An effectiveness measure for software clustering algorithms

Zhihua Wen and Vassilios Tzerpos
York University
Toronto, Ontario, Canada
{zhihua,bil}@cs.yorku.ca

Abstract

Selecting an appropriate software clustering algorithm that can help the process of understanding a large software system is a challenging issue. The effectiveness of a particular algorithm may be influenced by a number of different factors, such as the types of decompositions produced, or the way clusters are named.

In this paper, we introduce an effectiveness measure for software clustering algorithms based on MoJo distance, and describe an algorithm that calculates its value. We also present experiments that demonstrate its improved performance over previous measures, and show how it can be used to assess the effectiveness of software clustering algorithms.

1 Introduction

One of the main goals of software clustering research is to develop algorithms that automatically decompose large software systems into smaller, easier to comprehend subsystems. Many such techniques have been presented in the literature [11, 5, 10, 8, 2, 14, 3, 1]. They all demonstrate promising results when tested on example systems.

However, evaluating the effectiveness of various software clustering algorithms, and comparing their strengths and weaknesses, remains an open question. The software clustering field needs to designate a set of large, publically available software systems with well-understood decompositions that can be used as benchmarks.

Even if such a benchmark set existed though, the question of how to compare automatically created decompositions to the "gold standards" would remain a challenging one. Several researchers have attempted to tackle this difficult problem [7, 3, 6, 9]. One of the first approaches to be presented in the literature, was the MoJo distance measure [12, 15].

MoJo distance between two clusterings A and B of the same software system is defined as the minimum number of *Move* or *Join* operations one needs to perform in order to transform either A to B or vice versa. The smaller the MoJo distance between an automatically created decomposition A and the "gold standard" decomposition B, the more effective the algorithm that created A.

This indicates that MoJo distance can be helpful in comparing the relative effectiveness of various clustering algorithms. However, it is not particularly well-suited to assess the effectiveness of an algorithm in isolation, since the same distance value might indicate a good result if the algorithm was applied to a large software system, and a poor result if the software system in question was of small size.

For this reason, a "quality metric" given by the following formula was presented in [12]:

$$Q(M) = (1 - \frac{MoJo(A, B)}{n}) \times 100\% \qquad (1)$$

where M is the software clustering technique being examined, A is the automatically created decomposition, B is the "gold standard" one, and n is the number of software entities being clustered.

In this paper, we outline several shortcomings of this metric, and introduce a new "e**F**fectiveness **M**easure" based on MoJo distance that we call MoJoFM.

The structure of the rest of this paper is as follows: Section 2 introduces the features of MoJoFM that help overcome the shortcomings of the original "quality metric". Section 3 presents the parts of the algorithm for the calculation of MoJo distance presented in [15] that are necessary for the calculation of MoJoFM, which is described in section 4. Experiments that showcase the improved performance of MoJoFM, as well as its usefulness for assessing the effectiveness of software clustering algorithms are presented in section 5. Finally, section 6 concludes the paper.

2 MoJoFM features

The idea behind the original "quality metric" (we will refer to it as Q from now on) was that an algorithm that produces the farthest partition away from the "gold standard"

should have a quality of 0%, while an algorithm that produces the "gold standard" should have a quality of 100%. While this is the basis for our effectiveness measure as well, there were three shortcomings in the way this was implemented in [12]:

1. The original implementation was using a heuristic algorithm called HAM to calculate the value of MoJo distance. As shown in [15], the actual MoJo distance might have been significantly smaller than the calculated one (up to 19%, with an average difference of 4%).

2. MoJo distance is a non-symmetric measure, i.e. the minimum number of *Move* or *Join* operations to transform partition A to partition B is not necessarily the same as the minimum number of operations to transform B to A. We express this as: $mno(A, B) \neq mno(B, A)$. The distance between A and B is defined as $MoJo(A, B) = min(mno(A, B), mno(B, A)$. As a result, using $MoJo(A, B)$ in formula 1 is incorrect since we are only interested in how close A comes to B (expressed by $mno(A, B)$), and not in how close B comes to A (expressed by $mno(B, A)$ and possibly by $MoJo(A, B)$). Also, using $mno(A, B)$ instead of $MoJo(A, B)$ avoids the following paradox:

   Let ONE be a clustering algorithm that always produces a clustering of cardinality 1 (all objects in one cluster). The quality of such an algorithm should be very low. However, if we use $MoJo(A, B)$ in our quality measure, the computed value will be very high. Using TOBEY[1] as an example (its authoritative clustering contains 69 clusters and 939 objects) we would have $Q(ONE) = (1 - 68/939) \times 100\% = 92.8\%$, a value no "normal" clustering algorithm could ever hope to reach. This is because the authoritative clustering of TOBEY can be transformed into the output of algorithm ONE by just joining all its clusters. Using $mno(A, B)$ instead, we get the more reasonable value of 17.9% for the quality of algorithm ONE.

   Interestingly enough, the quality metric value of algorithm EACH, which always produces a clustering where each cluster contains exactly one object, is the same whether we use $mno(A, B)$ or $MoJo(A, B)$ in formula 1. In the case of TOBEY, this value is 7.3%.

3. The denominator in formula 1 was chosen to be n since it is trivial to show that $MoJo(A, B) < n$ (any partition of n objects can be transformed to any other partition of the same set of objects using less or equal to $n - 1$ *Move* operations). Therefore, the value of Q would always have been between 0% and 100%. However, using n as the denominator is not optimal, because it is the maximum MoJo distance between two specific partitions, the ones produced by algorithms ONE and EACH. For a given "gold standard" B, the maximum distance to it might be smaller. For example, consider the partition shown in figure 1. Though we use $n = 9$ as the denominator, the maximum MoJo distance to this partition is in fact 6. The value of Q in this case will be always between 33.3% and 100%. Therefore, its range has been shortened and its value over-evaluated. A more accurate calculation of the maximum distance to a given partition B is desirable.

[1] See section 5 for a description of TOBEY.

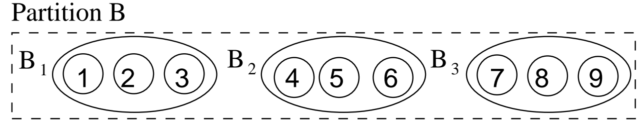

Figure 1. Example partition B.

The effectiveness measure presented in this paper overcomes the above shortcomings. MoJoFM has the following features:

1. It uses $mno(A, B)$ instead of $MoJo(A, B)$ in the numerator of its formula to avoid the paradox mentioned above.

2. It uses the optimal algorithm presented in [15] for the calculation of $mno(A, B)$.

3. It calculates the actual maximum distance to partition B for the denominator of its formula. We denote this by $max(mno(\forall A, B))$, and present its calculation in section 4.

Therefore, the MoJoFM formula is:

$$MoJoFM(M) = (1 - \frac{mno(A, B)}{max(mno(\forall A, B))}) \times 100\% \quad (2)$$

In the following section, we present the algorithm that computes $mno(A, B)$.

3 Calculating mno(A,B)

Assume S is a software system containing n objects, and A and B are two decompositions of S, $A = A_1, A_2, ..., A_l$ and $B = B_1, B_2, ..., B_m$. An example is shown in figure 2, where A and B are two partitions of the same system containing 16 objects.

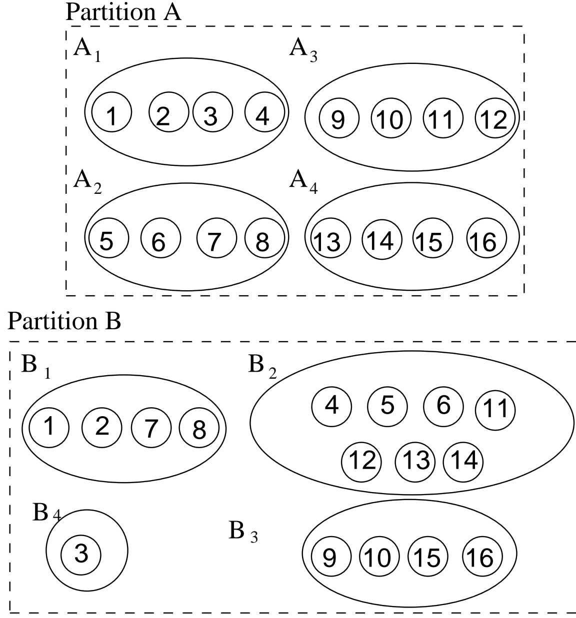

Figure 2. Example partitions A and B.

For each object in A, we find its corresponding subsystem in B. If an object in A_i belongs in B_j in partition B, we give this object a tag T_j. The tags are shown in figure 3 as a rectangle next to each object. For example, since object ④ is in B_2 in partition B, it is assigned tag T_2 (shown as $\boxed{2}$).

Let us denote the intersection between A_i and B_j as v_{ij}. Thus, $v_{ij} = |A_i \cap B_j|$ or $v_{ij} = |A_i(T_j)|$. For example, in figure 3, $v_{11} = 2, v_{12} = 1, v_{14} = 1, v_{21} = 2, v_{22} = 2$, etc.

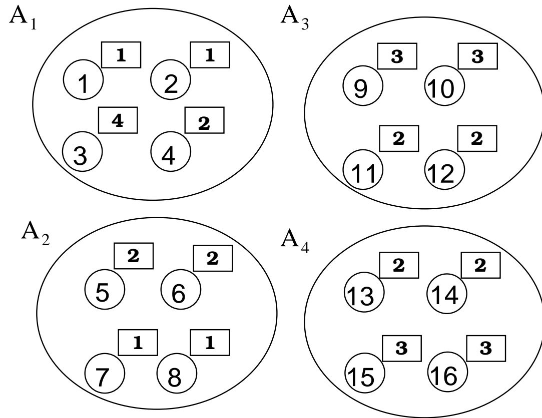

Figure 3. Partition A with the tags.

Our algorithm considers only the tags of each object from now on. Figure 4 presents partition A including only the tags of each object.

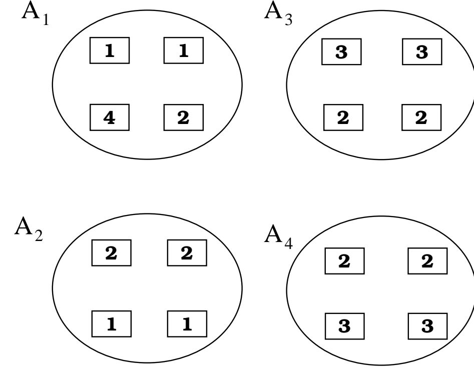

Figure 4. Partition A containing only tags.

Next, we create m sets G_k, one for each cluster in B (we will refer to set G_k as group G_k). For every A_i, we say that A_i belongs in group G_k if $v_{ik} = max_{j=1}^{m}(v_{ij})$. For example, in figure 4, A_1 contains two $\boxed{1}$s, one $\boxed{2}$ and one $\boxed{4}$. Thus, A_1 belongs in group G_1.

It is a requirement for our algorithm, that each cluster belong to exactly one group. However, the above definition of group assignment might create some ambiguity. In our example, A_2 contains two $\boxed{1}$s and two $\boxed{2}$s, which means it may belong in either G_1 or G_2. A_3 contains two $\boxed{2}$s and two $\boxed{3}$s, which means it may belong in either G_2 or G_3. A_4 contains two $\boxed{2}$s and two $\boxed{3}$s, which means it may belong in either G_2 on G_3. Our algorithm chooses the group for these clusters, so that the number of non-empty groups is maximized. We use g to denote the maximum number of non-empty groups. Figure 5 reflects such a group assignment. Cluster A_4 was assigned to group G_3 (assigning it to group G_2 would also work).

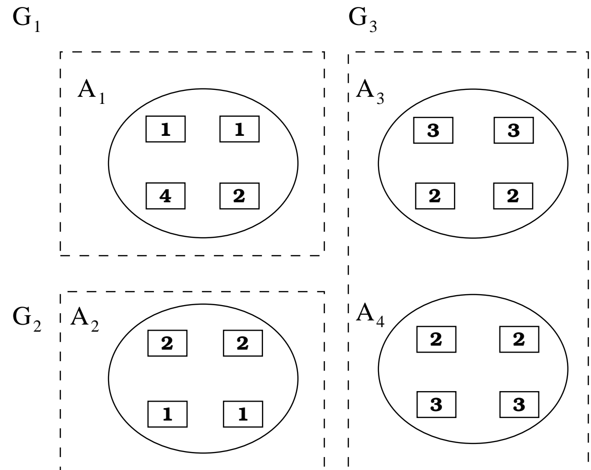

Figure 5. Group assignment for partition A.

We are now ready to perform *Move* and *Join* operations. As a preliminary step, we create and assign an empty cluster to each empty group. Next, we perform the following *Move* operations:

For each group G_k, we move all objects tagged with T_k that belong to clusters in other groups to any cluster in G_k. The result of this process to our example is shown in figure 6. G_1, G_2, and G_3 were non-empty, so we moved tags **1**, **2** and **3** to clusters in those groups. G_4 was an empty group, so we first created a new cluster A_5, then moved the lone tag **4** to it.

The total number of *Move* operations is as follows: Each cluster needs to move out all objects except those belonging to its own group. The moving cost for cluster A_i is $|A_i| - max_{j=1}^{m}(v_{ij})$. Thus the total cost is $n - \sum_{i=1}^{l} max_{j=1}^{m}(v_{ij})$. We use M to denote the total number of *Move* operations.

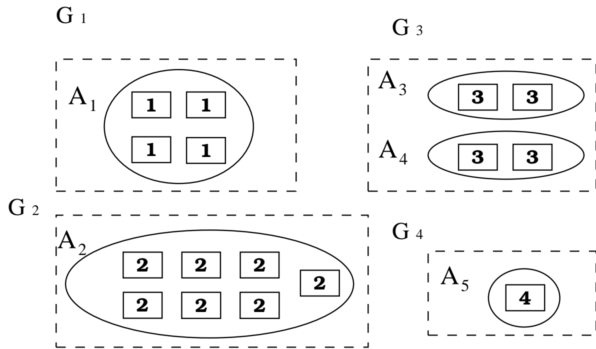

Figure 6. Partition A after all the *Move* operations.

After all the *Move* operations have been performed, any cluster in group G_k will contain only tags T_k. Then, we join all clusters belonging to the same group. After all the *Join* operations, every group is non-empty and contains exactly one cluster. Thus, our transformation is complete.

In our example, since only G_3 contains two or more clusters, we need only join A_3 and A_4. The resulting partition (shown in figure 7) is isomorphic to partition B.

The number of *Join* operations within group G_k is $|G_k| - 1$. Since *Join* operations happen only within original non-empty groups (i.e. non-empty before any *Move* and *Join* operations), the total cost of *Join* operations is $l - g$, where l is the number of clusters in partition A and g is the number of non-empty groups. Therefore, the total cost of our algorithm is $M + l - g$. This gives us some intuition on why we attempted to maximize g. The more non-empty groups, the smaller the cost.

This section presented a brief outline of the algorithm that calculates MoJo distance. Details on how to maximize

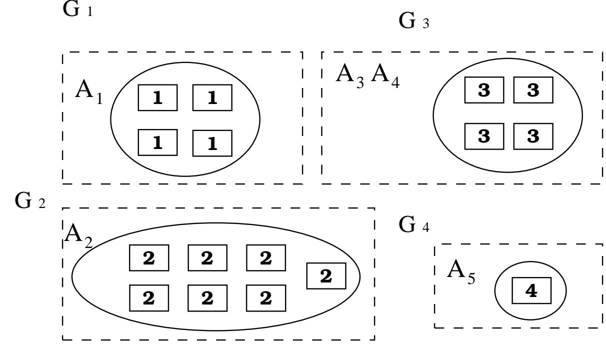

Figure 7. Partition A after all the *Join* operations.

g and the proof of correctness of this algorithm can be found in [15]. For this paper, it is sufficient to recall that MoJo distance is calculated as $M + l - g$ =

$$n - \sum_{i=1}^{l} max_{j=1}^{m}(v_{ij}) + l - g \quad (3)$$

4 Calculating MoJoFM

Having already described an algorithm that calculates $mno(A, B)$, we need only calculate the denominator in formula 2, i.e. $max(mno(\forall A, B))$. In order to do this, we will:

1. Prove that for any partition B, the most distant target partition from it is also the most distant source partition to it (section 4.1). This is expressed by the following formula:

$$max(mno(\forall A, B)) = max(mno(B, \forall A))$$

2. Show a method of finding the most distant target partition starting from partition B, i.e. calculate $max(mno(B, \forall A))$ (section 4.2).

4.1 The maximum distance to a specific target partition

We begin by showing that we need only worry about partitions for which $max_{j=1}^{m}(v_{ij}) = 1$.

Lemma 4.1. *For any source partition A that contains a cluster A_i for which $max(v_{ij}) > 1$, we can find another partition A' where $max(v_{ij}) = 1$ and $mno(A', B) \geq mno(A, B)$.*

Proof. For a given target partition B, we want to find a source partition A that has the maximum MoJo distance to B among all the partitions on the same set of objects. Since n is fixed in formula 3, our goal is to minimize $\sum_{i=1}^{l} max_{j=1}^{m}(v_{ij}) - l + g$.

If a cluster A_i has $max(v_{ij}) > 1$, we modify A_i in the following way. We extract one object from each kind of tag and then construct a new cluster. We do this iteratively until all $v_{ij} = 1$. For example, if A_1 contains $3T_1$, $2T_2$, $1T_3$, after the modification, A_1 will contain $1T_1$, a newly created A_2 will contain $1T_1$,$1T_2$ and another newly created A_3 will contain $1T_1$,$1T_2$ and $1T_3$ (figure 8). Let us analyze what is the effect to the MoJo distance.

Let us assume the original $max(v_{ij})$ of A_i is v. $\sum_{i=1}^{l} max_{j=1}^{m}(v_{ij})$ remains the same. l is increased by $v - 1$. We will prove that g will be increased by at most $v - 1$. This seems obvious at first because only $v - 1$ clusters were created, so g can be increased by at most $v - 1$. This is definitely true if A_i does not change its group after the modification. However it is possible that the original A_i also changes its group. If the $v - 1$ newly created clusters can increase g by $v - 1$ and A_i can also change g by 1 after changing its group, g will be increased by v in total. In the following, we will prove this is impossible.

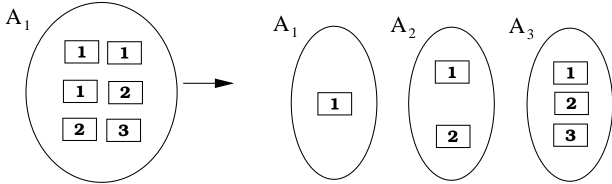

Figure 8. Decomposing A1 to A1, A2 and A3

We assume the original A_i belongs in group G_x, and it changes its group to G_y $(y \neq x)$ after the modification. We also know that A_i after the modification will contain only the original maximum tags. This means that in original A_i, $v_{ix} = v_{iy} = max(v_{ij})$ and $y \neq x$, i.e., G_y is also a possible group selection for original A_i. There are $v - 1$ newly created clusters. If any of these clusters belongs in group G_x or group G_y, g will be increased by at most $v-1$. Let us assume none of these clusters belongs in group G_x or G_y. If g can be increased by 1 after A_i moves from group G_x to G_y, it must hold that G_x had a cardinality of 2 or more and G_y was empty originally. This is not possible, because A_i could have chosen either G_x or G_y. A_i should have been assigned to G_y to maximize the non-empty groups in this case.

Thus, we have proven that g will be increased by at most $v - 1$ in all cases.

□

Lemma 4.2. *For any partition B, if partition A has any cluster A_i for which $max(v_{ij}) > 1$, we can find another partition B' for which $max(v_{ij}) = 1$ and $mno(A, B') \geq mno(A, B)$.*

Proof. For a given source partition A, we want to find a target partition B which satisfies that the MoJo distance from A to B is greater or equal than the MoJo distance from A to any other partition of the current software system. This time, both n and l are fixed in formula 3. As a result, if we want to maximize the MoJo distance, we need to minimize $\sum_{i=1}^{l} max_{j=1}^{m}(v_{ij}) + g$. We use S to denote this value.

Figure 9 is an example of how we decrease $max(v_{ij})$ for A_i to 1. For any tag T_j where $v_{ij} > 1$, we assign all objects with tag T_j to a new and unique cluster in partition B. Because partition B is not fixed, we can always change an object's tag as we wish. This shows that it is possible to construct B' so that all $v_{ij} = 1$.

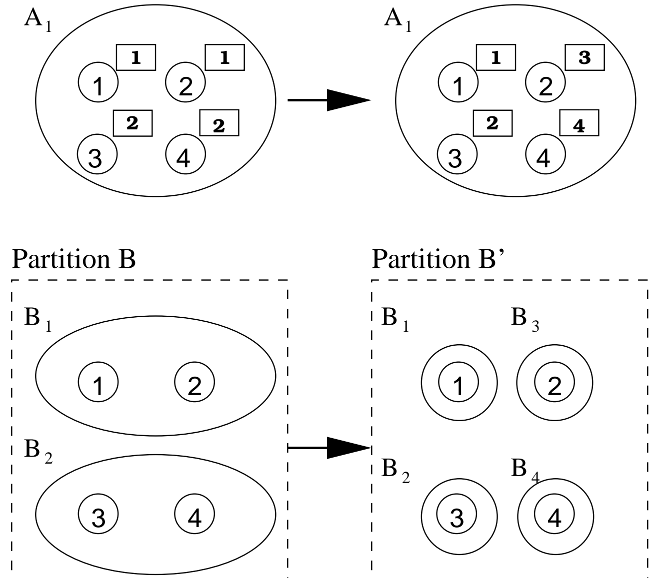

Figure 9. Modifying tags to ensure that all v_{ij}=1.

This way $max(v_{ij})$ for A_i changes to 1. If A_i changes its group, then g will be increased at most by 1. At the same time, $max(v_{ij})$ will be decreased at least by 1. Thus S will not be increased. □

Lemma 4.3. *If every v_{ij} is 1 for partition A and B, then mno(A,B)= mno(B,A).*

Proof. We first assume $mno(A, B) = x$ and $mno(B, A) = y$. If every v_{ij} is equal to 1, then according to the algorithm presented in section 3 (*Move* first, then *Join* within each group), *Join* operations can be replaced by *Move* operations.

This is because after all the *Move* operations, there will be at most 1 cluster containing more than one objects in each group. In order to join two clusters where at least one of them contains only one object, one can use either a *Join* or a *Move* operation. So all the *Join* operations can be replaced by *Move* operations.

We also know that *Move* operations are reversible. If partition A can be transformed to partition B using x *Move* operations, B can also be changed back to A using the same number of *Move* operations, which means $mno(B, A) = y \leq x = mno(A, B)$.

Moreover, it holds that $v_{ij} = v_{ji}$. Therefore, if every v_{ij} is 1, every v_{ji} is also 1, which means that partition B can also be transformed to A using $mno(B, A)$ *Move* operations. If partition B can be transformed to A using y *Move* operations, A can also be transformed to B using y *Move* operations, i.e. $mno(A, B) = x \leq y = mno(B, A)$. As a result, $x = y = mno(A, B) = mno(B, A)$. □

Theorem 4.4. *For any partition B, the maximum MoJo distance to it is also the maximum MoJo distance from it.*

Proof. We assume there exists a partition A so that the MoJo distance from B to A is the maximum MoJo distance from B to any other partition. We also assume that partition A' has the maximum MoJo distance to partition B among all partitions (Figure 10). Based on lemma 4.1 and 4.2, we can assume that according to partition B every v_{ij} of A and A' is equal to 1. Then we know that $mno(A, B) = mno(B, A)$ and $mno(A', B) = mno(B, A')$. Because $mno(B, A) \geq mno(B, A')$ and $mno(A', B) \geq mno(A, B)$, we conclude that $mno(B, A) = mno(A', B)$.

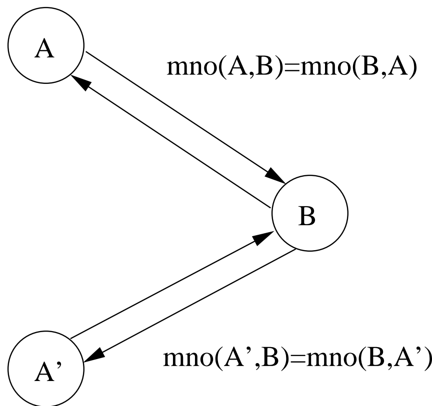

Figure 10. MoJo distances between A, B and A'.

□

4.2 The maximum distance from a specific source partition

Based on lemma 4.2, we know that if a partition B is the most distant partition from A, either every $v_{ij} = 1$, or we can find another B' that is as distant from A and satisfies that all v_{ij} are equal to 1. Then we need only consider those partitions B for whom each v_{ij} in partition A is equal to 1. In this case, the formula of MoJo distance becomes $n - l + l - g = n - g$.

Next, we describe the method of minimizing g when each v_{ij} in partition A is equal to 1. Because each $v_{ij} = 1 = max(v_{ij})$, each cluster A_i has $|A_i|$ groups for selection. To minimize the total maximum group selection, i.e. g, we want to make the group selection of each cluster as overlapping with each other as possible.

We construct B as follows: We first find the $max(|A_i|)$, then construct $max(|A_i|)$ clusters, from B_1 to $B_{max(|A_i|)}$. For each cluster A_i in partition A, we let it contain only tags T_1, T_2, ...,$T_{|A_i|}$. For example, in figure 11, we have $|A_1| = 1$, $|A_2| = |A_3| = 4$ and $|A_4| = 7 = max(|A_i|)$. As a result, we construct clusters B_1 to B_7, let A_1 contain only $1T_1$, let both A_2 and A_3 contain T_1 to T_4, and A_4 contain all kinds of tags, from T_1 to T_7. Without loss of generality, we assume that the clusters of A are ordered in ascending order of $|A_i|$. For example, in figure 11, we have $|A_1| \leq |A_2| \leq |A_3| \leq |A_4|$. In that case, we know that if $x < y$, then $|A_x| < |A_y|$ and $tags|A_x| \subseteq tags|A_y|$, where we use $tags|A_i|$ to denote the set of all tags in cluster A_i. As we see in figure 11, $tags|A_1| \subseteq tags|A_2| \subseteq tags|A_3| \subseteq tags|A_4|$.

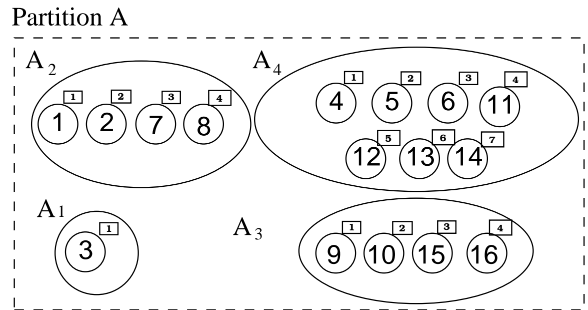

Figure 11. Tag assignment consistent with the most distant partition B.

Now we will prove why B is the most distant partition from A. We assume that partition A has clusters from A_1 to A_l. Then, we have $|A_1| \leq |A_2| \leq \cdots \leq |A_l|$. We also have $tags|A_1| \subseteq tags|A_2| \subseteq \cdots \subseteq tags|A_l|$.

Since A_l contains all possible tags, we have $g \leq |A_l|$. Because we have $tags|A_1| \subseteq tags|A_2| \subseteq \cdots \subseteq tags|A_{l-1}|$, A_{l-1} has all possible group selections for all

clusters except A_l. If A_l's group contains only one cluster A_l, then we have $|A_{l-1}| \geq g-1$. If A_l's group contains more than one clusters, then we have $|A_{l-1}| \geq g$. Therefore, in general we have $g-1 \leq |A_{l-1}|$. We can easily show by induction that for any $x \leq g$, we have $g-x \leq |A_{l-x}|$. For example, in figure 12, we have $g=4$, and
$|A_4| = 7 \geq g = 4$,
$|A_3| = 4 \geq g-1 = 3$,
$|A_2| = 4 \geq g-2 = 2$,
$|A_1| = 1 \geq g-3 = 1$.

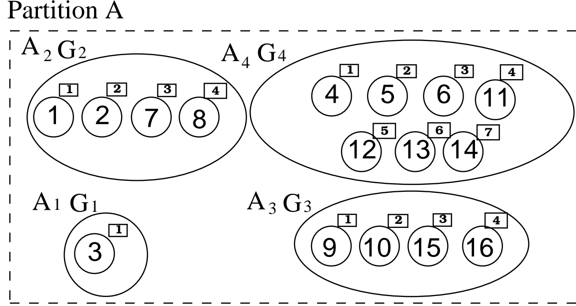

Figure 12. Group assignment for partition A

Let us assume there is another partition B' such that $\forall i, k$ $v_{ik} = 1$ in partition A and maximum possible groups in A, $g' < g$. Because $|A_l| \geq g > g'$, A_l has more than g' groups to choose from. This means that A_l could have chosen an empty group g_x. Therefore, A_l's group must contain only A_l, since otherwise we can increase the number of non-empty groups by assigning A_l to g_x instead.

As a result, clusters A_1 to A_{l-1} are assigned to $g'-1$ groups. Since $|A_{l-1}| \geq g-1 > g'-1$, we can similarly prove that A_{l-1} is also the only cluster in its group. This means that clusters A_l to $A_{l-g'}$ will also have to be in groups of cardinality 1. But then we will have at least $g'+1$ groups, which is a contradiction to our first assumption.

Having proven the correctness of our algorithm, we can also simplify it. We assume $|A_1| \leq |A_2| \cdots \leq |A_l|$, and use G_i to denote a mapping from clusters $A_1, \ldots, A_i$ to their groups, i.e., $\{A_1, \ldots, A_i\} \mapsto \{G_1, \ldots, G_m\}$. $Re(G_i)$ is defined as the range of G_i and $r_i = |Re(G_i)|$ is the cardinality of the range of G_i. We also use $g_i = max_{G_i}|Re(G_i)|$ to denote the maximum cardinality of the range of G_i, i.e., the maximum possible groups that clusters $A_1, \ldots, A_i$ can cover. Therefore, the maximum possible groups g is equal to g_l.

When $i = 1$, we have $g_1 = 1$. For any i such that $1 < i \leq l$, we know that A_i contains all possible group selections for clusters A_1 to A_i. Thus, we have $A_i \geq g_i \geq g_{i-1} \geq \ldots \geq g_1$.

If $r_{i-1} < |A_i|$, then A_i can select a new group. We can have $r_i = r_{i-1} + 1$. If $r_{i-1} = |A_i|$, then A_i can not belong in a new group. We will have $r_i = r_{i-1}$. If we have $g_{i-1} < |A_i|$, we can have $g_i = g_{i-1} + 1$. If we have $|A_i| = g_{i-1}$, there are two possible cases:

1. The group selection for clusters from A_1 to A_{i-1} is not maximized, i.e., $r_{i-1} < g_{i-1}$. In this case, we can have $r_i = r_{i-1} + 1 \leq g_{i-1}$.

2. The group selection is already maximized. Then, we can only have $r_i = g_{i-1} = g_i$.

Because in the second case we can get the maximum value of r_i, we should always maximize the value of r_{i-1}. This is true for all clusters from A_1 to A_l. Therefore, we can use this method to get the total maximum possible groups. We can write the algorithm in pseudo code as follows:

```
Sort A[i] in ascending order of |A[i]|
G := 0
for  i:=1 to l do begin
    if |A[i]| > G then G := G + 1
    assign A[i] to group G
end return G
```

5 Experiments

We have implemented the algorithm that calculates the value of MoJoFM. It is available at `http://www.cs.yorku.ca/~bil/downloads`. The implementation has been used to conduct several experiments. In this section we present some of these experiments, including the comparison of different software clustering techniques and the comparison between MoJoFM and Q.

5.1 Comparison of different clustering techniques

In order to assess the effectiveness of various software clustering techniques, we applied them to two large software systems of known authoritative decomposition, and computed their MoJoFM values.

The two large software systems we used for our experiments were of comparable size, but of different development philosophy:

1. **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.

2. **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. The authoritative decomposition of Linux was presented in [4].

The software clustering approaches we evaluated were the following:

1. **LIMBO**. LIMBO is a scalable hierarchical clustering algorithm based on minimizing information loss when clustering a software system. It combines structural and non-structural information in an integrated fashion. In the interest of fairness, only structural information was used for our experiments, since several of the algorithms cannot handle non-structural information [1].

2. **ACDC**. This is a pattern-based software clustering algorithm that attempts to recover subsystems commonly found in manually-created decompositions of large software systems [13].

3. **Bunch**. This is a suite of algorithms that attempt to find a decomposition that optimizes a quality measure based on high-cohesion, low-coupling. We experimented with two versions of a hill-climbing algorithm, we will refer to as NAHC and SAHC (for nearest- and shortest-ascend hill-climbing) [8].

4. **Cluster Analysis Algorithms**. We also used several hierarchical agglomerative cluster analysis algorithms for our experiments. We used the Jaccard co-efficient that has been shown to work best in a software clustering context [3]. We experimented with four different algorithms: single linkage (SL), complete linkage (CL), weighted average linkage (WA), and unweighted average linkage (UA).

The MoJoFM values for all these algorithms on both example software systems are shown in table 1.

These values denote that the above software clustering tools can be of help to reverse engineering projects since they appear to be able to correctly cluster a large portion of the software system at hand.

At the same time, the MoJoFM values can provide valuable intuition into which algorithm is better suited for a particular software system. For example, LIMBO and NAHC appear to perform best on Linux, while LIMBO and ACDC do better with TOBEY.

Of course, the MoJoFM value is not the only factor one needs to consider when choosing a clustering algorithm. For instance, when the difference is as small as 1.27%, as is the case when LIMBO and NAHC are applied on Linux, other features of these algorithms, such as the way they name clusters, will probably be the deciding factors. However, MoJoFM can help identify algorithms that are clearly ill-suited for a particular software system.

	LINUX	TOBEY
LIMBO	75.00%	65.82%
ACDC	63.19%	64.84%
NAHC	73.73%	58.02%
SAHC	62.76%	47.03%
SL	54.64%	24.40%
CL	61.92%	52.75%
UA	62.34%	49.01%
WA	37.03%	55.38%

Table 1. MoJoFM values for several algorithms on both example input systems.

5.2 Comparison of MoJoFM and Q

As mentioned before, Q used to over-evaluate the quality of the software clustering algorithms examined. We verified this experimentally and also observed how big is the difference between MoJoFM and Q.

We first compared the values of the two measures on TOBEY. Table 2 shows the values obtained for all the clustering algorithms presented above. The computed value was indeed over-evaluated by up to 2.25%.

However, all of these values were based on the same "gold standard" decomposition (the one for TOBEY). It is more interesting to investigate how is the MoJoFM value affected as the "gold standard" changes. If the over-evaluation is uniform, then Q could still be used for comparison purposes. If the amount of over-evaluation is significantly different, a case can be made for replacing Q with MoJoFM.

For this purpose, we developed a random partition generator that created both "candidate" (simulated automatically

	Q	MoJoFM
LIMBO	66.84%	65.82%
ACDC	65.88%	64.84%
NAHC	59.28%	58.02%
SAHC	48.61%	47.03%
SL	26.65%	24.40%
CL	54.16%	52.75%
UA	50.53%	49.01%
WA	56.72%	55.38%

Table 2. Comparison of the values of MoJoFM and Q for several algorithms, using TOBEY as input.

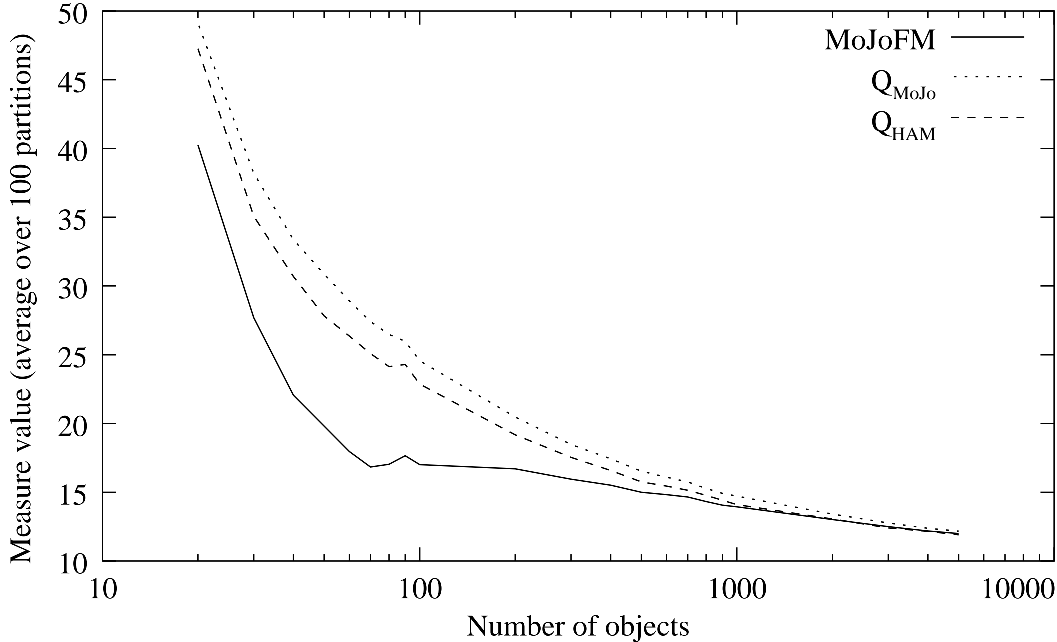

Figure 13. Comparison of the values of the three different measures as the number of objects increases.

created decompositions) and "reference" (simulated "gold standards") partitions.

For the first set of experiments, we fixed the number of clusters in both candidate and reference partitions to 10. The number of objects (n) to be clustered ranged from 20 to 5000. For each value of n, we created 100 random partitions and calculated the average value of both measures. In fact, in order to make the comparison fairer for Q, we calculated its value using both algorithms for the calculation of MoJo distance. We denote these two versions as Q_{HAM} and Q_{MoJo}.

Figure 13 presents our results. A first observation is that the amount of over-evaluation is not uniform. In some cases it is close to 0, while in others the value of Q is more than double the value of MoJoFM. This indicates that using Q may provide erroneous results. Furthermore, it is clear that using a better algorithm for MoJo distance calculation would not have improved the accuracy of Q.

To ensure that the aforementioned results were not influenced by the experiment setup used, we conducted a second set of experiments. This time we fixed the number of objects to 100, while allowing the number of clusters to range from 2 to 50. The results obtained are shown in figure 14.

We can again make the same observations. The amount of overevaluation is not uniform, while the accuracy of Q is not improved by the use of the optimal algorithm for MoJo distance calculation.

Finally, it is quite interesting to note that the value of all measures was very low when the reference partition contained a large number of clusters. This is a desirable property that agrees with our intuition that an algorithm, that produces course clusters when detailed ones are expected, is not very effective.

6 Conclusion

This paper introduced an effectiveness measure for software clustering algorithms called MoJoFM. We described an algorithm that calculates its value, and indicated the features that make it an improvement over existing measures. Finally, we presented experiments that demonstrate this improved performance, and showed how this measure can be used to assess the effectiveness of software clustering algorithms.

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. File clustering using naming conventions for legacy systems. In *Proceedings of CASCON 1997*, pages 184–195, Nov. 1997.

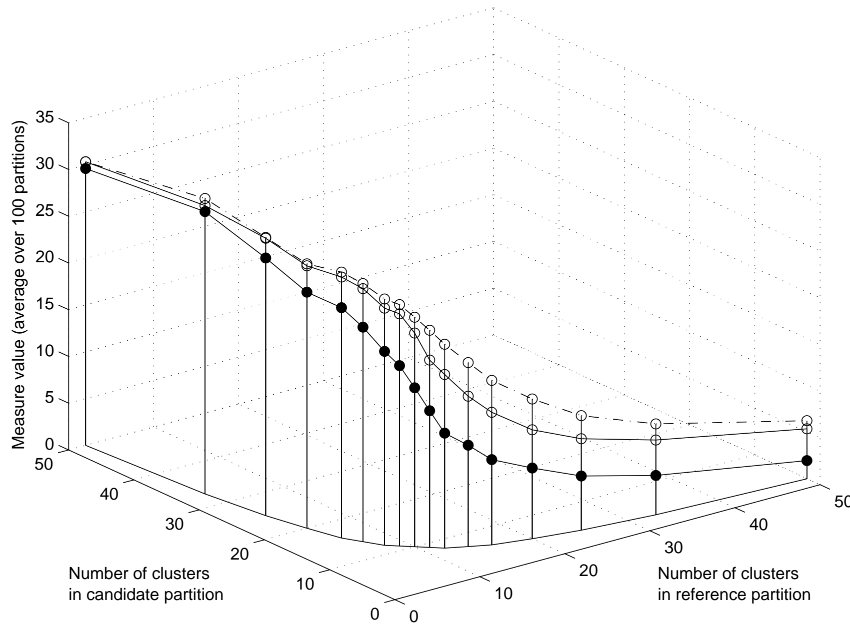

Figure 14. Comparison of the values of the three different measures as the number of objects remains fixed, and the number of clusters ranges from 2 to 50. Legend: The solid line with solid circles is MoJoFM. The solid line with unfilled circles is Q_{HAM}. The dashed line with unfilled circles is Q_{MoJo}.

[3] 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.

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] R. W. Schwanke. An intelligent tool for re-engineering software modularity. In *Proceedings of the 13th International Conference on Software Engineering*, pages 83–92, May 1991.

[12] 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.

[13] 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.

[14] A. van Deursen and T. Kuipers. Identifuing objects using cluster and concept analysis. In *Proceedings of the 21th International Conference on Software Engineering*, pages 246–255, May 1999.

[15] Z. Wen and V. Tzerpos. An optimal algorithm for MoJo distance. In *Proceedings of the 11th International Workshop on Program Comprehension*, pages 227–235, May 2003.