Software Clusterings with Vector Semantics and the Call Graph

Marios Papachristou
papachristou.marios@gmail.com
National Technical University of Athens
Athens, Attica, Greece

ABSTRACT
In this paper, we propose a novel method to determine a software’s modules without knowledge of its architectural structure, and empirically validate the method’s performance. We cluster files by combining document embeddings, generated with the Doc2Vec algorithm, and the call graph, provided by Static Graph Analyzers to an augmented graph. We use the Louvain Algorithm to determine its community structure and propose a module-level clustering. Our method performs better in terms of stability, authoritativeness, and extremity over other state-of-the-art clustering methods proposed in the literature and is able to decently recover the ground truth clustering of the Linux Kernel. Finally, we conclude that semantic information from vector semantics as well as the call graph can produce accurate results for software clusterings of large systems.

CCS CONCEPTS
• Computing methodologies → Cluster analysis; Machine learning algorithms; • Software and its engineering → Software architectures.

KEYWORDS
document embeddings, doc2vec, linux kernel, natural language processing, software architecture recovery, software clustering, static graph analysis, vector semantics

ACM Reference Format:

1 INTRODUCTION
In modern software systems, the need for adoption of modular software architectural schemata becomes inevitable for robust maintainability of their codebases. Since the architecture of a software system is the most fundamental realization of the software system, when there is no description of the architecture of it, software architects make attempts to recover it. In the past, methods have been proposed to identify the main components of a software system using hierarchical clustering algorithms [1, 9] mainly by clustering modules into subsystems, given using the source code. Recently, the possibilities offered by machine learning techniques and practices in the Natural Language Processing (NLP) can also be applied on codebases as corpora of text, thus enabling the extraction of useful information for the semantics of the source code inaugurating new modes of reviewing the codebase.

Our motivation, through this paper, is to use new features, namely vector semantics, and structural information via the call graph in order to perform community detection and arrive at module-level clusterings. We will study the Linux codebase and we will compare it with ground truth using the MoJo measure [15] as well as compare it with other state-of-the-art and baseline clustering algorithms in terms of stability, authoritativeness and extremity [9, 19].

2 RELATED WORK
Onaiza Maqbool and Haroon A. Babri [9] present an overview of the various approaches taken towards hierarchical clustering algorithms for software systems. In their study, they compare many clustering algorithms such as ACDC, LIMBO and other Traditional Clustering Algorithms such as Single Linkage, Complete Linkage and Weighted Linkage presenting their efficiency regarding multiple feature sets. Added to this, a semantics-based architectural view of the system, as discussed in reference [2] reveals significant aspects of a software system and its change over time, which suggests that semantic-based approaches should be followed for gaining better understandings of a software system.

Work in Software Clustering Algorithms (sca) has been extensively done by many authors in references [15], [17], [16] and [1] with the ACDC and LIMBO clustering algorithms. In particular, the ACDC algorithm leans toward to software components comprehension based on subsystem patterns. Their approach considers an initial structure of the system, without taking into account semantics, and tries to build comprehensive clusterings of the given ground structure. The authors also conducted a study on an older version of Linux. Moreover, Andritsos and Tzerpos in reference [1] present an Information-Theoretic approach of SCA by developing LIMBO which clusters modules upon inserting their Distributional Cluster Features to a B+-tree variant and then applying the Agglomerative Information Bottleneck algorithm.

3 METHOD
The codebase is initially preprocessed and the resulting data is fed into a Skip-Gram model for the initial components, which correspond to nodes in the call graph. Next, we add weights between the
nodes with the normalized similarity $w(i, j) = (1 + \cos(d_i, d_j)) / 2$ between the embeddings. Finally, we run the Louvain algorithm to obtain a clustering. The overall process is described below.

First of all, we tokenize the source code and remove the stopwords. After that, for each token, we split it into its constituent parts using dynamic programming [5] and lemmatize each individual sub-token via using the NLP package spaCy [7]. For example, the method `__zone_seqlock_init` corresponds to `zone`, `seqlock` and `init`, `inprogress` is split into `in` and `progress` and literals becomes `literal`. We, then, train a Doc2Vec model using Gensim [12]. The codebase is also processed by a Static Graph Analyzer. In our approach, focusing on C projects, we use CScour [14] in order to generate the directed call graph through function calls between the files. Each source code file is assigned to a module. Modules can be user-defined or automatically generated by their respective directories at a desired directory tree depth. The input to the clustering procedure is the call graph with normalized cosine similarities of the modules as edge weights. Then we run the Louvain Clustering Algorithm [3] in order to obtain the software clustering via maximizing the modularity function with a greedy approach. In case we want to consider edge directionality, we do a bipartite transformation, detect communities and merge the results as proposed in reference [19]. Note that ACDC, which takes into account only the structural properties of the files, produced a high granularity clustering that failed to catch the structure of Linux. In the authoritativeness criterion, our approach outperforms the other clusterings in terms of MoJo distance being very close (smaller is better) to the ground truth. The ground truth clustering is also present in the table. Our quality metrics for the clusterings are extremity, authoritativeness and stability as proposed in reference [19].

Our approach produces balanced clusterings thus demonstrating low extremity compared to the other clusterings, producing a number of clusters that is close to the ground truth with smaller standard deviation in the cluster sizes than the other clustering methods, without knowing the number of clusters a priori. Besides this, it gives balanced clusters with respect to Median Cluster criterion.

In Table 1, we present the results of the clusterings ordered in increasing MoJo distance with respect to the ground truth. We have included the number of clusters produced, the cluster size range, the average cluster size the standard deviation in the cluster sizes, the increasing MoJo distance with respect to the ground truth. We have ran the simulations multiple times and averaged the results. The experimentation with a large and complex system may constitute a threat to our findings’ validity since the results are not general.

## 4 RESULTS, EVALUATION & DISCUSSION

In Table 1, we present the results of the clusterings ordered in increasing MoJo distance with respect to the ground truth. We have included the number of clusters produced, the cluster size range, the average cluster size the standard deviation in the cluster sizes, the increasing MoJo distance with respect to the ground truth. We have ran the simulations multiple times and averaged the results. The experimentation with a large and complex system may constitute a threat to our findings’ validity since the results are not general.

### 5 CONCLUSIONS

This paper is set out to show how we can use a combination of vector semantics and information from the call graph in order to produce meaningful clusterings of a software system. We have outperformed state-of-the-art software clusterings and conventional agglomerative clustering algorithms in terms of extremity, authoritativeness and stability without even knowing the number of clusters of the ground truth. This evidence supports the further usage of vector semantics and the call graph for architecture recovery. Finally, the further integration with static analyzers and the development of evaluation policies with users should be taken into account, especially when dealing with old codebases lacking technical documentation.

### Table 1: Experimental Results for Linux 4.21

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Feature Dimension</th>
<th># Clusters</th>
<th>Size Range</th>
<th>$\bar{\delta}$</th>
<th>$\sigma$</th>
<th>Median Cluster</th>
<th>Mojo Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACDC [16]</td>
<td>–</td>
<td>9055</td>
<td>1 – 2425</td>
<td>5</td>
<td>46</td>
<td>2</td>
<td>33604</td>
</tr>
<tr>
<td>Complete Linkage [4]</td>
<td>300  21</td>
<td>1 – 1529</td>
<td>12317</td>
<td>21</td>
<td>725</td>
<td>1</td>
<td>2092</td>
</tr>
<tr>
<td>LIMBO ($B = 100$, $S = \infty$) [1]</td>
<td>300  21</td>
<td>1 – 5207</td>
<td>12317</td>
<td>21</td>
<td>412</td>
<td>19</td>
<td>1710</td>
</tr>
<tr>
<td>Ward Linkage [18]</td>
<td>300  21</td>
<td>1 – 948</td>
<td>12317</td>
<td>21</td>
<td>223</td>
<td>70</td>
<td>1138</td>
</tr>
<tr>
<td>SADe (Directed)</td>
<td>300  5 ± 2</td>
<td>1 – 1348</td>
<td>12317</td>
<td>21</td>
<td>341</td>
<td>11.0</td>
<td>–</td>
</tr>
</tbody>
</table>

In Table 1, we present the results of the clusterings ordered in increasing MoJo distance with respect to the ground truth. We have included the number of clusters produced, the cluster size range, the average cluster size the standard deviation in the cluster sizes, the increasing MoJo distance with respect to the ground truth. We have ran the simulations multiple times and averaged the results. The experimentation with a large and complex system may constitute a threat to our findings’ validity since the results are not general.

---

This paper is set out to show how we can use a combination of vector semantics and information from the call graph in order to produce meaningful clusterings of a software system. We have outperformed state-of-the-art software clusterings and conventional agglomerative clustering algorithms in terms of extremity, authoritativeness and stability without even knowing the number of clusters of the ground truth. This evidence supports the further usage of vector semantics and the call graph for architecture recovery. Finally, the further integration with static analyzers and the development of evaluation policies with users should be taken into account, especially when dealing with old codebases lacking technical documentation.
REFERENCES


