. GD is an error measure used to examine the distance of the solutions found by an algorithm (PF_{approx}) to the best solutions known (PF_{true}) [56]. IGD is an indicator based on GD, but with the goal of evaluating the distance from PF_{true} to PF_{approx}, *i.e.*, the inverse of which is considered by GD [53]. Values of GD and IGD closer to 0 are desired, which indicates that the solutions of both PF_{approx} and PF_{true} are close to each other.

ED is used to find the closest solution to the best theoretical objectives, *i.e.*, an ideal solution [14]. For our minimization optimization, to compute ED, we normalized each PF_{approx} between 0 and 1, and the ideal solution has a value equal to 0 for all objectives. The solution with the lowest value of ED represents the solution with the best trade-off among the objectives.

C measures the dominance between two sets of solutions [64]. $\mathrm{C}(\mathrm{PF}_a, \mathrm{PF}_b)$ represents a value between 0 and 1 according to how much the PF_b solutions are dominated by the PF_a solutions. Similarly, $\mathrm{C}(\mathrm{PF}_b, \mathrm{PF}_a)$ returns how many solutions in PF_a are dominated by solutions in PF_b. A value equal to 0 for C indicates that the solutions of the former set do not dominate any element of the latter set and, on the other hand, the value 1 indicates that all solutions of the latter set are dominated by elements of the former set.

ER is an error measure to compute the number of PF_{known} solutions that are not in PF_{true} [56]. Higher values of ER means that the algorithm does not have good convergence. The lower the ER, the better is the performance of the algorithm because a larger number of solutions of PF_{known} were found in PF_{true}.

4.5 Statistical Analysis

To analyze the statistical difference between NSGA-III and RS regarding the performance indicators (RQ2), firstly we evaluated the distribution of the values with the Shapiro-Wilk normality test [54], the same test used in the analysis of the correlation between criteria. Then we used the Wilcoxon rank-sum test [6], which is a non-parametric test based on the median of the values, and Welch Two Sample t-test [58], which is a parametric test based on the mean of the values. Furthermore, we also compute the effect size with $\hat{A}_{12}$ Vargha and Delaney [57] . These tests are widely used to assess search-based algorithms in Software Engineering [2, 16, 17].

5 Results and Analysis

Next we describe and analyze the results of our study to answer the RQs.

5.1 RQ1 - Objective Functions Correlation

Multi/Many-objective algorithms usually return a few solutions when the objectives are directly related since the optimization of one objective implies in the optimization of the other one. On the other hand, a high number of solutions is obtained in the presence of conflicting objectives since the optimization of one objective compromises the other one. In this way, as mentioned in RQ1, we evaluated whether there is a significant correlation between each pair of objective functions used by NSGA-III. The sample-set used as input to evaluate the correlation between the functions is the PF_{known} obtained by this algorithm. NSGA-III obtained the median of 76 non-dominated solutions per run, pointing out to the existence of negative correlations between the objectives.

Analysis. To analyze the correlation between the five objectives, we firstly applied the Shapiro-Wilk test to verify the distribution normality of the values for each objective function. Table 1 presents the results of the distribution test. The basis for this analysis is the PF_{known}, which is the set of non-dominated solutions obtained after the union of the solutions found in the 30 independent runs. The results point out that none of the distributions is normal, *i.e.*, p-value < 0.05.

Table 2 presents the results of the Spearman correlation test, for each pair of objective functions, including the p-value and the correlation level. This table also presents the interpretation of the correlation level, which provides evidence on the existence of a correlation between the investigated objective functions. Negative values mean that the correlation is negative, *i.e.*, if one

Table 1 Shapiro-Wilk test results for objective function values achieved by PF_{known} sets of NSGA-III.

Objective Function	**p-value**	**Conclusion**
Cohesion	4.117e-10	Non normal distribution
Coupling	2.731e-05	Non normal distribution
Feature Modularization	0.001058	Non normal distribution
Overhead	2.2e-16	Non normal distribution

function increases the other one always decreases. It is important to highlight that the criteria of feature modularization, cohesion, and reuse must be naturally maximized. However, as we are dealing with a minimization problem, in the objective functions related to these three criteria the values have been inverted.

Table 2 Spearman Rank Correlation Coefficient between PF_{known} sets.

Objective Functions	**p-value**	**Correlation Level**	**Conclusion**
Cohesion x Coupling	2.2E-16	-0.7699418	High negative correlation
Cohesion x Feature Modularization	2.2E-16	-0.4439882	Low negative correlation
Cohesion x Overhead	0.0002356	-0.1569073	Negligible correlation
Coupling x Feature Modularization	2.23E-02	0.1805772	Negligible correlation
Coupling x Overhead	0.002094	-0.1315126	Negligible correlation
Feature Modularization x Overhead	3.85E-10	-0.3043058	Low negative correlation

High correlation. The test indicates a significant negative correlation between cohesion and coupling, which means the objective functions are highly related but are inversely proportional. We can realize the correlation between these functions by observing the fitness of the solutions in Figure 4(a), since as coupling decreases, cohesion increases and vice-versa.

Low correlation. The Spearman coefficient also indicates a negative correlation between the pairs (cohesion, feature modularization) and (feature modularization, overhead), however, the correlation is low. We noticed that the other correlations are negligible, which points out the other pairs of functions are not interdependent. Hence, we can conclude that the optimization of all functions is important for the microservice identification problem as four criteria are not highly interdependent and these criteria allow evaluating different characteristics of each obtained solution.

Many-objective optimization is necessary. Regarding the conflict among the objective functions, we can see in the third column of Table 2 that all correlations are negative, except for the pair (coupling, feature modularization). Despite the correlation level, the negative correlations suggest that there is some kind of conflict between some pairs of objectives, namely (cohesion, coupling), (cohesion, feature modularization) and (feature modularization, overhead), justifying the need of many-objective optimization. The value of the objective function related to the reuse of all solutions is 1.0. This single fitness value impaired the application of statistical tests. Therefore, the objective function reuse was not considered for the correlation tests.

> **Answering RQ1:** The results reveal the objective functions for cohesion and coupling are highly conflicting and interdependent. The results also indicate the feature modularization function slightly compromises the cohesion and overhead functions. The other functions are not highly interdependent, justifying the need of applying a many-objective optimization algorithm.

5.2 RQ2 - Quantitative comparison between NSGA-III and RS

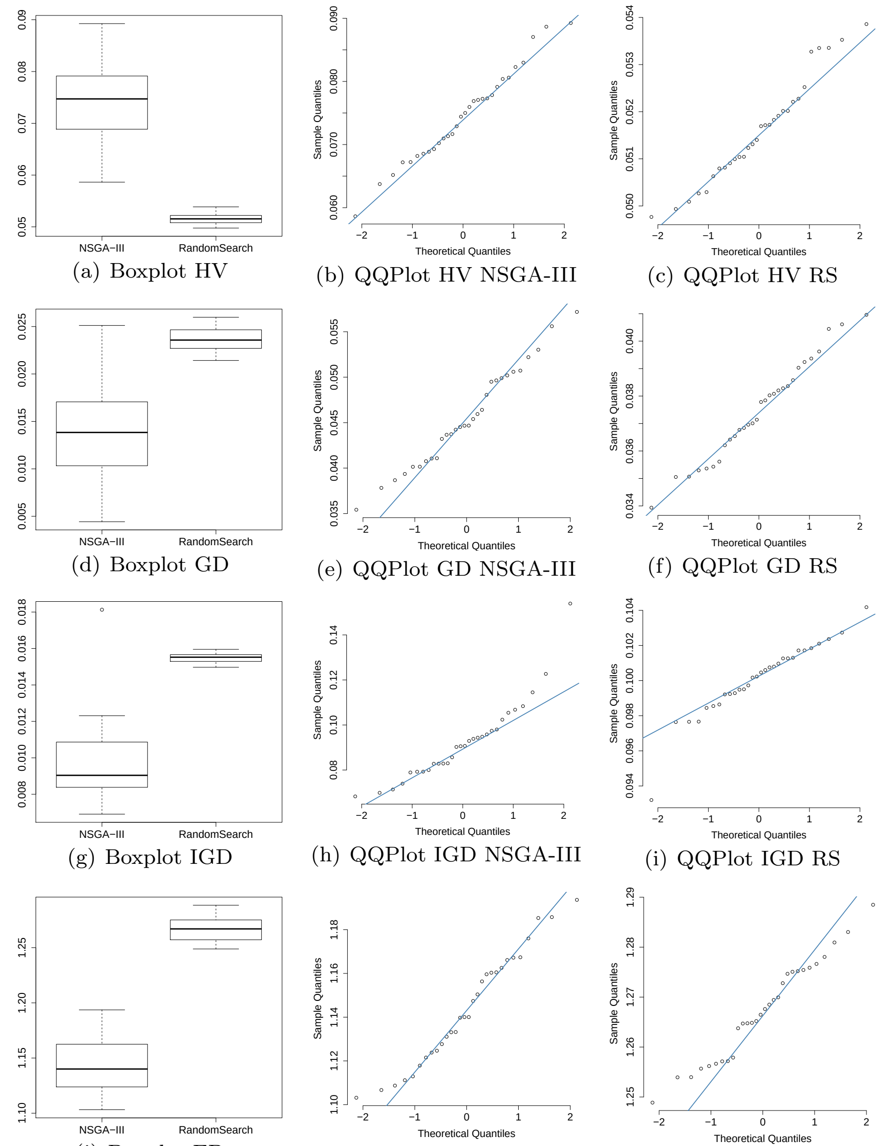
The quantitative analysis between NSGA-III and RS takes into account six performance indicators (see Section 4.4). These indicators have different Pareto fronts as a source of information (see Section 4.2). Four indicators, namely HV, GD, IGD, and ED, were computed for the PF_{approx} sets. Two indicators, which are ER and C, were computed based on PF_{known}.

Comparison of $\mathbf{PF}_{approx}$ sets. Table 3 presents the results for the performance indicators of HV, GD, IGD, and ED. To corroborate the analysis of these indicators, Figure 3 presents the boxplots. Regarding the individual behavior of each algorithm, the results of the Shapiro-Wilk test (second and third column in the table) present the distribution of the values computed for each performance indicator. Most of the values are normally distributed, except for IGD for NSGA-III. This strongly indicates the algorithms have a standard performance. In the qqplots of Figure 3 we can see that most of the data points are close to the reference line. For the exception case, IGD for NSGA-III, some points are distant from the reference (the right-side of Figure 3(h)). The values of the effect size measure are presented in the last two columns of the table, where we can observe that NSGA-III finds the best solutions in almost 100% of the runs for all indicators.

Regarding the comparison between the performance of NSGA-III (our approach) and RS, the fourth and fifth columns in Table 3 present the results of the Wilcoxon rank-sum test (based on the median) and Welch t-test (based on the mean). Both tests point out a significant difference between the two algorithms for all indicators. We can identify the best algorithm by observing the boxplots in Figure 3. On one hand, higher values of HV are the best

Table 3 Statistical Tests and Effect Size Measure Among PF_{approx} sets.

Indicator	**Shapiro-Wilk**		**Wilcoxon rank-sum**	**Welch t-test**	**$\hat{\mathbf{A}}_{12}$ Effect Size**	
	NSGA-III	**RS**			**NSGA-III**	**RS**
HV	0.88560	0.32040	2.20E-16	2.20E-16	1	0
GD	0.97780	0.73620	4.84E-13	7.00E-12	0.96667	0.03333
IGD	0.00018	0.43570	4.84E-13	4.15E-15	0.96667	0.03333
ED	0.37950	0.56310	2.20E-16	2.20E-16	1	0



(a) Boxplot HV (b) QQPlot HV NSGA-III (c) QQPlot HV RS

(d) Boxplot GD (e) QQPlot GD NSGA-III (f) QQPlot GD RS

(g) Boxplot IGD (h) QQPlot IGD NSGA-III (i) QQPlot IGD RS

(j) Boxplot ED (k) QQPlot ED NSGA-III (l) QQPlot ED RS

Fig. 3 Boxplots and QQPlots of the PF$_{approx}$ sets.

(Figure 3(a)); on the other hand, lower values GD, IGD, and ED are the best (Figures 3(d), 3(g), and 3(j)). NSGA-III has the best performance for these four indicators.

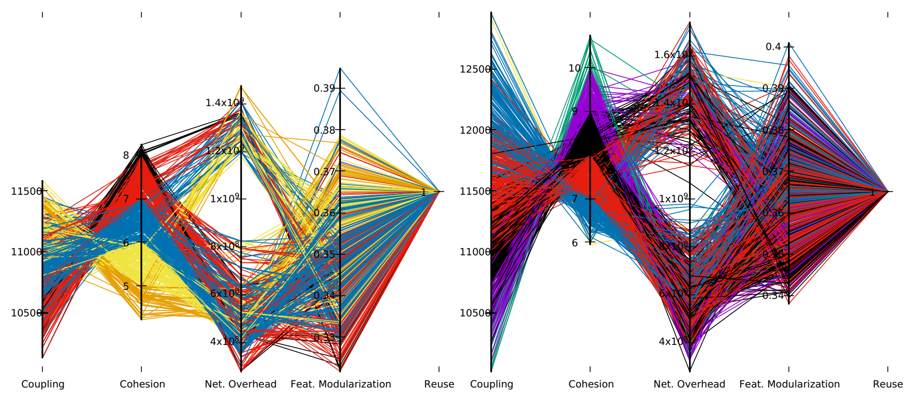
Comparison of PF$_{known}$ sets. Table 4 allows us to analyze the ability of the algorithms on finding the best solutions. ER considers the performance

Table 4 Error Ratio (ER) and Coverage (C) between PF_{known} sets.

Algorithm	**#PF_{true}**	**#PF_{known}**	**ER**	**C**
NSGA-III	554	545	1.63%	0.990044
RS		904	98.37%	0

on finding solutions in the PF_{true} and C compares the dominance between the PF_{known} sets. The PF_{true} set is composed of 554 solutions from which 545, which is the whole PF_{known} set, were found by NSGA-III; and 9 out of 904 found by RS. This huge difference in finding solutions in the PF_{true} is demonstrated by ER in the fourth column of Table 4. Regarding the paired comparison of the PF_{known} sets, C (fifth column in the table) indicates that the solutions of NSGA-III dominate 99.0044% of the RS solutions, which corresponds to 895 of 904 solutions. On the other hand, no solution of NSGA-III is dominated by solutions of RS. Hence, NSGA-III found the best solutions.

Trade-off visualization. Figure 4 demonstrates the trade-off among the criteria for both PF_{known} sets. At first, we can observe that the range of criteria values of solutions found by NSGA-III are better than solutions of RS. Our problem is a minimization and most of the NSGA-III solutions are in the lower parts of the figure. In both charts the conflicting interdependence between coupling and cohesion is clear. In addition, as discussed in the previous RQ, there is also some interdependence among the criteria of cohesion, feature modularization and network overhead as one can observe in the blue lines of Figure 4(a).



(a) NSGA-III (b) RS

Fig. 4 Solutions' trade-off among the five criteria of PF_{known} sets.

Answering RQ2: Both algorithms have a standard behavior as observed in the distribution of results. The comparison with the six performance indicators confirm NSGA-III has the best performance in comparison to RS.

6 Threats to Validity

In this section we discuss the main threats to the validity of our study.

Internal Validity. The internal validity of this study is threatened by the evolutionary algorithm adopted by `toMicroservices` and its parameter configuration. We adopted the state-of-the-art evolutionary algorithm NSGA-III [19] that has shown high accuracy to solve many-objective problems in the SBSE field. The algorithm parameters were set based on an existing work [38].Nonetheless, we acknowledge that the use of other parameter values may lead to different results. Still, conducting experiments with other algorithms and tuning parameters is an important next step, which is part of our future work.

The second threat is related to the set of microservice candidate solutions generated by NSGA-III. Due to the many-objective nature of NSGA-III, the solutions may converge to a different set of local optimum, *i.e.*, microservice candidates, in each run without finding the global optimum. To mitigate this threat, we executed 30 independent runs, as recommended in [2, 16, 17], and used performance indicators that analyze different solution sets, such as PF_{approx} and PF_{true}.

The generated solutions can also be subject to bias regarding the manual definition of the entry points for the feature mapping and the execution traces. The result is also dependent of the legacy system at hand, since all execution traces are generated using given test cases. To mitigate this threat in our study, the developer in charge of the microservice identification has sufficient knowledge of the system under analysis, including test cases, execution environment, and parameter settings. Thus, we believe that the test cases and defined entry points for the feature mapping cover the maximum number of business capabilities and thus can find suitable microservices.

Another limitation is the lack of an objective function to evaluate the database impact, which we intend to address in future work. Despite this limitation, the network overhead function – which measures the traffic of data between microservices – is able to partially capture such kind of data coupling softening this threat.

External Validity. A threat to external validity is related to the used case study. The obtained results cannot be generalized as our study is based on one industrial case study. Even though this system has the typical characteristics of any legacy code and is a consolidated real-world legacy system with more than 15 years of existence under the process of migration to microservices. We focused on a single system to be able at making an in-depth, robust analysis of the performance of our approach. Subsequently, we can perform such an analysis also over other systems by extending our current implementation to generalize our findings and improve our approach and its results.

7 Related Work

The problem of identifying microservice from legacy systems can be seen as a software remodularization problem [1, 5]. This problem have been widely investigated in the literature from different perspectives. For example, as a sequence of refactoring operations [61], by using structural and non-structural characteristics of the system under analysis [36], including expert knowledge in the process [32], and to organize feature of software product lines [50]. A critical analysis about the use of traditional criteria o coupling and cohesion for the remodularization is presented by Candela et al. [7]. Due the complexity of this problem, search-based approaches have been applied with satisfactory results [44, 48]. Despite all these pieces of work, next we focus on those ones in the scope of microservices.

A recent study mapped existing pieces of work focusing on migrating legacy systems to microservices [59]. The main result of this work is a roadmap process for conducting the migration. Among the activities that should be performed for the migration, the decomposition the legacy system for identifying the microservices is acknowledge as one of the most complex ones. Similarly, the study of Ponce et al. [52] reports a rapid review on 20 primary studies that performed the migration to microservices. These authors classified approaches for the identification of microservices in three groups: (i) *model-driven*, which rely on design elements as source of information, such as business objects, domain entities, functional and non-functional requirements, and data flow diagrams; (II) *static analysis*, approaches based on the source code that use the dependencies, couplings (static or evolutionary), and cohesion between source code entities for identifying the microservices; and (iii) *dynamic analysis*, approaches focusing on the analysis of the system functionalities at runtime, mainly using execution traces as source of information to group source code entities that will originate microservices. Fritzsch et al. [29] also describes categories of approaches for the decomposition of legacy systems into microservices. In addition to static, dynamic, and model-driven (named Meta-Data aided in this work), they included the (iv) *workload-data* category, in which approaches focus on finding suitable microservices cuts by measuring the application's operational data. Fritzsch et al. [29] also highlight that finding an appropriate granularity for the microservices can be seen as the main challenge during the migration. Yet, they mention about a decomposition around business capabilities, using bounded context [40].

Some studied are devoted to investigate the identification of microservices in practice. In a survey with practitioners that migrated legacy systems to microservices, Fritzsch et al. [30] reported that usually the identification of microservices is based functional decomposition. Less frequently, Domain-Driven Design was also observed among practitioners. Finally, some practitioners mentioned using non-systematic approaches. Surprisingly, some survey participants mentioned that they preferred rewrite the legacies using current technologies, because of the absence of a suitable decomposition approach. This reinforce the need of studies like ours presented in this paper. Another survey with experts was conducted by Carvalho et al. [10]. In this study, the authors investigated the criteria used for identifying microservice in legacy systems. The result of their study revealed that practitioners commonly consider at least four criteria simultaneously, which characterizes the microservices identification as a many-objective problem.

In order to identify common activities performed during the migration to microservices, Balalaie et al. [4] reported a set of migration patterns. Among the patterns, there are two related to the indentification of microservices: (i) using a domain-driven design to identify subdomains, which constitute a bounded context, of the business that the system is operating in, similarly to what was pointed by Fritzsch et al. [29]; and (ii) identifying microservices using a data ownership, by finding different cohesive sets of data entities that can be grouped together with their business logic into a microservice. However, these authors make it clear that a proper method for the identification depends on the context and characteristics of the legacy, which they refer to *situational method engineering* [34].

To deal with the complexity of the identification of microservices, same approaches uses search-based software engineering [16]. Existing search-based approaches for microservice identification apply evolutionary algorithms, such as genetic algorithms [38, 39, 62], to optimize some source code quality criteria extracted from execution traces, and thus, generate a set of microservice candidates. These solutions commonly make use of one or two criteria [38,39,45,62]. The two most conventional criteria adopted in the literature by automated approaches are coupling [38, 39, 45, 62] and cohesion [38, 39, 62]. The approach proposed in [62] also considers a non-functional criterion as their search-based approach is developed to optimize the identification of microservices with high-cohesion-low-coupling and load balance of CPU and memory consumption. In addition to the use of only a few criteria that does not entirely represent the practical needs of microservices identification, the performance of existing approaches is mostly evaluated on illustrative non-industrial systems.

To fill the gap existing in the literature and practice, `toMicroservices` uses five relevant in practice and well-defined criteria, namely coupling, cohesion, feature modularization, network overhead, and reuse [9, 10] (see criteria definition in Section 3.2). This implies that we turn the search-based microservice identification problem into a many-objective problem. To the best of our knowledge, there is no effort to solve such SBSE problems based on many objectives to formally represent the several criteria involved in the microservice identification. This new perspective represents an important challenge in this field. Yet, a recent exploratory study with practitioners has revealed that the efficiency of migrating legacy systems to microservice architectures depends on several criteria beyond coupling and cohesion [9,10] *e.g.*, the criterion network overhead is relevant given the architectural style of distributed components. In case of important criteria that are not considered by an approach, its solutions can hardly align with the desired benefits of microservices.

Another point in which our study is different from existing literature, is that work has a comparison of performance, that is, how the criteria are properly optimized by `toMicroservices` and random search. The performance comparison considers different criteria and search-based algorithms commonly used. Thus, our work has a complementary nature by conveying previous studies and further investigating additional criteria to better satisfy business needs. Furthermore, other recent studies, such as [31,41,43], have employed optimization algorithms to deal with problems related to microservice architectures, however their focus is not microservice identification as ours.

8 Conclusion

In this work, we quantitatively analyzed a many-objective optimization for the microservice identification problem in a two-fold perspective: (i) the correlation among the objective functions, and (ii) comparing the performance of NSGA-III against a RS. The experimental results pointed out that the problem deserves to be tackled as a many-objective problem because the criteria that are important to be considered from the developer's point of view are conflicting and interdependent. NSGA-III solved the referred problem more efficiently than a RS achieving a greater diversity of non-dominated solutions with different compromises among the objectives.

We now intend to improve our approach by investigating the application of crossover operators. We also plan to improve the reuse metric as it was not so sensitive to quantify the difference among the solutions regarding the solution's reuse degree. In addition, we evaluated an *extractive* approach for identifying microservices from the legacy system. However, we also plan to evaluate our approach to deal with the *proactive* approach, where a software system is designed from scratch as a set of microservices; and the *reactive* approach, in which the microservices are used to evolve a legacy system.

Acknowledgements This study was partially funded by CNPq grants 151723/2020-6, 428994/2018-0, 434969/2018-4, 312149/2016-6, 309844/2018-5, 421306/2018-1, and 408356/2018-9; CAPES grant 175956; FAPERJ grants 22520-7/2016, 010002285/2019, and PDR-10 Fellowship 202073/2020; FAPPR grants 51152 and 51435.

References

1. Anquetil, N., Lethbridge, T.C.: Experiments with clustering as a software remodularization method. In: 6th Working Conference on Reverse Engineering, pp. 235–255. IEEE (1999)

2. Arcuri, A., Briand, L.: A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Software Testing, Verification and Reliability **24**(3), 219–250 (2014)
3. Assunção, W.K.G., Colanzi, T.E., Carvalho, L., Pereira, J.A., Garcia, A., de Lima, M.J., Lucena, C.: A multi-criteria strategy for redesigning legacy features as microservices: An industrial case study. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 377–387 (2021). DOI 10.1109/SANER50967.2021.00042
4. Balalaie, A., Heydarnoori, A., Jamshidi, P., Tamburri, D.A., Lynn, T.: Microservices migration patterns. Software Practice and Experience **48**(11), 2019–2042 (2018)
5. Bavota, G., De Lucia, A., Marcus, A., Oliveto, R.: Software re-modularization based on structural and semantic metrics. In: 17th Working Conference on Reverse Engineering, pp. 195–204 (2010). DOI 10.1109/WCRE.2010.29
6. Bergmann, R., Ludbrook, J., Spooren, W.P.J.M.: Different Outcomes of the Wilcoxon-Mann-Whitney Test from Different Statistics Packages. The American Statistician **54**(1), 72–77 (2000)
7. Candela, I., Bavota, G., Russo, B., Oliveto, R.: Using cohesion and coupling for software remodularization: Is it enough? ACM Transactions on Software Engineering and Methodology **25**(3) (2016). DOI 10.1145/2928268
8. Capilla, R., Gallina, B., Cetina, C., Favaro, J.: Opportunities for software reuse in an uncertain world: From past to emerging trends. JSEP-ICSR'18-Special Issue **31**(8), 1–22 (2019). URL `http://www.es.mdh.se/publications/5550-`
9. Carvalho, L., Garcia, A., Assunção, W.K.G., Bonifácio, R., Tizzei, L.P., Colanzi, T.E.: Extraction of configurable and reusable microservices from legacy systems: An exploratory study. In: 23rd International Systems and Software Product Line Conference - Volume A, SPLC '19, pp. 26–31. ACM, New York, NY, USA (2019)
10. Carvalho, L., Garcia, A., Assunção, W.K.G., de Mello, R., de Lima, M.J.: Analysis of the criteria adopted in industry to extract microservices. In: Joint 7th Intl. Workshop on Conducting Empirical Studies in Industry and 6th Intl. Workshop on Software Engineering Research and Industrial Practice, CESSER-IP '19, pp. 22–29. IEEE Press, Piscataway, NJ, USA (2019)
11. Carvalho, L., Garcia, A., Colanzi, T.E., ao, W.K.G.A., Pereira, J.A., Fonseca, B., Ribeiro, M., de Lima, M.J., Lucena, C.: On the performance and adoption of search-based microservice identification with toMicroservices. In: IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE (2020). DOI 10.1109/icsme46990.2020.00060
12. Carvalho, L., Garcia, A., Colanzi, T.E., Assunção, W.K.G., Lima, M.J., Fonseca, B., Ribeiro, M.a., Lucena, C.: Search-based many-criteria identification of microservices from legacy systems. In: Genetic and Evolutionary Computation Conference Companion, GECCO '20, p. 305–306. ACM, New York, NY, USA (2020). DOI 10.1145/3377929.3390030
13. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Transactions on Software Engineering **20**(6), 476–493 (1994)
14. Cochrane, J., Zeleny, M.: Multiple Criteria Decision Making. University of South Carolina Press, Columbia (1973)
15. Coello, C.A.C., Lamont, G.B., Van Veldhuizen, D.A., et al.: Evolutionary algorithms for solving multi-objective problems. Springer Science & Business Media (2007)
16. Colanzi, T.E., Assunção, W.K.G., Farah, P.R., Vergilio, S.R., Guizzo, G.: A review of ten years of the symposium on search-based software engineering. In: Symposium on Search-Based Software Engineering (SSBSE), pp. 42–57 (2019)
17. Colanzi, T.E., Assunção, W.K.G., Vergilio, S.R., Farah, P.R., Guizzo, G.: The symposium on search-based software engineering: Past, present and future. Information and Software Technology **127**, 106372 (2020). DOI https://doi.org/10.1016/j.infsof.2020.106372
18. Curry, D.M., Dagli, C.H.: Computational complexity measures for many-objective optimization problems. Procedia Computer Science **36**, 185–191 (2014). DOI 10.1016/j.procs.2014.09.077

19. Deb, K., Jain, H.: An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. IEEE Transactions on Evolutionary Computation **18**(4), 577–601 (2014)
20. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002)
21. Di Francesco, P., Lago, P., Malavolta, I.: Architecting with microservices: A systematic mapping study. Journal of Systems and Software **150**, 77 – 97 (2019)
22. Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L.: Microservices: Yesterday, Today, and Tomorrow, pp. 195–216. Springer International Publishing, Cham (2017)
23. Durillo, J.J., Nebro, A.J.: jmetal: A java framework for multi-objective optimization. Advances in Engineering Software **42**(10), 760–771 (2011)
24. Escobar, D., Cárdenas, D., Amarillo, R., Castro, E., Garcés, K., Parra, C., Casallas, R.: Towards the understanding and evolution of monolithic applications as microservices. In: Latin American Computing Conference, pp. 1–11 (2016)
25. Eski, S., Buzluca, F.: An automatic extraction approach: Transition to microservices architecture from monolithic application. In: 19th International Conference on Agile Software Development: Companion, XP '18, pp. 25:1–25:6 (2018)
26. Fonseca, C.M., Paquete, L., Lopez-Ibanez, M.: An improved dimension-sweep algorithm for the hypervolume indicator. In: IEEE International Conference on Evolutionary Computation, pp. 1157–1163 (2006)
27. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley Longman Publishing Co., Inc., USA (1999)
28. Francesco, P.D., Lago, P., Malavolta, I.: Migrating towards microservice architectures: An industrial survey. In: International Conference on Software Architecture (ICSA), pp. 29:01–29:09 (2018)
29. Fritzsch, J., Bogner, J., Zimmermann, A., Wagner, S.: From monolith to microservices: a classification of refactoring approaches. In: International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment, pp. 128–141. Springer (2018)
30. Fritzsch, J., Bogner, J., Zimmermann, A., Wagner, S.: From monolith to microservices: A classification of refactoring approaches. In: J.M. Bruel, M. Mazzara, B. Meyer (eds.) Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment, pp. 128–141. Springer International Publishing, Cham (2019)
31. Gao, M., Chen, M., Liu, A., Ip, W.H., Yung, K.L.: Optimization of microservice composition based on artificial immune algorithm considering fuzziness and user preference. IEEE Access **8**, 26385–26404 (2020)
32. Hall, M., Walkinshaw, N., McMinn, P.: Effectively incorporating expert knowledge in automated software remodularisation. IEEE Transactions on Software Engineering **44**(7), 613–630 (2018). DOI 10.1109/TSE.2017.2786222
33. Harman, M., Tratt, L.: Pareto optimal search based refactoring at the design level. In: 9th Annual Conf. on Genetic and Evolutionary Computation (GECCO), pp. 1106–1113. ACM, New York, NY, USA (2007)
34. Henderson-Sellers, B., Ralyté, J., Ågerfalk, P.J., Rossi, M.: Situational method engineering. Springer (2014). DOI 10.1007/978-3-642-41467-1
35. Hinkle D.E. Wiersma W, J.S.: Applied Statistics for the Behavioral Sciences, 5th edn. Cengage Learning (2002)
36. Jalali, N.S., Izadkhah, H., Lotfi, S.: Multi-objective search-based software modularization: structural and non-structural features. Soft Computing **23**(21), 11141–11165 (2018). DOI 10.1007/s00500-018-3666-z
37. Jamil, M.A., Alhindi, A., Arif, M., Nour, M.K., Abubakar, N.S.A., Aljabri, T.F.: Multi-objective evolutionary algorithms nsga-ii and nsga-iii for software product lines testing optimization. In: 2019 IEEE 6th International Conference on Engineering Technologies and Applied Sciences (ICETAS), pp. 1–5 (2019). DOI 10.1109/ICETAS48360.2019.9117500

38. Jin, W., Liu, T., Cai, Y., Kazman, R., Mo, R., Zheng, Q.: Service candidate identification from monolithic systems based on execution traces. IEEE Transactions on Software Engineering pp. 1–1 (2019)
39. Jin, W., Liu, T., Zheng, Q., Cui, D., Cai, Y.: Functionality-oriented microservice extraction based on execution trace clustering. In: International Conference on Web Services (ICWS), pp. 211–218. IEEE (2018)
40. Lewis, J., Fowler, M.: Microservices: a definition of this new architectural term. `https://martinfowler.com/articles/microservices.html` (2014). Online
41. Lin, M., Xi, J., Bai, W., Wu, J.: Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud. IEEE Access **7**, 83088–83100 (2019)
42. Luz, W., Agilar, E., de Oliveira, M.C., de Melo, C.E.R., Pinto, G., Bonifácio, R.: An experience report on the adoption of microservices in three Brazilian government institutions. In: XXXII Brazilian Symposium on Software Engineering, pp. 32–41. ACM, New York, NY, USA (2018)
43. Ma, W., Wang, R., Gu, Y., Meng, Q., Huang, H., Deng, S., Wu, Y.: Multi-objective microservice deployment optimization via a knowledge-driven evolutionary algorithm. Complex & Intelligent Systems **7**, 1153–1171 (2021). DOI https://doi.org/10.1007/s40747-020-00180-1
44. Mahouachi, R.: Search-based cost-effective software remodularization. Journal of Computer Science and Technology **33**(6), 1320–1336 (2018). DOI 10.1007/s11390-018-1892-6
45. Mazlami, G., Cito, J., Leitner, P.: Extraction of microservices from monolithic software architectures. In: International Conference on Web Services (ICWS), pp. 524–531 (2017)
46. Mitchell, B.S., Mancoridis, S.: On the automatic modularization of software systems using the bunch tool. IEEE Transactions on Software Engineering **32**(3), 193–208 (2006)
47. Mkaouer, M.W., Kessentini, M., Bechikh, S., Deb, K., Ó Cinnéide, M.: High dimensional search-based software engineering: Finding tradeoffs among 15 objectives for automating software refactoring using nsga-iii. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, GECCO '14, p. 1263–1270. ACM, New York, NY, USA (2014). DOI 10.1145/2576768.2598366. URL `https://doi.org/10.1145/2576768.2598366`
48. Mkaouer, W., Kessentini, M., Shaout, A., Koligheu, P., Bechikh, S., Deb, K., Ouni, A.: Many-objective software remodularization using NSGA-III. ACM Trans. Softw. Eng. Methodol. **24**(3) (2015). DOI 10.1145/2729974
49. Newman, S.: Building microservices: designing fine-grained systems. O'Reilly Media, Inc. (2015)
50. Nicolodi, L.B., Colanzi, T.E., Assunção, W.K.G.: Architectural feature remodularization for software product line evolution. In: 14th Brazilian Symposium on Software Components, Architectures, and Reuse, SBCARS '20, pp. 31–40. Association for Computing Machinery, New York, NY, USA (2020). DOI 10.1145/3425269.3425271. URL `https://doi.org/10.1145/3425269.3425271`
51. Palakonda, V., Mallipeddi, R.: An evolutionary algorithm for multi and many-objective optimization with adaptive mating and environmental selection. IEEE Access **8**, 82781–82796 (2020). DOI 10.1109/ACCESS.2020.2991752
52. Ponce, F., Márquez, G., Astudillo, H.: Migrating from monolithic architecture to microservices: A rapid review. In: 38th International Conference of the Chilean Computer Science Society (SCCC), pp. 1–7. IEEE (2019)
53. Radziukyniene, I., Zilinskas, A.: Evolutionary Methods for Multi-Objective Portfolio Optimization. In: World Congress on Engineering 2008 Vol II (2008)
54. Shapiro, S.S., Wilk, M.B.: An analysis of variance test for normality (complete samples). Biometrika **52**(3-4), 591–611 (1965)
55. Spearman, C.: The proof and measurement of association between two things. The American Journal of Psychology **15**(1), 72–101 (1904)
56. Van Veldhuizen, D.A.: Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Tech. rep., Air Force Institute Of Technology, Wright-Patterson Air Force Base. School of Engineering (1999)
57. Vargha, A., Delaney, H.: A critique and improvement of the cl common language effect size statistics of McGraw and Wong. Journal of Educational and Behavioral Statistics **25**(2), 101–132 (2000)

58. Welch, B.L.: The generalization of student's problem when several different population variances are involved. Biometrika **34**(1/2), 28–35 (1947)
59. Wolfart, D., Assunção, W.K.G., da Silva, I.F., Domingos, D.C.P., Schmeing, E., Villaca, G.L.D., Paza, D.d.N.: Modernizing legacy systems with microservices: A roadmap. In: Evaluation and Assessment in Software Engineering, EASE 2021, p. 149–159. ACM, New York, NY, USA (2021). DOI 10.1145/3463274.3463334
60. Yen, G.G., He, Z.: Performance metric ensemble for multiobjective evolutionary algorithms. IEEE Transactions on Evolutionary Computation **18**(1), 131–144 (2013)
61. Zanetti, M.S., Tessone, C.J., Scholtes, I., Schweitzer, F.: Automated software remodularization based on move refactoring: A complex systems approach. In: 13th International Conference on Modularity, MODULARITY '14, p. 73–84. ACM, New York, NY, USA (2014). DOI 10.1145/2577080.2577097
62. Zhang, Y., Liu, B., Dai, L., Chen, K., Cao, X.: Automated microservice identification in legacy systems with functional and non-functional metrics. In: 2020 IEEE International Conference on Software Architecture (ICSA), pp. 135–145 (2020)
63. Zitzler, E., Laumanns, M., Thiele, L., et al.: Spea2: Improving the strength pareto evolutionary algorithm. In: Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems (EUROGEN), pp. 95–100. International Center for Numerical Methods in Engineering (2001)
64. Zitzler, E., Thiele, L.: Multiobjective optimization using evolutionary algorithms – a comparative case study. In: A.E. Eiben, T. Bäck, M. Schoenauer, H.P. Schwefel (eds.) Parallel Problem Solving from Nature – PPSN V, pp. 292–301. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
65. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., da Fonseca, V.G.: Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on Evolutionary Computation **7**, 117–132 (2003)