Refining Clustering Evaluation Using Structure Indicators

Mark Shtern and Vassilios Tzerpos
York University
Toronto, Ontario, Canada
{mark,bil}@cse.yorku.ca

Abstract

The evaluation of the effectiveness of software clustering algorithms is a challenging research question. Several approaches that compare clustering results to an authoritative decomposition have been presented in the literature. Existing evaluation methods typically compress the evaluation results into a single number. They also often disagree with each other for reasons that are not well understood.

In this paper, we introduce a novel set of indicators that evaluate structural discrepancies between software decompositions. They also allow researchers to investigate the differences between existing evaluation approaches in a reduced search space. Several experiments with real software systems showcase the usefulness of the introduced indicators.

1. Introduction

Software clustering has been shown to be an effective way to retrieve the high-level structure of a software system [14, 6, 3, 10, 15, 11]. Many software clustering algorithms that attempt to automatically construct decompositions of large pieces of software are presented in the literature. Different algorithms construct different decompositions [19, 8]. Therefore, it is important to have methods that evaluate the quality of automatic decompositions.

Several software clustering evaluation methods have been developed [8, 13, 16, 17, 19]. These methods typically attempt to measure the similarity between an automatically created decomposition and a pre-existing authoritative one. However, while the existing methods have been quite useful, there are certain issues that can be identified.

Firstly, the existing measures often disagree in their evaluation. Despite having the same goal, that of quantifying how similar an automatic and an authoritative decomposition are, they frequently disagree in their ranking of different automatic decompositions for the same software system. The reason for these discrepancies are not understood very well so far.

Secondly, existing measures provide a single number that is representative of the quality of the automatic decomposition. However, there could be many underlying reasons that result in higher or lower quality, since there are several ways in which an automatic decomposition can differ from an authoritative one. A better understanding of these partial differences would be beneficial to software maintenance activities.

In this paper, we investigate possible reasons for the discrepancies between two of the more popular evaluation methods, MoJoFM [22] and KE [8]. Our work focuses on structural properties of automatic decompositions, since theoretical analysis as well as experimental results indicate that the behaviour of these two measures depends on such properties.

We also introduce a set of structure indicators that a software maintainer could use in parallel with existing measures in order to better understand the properties of automatically created decompositions.

The structure of the rest of this paper is as follows. Section 2 describes relevant work in software clustering. Motivation for our work in evaluating structural differences between decompositions is presented in Section 3. Section 4 introduces the structure indicators that quantify these differences. We utilize these indicators in several experiments that demonstrate their usefulness in Section 5. Finally, Section 6 concludes the paper.

2. Related work

One of the early works on experimental evaluation of software clustering techniques was presented by Lakhotia and Gravely [9]. Their evaluation metric, however, can only be applied to hierarchical clustering algorithms that produce dendrograms, a feature that limits its applicability.

More recently, Anquetil and Lethbridge [4] used the well-known Information Retrieval measures of Precision and Recall in order to measure similarity between different clusterings of the same software system. Though their

work produced many interesting results, the limitations of the two measures are well-documented [13, 18].

The MoJo distance measure [19, 21] attempts to capture the distance between two decompositions as the minimum number of operations one has to perform in order to transform one into the other. The MoJoFM effectiveness measure [22] is based on MoJo distance but produces a number between 0 and 1 which is independent of the size of the decompositions.

Koschke and Eisenbarth [8] presented an elaborate framework that extends and removes the limitations of the approach taken by Lakhotia and Gravely. They incorporate the fact that one may have to cope with approximate matches between clusters in real life. Their approach includes an overall recall rate that captures the quality of a particular technique. We refer to this recall rate as KE. As is the case with MoJoFM, it is a number between 0 and 1, with 1 indicating a perfect match between the two decompositions.

Mitchell and Mancoridis [13] were the first to present a similarity (*EdgeSim*) and a distance (*MeCl*) measurement that consider relations between components as the important factor for the comparison of software system decompositions.

The EdgeMoJo measure [23] attempts to combine several aspects of existing techniques. It is the only comparison method that considers both the placement of objects into clusters, as well as the relations between them.

The Evaluation of Nested Decompositions (END) framework [16] allows a reverse engineer to compare nested decompositions of large software systems without having to lose information by transforming them to flat ones first. The END framework is able to compare two nested decompositions without any information loss by converting them into vectors of flat decompositions and applying existing comparison methods to all elements of these vectors. The overall similarity between the two nested decompositions is computed as the norm of the similarity vector generated from comparing the decomposition vectors.

3. Structure Evaluation

While the clustering evaluation methods presented in the previous section, such as MoJoFM and KE, can be quite useful, certain aspects of these measures could be improved upon. In particular, we identify the following two issues:

1. The stated goal of both measures is the same: to evaluate the quality of an automatically created decomposition by comparing it to an authoritative one. However, it has been shown [1, 17] that the two measures often disagree in their ranking of different automatic decompositions for the same software system.
2. Both measures provide a single number that is representative of the quality of the automatic decomposition. However, as outlined below, there are several ways in which an automatic decomposition can differ from an authoritative one. Measures of these partial differences could be very useful for clustering evaluation.

The goal of the research presented in this paper is to investigate possible reasons for the discrepancies between MoJoFM and KE and to identify further metrics that a software maintainer could use in parallel with MoJoFM and KE. We begin by presenting a motivating example.

Let us consider a software system that contains 8 entities, such as classes or source files. Expert analysis has indicated that there are three subsystems. Figure 1 shows the authoritative decomposition A for this example system.

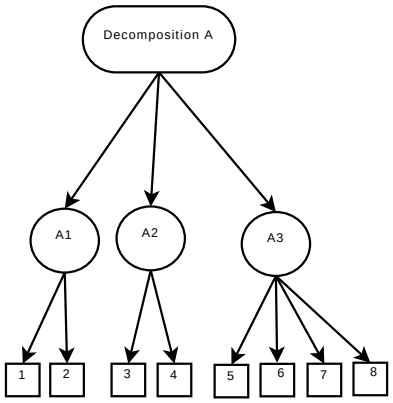

Figure 1. Authoritative decomposition A

Figures 2 and 3 show two automatic decompositions of the same software system. Let us assume that software clustering algorithm SCA_B created decomposition B, while algorithm SCA_C created decomposition C.

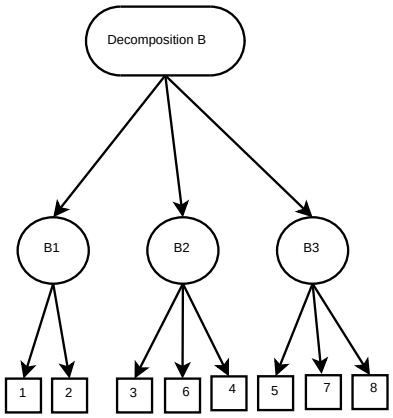

Figure 2. Automatic decomposition B

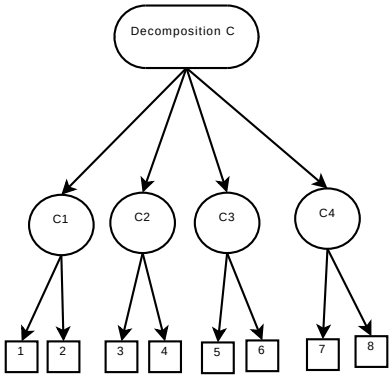

Figure 3. Automatic decomposition C

While both algorithms have recovered the system structure fairly well, it should be clear that algorithm SCA_B should be more helpful to someone attempting to understand the high level structure of this software system. It has recovered all subsystems correctly with only minor discrepancies with regard to their contents. On the other hand, the fact that subsystems $C3$ and $C4$ are closely related was missed by algorithm SCA_C.

Unfortunately, MoJoFM and KE do not agree with this assessment. Only one Move operation is required to transform B to A, while only one Join operation is required to transform C to A. As a result, MoJoFM considers B and C to be of equal quality. On the other hand, since KE does not penalize for joining clusters, its value for C will be 1, the maximum possible, while its value for B turns out to be 0.81. As a result, KE considers C to be the better automatic decomposition, contrary to our earlier assessment.

One can amplify the presented example by replacing every entity other than entity 6 with a large number of entities without affecting the validity of the example (the only change will be that the value of KE for B will go up, but it will still be less than 1).

This example illustrates that measures such as MoJoFM or KE are not sufficient for an accurate evaluation of a software decomposition. In Section 4, we present additional measures that augment MoJoFM and KE and capture the fact that decomposition B is closer to the authoritative one.

Furthermore, the above example presents a first hint at what might be the reasons for the occasional disagreements between MoJoFM and KE. The following example should help illuminate the situation further.

Let us consider a different software system that contains 16 entities. Its authoritative decomposition is shown in Figure 4.

Figures 5 and 6 show two automatic decompositions of the same software system.

While both automatic decompositions differ significantly from the authoritative one, they do so in a distinctly different fashion.

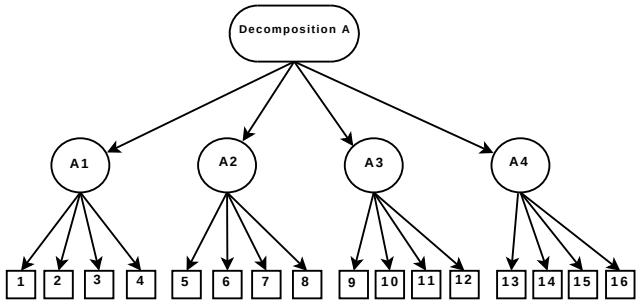

Figure 4. Authoritative decomposition A

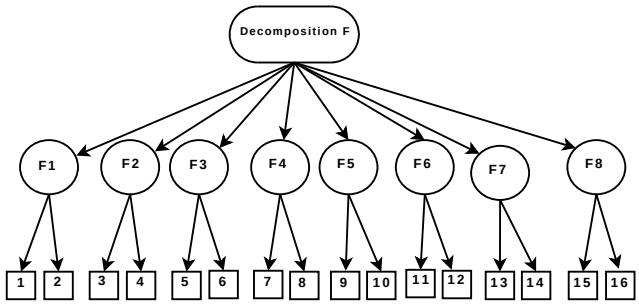

Figure 5. Automatic decomposition F

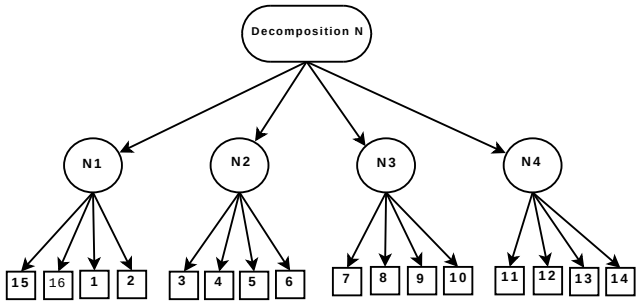

Figure 6. Automatic decomposition N

Decomposition F contains clusters whose elements are correctly related according to the authoritative decomposition. However, the close relation between many of the clusters is lost. We say that such a decomposition exhibits a large amount of *fragmentation*.

Decomposition N has the same number of clusters as the authoritative decomposition. However, many of the elements in these clusters are not correctly placed together. We say that such a decomposition exhibits a large amount of *noise*.

Table 1 presents the values of MoJoFM and KE for this example.

It should be evident by this admittedly extreme example that large amounts of fragmentation result in significantly higher values for KE, while large amounts of noise result

Method	Quality of F	Quality of N
MoJoFM	0.67	0.33
KE	1	0

Table 1. MoJoFM and KE values for decompositions F and N

in significantly lower values. The effect of such differences between decompositions could explain the occasional disagreements between MoJoFM and KE.

The two examples presented above provide the motivation for the work presented in this paper. There is a need to evaluate the structure of an automatic software decomposition in isolation, i.e. without considering the effects of noise in the clusters. This would allow for accurate evaluation of the decompositions in the first example, and would enable us to investigate the differences between MoJoFM and KE in a controlled setting, e.g. between decompositions with similar structure.

The next section presents three distinct structure indicators we have developed. In Section 5 we utilize these indicators in a number of experiments.

4. Structure Indicators

This section introduces three indicators that can be used to evaluate the structure of an automatic decomposition. They are all based on the intuition that two decompositions have similar structure when a software engineer gets the same picture about the software system by reading either of these decompositions. The indicators are presented in the context of flat decompositions. If necessary, one can use the END framework to apply any of these indicators to a nested decomposition as well.

We will present the structure indicators through the means of an example that illustrates the need for all three of them. Figure 7 presents the authoritative decomposition A of a software system with 13 entities. Figure 8 presents an automatic decomposition T that will refer to as the "test" decomposition.

The following observations can be made about the two decompositions:

- Clusters $A1$ and $A2$ from the authoritative decomposition are similar to the $T1$ and $T2$ clusters from the test decomposition (a small amount of noise does exist however).
- Cluster $T3$ does not resemble any cluster in A. In other words, the clustering algorithm has identified a meaningless cluster.

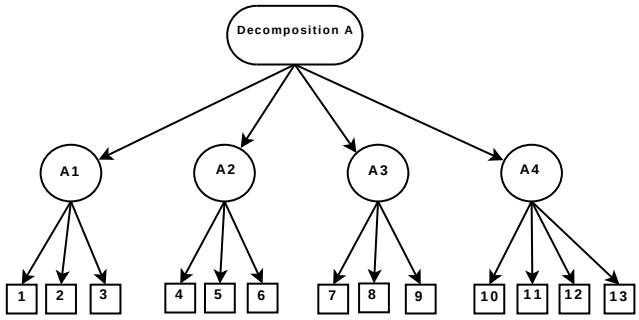

Figure 7. Decomposition A of a software system

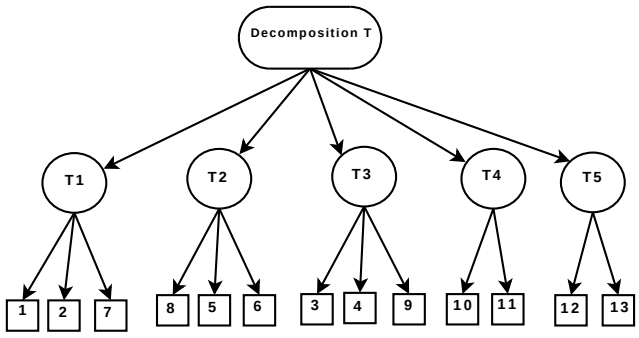

Figure 8. Decomposition T of the software system

- Cluster $A3$ does not resemble any cluster in T. In other words, information about cluster $A3$ is lost because it was never recovered by the software clustering algorithm.
- Cluster $A4$ is divided into two clusters $T4$ and $T5$ in the test decomposition.

Based on this example, we can identify three ways in which the structure of the test decomposition can differ from that of the authoritative one:

1. A cluster in the test decomposition does not have a corresponding cluster in the authoritative decomposition.
2. A cluster in the authoritative decomposition does not have a corresponding cluster in the test decomposition.
3. A cluster in the authoritative decomposition is fragmented into several clusters in the test decomposition.

The structure indicators presented in this section will gauge the magnitude of each type of difference presented above. In order to precisely define our indicators, we define the following:

Definition 4.1. A cluster T_i from the test decomposition is called a *segment* of cluster A_j from the authoritative decomposition if $\frac{|T_i \cap A_j|}{|T_i|} > 0.5$.

Cluster T_i is a segment of cluster A_j when the majority of the entities of T_i belong to A_j in the authoritative decomposition. This allows to define the three structure indicators as follows:

Extraneous Cluster Indicator (E): The number of clusters in the test decomposition that are not segments of any authoritative cluster. The value of E in our example is 1 due to cluster T_3.

Lost Information Indicator (L): The number of clusters in the authoritative decomposition that do not have any segments in the test decomposition. In our example, L = 1 due to cluster A_3.

Fragmentation Indicator (F): Let us denote by S the number of clusters in the test decomposition that are segments. The fragmentation indicator is defined as

$$\begin{cases} \frac{S}{|A|-L} & \text{when} \quad |A| > L \\ 0 & \text{when} \quad |A| = L \end{cases}$$

It is a measure of the average number of segments for all authoritative clusters that do have at least one segment. In our example, the value of F will be $\frac{4}{4-1} = 1.33$.

We often refer to the values of the three indicators as the ELF vector. In our example, the ELF vector would be (1,1,1.33).

Since the value of F can correspond to a number of different distributions for the number of segments per cluster, we can augment our three indicators with a *fragmentation distribution* graph.

Figure 9 shows the fragmentation distribution graph for our example. Each point in the x-axis represents a cluster in the authoritative decomposition that has at least one segment in decomposition T. The y-axis shows the number of segments for that cluster. Clusters are sorted in descending order of number of segments. More interesting fragmentation distribution graphs can be found in Section 5.

The three structure indicators presented above allow us to quantify the notion of similarity in decomposition structure. Test decompositions have similar structure to the authoritative one when the ELF vector is (0,0,1). As these values start to increase, the structures of the two decompositions are becoming more and more dissimilar.

The three distinct indicators also enable us to investigate particular types of structural differences and how they affect the values of measures, such as MoJoFM and KE. For example, one can limit their investigation to decompositions with low E and L, and high F to determine how fragmentation affects the behaviour of these measures.

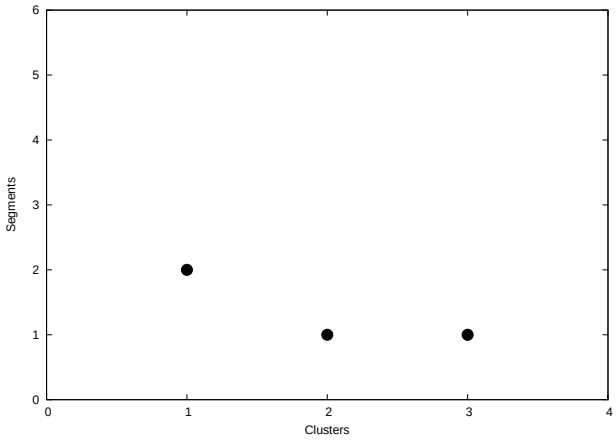

Figure 9. The fragmentation distribution graph for our example

Finally, it is interesting to note that the structure indicators would evaluate the two decompositions in the first example of Section 3 more accurately. The better decomposition has indeed an ELF vector of (0,0,1), while the other decomposition has (0,0,1.33).

In the next section, we assess the usefulness of these indicators by utilizing them in a number of experiments with real systems.

5. Experiments

The experiments presented in this section were run on decompositions of two large open source software systems:

- **Linux**. We experimented with version 2.0.27a of this free operating system that is probably the most famous open-source system. This version had 955 source files and approximately 750,000 lines of code. An authoritative decomposition for Linux was presented in [5].

- **Mozilla**. This is a widely used open-source web browser. We experimented with version 1.3 that was released in March 2003. It contains approximately 4.5 million lines of C and C++ source code. This version had 3559 source files. A decomposition of the Mozilla source files for version M9 was presented in [7]. We used an updated decomposition for version 1.3 [24].

Apart from the authoritative decompositions of these two systems, we also created automatic decompositions by clustering them with two well-known software clustering algorithms:

- **ACDC**. This is a pattern-based software clustering algorithm that attempts to recover subsystems com-

Abbreviation	Description	Clusters	Height
LF	Linux Flat Authoritative Decomposition	7	-
LFA	Linux Flat ACDC Decomposition	130	-
LFB	Linux Flat Bunch Decomposition	45	-
MF	Mozilla Flat Authoritative Decomposition	10	-
MFA	Mozilla Flat ACDC Decomposition	461	-
MFB	Mozilla Flat Bunch Decomposition	50	-
LN	Linux Nested Authoritative Decomposition	123	6
LNA	Linux Nested ACDC Decomposition	163	3
LNB	Linux Nested Bunch Decomposition	478	4
MN	Mozilla Nested Authoritative Decomposition	268	5
MNA	Mozilla Nested ACDC Decomposition	596	4
MNB	Mozilla Nested Bunch Decomposition	1646	5

Table 2. Decompositions used in the experiments.

Method	LF - LFA	LF - LFB	MF - MFA	MF - MFB
MoJoFM	0.64	0.70	0.60	0.62
KE	0.33	0.18	0.49	0.22
E	8 (130)	7 (45)	37 (461)	20 (50)
L	2 (7)	3 (7)	0 (10)	1 (10)
F	24.9	9.5	42.4	3.3

Table 3. Comparison results for flat decompositions. For E and L, the total number of clusters is shown in parentheses.

monly found in manually-created decompositions of large software systems [20].

- **Bunch**. This is a suite of algorithms that attempt to find a decomposition that optimizes a quality measure based on high-cohesion, low-coupling [12].

Both the authoritative and the automatic decompositions for Linux and Mozilla are nested, i.e. subsystems can be subdivided into smaller subsystems which themselves may be subdivided etc. As a result, we were able to run experiments for both the flat[1] and the nested versions of these decompositions.

Table 2 summarizes the decompositions used in the following experiments. It also presents the abbreviations we will use for them in this paper, as well as the number of clusters in each one. The height of each nested decomposition is also reported.

5.1. ELF evaluation

The target of the first series of experiments is to compare the results obtained from existing comparison methods to the results obtained by the structure indicators. We calculated the similarity between different flat decompositions using MoJoFM, KE, and the structure indicators. Table 3 presents a summary of the obtained results. The calculation of these values was practically instantaneous for all experiments.

These results allow for some interesting observations. First, the value of KE is always significantly lower than that of MoJoFM, a behaviour that has been observed in other similar experiments. It seems unlikely that an established tool such as Bunch has a quality of 0.18 for Linux. A possible explanation is that the threshold that KE uses to decide whether two clusters match each other is too strict. Unfortunately, the implementation that was available to us did not allow us to change the fixed threshold value of 0.7.

A second interesting observation is that KE clearly considers that ACDC produces better results, while MoJoFM gives the edge to Bunch, albeit by a smaller margin. This rather large discrepancy can be explained somewhat when one compares the KE values to those of the fragmentation indicator F. There appears to be a significant correlation between these measures. Large amounts of fragmentation result in large KE values, something that was suggested by our extreme example in Section 3. Such amounts of

[1] We transformed the nested decompositions into flat compact ones [16]

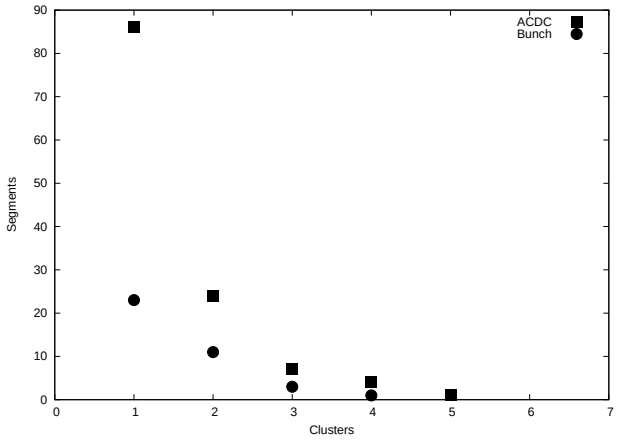

Figure 10. Fragmentation Distribution Graphs for Linux

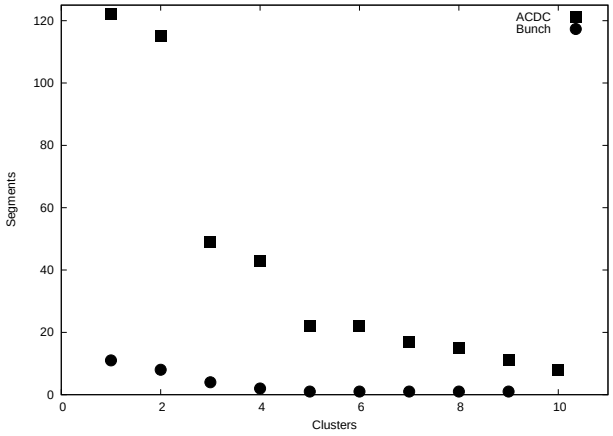

Figure 11. Fragmentation Distribution Graphs for Mozilla

fragmentation are, however, clearly counter-productive to a software maintainer. This would suggest that KE is not an appropriate evaluation method for algorithms that result in fragmented decompositions, which ACDC and Bunch seem to be.

To a lesser extent, a similar correlation can be detected between MoJoFM and the lost information indicator L. While both algorithms perform well with regard to this indicator for Mozilla, there is a significant number of Linux clusters that was never recovered. However, MoJoFM indicates that both algorithms did better for Linux than Mozilla. This can probably be attributed to the fact that MoJoFM does not directly penalize for lost clusters, it simply reconstructs them with Move and Join operations. KE penalizes directly for such mismatches, which explains why the KE values are larger for Mozilla.

Finally, it is interesting to note that the results for the extraneous cluster indicator E are mostly positive, with the exception of the case of Mozilla and Bunch. This indicates that both algorithms produce meaningful clusters. The main problem for both algorithms appears to be large amounts of fragmentation, more so in the case of ACDC.

This conclusion is also supported by the fragmentation distribution graphs presented in Figures 10 and 11. Both graphs show that only a small number of clusters have only one segment. Most authoritative clusters have several segments in the test decomposition. In the case of ACDC and Mozilla, the least fragmented cluster has 8 segments. A software maintainer will have a significantly different view about the software system by studying these test decompositions. This finding indicates that an important research direction for both algorithms, especially for ACDC, is to be able to produce more compact clusters.

We also performed a similar set of experiments using the nested version of all decompositions to investigate whether the above observations hold when all levels of a hierarchical decomposition are considered, as opposed to only the first one. Since no KE plugin for the END framework exists at this time, the experiments were performed only for MoJoFM and the structure indicators. These results are presented in Table 4.

The results are not as clear-cut as in the case of flat decompositions but they do not contradict any of the observations. It is interesting to note though that the good performance of the two algorithms with respect to the lost information indicator L is no longer evident. This indicates that both algorithms perform better at a course level of granularity but are unable to recover several finer-grained clusters. However, it is clear, that the fragmentation problem remains the most important one even for nested decompositions.

5.2. Segment Threshold Evaluation

The definition of segment in Section 4 uses a threshold value of 0.5. In this set of experiments, we investigate the effect that different thresholds could have in our results, by substituting an alternative threshold. We chose the value of 0.7 which is the threshold used by KE in a similar situation. The set of experiments with flat decompositions presented in the previous section was repeated with the new threshold. The results are shown in Table 5.

The results show that indicator F decreases but still remains high, while indicators E and L increase in value compared to the values of the corresponding indicators calculated with threshold 0.5. This is additional evidence that supports our conclusion about the significant structure differences between test and authoritative decompositions. More importantly, it shows that our choice of segment

Method	LN - LNA	LN - LNB	MN - MNA	MN - MNB
END (MoJoFM)	47.45	53.06	49.25	53.64
E	27.08 (163)	106.82 (478)	116.89 (596)	188.71 (1646)
L	33.68 (123)	29.87 (123)	65.05 (268)	72.3 (268)
F	11.21	4.79	19.55	3.21

Table 4. Comparison results for nested decompositions. For E and L, the total number of clusters is shown in parentheses.

Method	LF - LFA	LF - LFB	MF - MFA	MF - MFB
MoJoFM	0.64	0.70	0.60	0.62
KE	0.33	0.18	0.49	0.22
E	29 (130)	21 (45)	71 (461)	31 (50)
L	2 (7)	4 (7)	0 (10)	4 (10)
F	20.0	8.0	39.0	3.2

Table 5. Comparison results for flat decompositions using a segment threshold of 0.7.

threshold does not affect the evaluative power of the structure indicators.

5.3. MoJoFM and KE

The objective of this last set of experiments was to investigate the relationship between the MoJoFM and KE measures. The stated goal of both measures is to quantify the quality of a test decomposition with respect to an authoritative one. However, they attempt to measure quality in different ways often resulting in contradictory results [1, 2]. In the experiments presented below we attempted to remove the effect that structure discrepancies between the test and authoritative decomposition may have in the measures' assessment.

The structure indicators defined in Section 4 allowed us to compare MoJoFM and KE in a reduced search space i.e. when the decompositions involved have the same structure. In other words, the ELF vector between the test and authoritative decompositions in this set of experiments was always (0,0,1). To evaluate the relationship between the two measures, the congruity metric [17] was calculated. The congruity metric is a value between 0 and 100. Values close to 0 or 100 indicate strong correlation, while values closer to 50 indicate lack of correlation.

The experiments were performed on a series of 100 randomly generated decompositions. The number of entities in these decompositions was between 1000 and 2000, while the number of clusters varied between 10 and 20. The congruity metric value was always in the interval [42,49]. This indicates a lack of correlation between MoJoFM and KE even when they are comparing decompositions with the same structure. The randomized experiment was repeated 10 times with similar results.

A possible explanation for this rather surprising result (given that the goal of the two measures is the same) is the way they penalize for misplaced entities. MoJoFM applies the same penalty for any misplaced entity (one Move operation). KE applies a variable penalty depending on cluster size. When an entity is misplaced in a big cluster then the penalty is smaller than that for a misplaced entity in a small cluster. For example, misplacing a single entity in a decomposition of 2 clusters results in a KE value of 0.71 when the clusters have sizes 2 and 5. If the cluster sizes are 2 and 25, the KE value is 0.81.

It is also interesting to note that the value of KE was always smaller than that of MoJoFM in this set of experiments. For instance, the range of MoJoFM values was [0.97,1] while the corresponding range for KE was [0.83,1]. KE had values in the range [0.89, 0.98] when MoJoFM was held constant at 0.99.

The above results indicate strongly that MoJoFM and KE have significantly different behaviour even when they are applied to decompositions of similar structure. As a result, researchers need to select the comparison method whose properties better suit the needs of their task at hand.

6. Conclusions

This paper introduced a novel set of structure indicators that can augment the evaluation picture provided by traditional comparison methods, such as MoJoFM and KE.

Several sets of experiments indicate that these structure indicators can be valuable in better understanding the prop-

erties of automatically created decompositions, while they can also help software clustering researchers investigate the similarities and differences of existing comparison methods.

Acknowledgements

The authors would like to thank Brian Mitchell and Spiros Mancoridis for their help regarding the use of the Bunch API, as well as Rainer Koschke for sharing an implementation of the Koschke-Eisenbarth measure with us.

References

[1] B. Andreopoulos, A. An, V. Tzerpos, and X. Wang. Clustering large software systems at multiple layers. *Information & Software Technology*, 49(3):244–254, 2007.

[2] P. Andritsos and V. Tzerpos. Information-theoretic software clustering. *IEEE Trans. Softw. Eng.*, 31(2):150–165, 2005.

[3] N. Anquetil and T. Lethbridge. File clustering using naming conventions for legacy systems. In *Proceedings of CASCON 1997*, pages 184–195, Nov. 1997.

[4] N. Anquetil and T. Lethbridge. Experiments with clustering as a software remodularization method. In *Proceedings of the Sixth Working Conference on Reverse Engineering*, pages 235–255, Oct. 1999.

[5] I. T. Bowman, R. C. Holt, and N. V. Brewster. Linux as a case study: Its extracted software architecture. In *Proceedings of the 21st International Conference on Software Engineering*, May 1999.

[6] S. C. Choi and W. Scacchi. Extracting and restructuring the design of large systems. *IEEE Software*, pages 66–71, Jan. 1990.

[7] M. W. Godfrey and E. H. S. Lee. Secrets from the monster: Extracting mozilla's software architecture. In *Second International Symposium on Constructing Software Engineering Tools*, June 2000.

[8] R. Koschke and T. Eisenbarth. A framework for experimental evaluation of clustering techniques. In *Proceedings of the Eighth International Workshop on Program Comprehension*, pages 201–210, June 2000.

[9] A. Lakhotia and J. M. Gravley. Toward experimental evaluation of subsystem classification recovery techniques. In *Proceedings of the Second Working Conference on Reverse Engineering*, pages 262–269, July 1995.

[10] C. Lindig and G. Snelting. Assessing modular structure of legacy code based on mathematical concept analysis. In *Proceedings of the 19th International Conference on Software Engineering*, pages 349–359, May 1997.

[11] J. I. Maletic and N. Valluri. Automatic software clustering via latent semantic analysis. In *Proceedings of the Fourteenth IEEE International Conference on Automated Software Engineering*. IEEE Computer Society Press, 1999.

[12] S. Mancoridis, B. Mitchell, Y. Chen, and E. Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *Proceedings of the International Conference on Software Maintenance*. IEEE Computer Society Press, 1999.

[13] B. S. Mitchell and S. Mancoridis. Comparing the decompositions produced by software clustering algorithms using similarity measurements. In *Proceedings of the International Conference on Software Maintenance*, pages 744–753, Nov. 2001.

[14] H. A. Müller, M. A. Orgun, S. R. Tilley, and J. S. Uhl. A reverse engineering approach to subsystem structure identification. *Journal of Software Maintenance: Research and Practice*, 5:181–204, Dec. 1993.

[15] R. W. Schwanke, R. Altucher, and M. A. Platoff. Discovering, visualizing, and controlling software structure. In *International Workshop on Software Specification and Design*, pages 147–150. IEEE Computer Society Press, 1989.

[16] M. Shtern and V. Tzerpos. A framework for the comparison of nested software decompositions. In *Proceedings of the Eleventh Working Conference on Reverse Engineering*, pages 284–292, Nov. 2004.

[17] M. Shtern and V. Tzerpos. Lossless comparison of nested software decompositions. In *Reverse Engineering, 2007. Proceedings. 14th Working Conference on*, 2007.

[18] V. Tzerpos. Comprehension-driven software clustering. *Ph.D. Thesis, Department of Computer Science, University of Toronto*, July 2001.

[19] V. Tzerpos and R. C. Holt. MoJo: A distance metric for software clusterings. In *Proceedings of the Sixth Working Conference on Reverse Engineering*, pages 187–193, Oct. 1999.

[20] V. Tzerpos and R. C. Holt. ACDC: An algorithm for comprehension-driven clustering. In *Proceedings of the Seventh Working Conference on Reverse Engineering*, pages 258–267, Nov. 2000.

[21] Z. Wen and V. Tzerpos. An optimal algorithm for MoJo distance. In *Proceedings of the Eleventh International Workshop on Program Comprehension*, pages 227–235, May 2003.

[22] Z. Wen and V. Tzerpos. An effectiveness measure for software clustering algorithms. In *Proceedings of the Twelfth International Workshop on Program Comprehension*, pages 194–203, 2004.

[23] Z. Wen and V. Tzerpos. Evaluating similarity measures for software decompositions. In *Proceedings of the International Conference on Software Maintenance*, 2004.

[24] C. Xiao. Software clustering using static and dynamic data. *Master's thesis, Department of Computer Science, York University, in preparation*, 2004.