A Search-Based Approach to Multi-view Clustering of Software Systems

Amir M. Saeidi, Jurriaan Hage, Ravi Khadka, Slinger Jansen
Department of Information and Computing Sciences, Utrecht University, The Netherlands
{a.m.saeidi, j.hage, r.khadka, slinger.jansen}@uu.nl

***Abstract*—Unsupervised software clustering is the problem of automatically decomposing the software system into meaningful units. Some approaches solely rely on the structure of the system, such as the module dependency graph, to decompose the software systems into cohesive groups of modules. Other techniques focus on the informal knowledge hidden within the source code itself to retrieve the modular architecture of the system. However both techniques in the case of large systems fail to produce decompositions that correspond to the actual architecture of the system. To overcome this problem, we propose a novel approach to clustering software systems by incorporating knowledge from different viewpoints of the system, such as the knowledge embedded within the source code as well as the structural dependencies within the system, to produce a clustering. In this setting, we adopt a search-based approach to the encoding of multi-view clustering and investigate two approaches to tackle this problem, one based on a linear combination of objectives into a single objective, the other a multi-objective approach to clustering. We evaluate our approach against a set of substantial software systems. The two approaches are evaluated on a dataset comprising of 10 Java open source projects. Finally, we propose two techniques based on interpolation and hierarchical clustering to combine different results obtained to yield a single result for single-objective and multi-objective encodings, respectively.**

I. Introduction

Unsupervised software clustering involves organizing software units into clusters in an automatic fashion. Clusters represent high-level component abstractions of a system's organization. Clustering can not only be used to partition the system but also determine how much the architecture of the system has drifted as the system evolves from one version to the next [1]. In search-based approaches to clustering, the system decomposition is encoded as a search-based optimization problem, in which case what it means by a good decomposition is expressed in terms of an objective function [2].

There is a set of well-known coupling-based heuristics used to modularize software systems: 1) structure-based such as structural dependencies [3], [4], [5], [6] and file organization [7], 2) lexical-based [8], [9], [10], and 3) evolutionary-based [11]. Albeit these principles play an important factor in modularity, there appears to be no consensus as to what is the main deriving force in modularization. Beck et al. [12] investigate the congruence between these coupling concepts and the modularization of systems in the context of 16 open source projects and observe that *"none of the principles is exclusively dominating the modularity of the studied systems"*. However, the general observation [12], [13] is that amongst the coupling metrics for remodularization, the ones based on structural dependencies and conceptual metrics stand out. Hence, we have restricted ourselves to module dependency and lexical information viewpoints, however our method can be extended to include other viewpoints.

Most remodularizations are guided by structural dependencies [6]. An intrinsic problem with structural-based approaches is the fundamental assumption that the systems follow some software engineering principles for good design. First, many software systems have not followed these principles in their original design and were ill-formed from their conception. Second, due to the continuous evolution of software systems, the architecture of the system may deteriorate to a point where the original architecture becomes a "spaghetti" architecture. As a result, clustering solely based on the structural aspects of the software system fails to produce clusters that reflect the intended decomposition of the system. Glorie et al. [14] reported that applying structural techniques to systems in the industrial domain give results that are only partially acceptable to the people who are familiar with the architecture of the system.

On the other hand, lexical-based clustering exploits the knowledge embedded within the source code, such as comments and identifier names, to modularize the software system. There are several drawbacks to this approach. The first and foremost disadvantage is that since this approach extracts knowledge from identifier names and comments, its success heavily depends on the quality of the source code naming. Not all software systems, especially in the case of legacy systems, are written with good habits of naming in mind. Obsolete and useless comments make this task even more complicated. Second, clusterings that only rely on lexical information [8] are shown to emit results that do not correspond to the intended architecture. These findings strengthen the desire to produce a modularization by feeding-in different sources of knowledge about the system, in the hope of complementing each other.

Recent work [15], [16], [17], [18] suggest that incorporating different coupling measurements better captures the developers' perception of modularity of a software system. We aim to build on these findings to tackle the problem of unsupervised multi-view clustering in a systematic fashion. In this setting, not only the topology and weight of dependencies are considered but also the application and domain knowledge of the system are used to bias the results. Our fundamental hypothesis here is that each viewpoint of the system whether it is structural dependencies or the lexical similarity, gives a different abstraction of the same problem, and together they can provide a better solution to the problem. Our primary interest here is how much the produced partitioning resembles the authoritative decomposition or gold standard. Therefore,

the clustering result is evaluated based on its authoritativeness [11].

Our contributions in this paper are as follows:

- We present two different techniques to combine different sources of knowledge, one based on a linear combination of objective functions, and the other using a multi-objective formulation.
- We exploit interpolation and hierarchical clustering to merge results obtained from the aforementioned techniques to attain a single satisfactory solution.
- A comparative study of the two techniques is performed for a set of 10 relatively large open source Java projects.

This paper is structured as follows. We first give an overview of the current status of research in clustering software systems in the Section II. In Section III we motivate our approach by outlining our ideas and describe how we will materialize the multi-view clustering using both a linear combination encoding as well as a multi-objective approach. We proceed by introducing a benchmark for validating our approach as well as the implementation of our tooling for empirical evaluation in Section IV. In Section V, we give the results obtained by applying both of the techniques, and discuss our findings. In this section, we proceed by proposing two approaches to combine different results obtained to emit a single but yet satisfactory decomposition for any system under analysis. Finally, in Section VI we conclude and outline future work.

II. Related Work

Related work is organized into three categories: 1) structural-based approaches to clustering, 2) lexical-based approaches to clustering, and 3) hybrid approaches incorporating different sources of knowledge.

A. Structural-Based Approaches to Clustering

Most software clustering techniques use structural dependencies to approach the clustering problem. Mancoridis et al. [19] were the first to encode the clustering problem in terms of a search problem and introduced the notion of Modularity Quality (MQ) objective function to emit ”better” modularizations. MQ is defined as a trade-off between inter-connectivity and intra-connectivity. They have developed a toolset called Bunch [19] to perform structural-based clustering. Bunch provides a wide range of optimal and sub-optimal search-based algorithms including hill climbing, simulated annealing and genetic algorithms. A module dependency graph is all that is needed to perform clustering.

Praditwong et al. [4] demonstrated that formulating the low-coupling and high-cohesion objective in terms of a multi-objective optimization can significantly improve the MQ objective. They use the Pareto optimal multi-objective formulation to produce results that are far better for both weighted and unweighted graphs compared to single-objective search problems, however this improvement comes at a higher computational cost.

Bunch’s cohesion/coupling metric is evaluated in the context of a real-world case study, the Eclipse platform, by Anquetil and Laval [6]. The observation they made based on three restructurings of Eclipse framework was that after each restructuring, Bunch cohesion not only did not increase but the packages showed increased coupling. The conclusion they drew was that structural metrics alone may not be sufficient to extract the architecture of industrial projects.

B. Lexical-Based Approaches to Clustering

Kuhn et al. [8] propose *semantic clustering* by using Latent Semantic Indexing (LSI) to not only partition the system into clusters based on the use of similar vocabulary, but also derive a set of common linguistic topics within each cluster. They evaluate their approach using two real-world case studies, JBoss and JEdit to find out that the partition of a system based on lexical similarity does not necessarily correspond to its structural architecture. They have observed that many topics cross-cut through many packages.

To overcome the aforementioned problem, Corazza et al. [9] introduce the concept of zones (e.g., comments and class names) in the source code to assign different importance to the information extracted from different zones. To automatically give weights to these zones, they use a probabilistic model and apply the Expectation Maximization (EM) algorithm to derive values for these weights. In a followup paper [10], the authors build on their previous findings to further improve their approach. The modified approach is validated against 13 open source Java software systems.

The semantic clustering and conceptual metrics are adopted by Santos et al. [13] to investigate whether the vocabulary used in the source code can explain the intention behind reorganization of the architecture of a software system. They evaluate their approach using six real-world remodularizations of four software systems and conclude that developers indeed re-organize packages based on a common lexical concept.

C. Multi-view Clustering Approaches

The aforementioned techniques only rely on one viewpoint of software for modularization. One of the earliest works in the field of multi-view clustering belongs to that of Maletic and Marcus [20] where they explore the use of lexical and structural information to better understand systems for reverse engineering purposes. They use LSI to analyze semantic relationships between files and cluster them based on their similarity. Structural information such as file organization is then used to assess the semantic cohesion of the clusters and files. They demonstrate how their approach can be utilized to better understand a system called Mosaic.

Scanniello et al. [15] present a phased clustering approach to produce a hierarchical decomposition of object-oriented software systems. In this approach, the structural information is used to break up the system into horizontal layers, while each layer is partitioned into groups based on lexical similarity.

Patel et al. [16] describe a two-phase approach to combining different sources of information to decompose the software system. The proposed approach involves using dynamic analysis to build a core skeleton decomposition of the

system, followed by utilizing the static dependencies to add the remaining non-core components to the skeleton structure.

Bavota et al. [18] combine semantic and structural coupling measures to restructure packages to improve cohesion amongst classes within the package. In their approach, both the structural and semantic relationships between classes in a package are used to identify what they call chains of strongly related classes. The identified chains are then used to restructure the package. The observation they have made is that combining the latent topics and structural dependencies help give more meaningful recommendations.

III. Multi-view Clustering

The goal of multi-view clustering is to combine knowledge from different sources of information to yield a partition that "better" exposes the high-level decomposition of the system. In other words, each view contains unique information about the high-level configuration of the system, which together can help emit a more precise representation. The main hypothesis here is that there is a gain to be made when combining knowledge from different viewpoints.

To approach this problem in a systematic way, we will encode the problem as a search problem, where the goal is to find a solution, possibly more than one, that satisfies the aforementioned requirements. We employ two techniques, one based on a linear combination of different objective functions, while the second one relies on a multi-objective approach to clustering. The former approach tries to quantify what it means for a partition to be good, while the latter takes a more qualitative taking on what the good partitions for a system may be.

The main objective of search-based multi-view clustering is to find a trade-off between different views, while maximizing objective functions in each respective view. Each function is designed to maximize a heuristic in each view, such as the low-coupling, high-cohesion principle in the structural view. In the multi-view setting, the combined function in both encodings implicitly *maximizes agreement* between different views, while emboldening evident but yet distinct subpartitions in each view. The difference in the encodings stems from how the differences the between views are resolved. In the case of linear encoding, the function with higher weight has a bigger say in what cluster should any module belong to. As for the multi-objective encoding, the difference is resolved by admitting any different partitioning as a solution.

A. Linear Combination of Single-objectives Approach

To encode two objective functions in terms of one, we have to compose them in some ways. In our approach, we use linear combination to combine the functions into a single function. The composed function is parameterized over some weight parameter which denotes the significance of one function compared to the others. We assume the total weight of all functions contributing to the composed function is one. The formal definition of the encoding for the linear combination is defined below.

For n arbitrary objective functions $f_i(x)$ with respective weight λ_i, we define their linear combination $F(x)$ as follows:

$$F(x) = \sum_{i=1}^{n} \lambda_i . f_i(x) \tag{1}$$

$$\textit{s.t.} \sum_{i=1}^{n} \lambda_i = 1$$

B. Multi-objective Approach

Contrary to the single objective optimization, in multi-objective problems, there exists a set of solutions known as Pareto optimal solutions that dominate other non-optimal solutions. Under the Pareto interpretation of combined objective functions, no overall objective improvement occurs unless the solution is better according to at least one of the individual objective functions and no worse according to the others. The formal definition of the Pareto interpretation of combined objective functions, given in [4], is as follows:

$$F(x_1) > F(x_2) \tag{2}$$

$$\forall i . f_i(x_1) \geq f_i(x_2) \wedge \exists j . f_j(x_1) > f_j(x_2)$$

$F(x_i)$ denotes a function evaluating the fitness of some solution x_i in the Pareto front interpretation, comprising a set of functions $f_{i,j}$. The optimal solution to the single-objective encoding of the multi-objective problem using linear combination is a subset of its optimal Pareto front in the multi-objective setting. However, the major difference is that the linear-combination encoding usually consists of exactly one solution, whereas in the Pareto front formulation, the solution space is much larger, by a magnitude of 10, if not more.

C. Formulation of the Multi-view Clustering Problem

Now that we have described our approaches to tackling the multi-view clustering, we need to encode the problem in terms of a search problem. To formulate the problem in terms of a search problem, we have to define the representation and the objective function [2]. Since we have limited ourselves to structural modularity and lexical similarity, in the following subsections we describe the representation of problem for each viewpoint as well as its objective function.

1) Structural Modularity: There has been a considerable amount of work ([3], [4], [5], [6]) on partitioning systems based on structural properties. A design principle used for partitioning graphs is the low-cohesion, high-coupling principle. In this setting, a module dependency graph needs to be constructed to represent dependencies between modules within the system. We rely on source code analysis tools to construct such a graph. The MDG is then transformed into an adjacency matrix before it can be fed into the search algorithms.

The MQ objective function originally introduced in [19] was designed to balance the trade-off between the cohesion amongst the modules within a cluster and coupling between the modules in different clusters. For the sake of convenience and understandability, we have included the formal definition of MQ, given in equation (3). The measurement of MQ is given as the sum of the cluster factors (CF) for each cluster. In this equation, the internal edges of a cluster are denoted as

μ_i, whereas the inter-edges between two distinct clusters i and j are represented as $\epsilon_{i,j}$ and $\epsilon_{j,i}$, respectively.

$$MQ = \sum_{i=1}^{k} CF_i,\ CF_i = \begin{cases} 0 & \mu_i = 0 \\ \frac{2\mu_i}{2\mu_i + \sum_{j=1, i\neq j}^{k} (\epsilon_{i,j} + \epsilon_{j,i})} & otherwise \end{cases} \quad (3)$$

2) Lexical Similarity: We have closely followed the information retrieval method, Latent Semantic Indexing (LSI) [21] as outlined in Kuhn et al. [8] for software analysis to build a lexical similarity representation of the system. LSI computes the linguistic similarity between source code units by examining the distribution of co-occurring terms in documents (topics). LSI helps to overcome problems such as synonymy and by a lesser extent, polysemy[1] in the document by approximating the document-term matrix with a lower rank matrix. Reducing the noise (synonymy and polysemy) helps reveal the concepts in the system by bringing out the latent meaning of term occurrences.

The number of dimensions to approximate the document-term matrix is problem-specific. In natural-language corpa where the corpus may contain millions of documents, the document-term matrix is reduced to $200 - 500$ dimensions. In software analysis where the number of documents are much smaller and only range in the hundreds, as in the case of our evaluation benchmark, the number of dimensions can be as low as 10. In this paper, the number of dimensions used to approximate the document-term matrix is $r = (m * n)^{0.2}$, as suggested by Kuhn et al. [8], where m and n are the number of terms and documents, respectively.

LSI takes as input a weighted bag-of-words, representing the set of documents and the term occurrences normalized to balance out rare and common terms, to break it down into less dimensions. To build the corresponding bag-of-words, we apply a preprocessor to extract the vocabulary terms from the comments and identifier names in the source code. The preprocessor begins by breaking up all the composite terms found in the source code. We proceed by eliminating common terms that occur in a natural language such as "the", "a" and "an". Furthermore, all lexical items are normalized by applying a stemmer to emit a common radix. The result of this process is what is called a "bag-of-words", containing all occurring terms in the system with its frequency in each source document. One downside of the *bag-of-words* approach is that it gives the same weight to each term in the document, no matter where the term occurs in the document. For example, the term "interest" used in a class name should be treated differently from its occurrence in a comment within the same class. To overcome this problem, Corazza et al. [9] introduce a probabilistic model where they assign different importance to the information extracted from different zones (e.g. comments and class names) in the source code. For the weights associated for each document-term pair, we use the most widely adopted technique in information retrieval field, namely the *term frequency-inverse document frequency* ($tf - idf$).

The produced normalized bag-of-words can be fed into LSI to break it down into fewer dimensions, while preserving as much information about the distances between the documents. The similarity between the modules is computed using the the cosine of the angle between their corresponding feature vectors. Figure 1 illustrates the similarity matrix produced from raw and reduced document-term matrices for one of the case studies (Apache Ant). Due to the noise in the original document-term matrix, the relationships are not visible but by taking synonymy and polysemy into account, it is possible to bring out the latent conceptual relationships between the modules.

[1]LSI cannot truly capture polysemy, however it helps by eliminating less-likely meanings of words in the corpus.

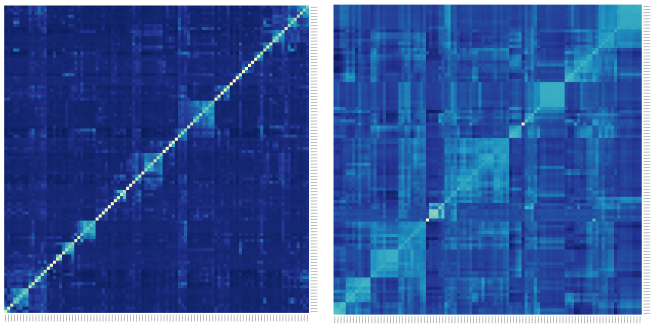


Fig. 1. The similarity matrices for Apache Ant: the one on left denotes the similarity matrix computed from raw document-term matrix; the right heatmap depicts the similarity matrix of the reduced document-term matrix

Most of the works in clustering (dis)similarity matrices are based on agglomerative hierarchical clustering, however, as stated above, to be able to encode this problem in terms of a search problem, we also need to define an objective function to find highly similar partitions. Hence, we propose an objective function, hereinafter referred to as *Conceptual Quality* (CQ), to bring together modules that are conceptually similar. The goal of this function is to maximize the lexical similarity between the modules within a cluster, while minimizing the dissimilarity among those modules with the modules that do not belong to that cluster.

The definition of CQ objective function is highly influenced by the definition of MQ function, originally introduced by Mancoridis et al. [19]. Similar to MQ, the CQ measurement for a clustering with k partitions is defined as the sum of the cluster factors (CF) for each cluster. The intra-lexical similarity measurement μ_i is defined as the similarity between modules within cluster i. On the other hand, $\epsilon_{i,j}$ denotes the inter-lexical similarity from modules within cluster i to j. Due to the symmetricity of similarity between modules, only the weight of outgoing edges are used in calculating the CQ.

$$CQ = \sum_{i=1}^{k} CF_i,\ CF_i = \begin{cases} 0 & \mu_i = 0 \\ \frac{\mu_i}{\mu_i + \sum_{j=1, i\neq j}^{k} (\epsilon_{i,j})} & otherwise \end{cases} \quad (4)$$

IV. Empirical Evaluation

A. The Dataset

We have created a benchmark for 10 open source Java-based software systems. Our choice for these systems is driven by the following factors: 1) size of the system, and 2) whether

a ground truth can be established, based on which the quality of the produced clusters can be measured.

- Apache Ant: A tool for automated software building.
- Apache Hadoop: A framework for distributed computing.
- Apache Log4j: A logging library for Java.
- Eclipse JDT Core: The infrastructure of Java Development Tools for Eclipse IDE.
- JDOM: A library for reading, manipulating, and outputting XML data from within Java code.
- JEdit: A text editor for programming.
- JFreeChart: A GUI framework for creating interactive and non-interactive charts.
- JHotDraw: A GUI framework for structured drawing editors.
- JUnit: A unit testing framework for Java.
- Weka: A collection of machine learning algorithms for data mining.

We have used the Chord program analysis framework for Java [22] to create the call dependency graph for the software systems. Chord offers different options for performing points-to and call-graph constructions. We have opted for the context-insensitive (0CFA) analysis to output the call graph information. As part of the analysis, the entry points must be specified. For systems with multiple entry points, a single class (through a set of stubs) is created to emulate single entry to the system. As such, the completeness of the produced call graph is constrained by our knowledge about possible entry points through different running configurations of the software, whether it is a plugin, an applet, an application, etc.

The only classes included are the ones deemed reachable through the entry point(s). Furthermore, the test suites and deprecated classes are not included in the dependency graph. The abstract classes as well as the interfaces are omitted.

Table I gives the dataset used for our evaluation. As stated before, only a subset of the classes for which a dependency graph could be constructed are included here. The number of lines of code (KLoC) corresponds to source files that were included. The benchmarks and results obtained are publicly available[2] .

TABLE I. THE BENCHMARK FOR EMPIRICAL EVALUATION OF MULTI-VIEW CLUSTERING APPROACHES

System	Version	Classes	Included Classes	KLOC	NoC
Apache Ant	1.9.3	691	96	31.07	15
Apache Hadoop	0.20.2	707	361	102.53	33
Apache Log4j	1.2.17	218	62	10.54	8
Eclipse JDT Core	3.8	1276	356	162.59	14
JDOM	2.0.5	140	63	21.93	8
JEdit	5.1.0	536	333	112.58	29
JFreeChart	1.2.0	655	210	100.05	25
JHotDraw	7.0.6	284	68	12.01	13
JUnit	4.12	167	69	6.09	13
Weka	3.6.11	1346	448	203.51	50

[2]www.servicifi.org/saner

The project Apache Hadoop, Apache Log4j and JDOM are evaluated for the first time in this paper. The projects JEdit, Ant, EasyMock, JUnit, JHotdraw, JFreeChart and Eclipse JDT Core were evaluated in the context of lexical-based clustering techniques [9], [8]. Scanniello et al. [15] apply their techniques to decompose JHotDraw and JFreeChart into hierarchical and vertical layers. Furthermore, as one of the case studies in [18], JHotDraw is used to restructure its package structure to improve cohesion. Patel et al. [16] combine static and dynamic analysis to derive the architectural decomposition of the Weka toolset.

It is rather difficult to make quantitative comparison between various techniques noted in the literature due to the following reasons: 1) incompatibility of case studies used, such as different versions of software, 2) the ground truth for many case studies are established subjectively, and 3) no standard evaluation metric exists for quantitative evaluation of produced results.

Assessing the authoritativeness of produced clusters requires 1) establishing a ground truth or an authoritative decomposition against which the quality of produced clusters can be evaluated, and 2) an evaluation metric to quantify what it means for a cluster to be of high quality when evaluated against the established ground truth.

Computing the authoritative decomposition or gold standard for a software system is not an easy task. It is only the developer or an expert familiar with the architecture of the system who can arrange the software units into partitions that correspond to the architecture of the system. Since we are not qualified to do this rather difficult task, we have adopted the approach used in works [9], [10], [11] to infer such partitions. This technique can be applied with a relative amount of certainty to extract the high-level decomposition of any well-engineered software system. The extraction process exploits the source folder structure of the system by recursively iterating over internal packages and putting together the modules that belong to the same package. The number of clusters (NoC) for each project as extracted from the package structure is given in Table I.

As for evaluating the authoritativeness of produced clusters, we have adopted the MoJo distance metric originally proposed by Tzerpos and Holt in [23]. The MoJo distance from partition A to B is defined as the minimum number of moves and joins required to reproduce the partition B. We have implemented the algorithm given by Wen and Tzerpos in [24] that calculates the exact MoJo distance in polynomial time. The MoJo distance is normalized over the number of modules N in terms of a similarity metric called $MoJoSim$. The higher the value of MoJoSim similarity, the more authoritative the produced clusters are. $MoJoSim$ is defined as below:

$$MoJoSim(A, B) = 1 - \frac{MoJo(A, B)}{N} \quad (5)$$

B. Implementation

As part of our Gelato toolset [25] for facilitating program comprehension and transformation for legacy software systems, we have created a package for both information retrieval techniques and local search algorithms. The infrastructure of the package used in this paper is implemented in the R

programming language[3]. Due to performance reasons, we have rewritten key functions such as evaluating objective functions in C++, using the Rcpp package[4]. To further improve performance, we have exploited parallel multi-core computation in our search algorithms.

C. Experimentation Setup

We have followed a set of strict guidelines to perform the experiments. The setup used to conduct the experiments for each of the approaches is as follows:

1) Linear combination of Single-objectives: In all studies performed so far on single-objective module clustering, the hill climbing algorithm has shown to outperform other search algorithms both in terms of the quality of clusters produced as well as running times [5]. Therefore, we will use the hill climbing algorithm to conduct the experiments for the single-objective encoding of multi-view clustering.

The hill climber developed and used in this paper implements the first-ascent, nearest-ascent and steepest ascent strategies. For the experimentation, we have employed the nearest ascent strategy, generating and evaluating at each step 20% of all possible neighbors. We have employed a two-phased search strategy involving an extensive search followed by a refined search. During the first phase, for a system of size N, a random population of size $0.2N$ is generated for intervals ranging over the number of possible clusters for the dependency graph. In the hope of further improving the results, the resulting partitions are ordered based on their quality. Based on the number of clusters of top 5 partitions, a random population of size 10 is generated for those in the vicinity of best results.

The weight λ for the linear combination encoding ranges from 0 to 1, where 0 yields a partition with the local best CQ value, and 1 gives the clustering with highest MQ. For the experiments, we have chosen λ over the intervals of 0.2, including the 50% mark.

2) Multi-objective: For the multi-objective optimization, we have implemented the two-archive genetic algorithm designed by Praditwong and Yao [26]. Before running the genetic algorithm on the case studies, it has to be tuned with the choice of parameters for the mutation rate, crossover probability and population size that work best for this problem domain. We have followed the genetic encoding employed in [4] to tune our algorithm. For a system of size N, the population size is equal to $10N$. The mutation rate is set to be $0.004 \log_2 N$. The maximum number of iterations is chosen to be $200N$, however, if no updates are made to the archives after 100 iterations, that is no generated offspring can dominate any members in any of the two archives, the algorithm terminates. The ratio of selecting parents from converging archive to the diversity archive is chosen to be $9 : 1$. Each experiment is run 3 times, and the one yielding the best result with respect to the MoJo distance is chosen as the solution front for the multi-objective approach.

[3]http://www.r-project.org/

[4]http://www.rcpp.org/

V. Findings and Discussions

In this section, we present our findings and give a comparison of different approaches to solve the multi-view clustering problem. We proceed by introducing a novel technique to move from a set of possibly good solutions to a single representative solution which can be used for unsupervised clustering of software systems.

A. Linear Combination Encoding

Table II depicts the results obtained from applying hill climbing algorithm to the linear combination encoding of the multi-objective problem. The observations made through conducting the experiments indeed confirm that incorporating knowledge from different viewpoints leads to results which are more desirable compared to the least-informative view. In case of JDOM, results show that up to 20% can be gained from performing multi-view clustering, when compared to the least-informative view. In case of Apache Log4j, JUnit, JFreeChart and Weka, multi-view clustering shows up to 5% improvement over the best of single-view cluster analyses, from authoritative perspective. In the rest of the cases, CQ metric produced the best results. The blue curve in Figure 2 illustrates the results for MoJo similarity of projects Apache Ant and JUnit plotted against different values of weights λ. The multi-objective formulation results in improvement in MoJo similarity for JUnit project, obtaining the best result at around 20% mark. No improvement is made in the case of Apache Ant, where the best solution is produced when only CQ metric was influencing in producing the partitioning.

B. Multi-objective Encoding

Table III give the results for running the multi-objective two-archive optimization algorithm on the case studies. The number of solutions in Pareto front obtained range from 35 to more than 200 solutions. The number of clusters as well as the similarity metrics are given for different cases. Figure 3 depicts the scatterplots for the non-dominated solution space attained for Apache Ant and JUnit case studies. The scatterplots clearly capture the concavity of the Pareto front on both cases. To visualize the quality of solutions, we map them to gray values: the lighter, the more similar to the reference decomposition. The best solutions are annotated with a red square.

To make direct comparison between the results obtained from the aforementioned encodings, we have scaled the multi-objective Pareto front to a single coordinate system, while preserving their relative distance in the original space. Hence, we have been able to plot the element-wise MoJo similarity of the Pareto front on the same plot as the linear encoding in Figure 2. Please note that the 'Weight' on the x-axis has a slightly different meaning here than the one in the linear encoding and it should be interpreted as the relative distance of any solution on the Pareto front against the two extremes of the front. The red curve in Figure 2 illustrates the quality of solutions on the Pareto front (i.e. MoJo similarity), plotted against the relative weight of the objective functions.

The Pareto front of Apache Ant comprises of two regions which show improvements over the rest of the solution space, one around 20% and a bigger bump around 60%. This pattern is in partial accord with that of the single objective

TABLE II. THE RESULTS FOR SINGLE-OBJECTIVE ENCODED PROBLEM OF MULTI-VIEW CLUSTERING

System	CQ case		MQ Case		Best Case			Worst Case			Average Case		Interpolated Max	
	NoC	MojoSim	NoC	MojoSim	λ	NoC	MojoSim	λ	NoC	MojoSim	NoC	MojoSim	λ	MojoSim
Apache Ant	14	0.57	25	0.53	0	14	0.57	0.4	26	0.52	23.14	0.54	0	0.57
Apache Hadoop	30	0.51	105	0.49	0	30	0.51	0.2	112	0.49	96.43	0.5	0	0.51
Apache Log4j	10	0.44	18	0.45	0.4	18	0.47	0	10	0.44	16.71	0.45	0.45	0.47
Eclipse JDT Core	23	0.55	108	0.53	0	23	0.55	0.5	107	0.53	95.14	0.54	0	0.55
JDOM	9	0.65	21	0.54	0	9	0.65	0.4	21	0.54	19.14	0.57	0	0.65
JEdit	34	0.59	107	0.56	0	34	0.59	0.2	110	0.55	96.86	0.56	0	0.59
JFreeChart	24	0.44	63	0.41	0.2	58	0.45	1	63	0.41	52.29	0.44	0.15	0.45
JHotDraw	12	0.59	21	0.5	0	12	0.59	1	21	0.5	18.29	0.54	0	0.59
JUnit	5	0.41	20	0.5	0.2	19	0.52	0	5	0.41	17.71	0.48	0.25	0.52
Weka	44	0.41	145	0.51	0.2	142	0.53	0	44	0.41	129	0.5	0.25	0.53

encoding, except that the results obtained from the multi-objective approach are in general more desirable. In case of JUnit, results tend to improve as MQ becomes dominating, implying the significance of the structural dependency of JUnit in determining its modularity. Contrary to the single objective encoding where the best result is obtained around 20% mark, in the multi-objective setting, MQ generates a better result from authoritative perspective. A possible reason for the difference observed in both cases can be due to the non-optimality of the local search algorithms used in this work. However, regardless of the encoding used, multi-view clustering indeed yields results that are at least more authoritative than the least-informative view.

C. A Compromise Solution

The goal of unsupervised clustering is to automatically generate a single partitioning for a system such that the generated partition closely aligns with the high-level decomposition of the system. In both encodings of multi-view clustering problem presented here, we are left with a set of solutions that need to be joined into a single solution. In the following, we present two techniques to attain a compromise solution, one based on interpolation and the other using hierarchical clustering.

1) Prediction using Interpolation: The single objective approach to clustering emits a single, possibly global optimal solution, however, in the context of linear combination encoding, the objective function is parameterized over some weight λ. As demonstrated, we have experimented with different values of weight to see where the best solution may occur. To predict the possible occurrence of best solution, we have employed spline interpolation to approximate the extreme points along all values of weight λ, using a few known data points constructed from empirical evaluation. The absolute extreme points predict the possible values for which the weights may give the best/worst results.

We have taken the median of absolute maximums for the spline interpolation of all the 10 case studies, and hence observed that the lexical information is a better indicator for architecture recovery of software systems. In six of our case studies, the conceptual measure produced superior results. These findings are in coherence with previous studies [17], [13] that suggest semantic measure aligns better with developers' perceptions of coupling between classes, and thus a better metric for remodularization of software systems. In presence of information about the reliability of views relative to each other, the weights can be adjusted accordingly. However, when no preference can be made over the views, as demonstrated with the case studies, conceptual measure is a good metric to derive the architecture.

2) Merging using Hierarchical Clustering: The Pareto front comprises a set of solutions which run well into hundreds of solutions, depending on the size of problem. The question is how one can merge the set of non-dominated solutions to a single solution that is satisfactory enough. The multi-objective encoding tries to maximize agreement between different views while resolving differences by accepting different solutions. In our approach to find a consensual solution from the pool of solutions, we try to *minimize disagreement* between different solutions. To help find a "good" consensual solution, we have adopted hierarchical clustering techniques.

In our approach, first an "agreement" matrix is constructed for the set of solutions that need to be merged. An agreement matrix denotes the number of partitions that any two particular modules share in the solution space. Since sharing a partition is a symmetric relationship, the computed agreement matrix is also symmetric. Once an agreement matrix is constructed, hierarchical clustering can be applied to obtain a dendrogram or a cluster tree. In this tree, the lower subtrees exhibit high agreement between their member modules, while higher subtrees denote weaker agreement.

We have run a set of trials using different hierarchical clustering algorithms to find an agglomerative clustering algorithm which gives the best solutions for our problem domain. The experiments conducted show that the average linkage algorithm yields results that are far better than other methods. The other advantage of the linkage algorithms is that unlike search algorithms, it is deterministic, and thus it produces the same result every time it is run on the same input. Once a tree is built, it has to be cut to get one solution. As for the choice of where the tree should be cut, we use the median of the number of clusters of the Pareto front. This choice is driven by the observations we have made through experimenting with different cutpoints. The hierarchical clustering approach gives results that are in most cases better than the average solution.

D. Comparison of Compromise Solutions

Table IV gives a comparison of the number of clusters as well as the quality of compromise solutions obtained using the above techniques. In four of the projects, that is Apache Ant, JDOM, JHotDraw and JUnit projects, unification of results obtained from the multi-objective approach are superior with respect to the MoJo similarity. In other cases, single-view clustering produces more authoritative results. At this stage, we are not certain the observations made are due to

TABLE III. THE RESULTS FOR MULTI-OBJECTIVE ENCODED VARIANT OF MULTI-VIEW CLUSTERING

System	Size of Solutions	Best Case		Worst Case		Median Case		Average Case	
		NoC	MojoSim	NoC	MojoSim	NoC	MojoSim	NoC	MojoSim
Apache Ant	120	18	0.62	21	0.52	23	0.56	20.59	0.57
Apache Hadoop	67	39	0.51	2	0.27	44	0.42	38.93	0.43
Apache Log4j	208	16	0.45	12	0.31	6	0.35	12.68	0.37
Eclipse JDT Core	91	42	0.54	13	0.37	33	0.46	31.66	0.46
JDOM	50	11	0.71	8	0.59	10	0.67	11.42	0.67
JEdit	79	32	0.46	2	0.24	31	0.42	33.97	0.42
JFreeChart	63	43	0.4	53	0.34	49	0.36	50.17	0.36
JHotDraw	76	12	0.63	16	0.51	14	0.59	14.13	0.58
JUnit	43	14	0.57	5	0.4	8	0.48	9.16	0.48
Weka	80	51	0.33	2	0.14	46	0.25	34.75	0.25

the sub-optimality of search algorithms or it is indeed an intrinsic difference between the two approaches, once a single solution is produced. We believe further experiments need to be conducted both on the case studies used in this paper as well as other projects to draw more concrete conclusions.

TABLE IV. THE COMPARISON OF QUALITY OF COMPROMISE SOLUTIONS FOR SINGLE-OBJECTIVE VS. MULTI-OBJECTIVE APPROACHES

System	SO-encoded		MO-encoded	
	NoC	MojoSim	NoC	MojoSim
Apache Ant	14	0.57	23	0.58
Apache Hadoop	30	0.51	44	0.50
Apache Log4j	10	0.44	5	0.36
Eclipse JDT Core	23	0.55	33	0.40
JDOM	9	0.65	10	0.7
JEdit	34	0.59	31	0.42
JFreeChart	24	0.44	49	0.37
JHotDraw	12	0.59	14	0.61
JUnit	5	0.41	8	0.43
Weka	44	0.41	46	0.20

E. General Observations

The following general observations, independent of the encoding used, were made through conducting the experiments:

First, multi-view clustering yields decompositions that are at least more authoritative than the single view decomposition of the least informative view. The results obtained show that by maximizing agreement between different views, it is possible to put a lower bound on the performance of the cluster analysis. Thus, in absence of information about reliability of views, multi-view clustering can be employed to not only improve general performance, but obtain results that are more authoritative.

Second, as the systems grow in size, the heuristics used for remodularization such as the structural cohesion/coupling dogma, do not seem to be the dominating force in deriving the package structure of the software system. In the case of JDOM, with only 63 classes, a MoJo similarity of 71% was obtained, whereas for Weka, with almost 7 times as many classes as JDOM, the MoJo similarity was no higher than 51%. Overall, there seems to be some 20% difference in MoJo similarity between those projects with fewer than 100 modules and those with more modules. This observation is in conformance with the previous works [6], [14] that question the coupling-based heuristics used here for architecture recovery of large systems.

Finally, the multi-objective approach to multi-view clustering gives results that are mostly superior to any results obtained through single-objective encoding, however the quality of solutions on the front vary vastly. When the goal of clustering is to seek different partitions of the system, as in alternative cluster analysis, multi-objective encoding is the preferred technique. On the other hand, in cases where a single solution is desired, the linear combination encoding is the preferred choice. If there is information about relative reliability of viewpoints, the weight can be adjusted to balance the weighing of different viewpoints on the produced partitioning. In absence of such information, it is recommended to either 1) opt for equal weighing (i.e. 50%) to give equal preference to the views, or 2) select CQ metric to perform clustering.

F. Threats to Validity

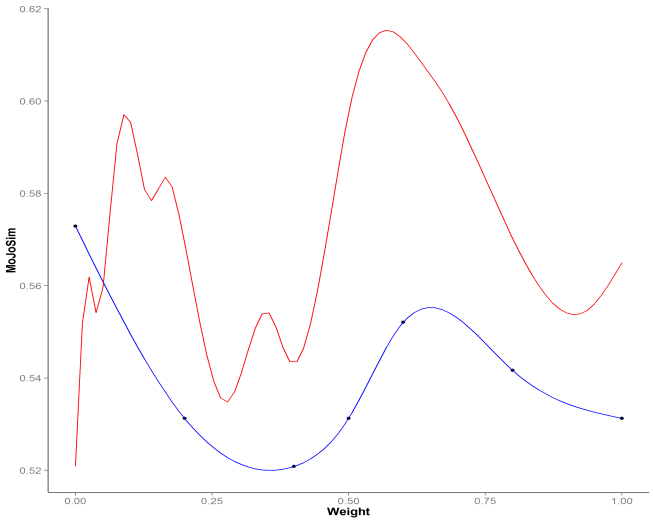
The non-optimality of local search algorithms is the greatest threat to the validity of this research. There is no guarantee that the results obtained through performing local search are in fact the optimal solutions to the problem. To help find better solutions, experiments are run multiple times and the one giving the best result is chosen as the solution. Furthermore, a heuristic-driven search strategy is adopted for the hill climbing algorithm, consisting of an extensive search followed by a narrowed area search.

Another threat to the validity of our research is the reliability of the ground truth constructed to evaluate the quality of produced clusters. We rely on the package structure of the case studies to build the authoritative decomposition. There are two major shortcomings to this approach. 1) It may be the case that the architecture of the system does not conform to its package structure. In fact, evaluating architecture quality is a subjective concept and there is no solid agreement on what constitutes a good architecture [13]. Although this is a serious shortcoming to the validation approach used here, however one can use the produced results to construct a package structure for a software system. 2) The evolution of software may cause a drift in its architecture. To mitigate this risk, we have chosen those projects that we believe are well-engineered or have gone through a migration phase during which their package structure was restructured.

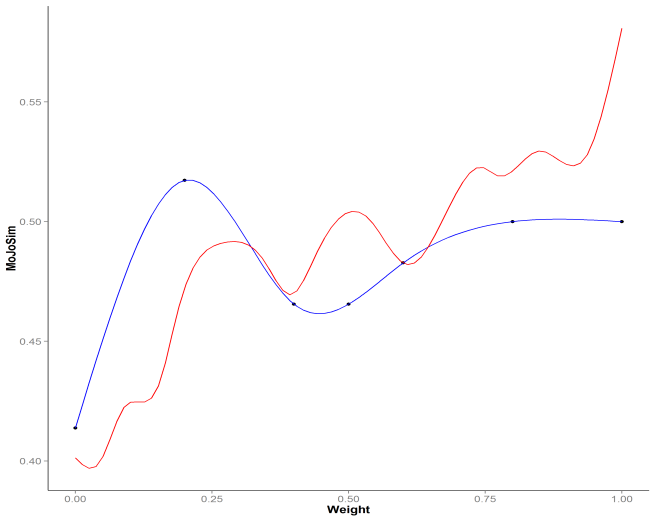
Although we have applied our approach on 10 relatively large systems, our findings cannot be generalized. As is the case with most empirical works, we believe our approach needs to be evaluated with larger systems, in particular on the systems that are not well-engineered to see if improvement can still be made.

VI. CONCLUSION AND FUTURE WORK

We have adopted a multi-view approach to clustering, where different sources of knowledge about the system are

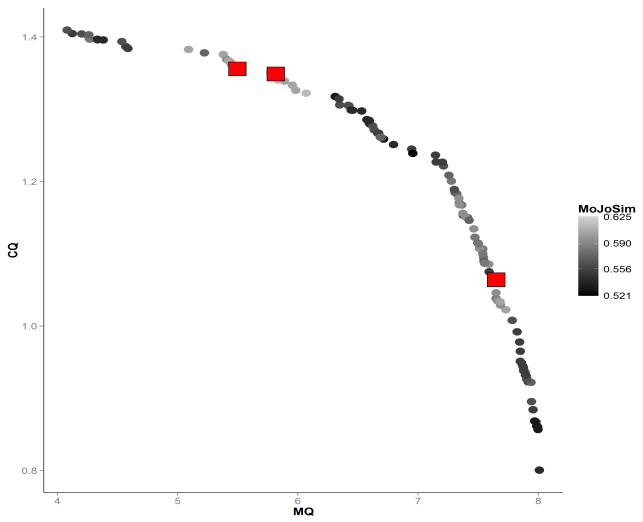


(a) Scatterplot for Apache Ant

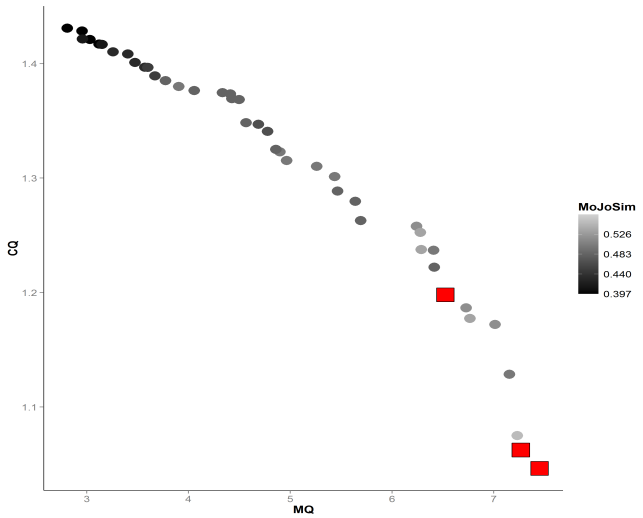


(b) Scatterplot for JUnit

Fig. 2. The scatterplots for Apache Ant and JUnit with MoJoSim similarity measurement plotted against weights λ



(a) Scatterplot for Apache Ant



(b) Scatterplot for JUnit

Fig. 3. The scatterplots for Apache Ant and JUnit with multi-objective Pareto front plotted against the MoJoSim similarity measurement

combined to produce superior results. The general observation made through conducting the experiments indeed confirm our hypothesis that incorporating knowledge from different viewpoints leads to results which are more desirable from the authoritative perspective. On the other hand, in single-objective approaches to unsupervised clustering, where no concrete conclusion can be drawn about whether one viewpoint is more reliable than the other, multi-view clustering can be used to not only help get a result which is at least better than the less reliable viewpoints, but in most cases improve the result altogether.

A prerequisite and indeed a limitation of our approach is that it requires access to different viewpoints of the system, for instance, the source code itself. In cases where only one source of information is available, the multi-view clustering cannot be employed. Especially in the industrial domain, where large companies are reluctant to give up their source code, this important source of information cannot be integrated into our approach. Nevertheless, as shown in this paper, the multi-view approach can be employed to combine other available viewpoints.

A domain which we believe has an urgent need for application of (semi-)automated clustering techniques is the legacy domain. The problems one needs to tackle in this domain range from a large dead/duplicate code base to arbitrary naming convention. We are currently investigating different techniques to facilitate legacy to SOA migration [27]. To facilitate this process, we would like to investigate the use of multi-view clustering for architecture recovery in the industrial domain and whether it also leads to improvements for legacy systems.

As part of future research direction, we plan to incorporate the probabilistic model introduced by Corazza et al. [9], [10]

as a preprocessing phase to compute the lexical similarity between modules before feeding the information for multi-view clustering. Furthermore, we want to include other aspects of knowledge including but not limited to evolutionary information such as revision transactions, ownership, as well as library usage.

REFERENCES

[1] N. Anquetil and T. Lethbridge, "Experiments with clustering as a software remodularization method," in *Proceedings of the 6th Working Conference on Reverse Engineering (WCRE)*, Oct 1999, pp. 235–255.

[2] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd, "Reformulating software engineering as a search problem," *IEE Proceedings–Software*, vol. 150, no. 3, pp. 161–175, June 2003.

[3] B. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the bunch tool," *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp. 193–208, March 2006.

[4] K. Praditwong, M. Harman, and X. Yao, "Software module clustering as a multi-objective search problem," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 264–282, March 2011.

[5] K. Mahdavi, M. Harman, and R. Hierons, "A multiple hill climbing approach to software module clustering," in *Proceedings of the International Conference on Software Maintenance (ICSM)*, Sept 2003, pp. 315–324.

[6] N. Anquetil and J. Laval, "Legacy software restructuring: Analyzing a concrete case," in *15th European Conference on Software Maintenance and Reengineering (CSMR)*, March 2011, pp. 279–286.

[7] N. Anquetil and T. Lethbridge, "Extracting concepts from file names: A new file clustering criterion," in *Proceedings of the 20th International Conference on Software Engineering*, ser. ICSE '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 84–93.

[8] A. Kuhn, S. Ducasse, and T. Grba, "Semantic clustering: Identifying topics in source code," *Information and Software Technology*, vol. 49, no. 3, pp. 230 – 243, 2007, 12th Working Conference on Reverse Engineering.

[9] A. Corazza, S. Di Martino, and G. Scanniello, "A probabilistic based approach towards software system clustering," in *14th European Conference on Software Maintenance and Reengineering (CSMR)*, March 2010, pp. 88–96.

[10] A. Corazza, S. D. Martino, V. Maggio, and G. Scanniello, "Investigating the use of lexical information for software system clustering," in *European Conference on Software Maintenance and Reengineering (CSMR)*, 2011, pp. 35–44.

[11] J. Wu, A. Hassan, and R. Holt, "Comparison of clustering algorithms in the context of software evolution," in *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM)*, Sept 2005, pp. 525–535.

[12] F. Beck and S. Diehl, "On the congruence of modularity and code coupling," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 354–364.

[13] G. Santos, M. Valente, and N. Anquetil, "Remodularization analysis using semantic clustering," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*, Feb 2014, pp. 224–233.

[14] M. Glorie, A. Zaidman, A. Van Deursen, and L. Hofland, "Splitting a large software repository for easing future software evolutionan industrial experience report," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 21, no. 2, pp. 113–141, 2009.

[15] G. Scanniello, A. D'Amico, C. D'Amico, and T. D'Amico, "Using the kleinberg algorithm and vector space model for software system clustering," in *18th International Conference on Program Comprehension (ICPC)*, June 2010, pp. 180–189.

[16] C. Patel, A. Hamou-Lhadj, and J. Rilling, "Software clustering using dynamic analysis and static dependencies," in *13th European Conference on Software Maintenance and Reengineering (CSMR)*, March 2009, pp. 27–36.

[17] G. Bavota, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "An empirical study on the developers' perception of software coupling," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 692–701.

[18] G. Bavota, M. Gethers, R. Oliveto, D. Poshyvanyk, and A. d. Lucia, "Improving software modularization via automated analysis of latent topics and dependencies," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 1, pp. 4:1–4:33, Feb. 2014.

[19] S. Mancoridis, B. Mitchell, C. Rorres, Y. Chen, and E. Gansner, "Using automatic clustering to produce high-level system organizations of source code," *International Conference on Program Comprehension*, vol. 0, p. 45, 1998.

[20] J. I. Maletic and A. Marcus, "Supporting program comprehension using semantic and structural information," in *Proceedings of the 23rd International Conference on Software Engineering*, ser. ICSE '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 103–112.

[21] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by Latent Semantic Analysis," *Journal of the American Society of Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[22] M. Naik, A. Aiken, and J. Whaley, "Effective static race detection for Java," in *Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '06. New York, NY, USA: ACM, 2006, pp. 308–319.

[23] V. Tzerpos and R. Holt, "MoJo: a distance metric for software clusterings," in *6th Working Conference on Reverse Engineering (WCRE)*, Oct 1999, pp. 187–193.

[24] Z. Wen and V. Tzerpos, "An optimal algorithm for MoJo distance," in *11th IEEE International Workshop on Program Comprehension*, May 2003, pp. 227–235.

[25] A. Saeidi, J. Hage, R. Khadka, and S. Jansen, "Gelato: GEneric LAnguage TOols for model-driven analysis of legacy software systems," in *20th Working Conference on Reverse Engineering (WCRE)*, Oct 2013, pp. 481–482.

[26] K. Praditwong and X. Yao, "A new multi-objective evolutionary optimisation algorithm: The two-archive algorithm," in *International Conference on Computational Intelligence and Security*, vol. 1, Nov 2006, pp. 286–291.

[27] R. Khadka, G. Reijnders, A. Saeidi, S. Jansen, and J. Hage, "A method engineering based legacy to SOA migration method," in *27th IEEE International Conference on Software Maintenance (ICSM)*, Sept 2011, pp. 163–172.