On the Comparability of Software Clustering Algorithms

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

Abstract

Evaluation of software clustering algorithms is typically done by comparing the clustering results to an authoritative decomposition prepared manually by a system expert. A well-known drawback of this approach is the fact that there are many, equally valid ways to decompose a software system, since different clustering objectives create different decompositions. Evaluating all clustering algorithms against a single authoritative decomposition can lead to biased results.

In this paper, we introduce and quantify the notion of clustering algorithm comparability. It is based on the concept that algorithms with different objectives should not be directly compared. Not surprisingly, we find that several of the published algorithms in the literature are not comparable to each other.

1 Introduction

Perhaps the most well-known tenet of clustering research is that there is no single correct answer to a clustering problem [6]. There are many, equally valid ways to decompose a data set into clusters. Similarly, in the context of software clustering, there are many, equally valid ways to decompose a software system into meaningful subsystems. Different software clustering algorithms employ different clustering objectives producing different results. All such clustering results could be useful to someone attempting to understand the software system at hand.

It is, therefore, rather strange that the typical way to evaluate a software clustering algorithm is to compare its output to a single authoritative decomposition, i.e. a decomposition created manually by a system expert. If the authoritative decomposition was constructed with a different point of view in mind than the algorithm, the evaluation results will probably be biased against the algorithm. However, no better way to evaluate software clustering algorithms exists at the moment.

In this paper, we introduce and quantify the notion of software clustering algorithm *comparability*. Comparable algorithms use the same or similar clustering objectives, produce similar results, and can therefore be evaluated against the same authoritative decomposition. Non-comparable algorithms use different clustering objectives, produce significantly different results, and cannot be evaluated by direct comparison to an authoritative decomposition.

Being able to distinguish between comparable and non-comparable algorithms allows us to define families of algorithms that share common clustering objectives. Conventional evaluation techniques apply only within the context of a given algorithm family.

As expected, our experimental results clearly show that several existing software clustering algorithms belong in different families, and therefore should not be directly compared against each other.

We also utilized the concept of algorithm families in order to answer the question of whether conventional evaluation measures, such as MoJoFM [15] and KE [8], behave in a congruent fashion by limiting our investigation only to comparable algorithms.

The structure of the rest of the paper is as follows. The notion of comparability as well as the way we quantify it is presented in Section 2. Experiments that verify whether several published algorithms are comparable, are presented in Section 3. The comparison of MoJoFM and KE in the context of comparability is shown in Section 4. Finally, Section 5 concludes the paper.

2 Clustering Algorithm Comparability

The various software clustering algorithms published in the literature have different clustering objectives, i.e. assumptions on what constitutes a meaningful decomposition of a software system. This leads to clustering results that, while significantly different from each other, offer equally valid points of view on the system's high level structure. Evaluating such clustering results by comparing them to

a single authoritative decomposition, that may or may not have been created with the same clustering objective in mind, can lead to significantly biased results.

For this reason, we define that two clustering algorithms that produce significantly different results are *non-comparable*, i.e. they should not be compared against the same authoritative decomposition since the obtained results will be meaningless if not misleading. Conversely, we define two clustering algorithms to be *comparable* when they produce similar results. Comparable algorithms can be compared using an authoritative decomposition.

We quantify what is meant by "similar clustering results" in Section 2.1. In order to do so, we require a large number of module dependency graphs (MDGs) extracted from software systems. Since this is impractical to do, we utilized an approach that generates simulated MDGs [11].

2.1 Quantifying Comparability

In this section, we present a method that determines whether two software clustering algorithms A and B are comparable or not.

Our method begins by generating a large number of MDGs using the generation approach mentioned above. The two algorithms are applied to all MDGs. In order to measure the similarity between the various clustering results, we use the MoJoFM measure [15]. A MoJoFM value of 100 means that the two algorithms construct the exact same decomposition. A MoJoFM value of 0 means that the two algorithms produce decompositions that are completely different, i.e. they disagree about the placement of all entities. Since MoJoFM is non-symmetric, we define the similarity between clustering algorithms A and B as $\min(MoJoFM(A,B), MoJoFM(B,A))$.

This allows us to compute the average similarity μ across all generated MDGs, as well as the standard deviation σ.

We consider two algorithms to be non-comparable if they produce significantly different results across a variety of software systems. More precisely, we define that two algorithms are non-comparable if in the majority of cases they produce results that are more different than similar, i.e. the similarity is less than 50%.

We use the well-known 3σ rule [10] to capture the notion of "the majority of cases". The 3σ rule states that 99.73% of the values of a normal distribution lie within 3 standard deviations of the mean. This allows us to disregard outliers that occur very infrequently. As a result, our definition becomes:

Two software clustering algorithms are non-comparable if it holds that

$$\mu + 3\sigma < 50$$

If the above does not hold, then the algorithms are considered comparable. This definition may appear biased towards making algorithms look comparable, since one could have used only the average in the definition. However, as will be shown in the next section, even with this definition most major algorithms are shown to be non-comparable.

3 Experiments

The first set of experiments attempts to determine the comparability of the following clustering algorithms: Bunch [9], ACDC [14], LIMBO [2], and several hierarchical clustering algorithms. We compared the performance of these algorithms on 10 simulated MDGs. The results of these experiments are presented in Table 1.

For our experiments, we used six different hierarchical clustering algorithms: all possible combinations of three update rule functions (complete linkage, weighted average, and unweighted average) and two similarity metrics (Jaccard and Sorensen–Dice). The cut-point heights were chosen so as to maximize the similarity values (again possibly biasing the results in favour of comparability). The designation Hierarchical in Table 1 refers to all six variations, i.e. the values reported are averaged across all hierarchical algorithms.

In order to determine how realistic the simulated MDGs are, Table 1 also provides the similarity values when the various algorithms are applied to three real systems: Linux, Mozilla, and TOBEY (a description of these systems is shown in the appendix). With the exception of the LIMBO-Hierarchical pair, all real similarity values fall within one standard deviation of the average simulated similarity values. This is a strong indication that the simulated MDGs closely resemble graphs from real systems.

An immediate observation is that most of the selected algorithms are not comparable to each other. For instance, the decompositions created by LIMBO are significantly different than those created by the other software clustering algorithms. A possible explanation is the fact that LIMBO has a unique objective when constructing software decompositions: the minimization of information loss.

The only case where a possible pair of comparable algorithms may be found is in the first row of Table 1. ACDC appears to be comparable to our set of hierarchical algorithms. In order to investigate further, we compared ACDC to each individual hierarchical algorithm, as shown in Table 2.

A first observation is that ACDC is comparable to a majority of the selected hierarchical algorithms. This can probably be attributed to ACDC's hierarchical nature: Subgraph dominator pattern instances are composed in agglomerative fashion in order to achieve the goal of limiting the number of elements in each subsystem [14].

Algorithm pair	μ	σ	$\mu + 3\sigma$	Comparable	Linux	Mozilla	TOBEY
Hierarchical and ACDC	35.07	6.74	57.29	Yes	41.6	26.92	28.12
Bunch and ACDC	25.07	7.66	48.35	No	31.32	19.97	34.06
Hierarchical and Bunch	16.47	4.77	30.78	No	33.34	20.56	19.03
LIMBO and ACDC	10.27	5.03	25.36	No	16.08	5.37	16.25
Bunch and LIMBO	8.28	3.33	18.27	No	9.09	5.35	15.77
LIMBO and Hierarchical	4.30	2.14	10.27	No	14.72	11.95	15.56

Table 1. Pair-wise comparability results for the selected clustering algorithms

Algorithm pair	μ	σ	$\mu + 3\sigma$	Comparable
ACDC and Complete Linkage with Jaccard	38.63	6.85	59.18	Yes
ACDC and Complete Linkage with Sorensen–Dice	40.72	4.53	54.31	Yes
ACDC and Weighted Average with Jaccard	33.46	4.57	47.1	No
ACDC and Weighted Average with Sorensen–Dice	32.88	8.29	57.75	Yes
ACDC and Unweighted Average with Jaccard	31.67	4.71	45.8	No
ACDC and Unweighted Average with Sorensen–Dice	32.98	7.65	56.83	Yes

Table 2. Comparability of ACDC to Hierarchical Algorithms

Moreover, the close comparability of ACDC to complete linkage suggests that ACDC constructs cohesive clusters, a well-known property of the complete linkage algorithm [3, 5]. This fact also supports our earlier finding that ACDC and Bunch are not comparable. Bunch attempts to produce decompositions with high cohesion and low coupling, while ACDC focuses mostly on high cohesion.

The above findings illustrate an important benefit of assessing algorithm comparability. One may be able to deduce properties of an algorithm they are considering for a software clustering task by studying the properties of comparable, well-known algorithms.

The experimental results presented above seem to validate the assumption we hinted at in the beginning of Section 2. Two clustering algorithms that have different objectives are by definition non-comparable, i.e. their results should not be compared against a single authoritative decomposition.

4 Comparability and Clustering Similarity

One of the open questions in software clustering is whether clustering similarity measures, such as MoJoFM [15] and KE [8], behave in a congruent fashion. The notion of comparability allows us to investigate this issue from a different angle: Is the behaviour of the two measures more congruent when evaluating comparable algorithms?

In order to measure congruity in the context of comparability we have extended a metric introduced earlier [12] that measures correlation between two evaluation metrics M and U. The original congruity metric generates N random pairs of decompositions of the same software system. Then it applies both M and U to each pair and obtains two different values m_i and u_i. We arrange the pairs of decompositions in such a way so that for $1 \leq i \leq N-1$ we have $m_i \leq m_{i+1}$. The value of the congruity metric is the number of distinct values of i for which $u_i > u_{i+1}$. Values for this metric range from 0 to $N-1$. A value of 0 means that both comparison methods rank all pairs in exactly the same order, while a value of $N-1$ probably indicates that we are comparing a distance measure to a similarity measure. Values significantly removed from both 0 and $N-1$ indicate important differences between the two comparison methods.

The main drawback of the original metric was that it was based on randomly generated decompositions that may be significantly different from each other, a situation that is unlikely when the two decompositions are actual clustering results. The new congruity metric generates the required pair of decompositions using the following process:

1. An MDG is generated randomly.
2. Two different algorithms construct two software decompositions based on the same MDG.

The improved congruity metric was used in a series of experiments that investigated the behaviour of MoJoFM and KE when evaluating comparable algorithms. The stated goal of both measures is to quantify the effectiveness of a software clustering algorithm. However, they attempt to

measure quality in different ways often resulting in contradictory results [1, 2]. In these experiments, we attempted to remove the effect that non-comparability may have in the measures' assessment.

The experiments were performed on a series of 100 decompositions generated using comparable algorithms. In the experiments, we used the complete linkage, single linkage, weighted average and unweighted average clustering algorithms. The number of entities in these decompositions was 1000.

The congruity metric value was always in the interval [43, 51]. This indicates a clear lack of correlation between MoJoFM and KE even when they are comparing decompositions produced by comparable algorithms. This corroborates similar results in the context of structure similarity [13], and strengthens our position that the two measures have fundamental differences that cannot be reconciled.

5 Conclusions

This paper introduced and quantified the notion of comparability for software clustering algorithms. Experiments showed that many of the published algorithms for software clustering are non-comparable.

We determined that the MoJoFM and KE evaluation measures do not exhibit congruent behaviour even when used to evaluate comparable algorithms.

A Experiment systems

In our experiments, we used the following three large software systems:

- **TOBEY**. This is a proprietary industrial system that is under continuous development. It serves as the optimizing back-end for a number of IBM compiler products. The version we worked with was comprised of 939 source files and approximately 250,000 lines of code. The authoritative decomposition of TOBEY was obtained over a series of interviews with its developers.

- **Linux**. We experimented with version 2.0.27a of the kernel 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. The authoritative decomposition of this version of the Linux kernel was presented in [4].

- **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 [16].

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. Experiments with clustering as a software remodularization method. In *Reverse Engineering, 1999. Proceedings. Sixth Working Conference on*, pages 235–255, 6-8 Oct. 1999.

[4] 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.

[5] J. Davey and E. Burd. Evaluating the suitability of data clustering for software remodularisation. In *Reverse Engineering, 2000. Proceedings. Seventh Working Conference on*, pages 268–276, 23-25 Nov. 2000.

[6] B. S. Everitt. *Cluster Analysis*. John Wiley & Sons, 1993.

[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] S. Mancoridis, B. Mitchell, C. Rorres, Y. Chen, and E. Gansner. Using automatic clustering to produce high-level system organizations of source code. In *IEEE proceedings of the 1998 International Workshop on Program Comprehension*. IEEE Computer Society Press, 1998.

[10] F. Pukelsheim. The three sigma rule. *The American Statistician*, 48(2):88–91, May 1994.

[11] M. Shtern. The factbase generator tool. Available online: `https://wiki.cse.yorku.ca/project/cluster/mdg_generator`.

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

[13] M. Shtern and V. Tzerpos. Refining clustering evaluation using structure indicators. In *ICSM '09: Proceedings of the 25st IEEE International Conference on Software Maintenance*, pages 297–305, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[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 effectiveness measure for software clustering algorithms. In *Proceedings of the Twelfth International Workshop on Program Comprehension*, pages 194–203, 2004.

[16] C. Xiao. Using dynamic analysis to cluster large software systems. Master's thesis, York University, 2004.