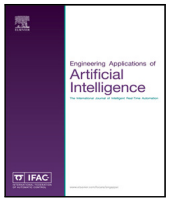
Engineering Applications of Artificial Intelligence 133 (2024) 108593

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai



Research paper

Multi-objective general variable neighborhood search for software maintainability optimization

Javier Yuste [a], Eduardo G. Pardo [a,*], Abraham Duarte [a], Jin-Kao Hao [b]

[a] Universidad Rey Juan Carlos, C/Tulipán s/n, Móstoles, 28933, Madrid, Spain
[b] LERIA, Université d'Angers, 2 Boulevard Lavoisier, Angers, 49045, France

ARTICLE INFO

Keywords:
Software maintainability
Search-based software engineering
Software module clustering
Heuristics
Multi-objective optimization

ABSTRACT

The quality of software projects is measured by different attributes such as efficiency, security, robustness, or understandability, among others. In this paper, we focus on maintainability by studying the optimization of software modularity, which is one of the most important aspects in this regard. Specifically, we study two well-known and closely related multi-objective optimization problems: the Equal-size Cluster Approach Problem (ECA) and the Maximizing Cluster Approach Problem (MCA). Each of these two problems looks for the optimization of several conflicting and desirable objectives in terms of modularity. To this end, we propose a method based on the Multi-Objective Variable Neighborhood Search (MO-VNS) methodology in combination with a constructive procedure based on Path-Relinking. As far as we know, this is the first time that a method based on MO-VNS is proposed for the MCA and ECA problems. To enhance the performance of the proposed algorithm, we present three advanced strategies: an incremental evaluation of the objective functions, an efficient exploration of promising areas in the search space, and an analysis of the objectives that better serve as guiding functions during the search phase. Our proposal has been validated by experimentally comparing the performance of our algorithm with the best previous state-of-the-art method for the problem and three reference methods for multi-objective optimization. The experiments have been performed on a set of 124 real software instances previously reported in the literature.

1. Introduction

The life cycle of a software project is initiated with a user need and concluded by the discontinued use of a product (International Organization for Standardization, 2017). To cover all the stages within it, the development of software projects is structured in a Software Development Life Cycle (SDLC). An SDLC involves the definition, ordering, and transition criteria of several phases, such as requirements definition, analysis, design, implementation, validation, and maintenance, among others. The main purpose of using an SDLC is to enhance the quality of the resulting software product. In this regard, the quality of a software project might be measured in terms of efficiency, security, understandability, robustness, cost, usability, or maintainability, among others. Maintainability and understandability are important aspects for the long-term success of software projects since they affect how easily a software system can be modified, corrected, improved, or adapted.

As software projects grow over time, their architecture often tends to deteriorate, creating technical debt (Li et al., 2015). Then, in the maintenance phase, several activities are required to provide support to the system by correcting bugs, improving its performance, adapting the product, etc. (Bakota et al., 2012). The maintenance phase itself is often responsible for up to more than 80% of the total costs (Chen et al., 2017). Interestingly, most work in this phase is dedicated to comprehend the existing software (Molnar and Motogna, 2021). A lack of comprehensive understanding of the code base makes its maintenance and correction costly and prone to errors. Therefore, facilitating the comprehension of a software project is highly beneficial in reducing its associated costs and improving its code quality. In addition, given the long-term benefits of having maintainable software, addressing these issues as soon as possible within the SDLC is highly recommended.

To simplify the comprehension of software projects, the source code is generally split in multiple parts so that each part or component can be understood separately. In addition, these components are traditionally organized into modules which ideally group closely related components. A good organization of these components follows the modularity principle and makes it easier to understand each component individually.

* Corresponding author.
E-mail addresses: javier.yuste@urjc.es (J. Yuste), eduardo.pardo@urjc.es (E.G. Pardo), abraham.duarte@urjc.es (A. Duarte), jin-kao.hao@univ-angers.fr (J.-K. Hao).

https://doi.org/10.1016/j.engappai.2024.108593
Received 10 October 2023; Received in revised form 22 March 2024; Accepted 6 May 2024
Available online 16 May 2024
0952-1976/© 2024 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

As defined by the International Organization for Standardization (2017), modularity refers to the *"software attributes that provide a structure of highly independent components"*. In a desirable modular structure, components within the same module should be closely connected among them (high cohesion) and weakly connected to the components in other modules (low coupling). There are different notions of what a software component is, such as a file, a class, a package, etc. Although in some contexts the words "module" and "component" are used interchangeably, here we will utilize the word "component" for individual elements (files and classes) and the word "module" for collections of components (packages or folders).

To optimize the quality of software projects, several tasks of software engineering are addressed as optimization problems in the Search-Based Software Engineering (SBSE) research area (Chaves-González et al., 2015; Moosavi and Bardsiri, 2017; Colanzi et al., 2020; Ramirez et al., 2019). Among the variety of issues within SBSE, the Software Module Clustering Problem (SMCP) is an optimization problem whose aim is to reduce the development costs of software projects by maximizing their modularity (Yuste et al., 2022c). To tackle this as an optimization problem, two main features are needed: a standardized representation of the possible solutions to the problem; and a method to compare these solutions and determine which one is better. Then, optimization, "the search process that seeks for the best possible solution to an optimization problem" (Duarte et al., 2007), can be performed. In the case of the SMCP, then, the objective is to reorganize the structure of software projects to increase their modularity.

For the majority of optimization problems of practical interest, there exist such a vast number of feasible solutions that exploring all of them is not affordable in reasonable computing times. In these cases, a procedure is needed to find good solutions in a limited time. These procedures, known as heuristics or metaheuristics (higher-level heuristics), often implement some sort of stochastic optimization and explore a subset of all possible solutions in a smart way. The SMCP is not an exception in this sense, as it is $\mathcal{NP}$-complete (Brandes et al., 2007).

The primary goal of SMCP problems, to maximize cohesion and minimize coupling, does not appear to be successful in practice (Barros et al., 2015). The best structure of any software project if our objective were to minimize coupling and maximize cohesion would be to put all components into a single module. In such a trivial solution, cohesion would be maximum (all dependencies would occur inside the same module) and coupling would be minimum (there would not be any dependency between components belonging to different modules). However, that trivial solution is not useful for maintainability purposes, since it would be as useful as not grouping the components into modules at all. In practice, a larger number of modules is usually preferred. Nevertheless, maximizing the number of modules would not reduce the effort required to understand the system. Instead, it would result in another trivial solution, where every component is located in a different module. Therefore, a balance must be found when considering coupling, cohesion, and the number of modules. Moreover, there exist other objectives that should also be considered in this context. For instance, it is generally undesirable to have isolated modules with one single component. Instead, a homogeneous distribution of components in modules is preferred, where the number of components in each module is roughly equal. Due to these reasons, it appears that a multi-objective strategy is more suitable for the SMCP, because (i) the consideration of different conflicting objectives is a more accurate reflection of the modularity of the system than the consideration of each objective in isolation; and (ii) providing a collection of modular organizations to a decision maker (e.g., a software developer) allows the stakeholders to prioritize some objectives over others depending on the context. Following these ideas, Praditwong et al. (2011) introduced two different multi-objective formulations for the SMCP: the Equal-size Cluster Approach (ECA) and the Maximizing Cluster Approach (MCA).

In recent years, there have been significant improvements in the results obtained on single-objective SMCP problems achieved by trajectory-based metaheuristics, mainly based on a deep exploration of neighborhoods (Pinto et al., 2014; Monçores et al., 2018; Yuste et al., 2022a, 2024). Despite the success of these methodologies, they have not been tested for multi-objective SMCP problems. Therefore, trajectory-based methods, such as MO-VNS, seem promising for multi-objective optimization SMCP problems, as it is the case of MCA and ECA.

In this work, we present a Multi-Objective General Variable Neighborhood Search (MO-GVNS) method for both the MCA and ECA problems. To enhance the performance of the method, we present three advanced strategies: an efficient evaluation of the solutions, an efficient exploration of promising areas in the neighborhoods proposed, and an analysis of the importance of the objectives contained in each problem as guiding functions during the search process. Moreover, a constructive procedure based on Path-Relinking is presented, and several criteria for the shake component of the method are proposed. Finally, we perform a comparison of the proposed method with several state-of-the-art approaches: a Two-Archive Artificial Bee Colony (TA-ABC) recently proposed to tackle both MCA and ECA (Amarjeet and Chhabra, 2018); the Non-dominated Sorting Genetic Algorithm III (NSGA-III) (Deb and Jain, 2013); the Modified Pareto Envelop-Based Selection Algorithm (PESA2) (Corne et al., 2001); and the Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) (Zhang and Li, 2007). The results show that the presented MO-GVNS method obtains better results in all the quality aspects evaluated: convergence, spread, uniformity, and cardinality.

The rest of this article is organized as follows. In Section 2, we present a literature review. In Section 3, we define the MCA and ECA problems. In Section 4, we describe the algorithm proposed. In Section 5, we propose several advanced strategies designed to enhance the efficiency of the method. In Section 6, we perform some preliminary experiments and evaluate the performance of the proposed algorithm. Finally, we draw some conclusions and future work in Section 7.

2. Literature review

Software engineers have traditionally organized software projects in modules based on their own experience in a subjective way. Thus, this process ends up being not necessarily systematic and repeatable (Barros et al., 2015). Alternatively, many researchers have suggested to tackle this task as an optimization problem, known as SMCP. Indeed, many variants of the same problem can be tracked due to the wide variety of quality measures (objective functions) used to evaluate the modularity of a software project. In Table 1, we present a chronologically sorted compilation of the main works addressing a variant of the SMCP. For each work, we report if a single-objective (S.O.) or multi-objective (M.O.) approach was used, the objective function or set of objective functions studied (O.F.), and the algorithm proposed (Algorithm). As can be observed in Table 1, Modularization Quality (MQ) is one of the first and most studied objective functions (Mancoridis et al., 1998). In MQ, the quality of a solution is computed as a trade-off between cohesion and coupling. This metric was later extended to TurboMQ, in order to accelerate its computation (Mitchell, 2002). Although MQ is one of the most widely used objective functions for the SMCP, some concerns have been highlighted in the literature about these metrics and the prevalence of the coupling-cohesion paradigm (Barros et al., 2015; Izadkhah and Tajgardan, 2019). As a response, alternative objective functions were proposed, such as: Modularization Quality measure based on similarity (MS) (Huang and Liu, 2016), Entropy-based Objective Function (EOF) (Izadkhah and Tajgardan, 2019), or Function of Complexity Balance (FCB) (Mu et al., 2020). Specifically, the latter tries to decrease over-cohesiveness and decrease the number of modules with only one isolated vertex (Mu et al., 2020).

Apart from the various individual modularity metrics that exist in the literature, a natural alternative to using only one of them is

Table 1
Chronological summary of previous proposals for the SMCP.

Reference	Type	O.F.	Algorithm
Mancoridis et al. (1998)	SO	MQ	HC+GA
Mitchell and Mancoridis (2002)	SO	MQ	HC+SA
Mahdavi (2005)	SO	MQ	HCGA
Mitchell and Mancoridis (2006)	SO	MQ	HCGA, HCSA
Mitchell and Mancoridis (2008)	SO	MQ	NAHC, SAHC
Mamaghani and Meybodi (2009)	SO	MQ	HGA
Abdeen et al. (2009)	SO	DQCQ	SA
Praditwong (2011)	SO	MQ	GA
Praditwong et al. (2011)	MO	MCA, ECA	GA
Mamaghani and Hajizadeh (2014)	SO	MQ	FA
Pinto et al. (2014)	SO	MQ	ILS
Mkaouer et al. (2015)	MO	SSH	NSGA-III
Izadkhah et al. (2016)	SO	MQ	E-CDGM
Jeet and Dhir (2016)	SO	MQ	ICA-GA
Tajgardan et al. (2016)	SO	MQ	EDA
Kumari and Srinivas (2016)	MO	MCA, ECA	MHypEA
Huang and Liu (2016)	SO	MS	HC, GA, MAEA
Huang et al. (2017)	SO	MQ	MAEA
Kargar et al. (2017)	SO	MQ	HCS
Amarjeet and Chhabra (2017)	SO	CCCS	HS
Hwa et al. (2017)	MO	MFMC	HC
Monçores et al. (2018)	SO	MQ	LNS
Prajapati and Chhabra (2018)	SO	MNCC	PSO
Ramirez et al. (2018)	MO	IFF	IEA
Amarjeet and Chhabra (2018)	MO	MCA, ECA	TA-ABC
Jalali et al. (2019)	MO	MOF	EoD
Izadkhah and Tajgardan (2019)	SO	EOF	GA
Chhabra (2018b)	MO	E-MCA, E-ECA	MaABC
Mu et al. (2020)	SO	FCB	HGA
Pourasghar et al. (2021)	SO	LCC	GMA
Yuste et al. (2022a)	SO	MQ	GRASP-VND
Arasteh et al. (2022)	MO	MCA, ECA	PESA
Prajapati (2022)	MO	E-MCA, E-ECA	GLMPSO
Yuste et al. (2022c)	SO	FCB	VND
Arasteh (2023)	SO	MQ	Chaos-based
Arasteh et al. (2023)	SO	MQ	Olympiad
Yuste et al. (2024)	SO	FCB	GVNS

Type: MO: Multi-Objective; SO: Single-Objective. **Objective Function (O.F.):** CCCS: Cohesion, Coupling, package Count index, and package Size index; DQCQ: Dependency Quality and Connection Quality; E-ECA: Extended Equal-size Cluster Approach; E-MCA: Extended Maximizing Cluster Approach; ECA: Equal-size Cluster Approach; EOF: Entropy-based Objective Function; FCB: Functions of Complexity Balance; IFF: Interactive Fitness Function; LCC: Linear Compound Criteria; MCA: Maximizing Cluster Approach; MFMC: Multi-Factor Module Clustering; MNCC: MQ, Non-extreme distribution, Coupling, and Cohesion; MOF: Multi-Objective Fitness function; MQ: Modularization Quality; MS: Similarity-based Modularity Quality; SSH: Structure of packages, Semantics coherence, and History of changes. **Algorithm:** E-CDGM: Evolutionary Call-Dependency Graph Modularization; EDA: Estimation of Distribution based Approach; EoD: Estimation of Distribution; FA: Firefly Algorithm; GA: Genetic Algorithm; GLMPSO: Grid-based Large-scale Many-objective Particle Swarm Optimization; GMA: Graph-based Modularization Algorithm; GRASP: Greedy Randomized Adaptive Search Procedure; GVNS: General Variable Neighborhood Search; HC: Hill Climbing; HCS: Hill Climbing based on Semantic dependency graph; HGA: Hybrid Genetic Algorithm; HS: Harmony Search; ICA: Imperialist Competitive Algorithm; IEA: Interactive Evolutionary Approach; ILS: Iterated Local Search; LNS: Large Neighborhood Search; MaABC: Many-objective Artificial Bee Colony; MAEA: Multi-Agent Evolutionary Algorithm; MHypEA: Multi-objective Hyper-heuristic Evolutionary Algorithm; NSGA-III: Non-dominated Sorting Genetic Algorithm III; PESA: Pareto Envelop-based Selection Algorithm; PSO: Particle Swarm Optimization; TA-ABC: Two-Archive Artificial Bee Colony; SA: Simulated Annealing; VND: Variable Neighborhood Descent.

to use multiple objectives in conflict in a multi-objective approach, representing the modularity of the system in a diverse way. Some works in the literature have highlighted the need for formulations that do not oversimplify the SMCP problems into a single metric that encapsulates different measures of coupling, cohesion, number of modules, etc., which often lead to class-to-package distributions having a very large number of packages and fewer classes per package (Barros et al., 2015; Mu et al., 2020). Furthermore, providing a collection of different solutions that conform a Pareto front (i.e., a set of non-dominated solutions with respect to different metrics) instead of just one, allows a decision maker to determine the most suitable solution for a particular context. In this sense, a software developer might be interested in solutions that are very good with respect to a given objective or in solutions that present a good balance between different desirable properties, introducing their preferences in the process. Due to these reasons, research in the SMCP has shifted towards multi-objective optimization problems, and different proposals can be found in the literature. To our knowledge, Abdeen et al. (2009) were the first to introduce a multi-objective optimization variant for the SMCP, where several metrics are used to measure the modularization quality of the entire architecture and each individual package within the organization of software projects. Later, Praditwong et al. (2011) proposed two of the most studied multi-objective problems in the area: the Equal-size Cluster Approach (ECA) and the Maximizing Cluster Approach (MCA). A total of six different objective functions were proposed, and each formulation uses five of them. Similarly, Mkaouer et al. (2015) and Hwa et al. (2017) proposed distinct multi-objective formulations, where an effort was made to simultaneously include metrics considering: the structure of the code, its semantics, as well as the history of changes of the software. Then, Ramirez et al. (2018) proposed an interactive multi-objective approach, where the preference of the software developer is incorporated into the search process. Among the different multi-objective proposals compiled in Table 1, MCA and ECA can be identified as the most studied multi-objective optimization problems in the SMCP literature. Moreover, they have recently been extended by Chhabra (2018b), which included two additional metrics. The resulting approaches were named Extended Maximizing Cluster Approach and Extended Equal-size Cluster Approach. Given the interest received by the scientific community (Praditwong et al., 2011; Kumari and Srinivas, 2016; Amarjeet and Chhabra, 2018; Arasteh et al., 2022) it is worth exploring the design of new search-based optimization algorithms for the MCA and ECA problems.

Regardless of the variant studied, in Table 1 we also summarize the algorithmic approach used to tackle each variant of the SMCP. Because the SMCP is known to be $\mathcal{NP}$-complete (Brandes et al., 2007), approximate search-based methods are better suited than exact methods to solve it (Harman et al., 2012; Ramirez et al., 2019). The objective of approximate search-based methods is to provide high-quality solutions (not necessarily optimal) in short computing times by introducing some intelligence into the search process. Among them, evolutionary approaches became prominent in the literature (Kumari and Srinivas, 2016; Huang and Liu, 2016; Huang et al., 2017; Prajapati and Chhabra, 2018; Arasteh et al., 2022; Prajapati, 2022; Arasteh et al., 2023; Arasteh, 2023). However, it is also possible to find trajectory-based approaches. From a historical perspective, the use of Hill Climbing (HC), often combined with Genetic Algorithms (GA) or Simulated Annealing (SA), was popular in the early years. However, a slow shift towards the use of GA and SA approaches without the HC component can be observed later in the literature (Abdeen et al., 2009; Praditwong, 2011; Praditwong et al., 2011). Recently, the use of trajectory-based metaheuristics for the SMCP has emerged as an effective alternative to the classic proposals (Pinto et al., 2014; Monçores et al., 2018; Yuste et al., 2022a, 2024, 2022b). Due to the recent improvements achieved by metaheuristics based on the exploration of neighborhood structures, the adoption of a method based on the Variable Neighborhood Search (VNS) (Mladenović and Hansen, 1997) methodology for the multi-objective problems MCA and ECA seems promising.

3. Problem definition

In order to address the SMCP, software projects need to be modeled into a graph structure, called the Module Dependency Graph (MDG). In this structure, vertices represent components, and edges represent dependencies between components. In addition, edges might be weighted depending on the strength of the dependency between components. Formally, an MDG is represented as an undirected graph with weighted

edges $G = (V, E, W)$, composed of a set of vertices (V), a set of edges (E), and a set of weights (W).

Given the previous representation of software projects, a solution for the SMCP is represented by a clustering of the vertices of the graph. Therefore, a solution is a set $M = \{m_1, m_2, \ldots, m_n\}$ of disjoint subsets of vertices in V, where n ($1 \leq n \leq |V|$) represents the number of modules (or clusters, groups). Following previous works (Sarhan et al., 2020), in this context a component of a software system is a class or a file. Therefore, a solution represents a grouping of classes/files into different packages.

We study two multi-objective optimization problems to evaluate the quality of the solutions to the SMCP, the Equal-size Cluster Approach (ECA) and the Maximizing Cluster Approach (MCA), proposed in Praditwong et al. (2011). These problems are based on the optimization of different conflicting objectives: cohesion, to be maximized (objective 1); coupling, to be minimized (objective 2); the number of modules, to be maximized (objective 3); MQ, to be maximized (objective 4); the number of isolated modules, to be minimized (objective 5); and the difference between the maximum and minimum number of vertices in a module, to be minimized (objective 6). In particular, MCA considers objectives 1, 2, 3, 4, and 5, while ECA considers objectives 1, 2, 3, 4, and 6. As can be noted, both problems share four of the five objectives considered. Therefore, they are usually studied together.

The first objective (cohesion) is computed as the sum of weights associated with the edges that link vertices that belong to the same module. The set of edges connecting vertices within the same module is also known as the set of intracluster edges, formally defined as:

$$Intra(m_i) = \{\{u, v\} \in E : u, v \in m_i\}. \tag{1}$$

Then, the sum of the weights of the edges that connect vertices that belong to the same module m_i is formally defined as:

$$\mu_i = \sum_{(u,v) \in Intra(m_i)} w(u, v). \tag{2}$$

Finally, the cohesion value of a solution is formally defined as:

$$\text{Cohesion} = \sum_{i=1}^{n} \mu_i. \tag{3}$$

In contrast to cohesion, the second objective of the MCA and ECA problems, coupling, is computed as the sum of weights associated with the intercluster edges, i.e., edges that link vertices that belong to different modules. Given two modules m_i and m_j, the set of intercluster edges between them is formally defined as:

$$Inter(m_i, m_j) = \{(u, v) \in E : u \in m_i \wedge v \in m_j \bigcup (u, v) \in E : u \in m_j \wedge v \in m_i\}. \tag{4}$$

Then, the coupling value of a solution is formally defined as:

$$\text{Coupling} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{(u,v) \in Inter(m_i, m_j)} w(u, v). \tag{5}$$

Trivially, the third objective (number of modules) is equal to the number of subsets of vertices (n). To evaluate the fourth objective (MQ), it is necessary to firstly calculate the Cluster Factor (CF_i) of each module m_i as follows:

$$CF_i = \begin{cases} 0, & \text{if } \mu_i = 0, \\ \frac{2\mu_i}{2\mu_i + \varepsilon_i}, & \text{otherwise} \end{cases} \tag{6}$$

where μ_i is the cohesion of module m_i, as described in Eq. (2), and ε_i is the coupling of module m_i (the sum of the weights of the edges that connect vertices belonging to m_i with vertices belonging to other modules), formally defined as:

$$\varepsilon_i = \sum_{m_j \in M \setminus m_i} \sum_{(u,v) \in Inter(m_i, m_j)} w(u, v). \tag{7}$$

Then, given a solution x, the MQ value is equal to the sum of the values of CF for every module:

$$MQ(x) = \sum_{i=1}^{n} CF_i. \tag{8}$$

The objective function presented in Eq. (8) is known as TurboMQ (Mitchell and Mancoridis, 2002), an extension of the original MQ function presented by Mancoridis et al. (1998). Finally, the size of a module is equal to the number of vertices contained in it. Thus, an isolated module is a module composed of one vertex (objective 5), and the size difference between two modules m_i and m_j is calculated as the absolute value of the number of vertices in m_i minus the number of vertices in m_j (objective 6).

In Fig. 1, we illustrate an example of a solution x for a software including 8 components. We also compile the values obtained after the evaluation of the aforementioned objectives within the ECA/MCA problems. As can be seen, the number of modules of the solution is equal to 4 (m_1, m_2, m_3, and m_4). The solution has a cohesion value of 4, since there are four edges (1–2, 1–3, 4–5, and 7–8) connecting vertices within the same module. Accordingly, coupling is equal to 5, since there are five edges (2–4, 3–6, 4–6, 5–8, and 6–7) connecting vertices belonging to different modules. The evaluation of MQ requires the evaluation of CF for each module first. The CF value for m_1 is calculated as $CF_1 = \frac{2 \cdot 2}{2 \cdot 2 + 2} = 0.67$, since there are 2 internal edges ($\mu_i = 2$) and 2 external edges ($\varepsilon_i = 2$). Similarly, the CF values for the rest of the modules are calculated as: $CF_2 = \frac{2 \cdot 1}{2 \cdot 1 + 3} = 0.4$, $CF_3 = 0.00$, and $CF_4 = \frac{2 \cdot 1}{2 \cdot 1 + 2} = 0.5$. Therefore, the MQ value is obtained as $MQ = CF_1 + CF_2 + CF_3 + CF_4 = 1.57$. Finally, there is 1 isolated module in the solution (m_3) and the size difference between the largest module (m_1) and the smallest module (m_3) is equal to 2.

4. Algorithmic proposal

In this work, we propose an algorithm based on the well-known Variable Neighborhood Search (VNS) framework (Mladenović and Hansen, 1997) which has been used in a wide variety of applications (Perez-Pelo et al., 2021; Cavero et al., 2022; Lai and Hao, 2016; Shi et al., 2023). Furthermore, given the improvements achieved in recent years for SMCP problems by algorithms based on the exploration of neighborhood structures, a VNS-based method for MCA and ECA problems is suitable. One of the main reasons behind this assumption is the different nature of the objectives studied within MCA/ECA. Thus, it is of interest to use a methodology able to successfully combine the exploration of different neighborhood structures. Therefore, we propose an algorithm based on the Multi-Objective Variable Neighborhood Search (MO-VNS) methodology (Duarte et al., 2015), which extends VNS to handle multi-objective optimization problems. As far as we know, this is the first time that a MO-VNS method is proposed for a problem in the SMCP literature.

The main idea of the VNS methodology is to perform a systematic change of the neighborhood structure to explore during the search process. The benefits of exploring different neighborhood structures in VNS instead of a single one are based on three facts: (i) "*a local minimum within one neighborhood structure is not necessarily so for another*"; (ii) "*a global minimum is a local minimum within all possible neighborhood structures*"; and (iii) "*for many problems, local minima within one or several neighborhoods are relatively close to each other*" (Hansen et al., 2017). In Fig. 2, we illustrate some of the main ideas behind the VNS framework. In particular, we illustrate some iterations of a VNS search process with two neighborhood structures. In Figs. 2(a), 2(b), 2(c), and 2(d), we depict the set of possible solutions for a given optimization problem in a plane: the x-axis represents the search space, while the y-axis represents the objective function value of each possible solution. In each figure, we show two neighborhood structures, N_a and N_b, illustrated with rectangles. In Fig. 2(a), we represent an initial solution x with a solid red circle. Two rectangles represent the areas of the search space

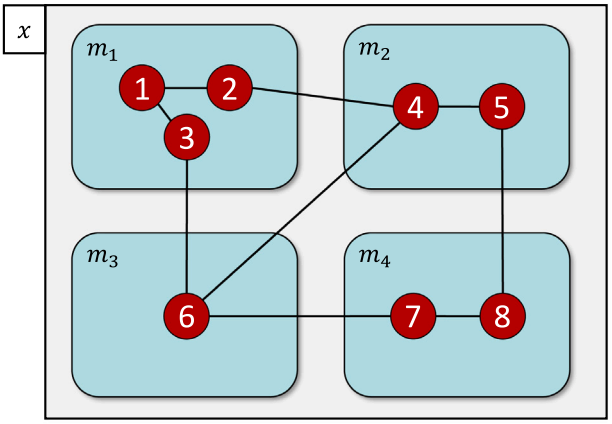


Fig. 1. Example of a solution x and evaluation of six objectives.

included in the neighborhood structures N_a (depicted in green) and N_b (depicted in gray) of the initial solution x. In this first step, $N_a(x)$ is explored, resulting in the application of a move operator to x in order to obtain solution x', the best solution in the neighborhood, which is represented by a red circle with a diagonal pattern. Next, in Fig. 2(b), as a second step, we represent the exploration of $N_a(x')$, where x' is the solution obtained in the previous figure. As it can be observed, there is no neighbor solution that is better than x' in $N_a(x)$. That is, x' is a local optimum in $N_a(x')$. Then, an alternative neighborhood structure, N_b, is explored in the third step, depicted in Fig. 2(c). In this figure, N_a is represented in gray, since it is not explored, while N_b is depicted in yellow. Although x' was a local optimum in $N_a(x')$, it is not a local optimum in the neighborhood structure $N_b(x')$. As a result, x'' is obtained. Finally, in Fig. 2(d), N_a is explored again. After the previous move operation, the current solution x'' is no longer a local optimum in N_a, and x''' is obtained. Finally, $N_a(x''')$ and $N_b(x''')$ are explored. However, as it can be observed, this solution is a global optimum in the search space represented in Fig. 2(d) and, therefore, a local optimum within all possible neighborhood structures. Thus, there are not better solutions in neighborhoods $N_a(x''')$ and $N_b(x''')$ than x'''.

Among the existing implementations proposed in Duarte et al. (2015), we use the Multi-Objective General Variable Neighborhood Search (MO-GVNS) schema. In Algorithm 1, we provide the pseudocode of MO-GVNS. The procedure expects five input parameters: a solution (SE), the maximum size of the perturbation to be made by the shake method (k_{max}), the number of distinct neighborhoods to explore (k'_{max}), the number of objective functions (r), and a time limit (t_{max}). It is worth mentioning that in this schema, a solution SE is defined as the set of efficient points in the Pareto front. Furthermore, as it is customary in the VNS methodology, the MO-GVNS schema does not establish any specific procedure to construct the initial solution, which should instead be defined depending on the particular tackled problem. Specifically, in Section 4.1 we describe the procedure proposed to construct the initial solution for the SMCP. The method presented in Algorithm 1 starts by setting the size of the perturbation to be made by the shake procedure to 1 (step 3). Then, while the stopping criteria is not met (step 4), the method performs three steps. First, it perturbs the solution by applying a shake procedure to each efficient point in SE (step 5). This step is performed in the MO-Shake procedure that is further described in Section 4.2. Second, it improves the solution by using a Multi-Objective Variable Neighborhood Descent (MO-VND) procedure (step 6). This procedure improves all efficient points in the solution according to each objective studied and it is further described in Section 4.3. Third, the procedure MO-NeighborhoodChange studies if the solution has been improved in this iteration and determines the size of the perturbation to be performed by the shake procedure in the next iteration (step 7). Specifically, in this context, an improvement consists of including at least a new efficient point (i.e., a non-dominated point) in the current solution. Therefore, in case of improvement, the new solution SE'' becomes the current best solution SE, and the value of k is reset to 1. Otherwise, SE is not updated and the value of k is incremented in one unit. Once an iteration is finished, the method verifies if the maximum time has been reached or the value of the variable k is greater or equal to the search parameter k_{max}. In that case, the algorithm returns the best solution found (step 10). Otherwise, the method performs a new iteration.

Algorithm 1 MO-GVNS method

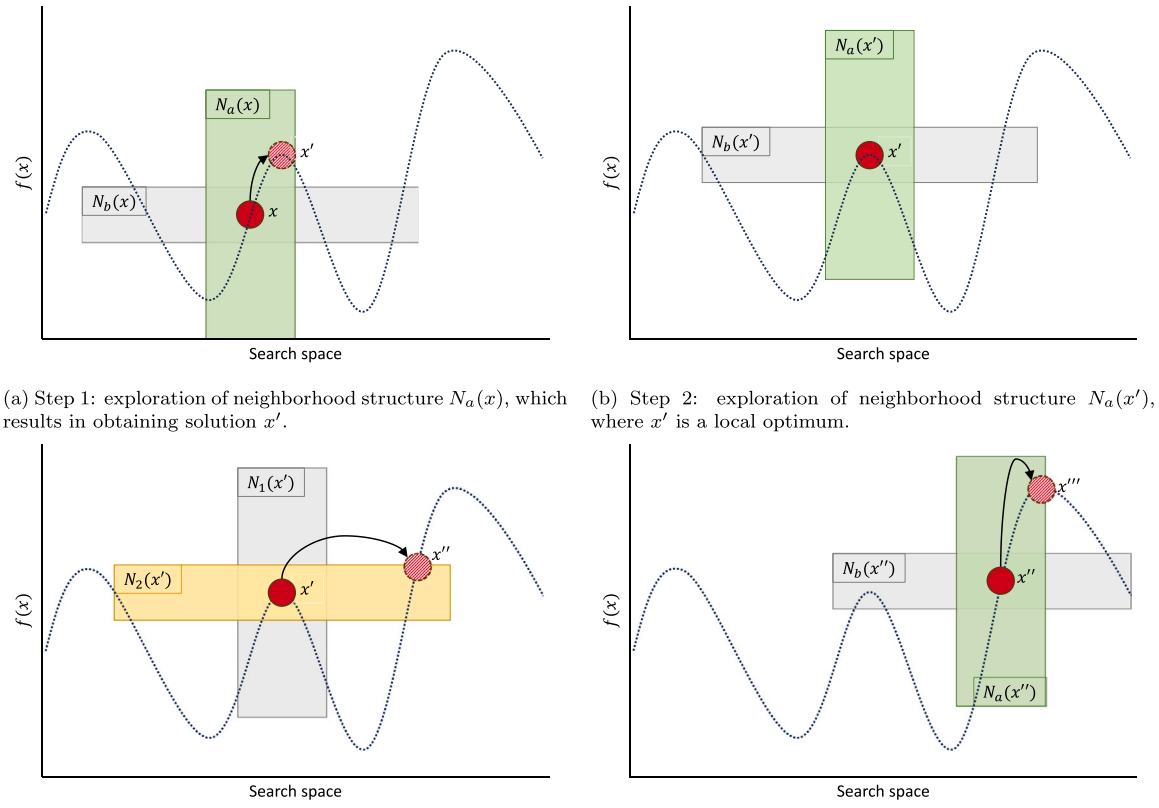
```
1: procedure MO-GVNS(SE,k_max,r,k'_max,t_max)
2:     t ← CPUTime()
3:     k ← 1
4:     while t ≤ t_max ∧ k ≤ k_max do
5:         SE' ← MO-Shake(SE,k)
6:         SE'' ← MO-VND(SE',k'_max,r)
7:         SE ← MO-NeighborhoodChange(SE,SE'',k)
8:         t ← CPUTime()
9:     end while
10:    return SE
11: end procedure
```

4.1. Constructive procedure

We propose an agglomerate constructive procedure to generate the initial solution for the MO-GVNS algorithm. The procedure starts by generating two trivial efficient points as follows:

1. The first efficient point is constructed by placing each vertex in a different module. This trivial structure has the maximum number of modules that any solution can have. However, it also has maximum coupling and minimum cohesion.
2. The second efficient point is constructed by placing all vertices of the input graph in the same module. This trivial organization has minimum coupling and maximum cohesion, which are desirable properties. However, it only has one module.

These two points are then used to generate additional non-dominated points by constructing a path between these two initial points. This idea is inspired by the Path Relinking methodology introduced by Glover (1997). In Fig. 3, we present an example of this procedure for a software with five components. As it can be seen, the method starts from the first trivial point x_s (Iteration 1) which has five modules. Then, in Iteration 2, it merges any two possible modules from the previous iteration and evaluates all obtained points with a particular function. Next, a point x' is selected based on a greedy criterion and will be used as the incoming



(a) Step 1: exploration of neighborhood structure $N_a(x)$, which results in obtaining solution x'.

(b) Step 2: exploration of neighborhood structure $N_a(x')$, where x' is a local optimum.

(c) Step 3: exploration of neighborhood structure $N_b(x')$, which results in obtaining solution x''.

(d) Step 4: Exploration of neighborhood structure $N_a(x'')$, which results in obtaining solution x'''.

Fig. 2. Illustration of two neighborhood structures, N_a and N_b, for different solutions in the search space.

point for the next iteration. The procedure is iteratively repeated until the second trivial point x_t is reached (Iteration 5).

As it can be noticed, the proposed constructive procedure traverses a path from a starting point to the target one, selecting only one point at each iteration. In this sense, only the points obtained from the selected point are evaluated, avoiding the exploration of the whole search space, which would be impractical given the combinatorial nature of the problem.

The choice of the modules to merge at each iteration is based on a greedy criterion. In this case, the constructive procedure illustrated in Fig. 3 evaluates the possible points at each iteration using the MQ function and greedily selects at each iteration the point with the largest MQ value. However, any other suitable function might be used. Moreover, the selection criterion does not need to be completely greedy.

4.2. Multi-objective shake procedure

In the VNS methodology, a shake procedure is used to randomly perturb an efficient point in order to escape from local optima (Mladenović and Hansen, 1997). A search parameter, denoted as k, indicates the number of perturbations to perform at each iteration. In the multi-objective adaptation of VNS (Duarte et al., 2015), the MO-Shake method extends the previous idea by applying the shake procedure to all efficient points in the solution.

Our shake procedures are based on the swap operator, which selects two vertices v_1 and v_2 belonging to two different modules m_1 and m_2, respectively, and interchanges them. That is, it removes v_1 from m_1 and inserts it into m_2. Accordingly, v_2 is removed from m_2 and then inserted into m_1. Based on this idea, we propose four different variants of the shake procedure, which use different criteria to select the vertices involved in the swap. Particularly, in Table 2 we summarize the type of criteria (greedy or random) used for the selection of the vertices for each shake variant. Next, we describe each proposal in a more detailed way:

Table 2
Selection criteria for the vertices involved in the swap operator for each of the proposed shake procedures.

	Selection criterion	
	First vertex	Second vertex
Shake 1	Random	Random
Shake 2	Greedy	Greedy
Shake 3	Random	Greedy
Shake 4	Greedy	Random

- **Shake 1**. Both vertices are selected at random from any of the modules in the solution.
- **Shake 2**. To select the first vertex, we greedily choose the module that has the worst CF value (see Eq. (6)). Then, a vertex is randomly selected from that module. The second vertex is selected as the one that has the strongest connection to the module which contains the first selected vertex. This strength is measured as the sum of the weights of the edges which connect the evaluated vertex to the aforementioned module.
- **Shake 3**. The first vertex is randomly chosen from any of the modules in the efficient point. The second vertex is selected as the one that has the strongest connection with the module of the first selected vertex.
- **Shake 4**. To select the first vertex, we greedily choose the module that has the worst CF value (see Eq. (6)). Then a vertex is randomly selected from that module. The second vertex is chosen at random from any of the modules in the efficient point.

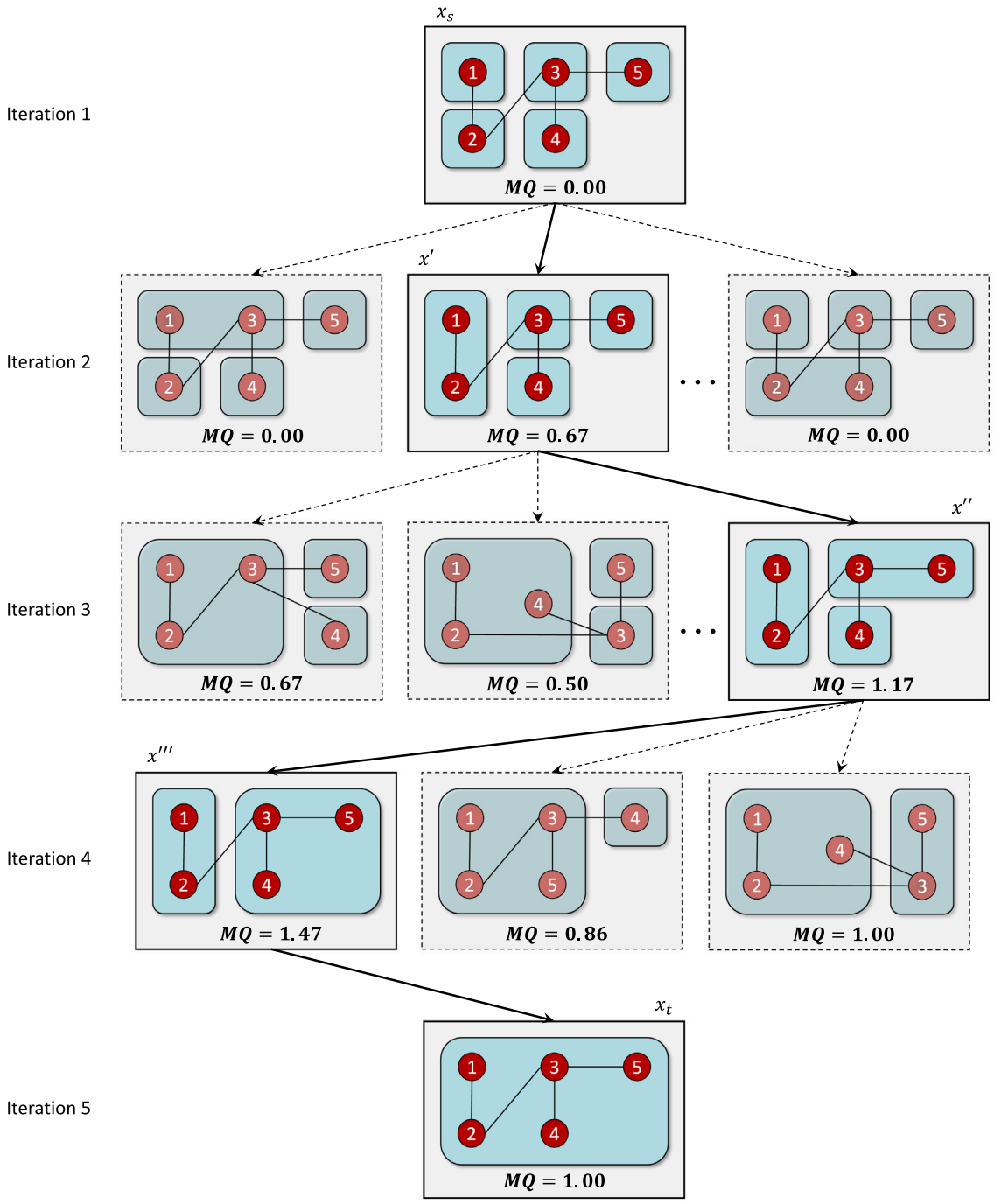


Fig. 3. Example of the agglomerate constructive procedure for a software project with five components, using MQ as the evaluation function.

The proposed variants of the shake procedure are empirically evaluated in Section 6.2, in order to select the most suitable one for the final configuration of the MO-GVNS algorithm.

4.3. Multi-objective variable neighborhood descent

In the VNS methodology, a VND procedure is used to improve an efficient point by exploring several neighborhoods in a systematic way. Once the search reaches a local optimum within all defined neighborhoods, the VND procedure ends. The MO-VND procedure also explores the set of predefined neighborhoods in a systematic way. However, this exploration must be performed for every efficient point in the solution and considering separately each of the objectives studied. Therefore, for each objective function $i \in r$, a VND-i procedure is defined, which explores all the neighborhoods proposed for all efficient points in the solution.

In Algorithm 2, we illustrate the MO-VND procedure, which receives three parameters: a number of neighborhoods (k'_{max}), a solution (SE), and a number of objectives (r). First, the variable i, which represents the objective being considered in each iteration, is set to 1 (step 2). Then, several sets of visited points, one for each of the objectives considered, are initialized (step 3). The procedure then improves the efficient points in solution SE according to objective i (steps 4–16). In each iteration, a VND-i method improves a non-exploited efficient point $x \in SE \setminus S_i$ (steps 5–9). Specifically, an efficient point x is selected at random among the non-visited efficient points in SE with respect to

the objective i (step 6). Then, this point is improved by the VND−i and the set of efficient points SE_i is obtained (step 7). Note that, at each iteration, the efficient points visited within the VND-i method are included in S_i to avoid exploring them twice (step 8). In step 10, the MO-Improvement checks if a new efficient point from S_i can be added to SE (i.e., an improvement has been made). If so, i is reset to 1 (step 12) and the solution SE is updated (step 11). Else, variable i is increased (step 14). Lastly, after exploring every efficient point within SE with respect to every objective, the method returns the solution SE (step 17).

Algorithm 2 MO-VND method

```
1: procedure MO-VND(k'_max, SE, r)
2:     i ← 1
3:     S_1 ← ∅, S_2 ← ∅, ..., S_r ← ∅
4:     while i ≤ r do
5:         while |SE \ S_i| ≥ 1 do
6:             x ← Pick(SE \ S_i)
7:             SE_i ← VND-i(x, k'_max)
8:             S_i ← S_i ∪ SE_i ∪ x
9:         end while
10:        if MO-Improvement(SE, S_i) then
11:            SE ← Update(SE, S_i)
12:            i ← 1
13:        else
14:            i ← i+1
15:        end if
16:    end while
17:    return SE
18: end procedure
```

For the search (step 7), four different neighborhoods are proposed:

- The first neighborhood (N_1) is defined by a swap operator, a classic move in trajectory-based heuristics (Croes, 1958; Gil-Borrás et al., 2021; Pardo et al., 2020; Yuste et al., 2022a). This operator selects two vertices and changes their respective modules. Therefore, the number of modules is not modified in this neighborhood. The size of this neighborhood is $\frac{|V|\cdot(|V|-1)}{2}$. In practice, it is usually smaller because vertices belonging to the same module are not exchanged. The complexity of exploring this neighborhood is $\mathcal{O}(|V|^2)$.
- The second neighborhood (N_2) is defined by an insertion operator. This is also a classic move in trajectory-based heuristics (Cavero et al., 2022; Gendreau et al., 1992; Pantrigo et al., 2012; Yuste et al., 2022a). This operator relocates a vertex from its current module into another one. Therefore, this neighborhood does not alter the number of modules. The size of this neighborhood is $|V|\cdot(|M|-1)$. Thus, the complexity of exploring this neighborhood is $\mathcal{O}(|V|\cdot|M|)$.
- The third neighborhood (N_3) is defined by a destruct operator, as described in Yuste et al. (2022a). This operator selects one module and removes it from the solution. The vertices that belonged to that module are then relocated into other modules. As a result, the solution contains one module less after applying a destruct operator. Therefore, this neighborhood is intended to reduce the number of modules. The size of this neighborhood is $|M|\cdot(|M|-1)^d$, where d is the average number of vertices within each module, as detailed in Yuste et al. (2022a). The complexity of exploring this neighborhood is $\mathcal{O}(|M|\cdot|M|^d)$.
- Finally, the fourth neighborhood (N_4) is defined by an extract operator, as described in Yuste et al. (2022a). This operator selects two or three vertices and inserts them into a new empty module. Therefore, this neighborhood is intended to increase the number of modules. The size of this neighborhood is $|V|\cdot(|V|-1)+|V|\cdot(|V|-1)\cdot(|V|-2)$, as detailed in Yuste et al. (2022a). Thus, the complexity of exploring this neighborhood is $\mathcal{O}(|V|^3)$.

5. Advanced strategies

First, in Section 5.1, we present a procedure to efficiently evaluate the quality of the efficient points in the solution. Next, in Section 5.2, we propose several strategies to explore only promising regions in the search space.

5.1. Efficient evaluation of the quality of an efficient point

Evaluating an objective function is frequently one of the most time-consuming tasks in optimization, since search algorithms explore a vast number of efficient points and each of them has to be evaluated. However, in trajectory-based search algorithms, an efficient point is usually obtained as a modification of a previous efficient point. Therefore, it is possible to evaluate a new efficient point by only determining the changes performed to the previous one, instead of recalculating the desired objective function from scratch. Next, we describe the strategy followed to efficiently evaluate an efficient point for the objective functions considered in this paper. This strategy is an extension of the ideas proposed in Yuste et al. (2022a) for the objectives presented in the MCA and ECA problems.

The MQ function is measured as the sum of the Cluster Factor (CF) of every module in the efficient point. Consequently, when a move is performed, only the CF of the affected modules has to be updated. Thus, after the first evaluation of an efficient point, we store the value of the CF of each module separately and we only update the value of the CF of a module when it is affected by a move. Similarly to MQ, the coupling and cohesion of each module change only when the module is modified. Therefore, we can store separately the coupling and cohesion values of each module and update them only when the module is affected after a move. In the case of the size difference between the smallest and the largest modules, the efficient evaluation of this metric is based again on the storage of the values which determine the largest and smallest modules. When a move is performed, it is checked if the affected modules have changed their size and, if so, it updates the value of the largest and smallest modules. Finally, the number of modules and the number of isolated modules are trivially calculated, so their evaluations are not enhanced.

5.2. Reduction of the size of the neighborhoods

When exploring a neighborhood, it is common that many of the moves performed result in a null improvement. Avoiding the exploration of unpromising areas (i.e., those containing non-improving moves) results in a more efficient exploration of the search space. Furthermore, when considering different objectives at the same time, as it is the case of multi-objective optimization problems, not all neighborhoods might be interesting for every objective. In this section we propose a reduction of both, the size and the number of neighborhoods, depending on the objective addressed.

When considering objectives 1, 2, and 3, the exploration of the neighborhoods might benefit from the theorem introduced in Köhler et al. (2013). In that theorem, the authors stated that: given an MDG graph $G(V,E)$, a vertex $v \in V$, and the set of adjacent vertices of v, denoted as $\gamma(v)$, then, in the optimal solution for the SMCP, v will be located in one of the modules where at least one vertex $u \in \gamma(v)$ is contained. Following this theorem, we can decrease the size of the neighborhoods when exploring objectives 1, 2, and 3, as reported in Yuste et al. (2022a). Specifically, we only consider moves that result in solutions where any moved vertex belongs to a module that contains at least one connected vertex. This filter reduces the computational complexity of exploring the neighborhoods proposed in the following ways:

- For N_1, the complexity is reduced from $\mathcal{O}(|V|^2)$ to $\mathcal{O}(|V| \cdot a \cdot d)$, where a is the average number of adjacent modules (those which contain an adjacent vertex) for each vertex and d is the average number of vertices in each module. Since the number of adjacent modules to any vertex is less or equal (at most) than the number of modules ($a \leq |M|$) and $|V| = |M| \cdot d$, it is clear that $a \cdot d \leq |V|$. In practice, for the SMCP, it is rarely the case where $a \cdot d = |V|$, since the graph would need to be fully connected.
- For N_2, given the above definition, since a vertex is only inserted in adjacent modules, the complexity is reduced from $\mathcal{O}(|V| \cdot |M|)$ to $\mathcal{O}(|V| \cdot a)$.
- For N_3, since a vertex is only inserted in adjacent modules, the complexity is reduced from $\mathcal{O}(|M| \cdot |M|^d)$ to $\mathcal{O}(|M| \cdot a^d)$.
- For N_4, since a vertex will only be placed in a new empty module together with adjacent vertices, the complexity is reduced from $\mathcal{O}(|V|^3)$ to $\mathcal{O}(|V| \cdot q^2)$, where q is the average number of adjacent vertices for each vertex. As it can be noticed, $q \leq |V|$. Again, for the SMCP, it is rarely the case where $q = |V|$.

As it can be observed, the sparser the MDG, the greater the reduction in size of the neighborhoods achieved with this strategy. In the case of real software projects, this strategy is particularly useful, since the graphs of dependencies are frequently sparse, with densities that vary between 1.77% and 21.52% (Mitchell and Mancoridis, 2008). In the dataset used in this work (described in Section 6), the average number of vertices is 156.37, while the average number of adjacent vertices per vertex is 10.22. Therefore, at least for this dataset, $q \ll |V|$ and $a \ll |V|$.

When considering objectives 4, 5, and 6, the previous reduction is not suitable. Instead, for objective number 4, we introduce a new strategy, where we avoid exploring any neighborhood which does not increase the number of modules. Particularly, we can avoid exploring N_1, N_2, and N_3, since they are designed to not increment the number of modules.

Similarly, for objective number 5, we introduce a new strategy where only moves that relocate vertices contained in isolated modules are considered. Therefore, we avoid exploring neighborhoods N_1, N_3, and N_4. Moreover, in N_2, we are only interested in the moves that insert vertices contained in isolated modules into other modules. Therefore, the complexity of N_2 is reduced from $\mathcal{O}(|V| \cdot |M|)$ to $\mathcal{O}(|V| \cdot |I|)$, where I represents the set of isolated modules and $|I| \leq |M|$.

Finally, for objective number 6, we introduce a new strategy where the exploration of N_1 is not performed, since it does not modify the size of the modules involved in the moves within N_1. Furthermore, among the rest of the neighborhoods (N_2, N_3, and N_4) we only consider moves that involve modules with the largest or smallest size.

6. Experiments

In this section, we perform some preliminary experiments devoted to either configure the parameters of the method proposed, or to study the impact of some of the advanced strategies previously introduced. Then, we perform a comparison of our method with the best previous state-of-the-art algorithms.

To test our algorithms, we have used a dataset of 124 instances previously introduced by Monçores et al. (2018). The dataset contains instances obtained from real software projects, written in C/C++ or Java, which have been already tested in related literature (Pinto et al., 2014; Monçores et al., 2018; Barros, 2012; Mitchell, 2002). The MDGs provided in the dataset that represent the software systems studied have considered files as components in the case of C/C++ and classes as components in the case of Java. The instances of the dataset are of varying sizes, including small and large projects with up to 1161 components and 11,722 dependencies. In particular, the average number of components is 156.37, with a standard deviation of 215.85. The average number of dependencies is 948.79, with a standard deviation of 1744.78. The average density of the resulting MDGs of the instances is 8.83%, with a standard deviation of 11.60%. To perform some preliminary experiments, ten instances have been randomly chosen from the aforementioned dataset: bison, bunch2, cia, crond, dot, forms, jscatterplot, mailx, micq, and netkit-ftp. The rest of the instances are used in the comparison to the state-of-the-art algorithms reported in Section 6.7.

Table 3
QIs and various quality aspects. A "+" means that the quality aspect is fully covered by a QI. A "–" means that the quality aspect is partially covered by a QI.
Source: Table adapted from Ali et al. (2020).

Quality aspect	HV	IGD	PFS	GS	C
Convergence	+	+			–
Spread	+	+		–	
Uniformity	+	–		+	
Cardinality	–	–	+		

6.1. Quality indicators

To assess the quality of the solutions in multi-objective optimization problems, different authors in the literature recommend comparing the results of different algorithms using appropriate Quality Indicators (QIs) instead of comparing individual objectives in isolation, since it might lead to inaccurate results (Li and Yao, 2019; Li et al., 2020; Ali et al., 2020). However, selecting the appropriate QIs is not a trivial task, and there is no agreement in the SBSE area on which QIs are the most suitable for the problems at hand (Ali et al., 2020).

As reported in Ali et al. (2020) and Li and Yao (2019), QIs can evaluate four different quality aspects of solutions: convergence, spread, uniformity, and cardinality. In Table 3, we report the quality aspects that are covered by different QIs, as reported in the related literature (Ali et al., 2020). In particular, we report the following QIs: Hypervolume (HV), Inverted Generational Distance (IGD), Coverage (C), Generalized Spread (GS), and Pareto Front Size (PFS). As it can be observed, there is not a single QI that completely covers all quality aspects. To select the most suitable QIs for the studied problems, we follow the suggestions of Wang et al. (2016), Ali et al. (2020), and Li et al. (2020), who offer practical guidelines for selecting representative QIs for SBSE problems. In particular, Wang et al. (2016) suggests using one of the following two indicators: HV and IGD. Ali et al. (2020) suggests using multiple QIs to cover all quality aspects, including C, GS, and PFS. Finally, for problems where the preferences of decision makers are not clear, Li et al. (2020) suggest either using multiple QIs that cover all quality aspects or using a comprehensive QI, such as HV or IGD. This is the case in SMCP problems, where software developers introduce their subjective experience in the process, which makes it neither systematic nor repeatable (Barros et al., 2015). For these reasons, we decide to include in the evaluation of the results both comprehensive QIs (i.e., HV and IGD) and complementary QIs (i.e., C, GS, and PFS). Moreover, since other authors have highlighted deficiencies in IGD for real-world problems, we replace it with IGD+, as suggested in Ishibuchi et al. (2015). Therefore, for the evaluation of solutions, we report five QIs, as suggested in the related literature for SBSE problems: HV, IGD+, C, GS, and PFS.

The HV indicator evaluates the convergence, spread, and uniformity of solutions. It is one of the most popular quality indicators in multi-objective optimization. The higher its value, the better the front. C measures the percentage of efficient points in a solution SE that are dominated by a reference set R. IGD+ measures the proximity of the efficient points in a SE to the efficient points in R. GS measures the uniformity and spread of the efficient points contained in a SE. For coverage, IGD+, and spread, the lesser their value, the better the quality of the front. Finally, PFS measures the number of efficient points in a SE. The higher its value, the better the quality of the front.

Notice that, for most QIs described above, a reference set R is needed to evaluate a given solution. Ideally, the reference set would

Table 4
Comparison of different shake procedures for the MCA problem.

Shake	CPUt (s)	PFS	HV	C	IGD+	GS
Shake 1	21428.59	748.10	**0.2903**	**0.1475**	**0.0024**	0.5665
Shake 2	266.16	373.00	0.2687	0.6389	0.0202	**0.5428**
Shake 3	8722.78	671.00	0.2868	0.3585	0.0053	0.5755
Shake 4	15307.98	688.30	0.2888	0.2589	0.0034	0.5578

Table 5
Comparison of different shake procedures for the ECA problem.

Shake	CPUt (s)	PFS	HV	C	IGD+	GS
Shake 1	131874.37	1801.60	**0.2667**	**0.1821**	**0.0024**	0.5626
Shake 2	3626.04	989.30	0.2449	0.5905	0.0148	**0.5313**
Shake 3	39516.04	1587.90	0.2633	0.3293	0.0043	0.5583
Shake 4	101012.11	1758.50	0.2665	0.2181	0.0025	0.5546

be the optimal Pareto front, but it is frequently unknown. Here, we use an approximate reference front, obtained by combining the Pareto fronts generated by all the methods being compared, as it is common in the literature (Ali et al., 2020).

6.2. Comparison of shake procedures

In this paper we introduced four different shake procedures (see Section 4.2) for the proposed MO-GVNS. Here, we compare the performance of the four shake methods for both MCA and ECA.

The configuration of the algorithm to compare the different shake procedures includes the exploration of the neighborhoods within the MO-VND, in the following order: N_1, N_3, N_2, and N_4. For the stopping criteria, we set $k_{max} = 5$. In order to compare the convergence of the proposed method over time with the different shake procedures, we do not set a time limit in this experiment. Finally, the objectives are tackled in the following order: MQ, Cohesion, Coupling, Number of modules, and Number of isolated modules (in the case of MCA) / Difference between the maximum and minimum size of modules (in the case of ECA).

In Tables 4 and 5, we present the results obtained for the MCA and ECA problems, respectively. In particular, we report the indicators previously described, highlighting the best result in bold font. As it can be observed, the first shake procedure achieves the best result in three out of four quality indicators for both problems. Although it consumes more time than the other approaches, this is due to the fact that the algorithm is able to explore a wider area of the search space before reaching the stopping criterion. Therefore, we select the Shake 1 to be part of the final configuration of the MO-GVNS.

6.3. Comparison of different values for k_{max}

In this experiment, we compare the performance of the MO-GVNS with different values of k_{max}. This parameter specifies the maximum value that the variable k can take in the algorithm. Therefore, this parameter is used to control the magnitude of the perturbation to be made within the shake procedure (i.e., the largest neighborhood that will be stochastically explored). Here, we analyze the impact of the use of different values of k_{max} (1, 2, 3, 4, and 5) using the same configuration of the MO-GVNS previously reported in Section 6.2, but considering the Shake 1, which was selected in the previous preliminary experiment. Additionally, we also introduce in the comparison the MO-VND method used within the MO-GVNS to test the contribution of using a MO-GVNS schema instead of just a MO-VND one. In this experiment, we let the algorithms run as far as they produce an improvement in the solution and we reported the computational time consumed and the best solution obtained.

In Tables 6 and 7 we present, respectively, the results achieved for both MCA and ECA. Firstly, we observe that the combination of deterministic and stochastic exploration performed by any of the MO-GVNS configurations tried improves the results obtained by the MO-VND, which only performs a deterministic exploration. Among the different MO-GVNS configurations, the results obtained are similar. For both problems, the configuration with $k_{max} = 5$ achieved the best results in terms of Hypervolume, Coverage, and IGD+. However, the increase in the performance with respect to $k_{max} = 4$ is the smallest in the comparison, so we did not try larger values of k_{max}. It is worth noting that using a higher value for the parameter k_{max} leads to a further exploration of the search space and, therefore, larger computing time. Generally speaking, a larger exploration usually achieves better results, but it is necessary to find a trade off between the quality of the solutions and the time consumed.

In this sense, we selected $k_{max} = 5$ for the following experiments, since it obtained the best results, but in order to bound the running time of the algorithm, a combined stopping criteria based on the size of the instance, should be considered.

6.4. Influence of the incremental evaluation

Here, we study the impact of the strategy presented in Section 5.1. In particular, in this experiment, we executed the MO-GVNS algorithm twice using the same configuration but with one difference: one of them implements the incremental evaluation (Incremental) while the other one does not (Complete).

The results obtained for both MCA and ECA are reported in Table 8 and Table 9, respectively. As it can be seen, the obtained results are identical in terms of quality. However, when using the incremental evaluation, the computational time consumed by the algorithm was reduced by 73.37% in the case of MCA and by 67.23% in the case of ECA. The incremental evaluation is therefore included to evaluate the efficient points in the proposed MO-GVNS algorithm.

6.5. Influence of the reduction of the size of the neighborhoods

In Section 5.2, we presented a strategy to reduce the size of the neighborhoods, exploring only promising areas within the search space. Here, we study the impact of this strategy on the performance of the algorithm. In particular, we execute the MO-GVNS algorithm twice using the same configuration, but one of them exploring only the reduced neighborhoods as explained above.

We present the results obtained for both MCA and ECA in Table 10 and Table 11, respectively. As it can be observed, there are slight differences between the solutions obtained in terms of quality, but a very large reduction of the computing time. Particularly, 68.11% and 45.13% of the time consumed for MCA and ECA, respectively. Therefore, the reduction of the size of the neighborhoods is included in the proposed MO-GVNS algorithm to avoid exploring unpromising areas in the search space.

6.6. Contribution of the objectives

Yuan et al. (2017) proposed a methodology to reduce the number of objectives in multi-objective optimization problems. Among other benefits, removing redundant objectives reduces the computational cost of the optimization process. In this case, given the nature of the proposed algorithm (a MO-GVNS method) the number of objectives directly impacts the computational cost of the algorithm, since a new VND is created for each of the objectives of the problem. Therefore, it is interesting to analyze the objectives considered in both the MCA and ECA problems, in order to identify redundant objectives. Notice that in this case, we are not interested in reducing the number of objectives of the problem, but the number of objectives considered during the search.

Yuan et al. (2017) proposed three different methods to reduce the number of objectives. Given a Pareto front, obtained by considering a set of objectives, these methods calculate an error rate by considering

Table 6
Comparison of different values of k_{max} for the MCA problem.

Method	k_{max}	CPUt (s)	PFS	HV	C	IGD+	GS
MO-VND	NA	10.94	291.09	0.2586	0.4092	0.0217	**0.4835**
MO-GVNS	1	5072.68	624.70	0.2853	0.2925	0.0034	0.5589
MO-GVNS	2	7919.36	664.30	0.2873	0.2049	0.0017	0.5605
MO-GVNS	3	12790.68	705.60	0.2888	0.1163	0.0006	0.5650
MO-GVNS	4	19008.96	737.80	0.2898	0.0336	0.0001	0.5657
MO-GVNS	5	21428.59	748.10	**0.2903**	**0.0004**	<**0.0000**	0.5665

Table 7
Comparison of different values of k_{max} for the ECA problem.

Method	k_{max}	CPUt (s)	PFS	HV	C	IGD+	GS
MO-VND	NA	26.42	435.90	0.2293	0.7394	0.0328	**0.4566**
MO-GVNS	1	22887.41	1528.40	0.2629	0.2682	0.0026	0.5534
MO-GVNS	2	42556.78	1641.70	0.2644	0.1675	0.0012	0.5590
MO-GVNS	3	69516.32	1702.60	0.2653	0.1040	0.0006	0.5584
MO-GVNS	4	111005.34	1773.80	0.2663	0.0316	0.0001	0.5597
MO-GVNS	5	131874.37	1801.60	**0.2667**	**0.0025**	<**0.0000**	0.5626

Table 8
Comparison of the results obtained with (Incremental) and without (Complete) including the incremental evaluation for the MCA problem.

Evaluation	CPUt (s)	PFS	HV	C	IGD+	GS
Complete	80457.50	748.10	0.2903	0.0000	0.0000	0.5665
Incremental	21428.59	748.10	0.2903	0.0000	0.0000	0.5665

Table 9
Comparison of the results obtained with (Incremental) and without (Complete) including the incremental evaluation for the ECA problem.

Evaluation	CPUt (s)	PFS	HV	C	IGD+	GS
Complete	402415.48	1801.60	0.2667	0.0000	0.0000	0.5626
Incremental	131874.37	1801.60	0.2667	0.0000	0.0000	0.5626

only a subset of the objectives to evaluate the Pareto front. If the objectives are in conflict, the fewer the objectives considered, the lesser the number of non-dominated efficient points (and the higher the error rate). The objective is to find a trade-off between the error rate and the number of objectives considered (both to be minimized). To calculate the error rate, the authors proposed three different measures, δ, η, and γ, based on the dominance structure of the front and the correlation between the objectives. In particular, δ and η are based on the dominance structure. The δ criterion, proposed in Brockhoff and Zitzler (2009), measures the degree of dominance structure change in a Pareto front N between a set of objectives F_0 and a subset of objectives $F \subseteq F_0$. The η criterion is proposed in Yuan et al. (2017) as an alternative to some deficiencies identified in the δ criterion. To calculate the error rate η, the set N is divided into two subsets: N_{DS} and N_{NS}. N_{DS} contains the efficient points in N that are dominated by other efficient points when the subset of objectives $F \subseteq F_0$ is considered instead of the original set F_0. On the contrary, N_{NS} contains the efficient points that are not dominated ($N_{NS} = N \setminus N_{DS}$). Then, η is formally defined as:

$$\eta = |N_{DS}|/|N|. \tag{9}$$

Finally, the error rate γ is based on the correlation between each pair of objectives. It is proposed in Yuan et al. (2017) as an off-the-shelf criterion that evaluates any given subset of objectives based on a correlation analysis. As described by the authors, it can be seen as a measurement of the degree of correlation structure change between F and F_0. The more in conflict an objective is with the rest of objectives, the worse the error rate γ when that objective is not considered in the subset F. For a detailed description of the calculation of each error rate, we refer interested readers to relevant sources (Yuan et al., 2017; Brockhoff and Zitzler, 2009).

Here, we analyze the error rates of all the possible subsets of objectives for both the MCA and ECA problems. For the calculation of the error rates, we use the Pareto fronts obtained by the proposed MO-GVNS for the preliminary dataset. In Tables 12 and 13, we present the results obtained for both the MCA and ECA problems, respectively. For each subset of objectives, we report the mean error rate, averaged among the set of instances and the three error measures δ, η, and γ. The different combinations of objectives studied are sorted in ascending order depending on the average error rate obtained. Moreover, for the sake of brevity, we have cropped the table, showing only the 10 best subsets of objectives (i.e., those with the smallest error rates). As it can be observed in Tables 12 and 13, removing either coupling or cohesion results in an error rate close to 0% for both problems. This can be explained due to the fact that coupling and cohesion are antagonist objectives (coupling can be calculated as the number of edges minus cohesion). Therefore, they seem not to be in conflict.

As it was aforementioned, the interest of this analysis resides in the possibility of reducing the number of VND components in the MO-GVNS method. Therefore, considering only a subset of objectives reduces the number of VND components used in the exploration. Here, we analyze the results obtained by considering only a subset of the objectives for the VND components. To avoid comparing all the possible combinations of objectives, we consider only the subsets with the lowest error rates as reported in Tables 12 and 13. Notice that, regardless of the subset of objectives considered for the VND components, all the objectives are reported when evaluating a solution in order to be qualified for entering in the Pareto front. That is, we are reducing the search space explored within the MO-GVNS method, but we are not reducing the number of objectives considered in the MCA and ECA problems.

The results obtained are shown in Tables 14 and 15. As it can be observed, considering only a subset of objectives for the VND components results in worse solutions in terms of quality. However, the difference is sometimes in the third decimal. For instance, not considering coupling, cohesion, and/or the number of isolated modules for the MCA problem results in an almost identical value for the hypervolume indicator. In contrast, the consumed time is reduced in up to 21.83%. In the case of MQ, not exploring this objective results in a greater reduction of the quality of the solutions. In the case of ECA, the results are similar. When not considering coupling nor the size difference between the smallest and largest modules for the VND components, the method achieves a reduction of 48.13% in the consumed time with just a small detriment in the quality of the solutions.

To find a trade-off between quality and computational cost, we configure our algorithm so that it does not consider coupling and the number of isolated modules (in the case of MCA), and coupling and the size difference between the smallest and largest modules (in the case of ECA), during the search process.

Table 10
Comparison of the results obtained with (Reduced) and without (Complete) the strategy presented in Section 5.2 for the MCA problem.

Size of neighborhoods	CPUt (s)	PFS	HV	C	IGD+	GS
Complete	67193.25	1216.20	**0.3025**	**0.0869**	**0.0014**	**0.5551**
Reduced	21428.59	748.10	0.2903	0.2885	0.0111	0.5665

Table 11
Comparison of the results obtained with (Reduced) and without (Complete) the strategy presented in Section 5.2 for the ECA problem.

Size of neighborhoods	CPUt (s)	PFS	HV	C	IGD+	GS
Complete	240329.02	2288.50	**0.2737**	**0.1161**	**0.0017**	**0.5504**
Reduced	131874.37	1801.60	0.2667	0.2951	0.0078	0.5626

Table 12
Comparison of the average error rates obtained by removing some of the considered objectives in the MCA problem for the evaluation of the solutions.

Considered objectives	Number of objectives	Avg. error rate
1,2,3,4,5	5	<0.00%
1,3,4,5	4	<0.00%
2,3,4,5	4	<0.00%
1,2,3,4	4	21.97%
1,3,4	3	21.97%
2,3,4	3	21.97%
1,2,4,5	4	47.02%
1,4,5	3	47.02%
2,4,5	3	47.02%

Table 13
Comparison of the average error rates obtained by removing some of the considered objectives in the ECA problem for the evaluation of the solutions.

Considered objectives	Number of objectives	Avg. error rate
1,2,3,4,6	5	0.00%
1,3,4,6	4	0.00%
2,3,4,6	4	0.00%
1,2,4,6	4	56.47%
1,4,6	3	56.47%
2,4,6	3	56.47%
1,2,3,4	4	62.89%
1,3,4	3	62.89%
2,3,4	3	62.89%

6.7. Comparison with the state of the art

In this section, we compare the performance of the best configuration of the MO-GVNS algorithm to the state-of-the-art methods available for the SMCP. The MO-GVNS method is configured as follows: Shake 1 is used as the shake procedure; the maximum value of k is set to $k_{max} = 5$; the incremental evaluation of objective functions and the efficient exploration of the search space are implemented; and coupling, the number of isolated modules, and the difference in size between the largest and smallest modules are not considered as guiding functions. For the comparison, we use a dataset comprised of 124 instances previously introduced by Monçores et al. (2018). These instances were obtained from different real-life software projects, including projects with up to 1161 components and 11,722 dependencies.

We compare the proposed method with four different algorithms: a Two-Archive Artificial Bee Colony (TA-ABC) recently proposed to tackle the MCA and ECA problems (Amarjeet and Chhabra, 2018); the Non-dominated Sorting Genetic Algorithm III (NSGA-III) (Deb and Jain, 2013); the Modified Pareto Envelop-Based Selection Algorithm (PESA2) (Corne et al., 2001); and the Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) (Zhang and Li, 2007). It is worth mentioning that the TA-ABC method included in the comparison can be considered as the current state-of-the-art algorithm for the MCA/ECA problems. This procedure extends the ideas of the ABC framework, and it was particularly designed for these problems. Additionally, we have included three general-purpose algorithms for multi- and many-objective optimization: NSGA-III, MOEA/D and PESA2. These algorithms have been widely used as reference algorithms for multi- and many-objective optimization problems. Moreover, they have also been used in the literature for different SMCP problems (Mkaouer et al., 2015; Chhabra, 2018b,a; Prajapati and Chhabra, 2020; Arasteh et al., 2022). Since different authors in the literature do not recommend comparing individual objectives in isolation, because it might lead to inaccurate results (Li and Yao, 2019; Li et al., 2020; Ali et al., 2020), we do not use other methods in the literature that have individually studied some of the metrics included in the MCA/ECA problems following a mono-objective approach.

The experiments were conducted on an Intel Xeon Processor with 64 cores and 124 GB of RAM, running Ubuntu 22.04 LTS as the Operating System. Our proposal and the TA-ABC algorithm were implemented in Java 17.0.5 and using the Metaheuristic Optimization framewoRK (MORK) v0.12 (Martín-Santamaría et al., 2022). For NSGA-III, PESA2, and MOEA/D, we used the implementations that are accessible in the jMetal framework v5.11 (Durillo and Nebro, 2011). Finally, to perform a fair comparison, we set a time limit for the execution of the proposed MO-GVNS. In particular, we propose a hybrid stopping criteria, the k_{max} parameter with a maximum CPU time in seconds, as specified in Algorithm 1. On one hand, the method stops when no further improvements are found and the maximum k_{max} is reached. On the other hand, the method stops if the time of four times the number of vertices ($t_{max} = 4 \cdot |V|$) is reached.

The results obtained are shown in Tables 16 and 17. As can be observed, the MO-GVNS method is able to obtain solutions with better quality for all the analyzed indicators, considering both convergence to the reference set and distribution along the objective space. The margin is wide for all indicators, and the difference is an order of magnitude in some cases, such as hypervolume, coverage, and IGD+. In addition, the showcased results are obtained consuming less CPU time than the next fastest method, and the returned Pareto front contains an order of magnitude more efficient points.

6.8. Analysis of Pareto fronts generated by the compared methods for pairs of objectives

In this section, we analyze the Pareto fronts for several interesting pairs of objectives studied, generated by the methods compared in this paper. In particular, we have randomly selected two instances with a diverse number of vertices, i.e., a medium size instance, and a very large instance, from the data set: gae_plugin_core, which contains 139 vertices and 375 edges; and apache_ant, which contains 1085 vertices and 5329 edges. To ease readability, we represent only those efficient points from the solution that are not dominated with respect to the two objectives depicted in each figure. In addition, to ease readability, the values of the objectives that should be minimized have been multiplied by -1. Therefore, in every figure, the greater the value of each objective, the better the efficient point.

In Fig. 4, we represent the non-dominated efficient points obtained by each method for the ECA problem considering cohesion and the

Table 14
Comparison of the results obtained by considering different sets of objectives as guiding functions for the MCA problem.

Guiding functions	CPUt (s)	PFS	HV	C	IGD+	GS
All	21428.59	748.10	**0.2903**	**0.2066**	0.0029	0.5665
All \ {Coupling}	17268.77	720.80	0.2893	0.2272	**0.0028**	0.5658
All \ {Cohesion}	17243.87	720.80	0.2893	0.2272	**0.0028**	0.5658
All \ {Isolated}	17210.76	705.50	0.2867	0.2748	0.0044	0.5706
All \ {Coupling, Isolated}	16750.98	703.70	0.2866	0.2673	0.0041	0.5777
All \ {Cohesion, Isolated}	16937.45	703.70	0.2866	0.2673	0.0041	0.5777
All \ {MQ}	6428.05	325.10	0.2644	0.4510	0.0288	0.5220
All \ {Coupling, MQ}	3131.02	287.60	0.2595	0.4859	0.0339	**0.5005**
All \ {Cohesion, MQ}	3085.88	287.60	0.2595	0.4859	0.0339	**0.5005**

Table 15
Comparison of the results obtained by considering different sets of objectives as guiding functions for the ECA problem.

Guiding functions	CPUt (s)	PFS	HV	C	IGD+	GS
All	131874.37	1801.60	**0.2667**	**0.2035**	0.0027	0.5626
All \ {Coupling}	99374.69	1779.70	0.2666	0.2340	**0.0021**	0.5608
All \ {Cohesion}	98563.21	1779.70	0.2666	0.2340	**0.0021**	0.5608
All \ {Diff}	89541.81	1506.10	0.2533	0.2897	0.0097	0.5520
All \ {Coupling, Diff}	68402.46	1489.90	0.2531	0.3140	0.0089	0.5551
All \ {Cohesion, Diff}	64126.70	1489.90	0.2531	0.3140	0.0089	0.5551
All \ {MQ}	24589.92	531.80	0.2377	0.4281	0.0471	0.5599
All \ {Coupling, MQ}	18796.65	486.10	0.2347	0.4153	0.0486	**0.5423**
All \ {Cohesion, MQ}	19078.99	486.10	0.2347	0.4153	0.0486	**0.5423**

Table 16
Comparison of the proposed MO-GVNS to several state-of-the-art methods for the MCA problem.

Method	CPUt (s)	PFS	HV	C	IGD+	GS
MO-GVNS	**311.18**	**507.60**	**0.2213**	**0.0264**	**0.0443**	**0.5175**
MOEA/D (Zhang and Li, 2007)	336.43	300.00	0.0982	0.5254	0.2184	0.6844
NSGA-III (Deb and Jain, 2013)	596.41	479.56	0.0937	0.2719	0.2587	0.5891
PESA2 (Corne et al., 2001)	643.16	99.63	0.0604	0.7335	0.3209	0.6994
TA-ABC (Amarjeet and Chhabra, 2018)	1037.31	97.70	0.0277	0.2145	0.3991	0.8026

Table 17
Comparison of the proposed MO-GVNS to several state-of-the-art methods for the ECA problem.

Method	CPUt (s)	PFS	HV	C	IGD+	GS
MO-GVNS	**311.16**	**520.64**	**0.1939**	**0.0122**	**0.0180**	**0.5614**
MOEA/D (Zhang and Li, 2007)	345.06	300.00	0.0724	0.8010	0.3312	0.6032
NSGA-III (Deb and Jain, 2013)	745.81	209.56	0.0889	0.5918	0.3690	0.6777
PESA2 (Corne et al., 2001)	408.69	89.59	0.0443	0.7386	0.4777	0.7599
TA-ABC (Amarjeet and Chhabra, 2018)	914.74	30.66	0.0284	0.3051	0.5132	0.8932

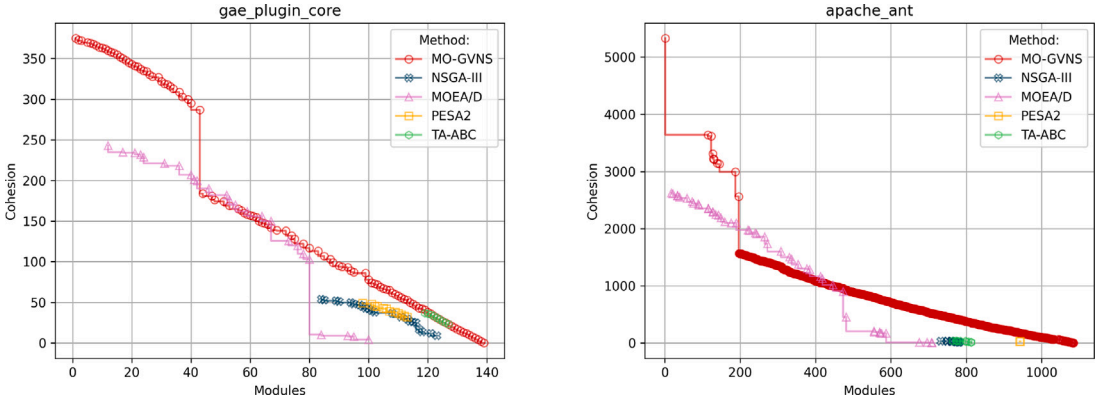


Fig. 4. Non-dominated fronts considering only cohesion and the number of modules, obtained for instances gae_plugin_core and apache_ant in the ECA problem.

number of modules. As it can be observed, the MO-GVNS method is able to generate Pareto fronts that are better distributed along the objective space than previous methods. However, there exist some solutions generated by MOEA/D that are not dominated by the solutions obtained with the MO-GVNS method.

In Fig. 5, we represent the non-dominated efficient points obtained by each method for the ECA problem considering the MQ metric and the number of modules. As it can be seen, the solutions generated by the MO-GVNS completely dominate the Pareto front generated by MOEA/D for both instances. This suggests that, although having less total cohesion in some cases, the components of the systems are better distributed among modules in the solution. That is, the relation of classes to packages is more homogeneous. This fact is further supported by Fig. 6, where we represent the non-dominated solutions obtained

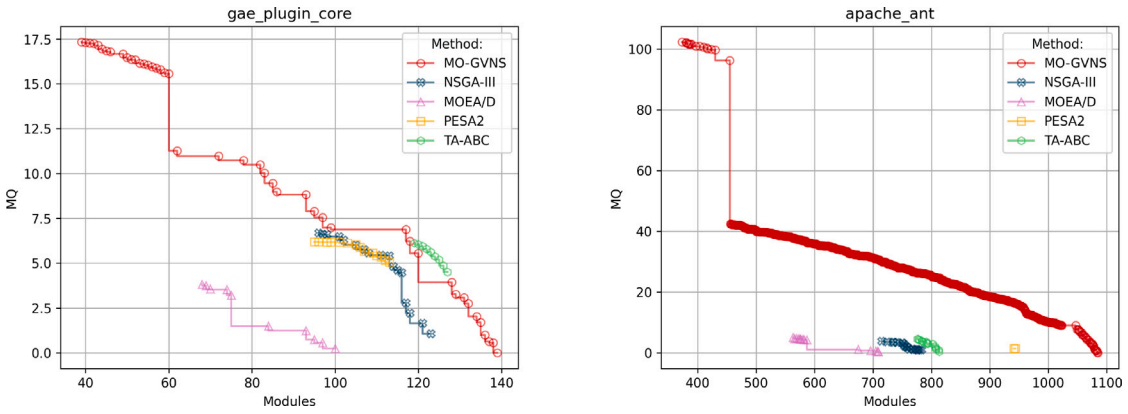


Fig. 5. Non-dominated fronts considering only MQ and the number of modules, obtained for instances gae_plugin_core and apache_ant in the ECA problem.

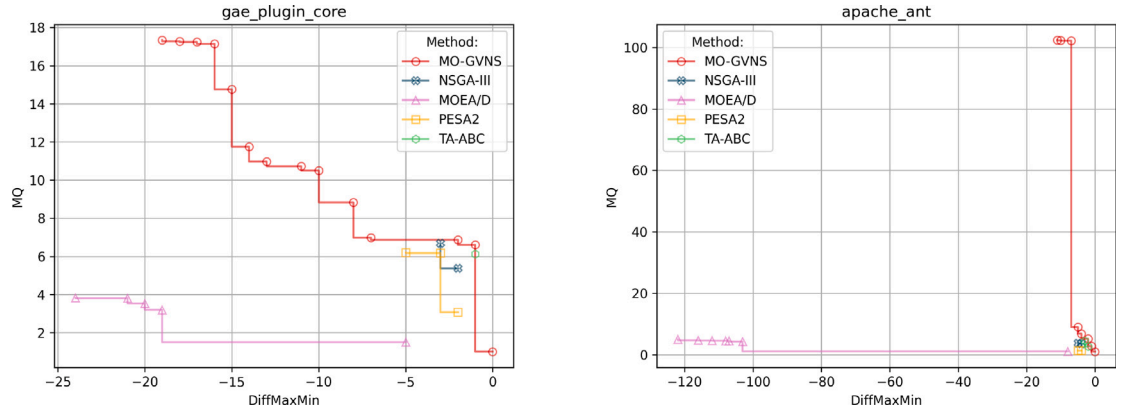


Fig. 6. Non-dominated fronts considering only MQ and the difference in size between the largest and smallest modules, obtained for instances gae_plugin_core and apache_ant in the ECA problem.

by each method for the ECA problem considering the difference in size between the largest and smallest modules and the MQ metric.

In Fig. 7, we represent the non-dominated efficient points obtained by each method for the ECA problem considering the MQ metric and cohesion. As it can be observed, although MOEA/D obtained some efficient points that were not dominated when considering cohesion and the number of modules, the Pareto front generated by the MO-GVNS method completely dominated the rest of the solutions when considering both MQ and cohesion.

In Fig. 8, we represent the non-dominated efficient points obtained by each method for the MCA problem considering the number of isolated modules and MQ. As it can be observed, the MO-GVNS method is able to obtain efficient points with few or non-isolated modules and an MQ value much higher than the efficient points found by the other approaches.

In Fig. 9, we represent the non-dominated efficient points obtained by each method for the MCA problem considering the number of modules and the number of isolated modules. As it can be observed, as happened in the previous examples, the MO-GVNS method is able to better populate the front along the objective space. Moreover, it finds good efficient points, with very few isolated modules (up to zero), even when the number of modules is high (more than 40 in the case of gae_plugin_core and more than 400 in the case of apache_ant). However, other methods are able to obtain better efficient points in the "center" of the objective space, with a good compromise between the number of modules and the number of isolated modules.

The visualization of the Pareto fronts generated by the different methods indicates that the MO-GVNS method obtains solutions that are better distributed along the objective space. This fact can be explained by the design of the algorithm. First, the constructive procedure proposed in Section 4.1 generates efficient points along the entire objective space considering MQ and the number of modules. Then, the MO-GVNS component of the algorithm explores the search space by improving each objective in isolation, successfully "pushing" each efficient point in different directions along the objective space. However, the isolated focus on different objectives during the search process may have some drawbacks. In Figs. 4, 5, 6, and 7, as it can be observed, there exist some gaps along the front generated by the MO-GVNS method. That is, there exist some drastic changes in the value of an objective with a small change in the value of the other objective. For example, see the change in cohesion for instance apache_ant when the number of modules increases to 200. The same drastic change does not occur in the Pareto front generated by MOEA/D, which is able to obtain better efficient points, in terms of MQ, when the number of modules is between 200 and 400. Moreover, in Fig. 9, the MO-GVNS method, although finding good efficient points with a high number of modules and few isolated modules, seems to have difficulties filling the center area of the figure for the pair of objectives studied. In future work, it would be interesting to analyze whether combining the isolated improvement of different objectives proposed by the MO-GVNS method with other approaches would improve its performance.

6.9. Discussion about the key strategies proposed

After improving previous state of the art procedures for the MCA/ECA, according to convergence, spread, uniformity, and cardinality, we can identify several key aspects of our research that might be useful for other researchers and that we review next.

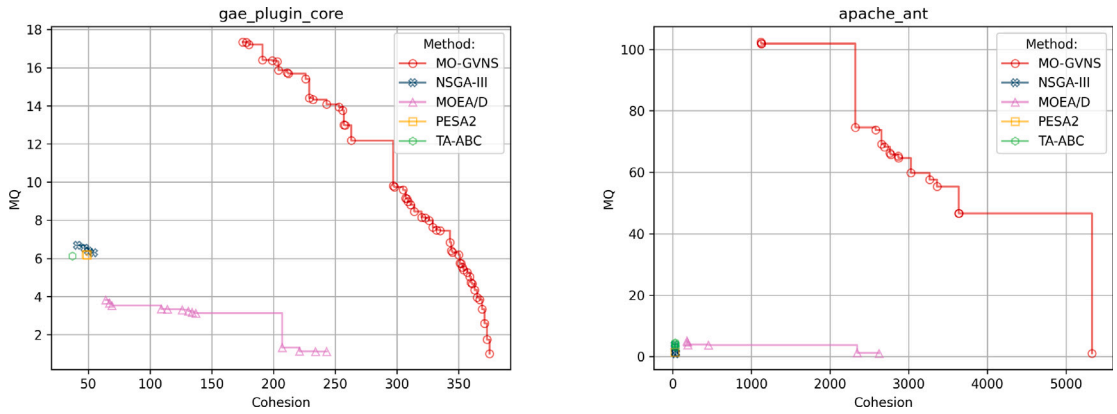


Fig. 7. Non-dominated fronts considering only MQ and cohesion, obtained for instances gae_plugin_core and apache_ant in the ECA problem.

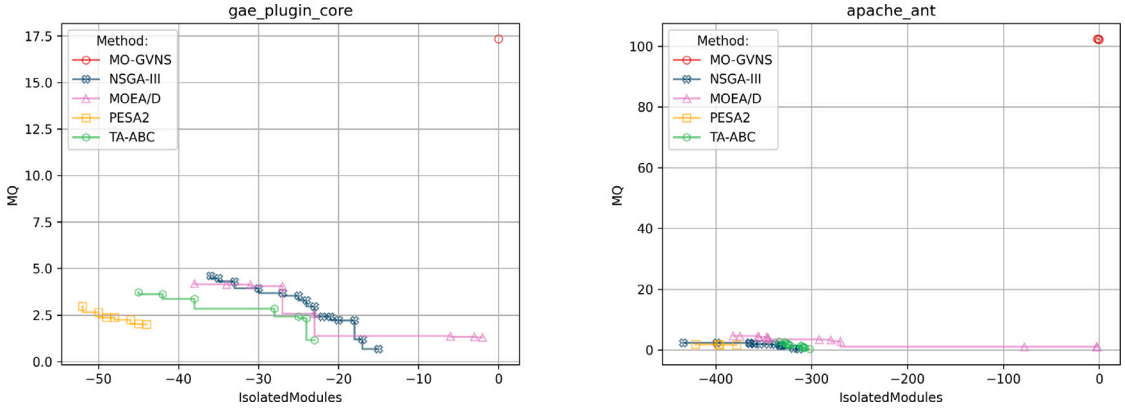


Fig. 8. Non-dominated fronts considering only the number of isolated modules and MQ, obtained for instances gae_plugin_core and apache_ant in the MCA problem.

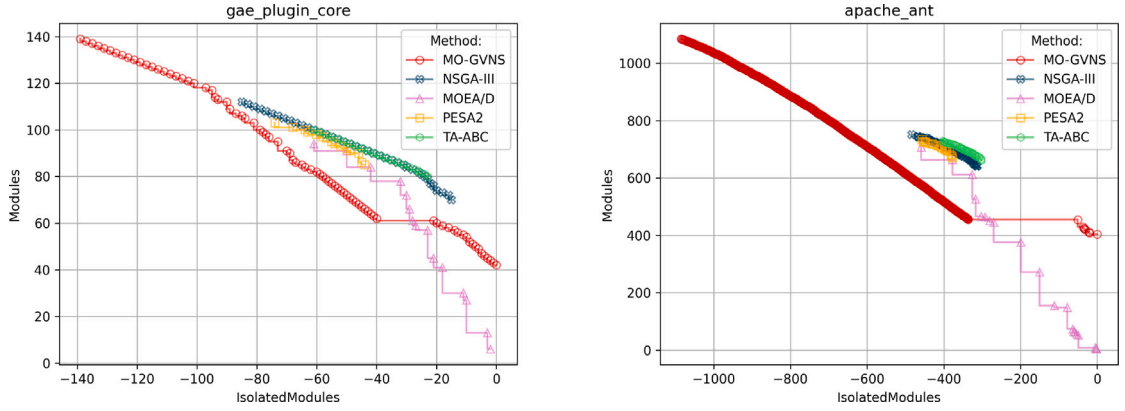


Fig. 9. Non-dominated fronts considering only the number of modules and the number of isolated modules, obtained for instances gae_plugin_core and apache_ant in the MCA problem.

From an algorithmic point of view, the proposed constructive approach, based on the ideas of Path-Relinking, generates a set of initial solutions that are well distributed along the objective space. This presents a good starting point for methods based on local search. Then, exploring each objective function in isolation (i.e., using a specific local search procedure guided by each objective) results in a very effective strategy. However, since this might be very time-consuming, the exploration performed by the method might need to be truncated.

Additionally, when studying multiple objectives simultaneously, we might need to use different neighborhoods with a diverse nature. In this sense, the exploration of the search space has to be performed very carefully, since exploring a particular neighborhood might be useless for some objectives. In this sense, several neighborhoods should be tested and then classified into categories, which could lead to the design of a more effective method, avoiding the exploration of very similar neighborhoods and selecting neighborhoods which complement each other.

Further than the quality of the results we must pay attention to the efficiency of the methods proposed. A more efficient method usually leads to a better exploration and, therefore, to better quality results. The better performance of our proposal with respect others, in terms of computing time, is mainly due to the advanced strategies implemented to improve the efficiency of the method: an efficient evaluation of the objective functions, a reduction of the size of the neighborhoods

depending on the objective function, and an analysis of the objective functions used to guide the search.

7. Conclusions and future work

In this work, we have studied two well-known optimization problems in the context of software quality optimization: the Equal-Size Cluster Approach (ECA) and the Maximizing Cluster Approach (MCA). The problems studied have a practical interest for software developers, who might be keen on reducing the complexity of software projects to improve their understandability. However, addressing optimization tasks in the context of software quality is not straightforward, since software developers often introduce their personal preferences, resulting in a task that is not systematic. Thus, we believe that a multi-objective approach is much better for this task than a single-objective one, since it allows a stakeholder to introduce their subjective experience to select one solution from a set of previously filtered efficient solutions.

The contribution of this work can be divided into three different aspects. First, we have evaluated the performance of a trajectory-based method, based on MO-GVNS, for two multi-objective optimization SMCP problems: MCA and ECA. This is a novel approach for the problems tackled since, as it can be observed in the literature review, the procedures previously proposed for multi-objective SMCP problems are population-based evolutionary algorithms. Specifically, we focused on a methodology which is based on a deep exploration of neighborhood structures, since it has been shown successful for single-objective SMCP problems.

Secondly, we have introduced several strategies in combination with the proposed MO-GVNS algorithm to improve its efficiency, including: a constructive procedure that resembles the ideas of the Path-Relinking methodology to generate the initial set of efficient points; four different shake procedures, which combine random and greedy strategies; a compilation and classification of useful neighborhoods for the MCA and ECA problems; an efficient evaluation of the objectives considered in MCA and ECA; a reduction of the size of the explored neighborhoods for each of the objectives considered; and an analysis of the objectives that better serve as guiding functions during the search phase.

Finally, we favorably compared the performance of the proposed method with the best state-of-the-art algorithms for the MCA and ECA problems. The results showed that the solutions obtained by the MO-GVNS method are better in terms of HV, C, IGD+, PFS and GS. The results obtained indicate that the MO-GVNS method obtains better solutions than those of the state of the art with respect to the quality aspects of convergence, spread, uniformity, and cardinality.

Interestingly, when analyzing the relevance of the objective functions when used to guide the search within the MO-GVNS for the MCA and ECA problems, we found that not considering some of the objectives within the VND components results in a saving of time, at a small quality cost. In particular, we found that not considering coupling, the number of isolated modules, and the difference in size between the largest and the smallest modules to guide VND components resulted in solutions of similar quality than considering all the objectives during the search, while reducing up to 48.13% the computing time of the algorithm.

In multi-objective optimization, the use of evolutionary algorithms is largely extended due to their inherent nature to work with sets of efficient points and they have been proved to be very efficient in this context. On the other hand, trajectory-based search algorithms have not been as popular as population-based methods. However, they could be advantageously combined with evolutionary algorithms to obtain more powerful methods. In this sense, trajectory-based algorithms like MO-GVNS can be used to ensure the intensification role, additionally involving some domain knowledge in the design. Furthermore, due to the general nature of the strategies proposed in this paper, we believe that they can be adapted to improve the results for other multi-objective optimization problems, especially those related to the SMCP.

The study of the software quality optimization through a multi-objective optimization approach allows the stakeholders to introduce their subjective experience in the process. Particularly, developers can select an organization among a set of high-quality solutions. However, we believe that there is still work to be done in the search for the optimal set of objectives that reflect the needs of software developers. Specifically, important aspects such as semantic similarity and history of previous changes should be taken into account. This might be beneficial for the integration of software quality optimization in the Software Development Life Cycle. In future work, we plan to explore the optimization of these objectives in a multi-objective approach.

CRediT authorship contribution statement

Javier Yuste: Conceptualization, Data curation, Investigation, Software, Visualization, Writing – original draft, Writing – review & editing. **Eduardo G. Pardo:** Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Supervision, Validation, Writing – original draft, Writing – review & editing. **Abraham Duarte:** Funding acquisition, Investigation, Project administration, Supervision. **Jin-Kao Hao:** Investigation, Methodology, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. This research has been partially supported by: grants PID2021-125709OA-C22, PID2021-126605NB-I00, funded by MCIN/AEI/10.13039/501100011033, Spain and by "ERDF A way of making Europe"; grant P2018/TCS-4566, funded by the Comunidad de Madrid, Spain and cofinanced by the European Structural Funds ESF and FEDER, Spain; grant CIAICO/2021/224 funded by Generalitat Valenciana, Spain; grant M2988 funded by "Proyectos Impulso de la Universidad Rey Juan Carlos 2022, Spain"; "Cátedra de Innovación *y* Digitalización Empresarial entre Universidad Rey Juan Carlos *y* Second Episode, Spain" (Ref. ID MCA06); and "Red Española de optimización heurística 4.0 digitalización, Spain" (Ref. RED2022-134480-T).

Appendix. Dataset

In this section, we detail the instances contained in the dataset used in the experiments. The dataset is made up of 124 real software instances proposed in previous works (Monçores et al., 2018). These instances are of varying sizes, having between 2 and 1161 vertices and between 2 and 11,722 edges. On average, these instances have 156.37 vertices (with a standard deviation of 216.72) and 948.79 edges (with a standard deviation of 1751.86). In the work where this dataset was proposed for the first time, the instances were divided into four different categories according to their size: 64 small instances (up to 68 vertices), 29 medium instances (from 74 to 182 vertices), 18 large instances (from 190 to 377 vertices), and 13 very large instances (from 413 to 1161 vertices). Following this classification, we present the instances in four different tables: small instances are presented in Table A.18, medium instances are presented in Table A.19, large instances are presented in Table A.20, and very large instances are presented in Table A.21. In each table, we report the number of vertices, the number of edges, and the density of all instances.

Table A.18
Small instances contained in the dataset. These instances have a number of vertices between 2 and 68.

Instance	$\|V\|$	$\|E\|$
squid	2	2
small	6	5
compiler	13	32
random	13	30
regexp	14	20
jstl	15	20
lab4	15	18
netkit-ping	15	15
nss_ldap	15	16
nos	16	52
lslayout	17	43
boxer	18	29
netkit-tftpd	18	23
sharutils	19	36
mtunis	20	57
spdb	21	17
xtell	22	57
bunch	23	62
ispell	24	103
netkit-inetd	24	25
nanoxml	25	64
ciald	26	64
jodamoney	26	102
Modulizer	26	66
bootp	27	75
jxlsreader	27	73
sysklogd-1	28	74
telnetd	28	81
crond	29	112
netkit-ftp	29	95
rcs	29	163
seemp	30	61
dhcpd-2	31	122
cyrus-sasl	32	100
tcsh	32	105
micq	33	156
apache_zip	36	86
star	36	89
bison	37	179
cia	38	185
stunnel	38	97
minicom	40	257
mailx	41	331
dot	42	255
screen	42	292
slang	45	242
slrn	45	323
net-tools	48	183
graph10up49	49	1650
wu-ftpd-1	50	230
joe	51	540
hw	53	51
imapd-1	53	298
wu-ftpd-3	54	278
udt-java	56	227
javaocr	58	155
dhcpd-1	59	571
icecast	60	650
pfcda_base	60	197
servletapi	61	131
php	62	191
bunch2	65	151
forms	68	270

Table A.19
Medium instances contained in the dataset. These instances have a number of vertices between 74 and 182.

Instance	$\|V\|$	$\|E\|$
jscatterplot	74	232
jxlscore	79	330
elm-2	81	683
jfluid	81	315
grappa	86	295
elm-1	88	941
gnupg	88	601
inn	90	624
bash	92	901
jpassword	96	361
bitchx	97	1653
junit	99	276
xntp	111	729
acqCIGNA	114	179
bunch_2	116	364
exim	118	1255
xmldom	118	209
cia++	124	369
tinytim	129	564
mod_ssl	135	1095
jkaryoscope	136	460
ncurses	138	682
gae_plugin_core	139	375
lynx	148	1745
javacc	153	722
lucent	153	103
JavaGeom	171	1445
incl	174	360
jdendogram	177	583
xmlapi	182	413

Table A.20
Large instances contained in the dataset. These instances have a number of vertices between 190 and 377.

Instance	$\|V\|$	$\|E\|$
jmetal	190	1137
graph10up193	193	9190
dom4j	195	930
nmh	198	3262
pdf_renderer	199	629
Jung_graph_model	207	603
jung_visualization	208	919
jconsole	220	859
pfcda_swing	248	885
jml-1.0b4	267	1745
jpassword2	269	1348
notelab-full	293	1349
Poormans_CMS	301	1118
log4j	305	1078
jtreeview	320	1057
bunchall	324	1339
JACE	338	1524
javaws	377	1403

Table A.21
Very large instances contained in the dataset. These instances have a number of vertices between 413 and 1161.

Instance	$\|V\|$	$\|E\|$
swing	413	1513
lwjgl-2.8.4	453	1976
res_cobol	470	7163
ping_libc	481	2854
y_base	556	2510
krb5	558	3793
apache_ant_taskdef	626	2421

(*continued on next page*)

Table A.21 (*continued*).

Instance	$\|V\|$	$\|E\|$
itextpdf	650	3898
apache_lucene_core	738	3726
eclipse_jgit	909	5452
linux	916	11722
apache_ant	1085	5329
ylayout	1161	5770

References

Abdeen, H., Ducasse, S., Sahraoui, H., Alloui, I., 2009. Automatic package coupling and cycle minimization. In: 2009 16th Working Conference on Reverse Engineering. IEEE, pp. 103–112.

Ali, S., Arcaini, P., Pradhan, D., Safdar, S.A., Yue, T., 2020. Quality indicators in search-based software engineering: An empirical evaluation. ACM Trans. Softw. Eng. Methodol. (TOSEM) 29 (2), 1–29.

Amarjeet, Chhabra, J.K., 2017. Harmony search based remodularization for object-oriented software systems. Comput. Lang. Syst. Struct. 47, 153–169.

Amarjeet, Chhabra, J.K., 2018. TA-ABC: two-archive artificial bee colony for multi-objective software module clustering problem. J. Intell. Syst. 27 (4), 619–641.

Arasteh, B., 2023. Clustered design-model generation from a program source code using chaos-based metaheuristic algorithms. Neural Comput. Appl. 35 (4), 3283–3305.

Arasteh, B., Fatolahzadeh, A., Kiani, F., 2022. Savalan: Multi objective and homogeneous method for software modules clustering. J. Softw. Evol. Process 34 (1), e2408.

Arasteh, B., Sadegi, R., Arasteh, K., Gunes, P., Kiani, F., Torkamanian-Afshar, M., 2023. A bioinspired discrete heuristic algorithm to generate the effective structural model of a program source code. J. King Saud Univ. Comput. Inf. Sci. 35 (8), 101655.

Bakota, T., Hegedus, P., Ladanyi, G., Kortvelyesi, P., Ferenc, R., Gyimothy, T., 2012. A cost model based on software maintainability. In: IEEE International Conference on Software Maintenance. ICSM, pp. 316–325.

Barros, M.d.O., 2012. An analysis of the effects of composite objectives in multiobjective software module clustering. In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation. pp. 1205–1212.

Barros, M.d., Farzat, F.d., Travassos, G.H., 2015. Learning from optimization: A case study with Apache Ant. Inf. Softw. Technol. 57, 684–704.

Brandes, U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., Wagner, D., 2007. On modularity clustering. IEEE Trans. Knowl. Data Eng. 20 (2), 172–188.

Brockhoff, D., Zitzler, E., 2009. Objective reduction in evolutionary multiobjective optimization: Theory and applications. Evol. Comput. 17 (2), 135–166.

Cavero, S., Pardo, E.G., Duarte, A., 2022. A general variable neighborhood search for the cyclic antibandwidth problem. Comput. Optim. Appl. 1–31.

Chaves-González, J.M., Pérez-Toledano, M.A., Navasa, A., 2015. Teaching learning based optimization with Pareto tournament for the multiobjective software requirements selection. Eng. Appl. Artif. Intell. 43, 89–101.

Chen, C., Alfayez, R., Srisopha, K., Boehm, B., Shi, L., 2017. Why is it important to measure maintainability and what are the best ways to do it? In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion. ICSE-C, IEEE, pp. 377–378.

Chhabra, J.K., 2018a. FP-ABC: Fuzzy-Pareto dominance driven artificial bee colony algorithm for many-objective software module clustering. Comput. Lang. Syst. Struct. 51, 1–21.

Chhabra, J.K., 2018b. Many-objective artificial bee colony algorithm for large-scale software module clustering problem. Soft Comput. 22 (19), 6341–6361.

Colanzi, T.E., Assunção, W.K., Vergilio, S.R., Farah, P.R., Guizzo, G., 2020. The symposium on search-based software eengineering: Past, present and future. Inf. Softw. Technol. 127, 106372.

Corne, D.W., Jerram, N.R., Knowles, J.D., Oates, M.J., 2001. PESA-II: Region-based selection in evolutionary multiobjective optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO 2001, pp. 283–290.

Croes, G.A., 1958. A method for solving traveling-salesman problems. Oper. Res. 6 (6), 791–812.

Deb, K., Jain, H., 2013. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. IEEE Trans. Evol. Comput. 18 (4), 577–601.

Duarte, A., Pantrigo, J.J., Gallego, M., 2007. Metaheurísticas. Madrid: Dykinson.

Duarte, A., Pantrigo, J.J., Pardo, E.G., Mladenovic, N., 2015. Multi-objective variable neighborhood search: an application to combinatorial optimization problems. J. Global Optim. 63 (3), 515–536.

Durillo, J.J., Nebro, A.J., 2011. jMetal: A Java framework for multi-objective optimization. Adv. Eng. Softw. 42 (10), 760–771.

Gendreau, M., Hertz, A., Laporte, G., 1992. New insertion and postoptimization procedures for the traveling salesman problem. Oper. Res. 40 (6), 1086–1094.

Gil-Borrás, S., Pardo, E.G., Alonso-Ayuso, A., Duarte, A., 2021. A heuristic approach for the online order batching problem with multiple pickers. Comput. Ind. Eng. 160, 107517.

Glover, F., 1997. Tabu search and adaptive memory programming—advances, applications and challenges. In: Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies. Springer, pp. 1–75.

Hansen, P., Mladenović, N., Todosijević, R., Hanafi, S., 2017. Variable neighborhood search: basics and variants. EURO J. Comput. Optim. 5 (3), 423–454.

Harman, M., Mansouri, S.A., Zhang, Y., 2012. Search-based software engineering: Trends, techniques and applications. ACM Comput. Surv. 45 (1), 1–61.

Huang, J., Liu, J., 2016. A similarity-based modularization quality measure for software module clustering problems. Inform. Sci. 342, 96–110.

Huang, J., Liu, J., Yao, X., 2017. A multi-agent evolutionary algorithm for software module clustering problems. Soft Comput. 21 (12), 3415–3428.

Hwa, J., Yoo, S., Seo, Y.-S., Bae, D.-H., 2017. Search-based approaches for software module clustering based on multiple relationship factors. Int. J. Softw. Eng. Knowl. Eng. 27 (07), 1033–1062.

International Organization for Standardization, 2017. ISO/IEC/IEEE 24765:2017 Systems and software engineering — Vocabulary.

Ishibuchi, H., Masuda, H., Nojima, Y., 2015. A study on performance evaluation ability of a modified inverted generational distance indicator. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 695–702.

Izadkhah, H., Elgedawy, I., Isazadeh, A., 2016. E-CDGM: an evolutionary call-dependency graph modularization approach for software systems. Cybern. Inf. Technol. 16 (3).

Izadkhah, H., Tajgardan, M., 2019. Information Theoretic Objective Function for Genetic Software Clustering. In: Multidisciplinary Digital Publishing Institute Proceedings, Vol. 46. p. 18.

Jalali, N.S., Izadkhah, H., Lotfi, S., 2019. Multi-objective search-based software modularization: structural and non-structural features. Soft Comput. 23 (21), 11141–11165.

Jeet, K., Dhir, R., 2016. Software module clustering using hybrid socio-evolutionary algorithms. Int. J. Inf. Eng. Electron. Bus. 8 (4), 43.

Kargar, M., Isazadeh, A., Izadkhah, H., 2017. Semantic-based software clustering using hill climbing. In: 2017 International Symposium on Computer Science and Software Engineering Conference. CSSE, IEEE, pp. 55–60.

Köhler, V., Fampa, M., Araújo, O., 2013. Mixed-integer linear programming formulations for the software clustering problem. Comput. Optim. Appl. 55 (1), 113–135.

Kumari, A.C., Srinivas, K., 2016. Hyper-heuristic approach for multi-objective software module clustering. J. Syst. Softw. 117, 384–401.

Lai, X., Hao, J.-K., 2016. Iterated variable neighborhood search for the capacitated clustering problem. Eng. Appl. Artif. Intell. 56, 102–120.

Li, Z., Avgeriou, P., Liang, P., 2015. A systematic mapping study on technical debt and its management. J. Syst. Softw. 101, 193–220.

Li, M., Chen, T., Yao, X., 2020. How to evaluate solutions in Pareto-based search-based software engineering: A critical review and methodological guidance. IEEE Trans. Softw. Eng. 48 (5), 1771–1799.

Li, M., Yao, X., 2019. Quality evaluation of solution sets in multiobjective optimisation: A survey. ACM Comput. Surv. 52 (2), 1–38.

Mahdavi, K., 2005. A Clustering Genetic Algorithm for Software Modularisation with a Multiple Hill Climbing Approach (Ph.D. thesis). Brunel University, UK.

Mamaghani, A.S., Hajizadeh, M., 2014. Software modularization using the modified firefly algorithm. In: 2014 8th. Malaysian Software Engineering Conference. MySEC, IEEE, pp. 321–324.

Mamaghani, A.S., Meybodi, M.R., 2009. Clustering of software systems using new hybrid algorithms. In: 2009 Ninth IEEE International Conference on Computer and Information Technology, vol. 1, IEEE, pp. 20–25.

Mancoridis, S., Mitchell, B.S., Rorres, C., Chen, Y.-F., Gansner, E.R., 1998. Using automatic clustering to produce high-level system organizations of source code. In: 6th International Workshop on Program Comprehension. IWPC'98, IEEE, pp. 45–52.

Martín-Santamaría, R., Cavero, S., Herrán, A., Duarte, A., Colmenar, J.M., 2022. A practical methodology for reproducible experimentation: an application to the Double-row Facility Layout Problem. Evol. Comput. 1–35.

Mitchell, B.S., 2002. A Heuristic Search Approach to Solving the Software Clustering Problem (Ph.D. thesis). Drexel University, USA, AAI3039424.

Mitchell, B.S., Mancoridis, S., 2002. Using heuristic search techniques to extract design abstractions from source code. In: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation. pp. 1375–1382.

Mitchell, B.S., Mancoridis, S., 2006. On the automatic modularization of software systems using the bunch tool. IEEE Trans. Softw. Eng. 32 (3), 193–208.

Mitchell, B.S., Mancoridis, S., 2008. On the evaluation of the Bunch search-based software modularization algorithm. Soft Comput. 12 (1), 77–93.

Mkaouer, W., Kessentini, M., Shaout, A., Koligheu, P., Bechikh, S., Deb, K., Ouni, A., 2015. Many-objective software remodularization using NSGA-III. ACM Trans. Softw. Eng. Methodol. (TOSEM) 24 (3), 1–45.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Comput. Oper. Res. 24 (11), 1097–1100.

Molnar, A.-J., Motogna, S., 2021. A study of maintainability in evolving open-source software. In: Evaluation of Novel Approaches To Software Engineering: 15th International Conference, ENASE 2020, Prague, Czech Republic, May 5–6, 2020, Revised Selected Papers 15. Springer, pp. 261–282.

Monçores, M.C., Alvim, A.C.F., Barros, M.O., 2018. Large neighborhood search applied to the software module clustering problem. Comput. Oper. Res. 91, 92–111.
Moosavi, S.H.S., Bardsiri, V.K., 2017. Satin bowerbird optimizer: A new optimization algorithm to optimize ANFIS for software development effort estimation. Eng. Appl. Artif. Intell. 60, 1–15.
Mu, L., Sugumaran, V., Wang, F., 2020. A hybrid genetic algorithm for software architecture re-modularization. Inf. Syst. Front. 22 (5), 1133–1161.
Pantrigo, J.J., Martí, R., Duarte, A., Pardo, E.G., 2012. Scatter search for the cutwidth minimization problem. Ann. Oper. Res. 199, 285–304.
Pardo, E.G., García-Sánchez, A., Sevaux, M., Duarte, A., 2020. Basic variable neighborhood search for the minimum sitting arrangement problem. J. Heuristics 26, 249–268.
Perez-Pelo, S., Sanchez-Oro, J., Gonzalez-Pardo, A., Duarte, A., 2021. A fast variable neighborhood search approach for multi-objective community detection. Appl. Soft Comput. 112, 107838.
Pinto, A.F., Alvim, A.C.F., Barros, M.O., 2014. ILS for the Software Module Clustering Problem. pp. 1972–1983, XLVI Simpósio Brasileiro de Pesquisa Operacional. Salvador:[sn].
Pourasghar, B., Izadkhah, H., Isazadeh, A., Lotfi, S., 2021. A graph-based clustering algorithm for software systems modularization. Inf. Softw. Technol. 133, 106469.
Praditwong, K., 2011. Solving software module clustering problem by evolutionary algorithms. In: 2011 Eighth International Joint Conference on Computer Science and Software Engineering. JCSSE, IEEE, pp. 154–159.
Praditwong, K., Harman, M., Yao, X., 2011. Software module clustering as a multi-objective search problem. IEEE Trans. Softw. Eng. 37 (2), 264–282.
Prajapati, A., 2022. Software module clustering using grid-based large-scale many-objective particle swarm optimization. Soft Comput. 1–22.
Prajapati, A., Chhabra, J.K., 2018. A particle swarm optimization-based heuristic for software module clustering problem. Arab. J. Sci. Eng. 43 (12), 7083–7094.
Prajapati, A., Chhabra, J.K., 2020. Information-theoretic remodularization of object-oriented software systems. Inf. Syst. Front. 22, 863–880.
Ramirez, A., Romero, J.R., Ventura, S., 2018. Interactive multi-objective evolutionary optimization of software architectures. Inform. Sci. 463, 92–109.
Ramirez, A., Romero, J.R., Ventura, S., 2019. A survey of many-objective optimisation in search-based software engineering. J. Syst. Softw. 149, 382–395.
Sarhan, Q.I., Ahmed, B.S., Bures, M., Zamli, K.Z., 2020. Software module clustering: An in-depth literature analysis. IEEE Trans. Softw. Eng. 48 (6), 1905–1928.
Shi, Y., Liu, W., Zhou, Y., 2023. An adaptive large neighborhood search based approach for the vehicle routing problem with zone-based pricing. Eng. Appl. Artif. Intell. 124, 106506.
Tajgardan, M., Izadkhah, H., Lotfi, S., 2016. Software systems clustering using estimation of distribution approach. J. Appl. Comput. Sci. Methods 8, 99–113.
Wang, S., Ali, S., Yue, T., Li, Y., Liaaen, M., 2016. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In: Proceedings of the 38th International Conference on Software Engineering. pp. 631–642.
Yuan, Y., Ong, Y.-S., Gupta, A., Xu, H., 2017. Objective reduction in many-objective optimization: evolutionary multiobjective approaches and comprehensive analysis. IEEE Trans. Evol. Comput. 22 (2), 189–210.
Yuste, J., Duarte, A., Pardo, E.G., 2022a. An efficient heuristic algorithm for software module clustering optimization. J. Syst. Softw. 190, 111349.
Yuste, J., Pardo, E.G., Duarte, A., 2022b. Multi-objective variable neighborhood search for improving software modularity. In: International Conference on Variable Neighborhood Search. Springer, pp. 58–68.
Yuste, J., Pardo, E.G., Duarte, A., 2022c. Variable neighborhood descent for software quality optimization. In: Metaheuristics International Conference. Springer, pp. 531–536.
Yuste, J., Pardo, E.G., Duarte, A., 2024. General variable neighborhood search for the optimization of software quality. Comput. Oper. Res. 106584.
Zhang, Q., Li, H., 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. IEEE Trans. Evol. Comput. 11 (6), 712–731.