A Framework for the Comparison of Nested Software Decompositions

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

Abstract

The evaluation of results obtained from software clustering algorithms has attracted the attention of many reverse engineering researchers. Several methods that compare flat decompositions of software systems have been presented in the literature. However, software clustering algorithms often produce nested decompositions. Converting nested decompositions to flat ones in order to compare them may remove significant information.

In this paper, we introduce a framework called END that reuses comparison methods for flat decompositions in order to compare nested decompositions without loss of information. We also present experimental results with END using several existing methods as plugins that demonstrate its usefulness.

1 Introduction

It is difficult to imagine a world without computers nowadays. Many organizations such as banks, airports, or even smaller companies are relying on software systems in order to function properly. As a result, software systems become more and more complicated, as they attempt to improve constantly to meet market requirements. Often the market demands new software features in a short time. One of the consequences of this trend is that software systems often have poor documentation. In many cases, design documents are never updated.

At the same time, software developers often move to other projects or even different companies. Moreover, software developers are often using code components that were not developed in-house. Finally, research on code cloning indicates that the exchange of source code through the indiscriminate use of copy/paste facilities is a common practice between software developers. As a result, developers often do not know exactly what is going on in the code they are working with.

These factors create situations where software developers can not predict the system's behaviour. Maintenance activities, such as fixing bugs or developing new features, require a lot of effort from developers and testers. A large amount of important business software is in operation today that developers are simply afraid to change. Organizations are often faced with the dilemma of either switching to a new software product, or re-engineering the existing one. Risk assessment processes often indicate the latter approach as the most viable one.

Retrieving design information from the source of a software system is an important problem that affects all stages of the life cycle of a software system. Usually, complicated systems consist of different subsystems that can help divide such a system into logical components. The natural decomposition of a software system is usually presented as a nested decomposition [6]. Many different methodologies and approaches that attempt to create such decompositions automatically have been presented the literature[1, 4, 6, 9, 11, 14]. Most of them produce nested decompositions.

Evaluating the effectiveness of software clustering approaches is a challenging issue. Comparing results produced by different tools is a complicated problem. A number of approaches that attempt to tackle this problem have been presented [7, 10, 13]. However, all of them assume a flat decomposition.

In this paper, we present a framework called END that allows one to reuse a technique developed for flat decompositions in order to compare nested ones. The END framework takes into account all levels of a nested decomposition when making the comparison. We apply several existing comparison methods to the END framework, and we use it to compare nested decompositions for two large software systems. Our experiments show that the END framework can provide more accurate results than the original methods.

The structure of the rest of this paper is as follows. Section 2 describes related work concentrating on existing methods for the comparison of flat decompositions. Section 3 presents the issues that arise when nested decompositions are flattened in order to be compared using existing methods. Our solution to this problem, the END framework, is presented in Section 4. Experiments that showcase the usefulness of the framework are presented 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 [8]. 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 [2] 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 [10, 12].

The MoJo distance measure [13, 15] 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 [16] is based on MoJo distance but produces a number in the range 0-100 which is independent of the size of the decompositions.

Koschke and Eisenbarth [7] 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.

Mitchell and Mancoridis [10] 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.

Finally, the EdgeMoJo measure [17] 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.

3 Flat vs. nested decompositions

The methods presented in the previous section have been developed in order to compare flat decompositions. Since clustering algorithms commonly create nested decompositions, various methods have been devised in order to evaluate clustering results using these methods. The most common approach is to convert the nested decompositions into flat ones before applying the comparison method. However, this approach has limitations.

Converting a nested decomposition to a flat one is commonly done in two different ways depending on whether a compact or a detailed flat decomposition is required:

1. Converting a nested decomposition to a compact flat one. Each object is assigned to its ancestor that is closest to the root of the containment tree. The flat decomposition obtained contains only the top-level clusters of the original one (Figure 1 presents an example of such a conversion).

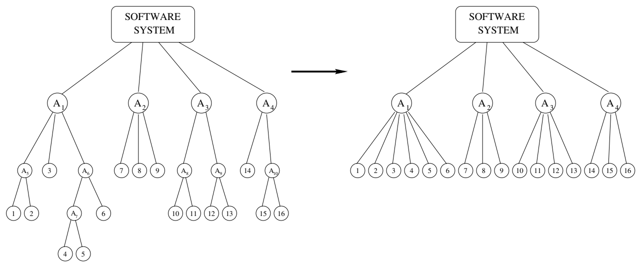


Figure 1. Conversion of a nested decomposition to a compact flat decomposition.

2. Converting a nested decomposition to a detailed flat one. Each cluster and its sub-tree is assigned directly to the root of the containment tree. The decomposition obtained contains any cluster that contained at least one object in the original decomposition (an example with the same nested decomposition as in Figure 1 is presented in Figure 2).

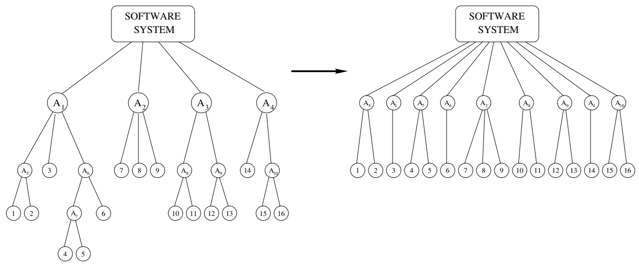


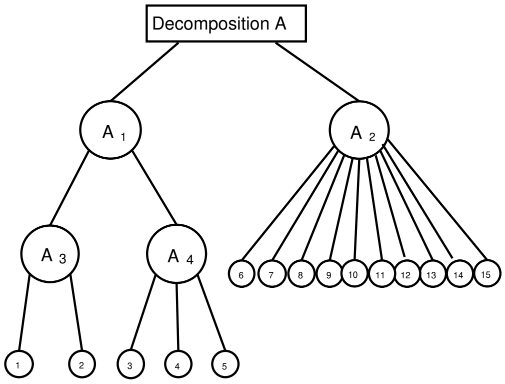
Figure 2. Conversion of a nested decomposition to a detailed flat one.

Figure 3 presents three decompositions of the same software system. Clearly, decompositions (a) and (b) have different hierarchical structures. Such decompositions cannot be compared directly using one of the methods presented already.

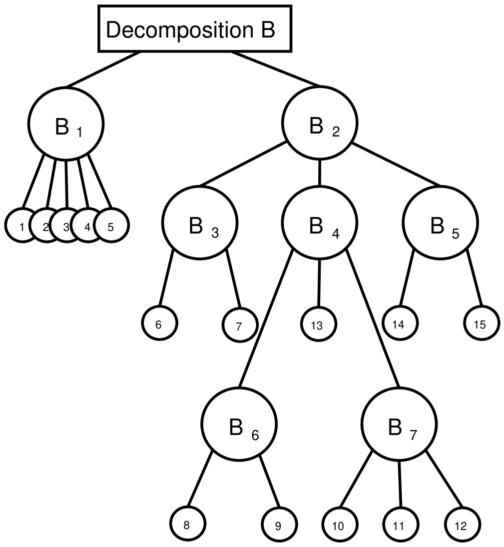
If we convert the decompositions in Figure 3 to compact flat form, they will both become equivalent to decomposition (c). However, the original decompositions were quite different. A significant amount of information has been lost. Figure 4 presents an example where detailed transformation creates a similar problem.

These examples illustrate clearly that converting nested decompositions to flat ones removes significant information that could impact the evaluation process.

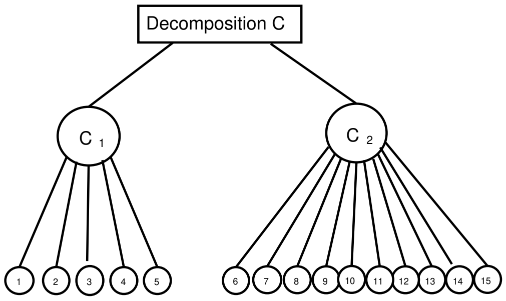
The next section discusses a framework that alleviates this problem.



(a) A nested decomposition of a software system

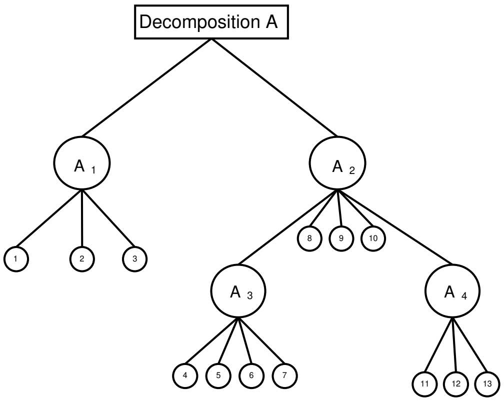


(b) Another nested decomposition of the same software system

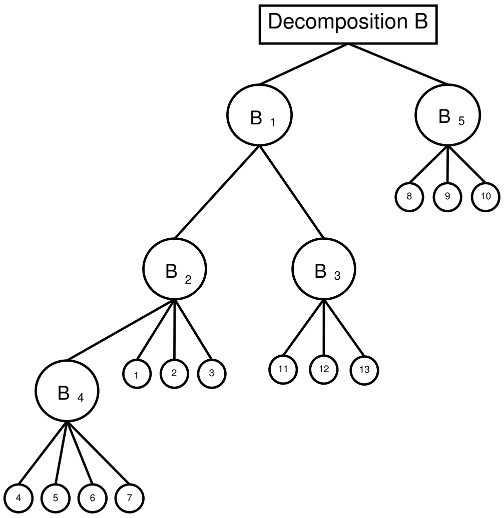


(c) The compact flat form of both decompositions (a) and (b)

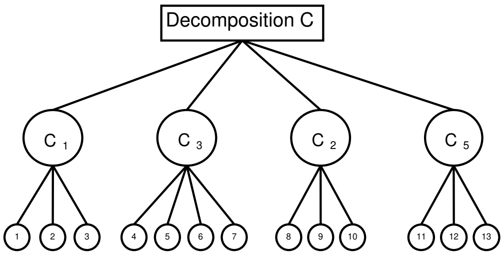
Figure 3. Limitation of compact flat decompositions.



(a) A nested decomposition of a software system



(b) Another nested decomposition of the same software system



(c) The detailed flat form of both decompositions (a) and (b)

Figure 4. Limitation of detailed flat decompositions.

4 The Evaluation of Nested Decompositions (END) framework

The Evaluation of Nested Decompositions (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 selected elements of these vectors. This section presents this process in detail.

We begin by introducing certain terms that will help our discussion.

Each cluster in the nested decomposition is assigned a *level* that is equal to the length of the path from the root of the decomposition to itself. For example, in Figure 3b, cluster B_2 has a level of 1, while cluster B_7 has a level of 3. The *height* of a nested decomposition is defined as its maximum cluster level.

The algorithm that transforms a nested decomposition to a vector of flat ones works as follows:

For each cluster level ℓ in the nested decomposition we construct a flat decomposition as described below:

1. Each object in a cluster of level larger than ℓ is assigned to its ancestor of level ℓ. This results is a nested decomposition T of height ℓ.

2. Convert T to a detailed flat decomposition, as described in Section 3.

We denote a flat decomposition corresponding to level ℓ with P_ℓ.

Figure 5 shows P_2 for the decomposition in Figure 3b.

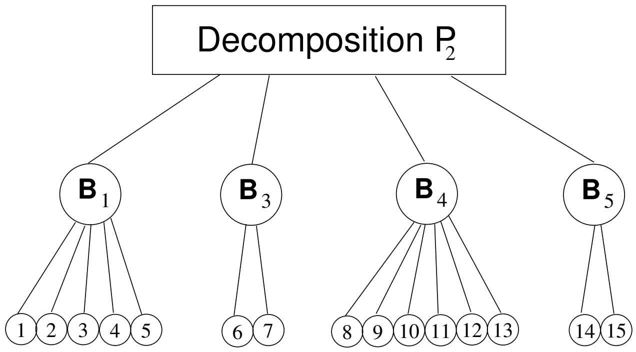


Figure 5. Flat Decomposition P_2 for the decomposition in Figure 3b.

Definition 4.1. Nested decomposition vector (NDV) of a nested decomposition is a vector, where NDV_i=P_i.

The size of an NDV is equal to the height of its corresponding nested decomposition. It is interesting to note that the first element in the NDV is a compact flat decomposition, while the detailed flat decomposition is its last element.

Our method for the comparison of two nested decompositions depends on the fact that a technique to compare flat decompositions already exists. Let us call this technique *M*. Consider two nested decomposition vectors V and U constructed from different software decompositions of the same software system. If the two vectors have different sizes, then the shorter one is expanded so that the two sizes are equal. Let us assume without loss of generality, that V is shorter than U. If the size of V is s, then all new elements appended to V are equal to V_s. In other words, any extra flat decompositions appended to V are equal to the detailed flat decomposition.

Definition 4.2. Similarity vector S_M of two nested decompositions with NDVs V and U is a vector of equal size to V and U, where $S_{M_i} = M(V_i, U_i)$.

As a result, the similarity vector S_M contains the value of the comparison method M for all cluster levels.

The overall similarity between the two nested decomposition will be computed based on the similarity vector. There are many ways to convert the similarity vector to a number, such as taking the norm of the vector. A more flexible approach would allow different weights for different coordinates. The formula would be the following:

$$\sqrt{\sum (w_i {S_{M_i}}^2)} \tag{1}$$

where $\sum w_i = 1$ so that the overall result is normalized and can be compared to numbers obtained without using the END framework.

It is interesting to note that if the vector of weights is (1,0,...,0), then END becomes equivalent to comparing flat compact decompositions, while if the vector is (0,...,0,1), then END is equivalent to comparing flat detailed decompositions. In this sense, the END framework generalizes existing approaches, while providing more flexibility to the reverse engineer.

In the next section, we present several experiments with an implementation of the END framework. We have created plugins for several of the comparison methods presented in Section 2. Unfortunately, the implementation of MeCl that was available to us contained a bug since the output was frequently outside the expected range of 0-100. For this reason, we only present results for MoJo, MoJoFM, Koschke-Eisenbarth, EdgeSim, and EdgeMoJo.

5 Experiments

The experiments presented in this section were run on nested 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 [3].

- **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. A decomposition of the Mozilla source files for version M9 was presented in [5]. For the evaluation portion of our work, we used an updated decomposition for version 1.3 [18].

Apart from the authoritative decompositions of these two systems, we also created nested 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 commonly found in manually-created decompositions of large software systems [14].

- **Bunch**. This is a suite of algorithms that attempt to find a decomposition that optimizes a quality measure based on high-cohesion, low-coupling [9].

Table 1 summarizes the decompositions used in the following experiments. It also presents the abbreviations we will use for them in this paper, as well as their height.

Sign	Description	Height
L	Linux Authoritative Decomposition	6
LA	Linux ACDC Decomposition	3
LB	Linux Bunch Decomposition	4
M	Mozilla Authoritative Decomposition	5
MA	Mozilla ACDC Decomposition	4
MB	Mozilla Bunch Decomposition	5

Table 1. Nested decompositions used in the experiments.

We conducted two different types of experiments as outlined in the next two sections.

5.1 END vs Existing Approaches

The target of the first series of experiments is to compare the results obtained from existing comparison methods to the results obtained by the END framework using the same measurement as a plug-in. We calculated the (dis)similarity between different nested decompositions using both the END framework and the standalone approaches. For each standalone approach, we calculated two values: the value obtained when using the compact flat decomposition, and the value obtained when using the detailed one. In this series of experiments, END is using the same weight for all components of the similarity vector.

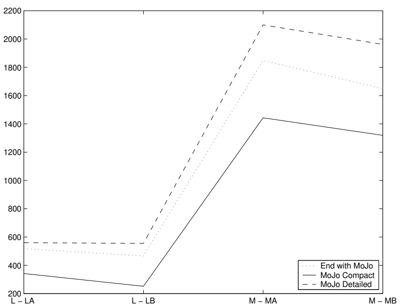
Table 2 presents a summary of the obtained results. A graphical representation of the results is also given in Figure 6. In all cases, a lower value indicates that the two decompositions were more similar. Note that values for the Koschke-Eisenbarth measure range from 0 to 1, MoJoFM and EdgeSim values range from 0 to 100, while MoJo and EdgeMoJo values are positive and bounded by s and $2s$ respectively, where s is the number of elements in each decomposition.

A number of interesting observations can be made based on these results.

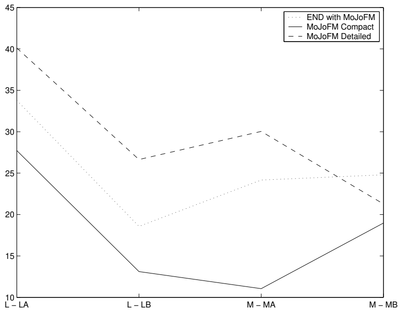
To begin with, it is noteworthy that if one assumes balanced decompositions in terms of the number of objects per cluster, as well as the number of clusters per level, the expected order of the three plots in each graph would be: the plot corresponding to the detailed decomposition should be

Method	L - LA	L - LB	M - MA	M - MB
MoJo C	342	252	1443	1319
MoJo D	560	555	2100	1962
MoJo E	517.67	466.02	1848.80	1649.37
MoJoFM C	27.73	13.11	11.05	18.97
MoJoFM D	40.11	26.64	30.04	21.28
MoJoFM E	33.78	18.57	24.17	24.80
Koschke C	0.07	0.14	0.06	0.08
Koschke D	0.36	0.15	0.17	0.12
Koschke E	0.23	0.12	0.15	0.12
EdgeSim C	55.66	74.24	59.69	71.68
EdgeSim D	70.75	86.17	68.88	76.37
EdgeSim E	64.55	80.46	65.80	74.66
EdgeMoJo C	1317.10	1581.56	6193.23	5336.21
EdgeMoJo D	987.88	1331.72	4303.46	5387.47
EdgeMoJo E	1161.72	1508.87	5066.31	4881.42

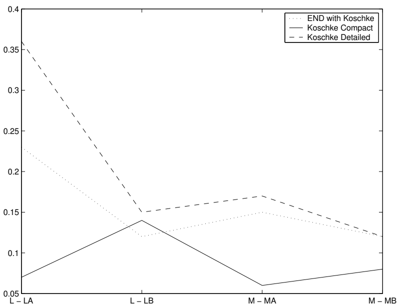
Table 2. Summary of Comparing Nested Decompositions (C = compact, D = detailed, E = END).



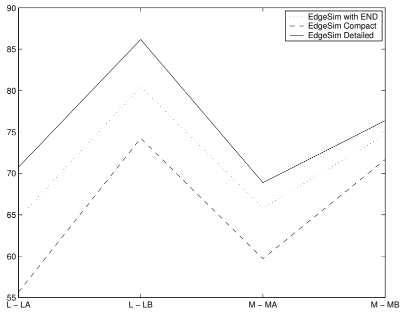
(a) MoJo



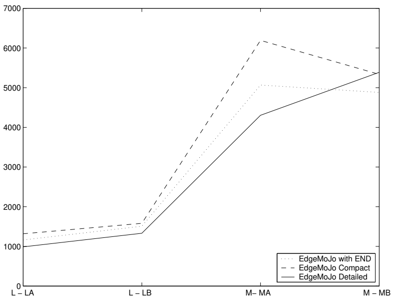
(b) MoJoFM



(c) Koschke-Eisenbarth



(d) EdgeSim



(e) EdgeMoJo

Figure 6. Comparing END to various standalone approaches.

at the top, followed by the one corresponding to the END framework, while the plot corresponding to the compact decomposition should be the lowest. As can be seen in Figure 6, MoJo and EdgeSim follow this pattern. Any deviations from this - usually indicated by a crossing between the plots - are worthy of discussion.

For example, in the MoJoFM graph the "END plot" is higher than the "detailed plot" for the data point that corresponds to Mozilla and Bunch. This indicates that considering the intermediate levels as well helps Bunch achieve a better result. In other words, it appears that Bunch selects good top-level clusters, but does not necessarily do as well for the fine-grained ones. Since Bunch provides an option to create a decomposition at any level requested by the user, our experiments would indicate that selecting a higher level is recommended.

Another interesting observation is that the expected order for EdgeMoJo is actually reversed. This indicates that the edge component of EdgeMoJo penalizes more at the compact decomposition level. This can be explained by the fact that at that level, a *Move* operation between two clusters would result in the displacement of a large number of edges. This interesting property of the EdgeMoJo metric was revealed through the use of the END framework for the first time.

The results for the Koschke-Eisenbarth measure corroborate our previous observation for Bunch, since the END framework and the detailed decomposition produce the same value for Mozilla. However, this time we see a different phenomenon for Linux. The compact and detailed plots are far apart for ACDC, but they are almost equal for Bunch. This is exactly the motivation for this work. Concentrating on a particular flat decomposition can provide biased results. Considering all levels (possibly with different weights) can give a better view of the quality of a particular decomposition.

Finally, it is interesting to note that in all experiments the values obtained for ACDC conformed to the expected order (with the already explained exception for EdgeMoJo). This is probably due to the fact that ACDC creates clusters of bounded cardinality, a restriction that does not apply to Bunch.

5.2 Various weighting schemes

As explained above, the END framework does not specify exact rules for the conversion of the similarity vector to a number. This can be done in a way that fits the specific needs of the reverse engineer using the framework. The previous experiments used equal weights for all elements in the similarity vector. However, other weighting schemes are also possible. We present and experiment with five of them:

1. The first element of the similarity vector has larger weight than the rest, i.e. more importance is assigned to the compact flat decomposition. We denote this weighting scheme by F.
2. The last element of the similarity vector has larger weight than the rest, i.e. more importance is assigned to the detailed flat decomposition. We denote this weighting scheme by L.
3. All elements of the similarity vector have the same weight (denoted by ALL).
4. Each element in the vector has a weight equal to its index, i.e. the first element has a weight of 1, the second a weight of 2 etc. These weights are of course normalized before they are applied. We denote this scheme by UP.
5. Similar to UP, but in reverse order, i.e. the last element has a weight of 1, the second last a weight of 2 etc. (denoted by DOWN).

The actual formulas that were used in each case are given in Table 3. For simplicity, in these formulas $x_i = S_{M_i}$.

Abbr	Formula
F	$\sqrt{(}\sum(x_i^2 \frac{1}{N+1}) + x_0^2 \frac{2}{N+1})$
L	$\sqrt{(}\sum(x_i^2 \frac{1}{N+1}) + x_N^2 \frac{2}{N+1})$
ALL	$\sqrt{(}\sum(x_i^2 \frac{1}{N}))$
UP	$\sqrt{(}\sum(x_i^2 \frac{2i}{N(N+1)}))$
DOWN	$\sqrt{(}\sum(x_i^2 \frac{2(N-i+1)}{N(N+1)}))$

Table 3. Formulas for all weighting schemes

Table 4 presents the results of these experiments. They are also presented in graphical form in Figure 7.

Assuming balanced decompositions, the expected order for the various plots in these graphs is the following (in order from top to bottom): UP, L, ALL, F, DOWN. As can be seen in Figure 7, the values for MoJo follow this pattern. As before, crossings in these graphs make for interesting observations.

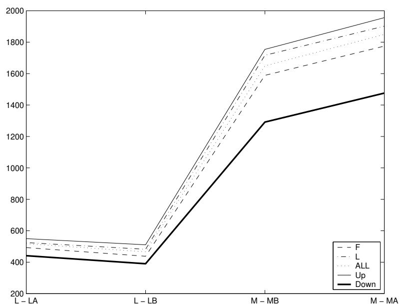
MoJo	L - LA	L - LB	M - MA	M - MB
F	492.76	437.68	1775.08	1588.80
L	524.97	481.99	1901.70	1716.46
ALL	517.67	466.02	1848.80	1649.37
UP	549.78	510.87	1955.81	1754.17
DOWN	441.27	390.12	1475.81	1291.45
MoJoFM	L - LA	L - LB	M - MA	M - MB
F	32.85	17.77	22.17	23.75
L	34.91	20.14	25.45	24.14
ALL	33.78	18.57	24.17	24.80
UP	35.74	20.85	26.81	25.12
DOWN	27.35	14.05	20.02	21.34
Koschke	L - LA	L - LB	M - MA	M - MB
F	0.21	0.12	0.13	0.12
L	0.25	0.13	0.15	0.12
ALL	0.23	0.12	0.15	0.12
UP	0.27	0.12	0.16	0.13
DOWN	0.17	0.08	0.13	0.11
EdgeSim	L - LA	L - LB	M - MA	M - MB
F	63.16	79.46	64.62	74.07
L	65.63	81.44	65.80	75.01
ALL	65.55	80.46	67.30	74.66
UP	66.81	82.33	52.00	75.48
DOWN	53.27	65.83	74.07	58.28
EdgeMoJo	L - LA	L - LB	M - MA	M - MB
F	1189.02	1521.23	5310.86	4975.70
L	1134.59	1480.82	4923.21	4986.74
ALL	1161.72	1508.87	5066.31	4881.42
UP	1105.45	1466.79	4727.89	4897.23
DOWN	948.11	1252.26	3693.19	3504.79

Table 4. Experimental results with various weighting schemes.

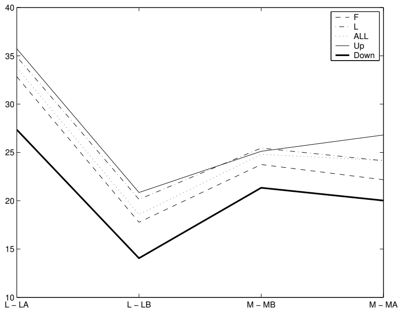
For instance, in the MoJoFM graph, we have that UP is lower than L in the case of Mozilla and Bunch. This means that when a lower weight is assigned to the compact decomposition, the result becomes worse for Bunch. This is in agreement with the observation in the previous section that Bunch performs better at coarse-grained levels.

A similar phenomenon takes place for the Koschke-Eisenbarth measure but in the case of Linux and Bunch. It is quite intriguing that a similar deviation was observed for this measure in the previous section as well. Further investigation is required to determine what feature of this metric is responsible for this behaviour with regard to the Linux decompositions.

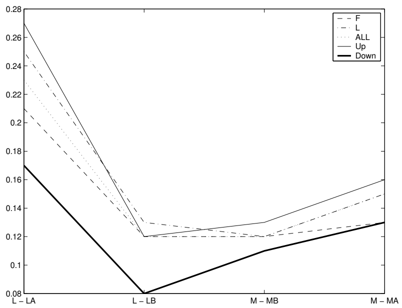
The results for EdgeMoJo are again in the reverse order than expected for the reasons described above. At the same time, the EdgeSim results follow the expected order except



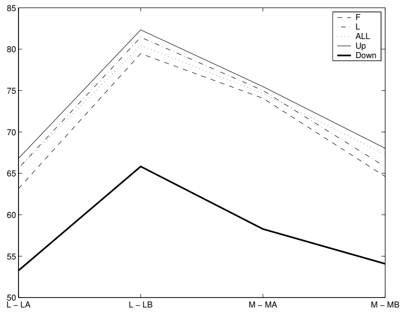
(a) MoJo



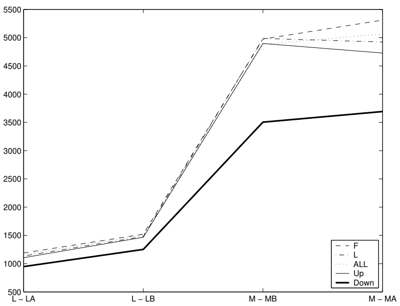
(b) MoJoFM



(c) Koschke-Eisenbarth



(d) EdgeSim



(e) EdgeMoJo

Figure 7. Weighting scheme results.

that the result for ALL is better than that for L in the case of Mozilla and ACDC, the only deviation from the expected order we observed for ACDC.

Finally, it is quite interesting to note that through all these experiments, the results produced by ACDC were rated higher that those of Bunch in approximately 50% of the experiments. Perhaps not surprisingly, MoJo rates the ACDC results consistently better, while EdgeSim does so for the results of Bunch. A more detailed comparative study of this behaviour is part of our future plans.

Our overall observation from these experiments is that the END framework has revealed many interesting properties of the clustering algorithms we experimented with as well as the comparison methods we used as plugins. We are hopeful that reverse engineers that are more familiar with the software system they are dealing with, as well as the clustering algorithm they are using, will be able to benefit even more from the use of this framework.

6 Conclusions

This paper presented a framework called END that allows a reverse engineer to compare nested decompositions of large software systems without having to lose information by transforming them to flat ones first. We also presented several experiments that demonstrated that the END framework can provide useful insights into the nature of software clustering algorithms, as well as the behaviour of existing comparison methods for flat decompositions.

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] P. Andritsos and V. Tzerpos. Software clustering based on information loss minimization. In *Proceedings of the Tenth Working Conference on Reverse Engineering*, pages 334–344, Nov. 2003.

[2] 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.

[3] 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.

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

[5] 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.

[6] D. H. Hutchens and V. R. Basili. System structure analysis: Clustering with data bindings. *IEEE Transactions on Software Engineering*, 11(8):749–757, Aug. 1985.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] H. A. Müller and J. S. Uhl. Composing subsystem structures using (k,2)-partite graphs. In *Conference on Software Maintenance*, pages 12–19, Nov. 1990.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

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