STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIV**, Number 1, 2009

A COMPARATIVE ANALYSIS OF CLUSTERING ALGORITHMS IN ASPECT MINING

GRIGORETA SOFIA COJOCAR, GABRIELA CZIBULA AND ISTVAN GERGELY CZIBULA

Abstract. *Aspect mining* is a research direction that tries to identify crosscutting concerns in already developed software systems, without using aspect oriented programming. The goal is to identify them and then to refactor them to aspects, to achieve a system that can be easily understood, maintained and modified. In this paper we aim at comparatively analyzing four clustering algorithms in aspect mining. The comparison is made using a set of quality measures previously introduced in aspect mining literature.

1. Introduction

Separation of concerns [12] is a very important principle of software engineering that, in its most general form, refers to the ability to identify, encapsulate and manipulate those parts of software that are relevant to a particular concept, goal, or purpose.

Crosscutting concerns [8] are parts of a program which affect or crosscut other concerns. Usually these concerns cannot be cleanly decomposed from the rest of the system, and they are mixed with many core concerns from the system leading to code scattering and code tangling, and, also, to systems that are hard to explore and understand. Identifying crosscutting concerns automatically improves both the maintainability and the evolution of the software system. Crosscutting concerns are a relevant source of problems to program comprehension and software maintenance. Examples of crosscutting concerns are persistence, synchronization, exception handling, error management and logging.

The aspect oriented paradigm (AOP) is one of the approaches proposed, so far, for designing and implementing crosscutting concerns [8]. Aspect oriented

Received by the editors: April 15, 2009.

2000 *Mathematics Subject Classification.* 62H30, 68N99.

1998 *CR Categories and Descriptors.* I.5.3 [**Computing Methodologies**]:Pattern Recognition – *Clustering*; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *Restructuring, reverse engineering, and reengineering*.

Key words and phrases. Aspect mining, Clustering, Crosscutting concerns.

techniques allow crosscutting concerns to be implemented in a new kind of module called *aspect*, by introducing new language constructs like pointcuts and advices.

Aspect mining is a research direction that tries to identify crosscutting concerns in already developed software systems, without using AOP. The goal is to identify them and then to refactor them to aspects, to achieve a system that can be easily understood, maintained and modified. There exist many reasons for migrating a legacy system to an aspect oriented based system. An inadequate solution for crosscutting concerns implementation has a negative impact on the final system with consequences like duplicated code, *scattering* of concerns throughout the entire system and *tangling* of concern-specific code with that of other concerns. The *code scattering* symptom means that the code that implements a crosscutting concern is spread across the system and the *code tangling* symptom indicates that the code that implements some concern is mixed with code from other (crosscutting) concerns. These consequences lead to software systems that are hard to maintain and to evolve. When aspect oriented techniques are used, the crosscutting concerns are cleanly separated from the core concerns, the latters becoming oblivious of them.

Clustering [6] has been already applied in aspect mining, as clustering aims at identifying groups of similar objects, and crosscutting concerns in legacy systems can be viewed as such groups (of methods, statements, etc.). Several partitional and hierarchical clustering algorithms were developed in [1, 2, 13, 14] for crosscutting concerns identification.

In this paper we are focusing on comparing the results obtained by four clustering algorithms (kAM, HAM, $PACO$ and $HACO$) that were previously introduced in the aspect mining literature.

The paper is structured as follows. Section 2 presents the clustering based approach that we have used for the evaluation that we aim at performing. The clustering algorithms used in our evaluation for identifying crosscutting concerns are described in Section 3. Section 4 provides the comparative analysis of the considered clustering algorithms, and Section 6 gives some conclusions of the paper and further work.

2. Clustering Based Aspect Mining

In this section we present the problem of identifying *crosscutting concerns* as a clustering problem [11].

Let $S = \{es_1, es_2, \ldots, es_n\}$ be a software system, where $es_i, 1 \leq i \leq n$, is an *element* from the software system. An *element* can be a statement, a method, a class, a module, etc. We denote by n ($|S|$) the number of elements of the system.

In the following, we will consider a crosscutting concern as a set of elements $C \subset S$, $C = \{e_1, e_2, ..., e_{cn}\}$, elements that implement this concern. The number of elements in the crosscutting concern C is $cn = |C|$. Let $CCC = \{C_1, C_2, ..., C_q\}$ be the set of all crosscutting concerns that exist in the system S. The number of crosscutting concerns in the system S is $q = |CCC|$. We suppose that two different crosscutting concerns do not have elements in common, meaning that $C_i \cap C_j = \emptyset, \forall i, j, 1 \leq i, j \leq q, i \neq j$.

Definition 1. [11] ***Partition of a system S.***
*The set $\mathcal{K} = \{K_1, K_2, ..., K_p\}$ is called a **partition** of the system S iff:*
(1) $1 \leq p \leq n$;
(2) $K_i \subseteq S, K_i \neq \emptyset, \forall\, i, 1 \leq i \leq p$;
(3) $S = \bigcup\limits_{i=1}^{p} K_i$;
(4) $K_i \cap K_j = \emptyset,\ \forall\, i, j, 1 \leq i, j \leq p, i \neq j$.

We will refer to $\mathcal{K}$ as a *set of clusters* and to K_i as the i-th *cluster* of $\mathcal{K}$.

In fact, the problem of aspect mining can be viewed as the problem of finding a partition $\mathcal{K}$ of the system S.

A partition of a software system S can be obtained by different kinds of algorithms (i.e., a clustering algorithm).

Definition 2. [11] ***Optimal partition of a system S.***
*Being given a partition $\mathcal{K} = \{K_1, K_2, ..., K_p\}$ of the system S, $\mathcal{K}$ is called an **optimal partition** of system S with respect to the set $CCC = \{C_1, C_2, ..., C_q\}$ of all crosscutting concerns, iff:*
(1) $p \geq q$;
(2) $\forall C \in CCC,\ \exists K_C \in \mathcal{K}$ *such that* $C = K_C$.

Intuitively, $\mathcal{K}$ is an optimal partition of the system S if all the elements implementing a crosscutting concern C_i $(1 \leq i \leq q)$ are in the same cluster K_{j_i} $(1 \leq j_i \leq p)$ and they are the only elements in K_{j_i}.

2.1. **Identification of Crosscutting Concerns.** In order to discover the crosscutting concerns from the system, we analyze the source code of the software system to be mined. All classes, methods and relations between them are computed. Afterwards, a clustering algorithm is used to identify a partition of a software system S in which the methods belonging to a crosscutting concern should be grouped together. The final step is to manually analyzed the obtained results.

Let us consider that the software system to be mined consists of a set of classes $\mathcal{C} = \{c_1, c_2, \ldots, c_s\}$, each class containing one or more methods. In our clustering approach, the objects to be clustered are the methods from the software system S, i.e., $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$. Our focus is to group the methods such that the ones belonging to the same crosscutting concern to be placed in the same cluster.

3. Clustering Algorithms for Crosscutting Concerns Identification

In order to group the methods in clusters we have used four clustering algorithms especially defined for aspect mining: *kAM* introduced in [14], *HAM* introduced in [13], *PACO* introduced in [1], and *HACO* introduced in [2]. In the following we briefly describe the algorithms used for our evaluation.

3.1. **kAM.** This is based on *k-means* clustering techniques, but it tries to avoid the two main disadvantages of *k-means* algorithm: the dependence on the number of clusters given as input and the dependence on the initial choice of the centroids. In order to accomplish this, *kAM* uses a heuristic for choosing the optimal number p of clusters, and the initial centroids. This heuristic provides a good enough choice of the initial centroids and is particular to aspect mining. The main idea of *kAM*'s heuristic is the following:

(i) The initial number p of clusters is n (the number of methods from the system).
(ii) The method chosen as the first centroid is the most "distant" method from the set of all methods (i.e., the method that maximizes the sum of distances from all other methods).
(iii) For each remaining methods (that were not chosen as centroids), the minimum distance (*dmin*) from the method and the already chosen centroids is computed. The next centroid is chosen as the method m that maximizes *dmin* and this distance is greater than a positive given threshold (*distMin*). If such a method does not exist it means that m is very close to its nearest centroid nc and should not be chosen as a new centroid (m and nc should belong to the same cluster). In this case, the number p of clusters will be decreased.
(iv) The step (iii) will be repeatedly performed, until the number p of clusters and the number of centroids are equal.

3.2. **HAM.** This algorithm is based on the idea of hierarchical agglomerative clustering, but uses a heuristic for merging two clusters, heuristic that is particular for aspect mining. In this algorithm, the distance between two clusters K_i and K_j is considered to be the largest distance between the objects from the clusters, i.e., it uses the complete linkage metric [6].

The heuristic used in *HAM* is that, at a given step, the most two similar clusters (the pair of clusters that have the smallest distance between them) are merged only if the distance between them is less or equal to a given threshold, $distClusMin$.

Both, *kAM* and *HAM* algorithms, use a vector space model in order to compute the dissimilarity between two methods. The following vector space model is used: a method m is characterized by an $(s+1)$-dimensional vector $\{FIV, B_1, B_2, ...B_s\}$, where s is the number of classes from the software system S (called application classes), FIV is the value of the *fan-in* metric and B_i is the value of the attribute corresponding to the application class c_i $(1 \leq i \leq s)$, as follows:

$$B_i = \begin{cases} 1 & \text{if } m \text{ is called from at least one method belonging to} \\ & \text{ application class } c_i \\ 0 & \text{otherwise} \end{cases} .$$

For both *kAM* and *HAM* algorithms, the distance between two methods is computed using the *Euclidian* distance.

3.3. **PACO.** This algorithm is based on *k-medoids* or *PAM* (*Partitioning around medoids*) algorithm [7], that finds representative objects, called *medoids*, in clusters. The algorithm starts with p initial representative objects for the clusters (medoids), then iteratively recalculates the clusters (each object is assigned to the closest cluster - medoid), and their medoids until convergence is achieved. At a given step, a medoid of a cluster is replaced by a non-medoid if it improves the total distance of the resulting clustering [7]. *k-medoids* algorithms have the same disadvantages as *k-means* algorithm: the dependence on the number of clusters given as input and the dependence on the initial choice of the medoids.

In order to avoid these disadvantages, *PACO* algorithm uses a heuristic for choosing the number of medoids (clusters) and the initial medoids. The heuristic is similar to the one used for *kAM*.

After selecting the initial medoids, *PACO* behaves like the classical *k-medoids* algorithm.

In order to compute the dissimilarities between methods three distance metrics were used:

- *Scattering* distance that captures the *scattering* symptom of crosscutting concerns.

$$(1) \qquad d_S^P(m_i, m_j) = \begin{cases} 1 - \frac{|in(m_i) \cap in(m_j)|}{|in(m_i) \cup in(m_j)|} & if\ in(m_i) \cap in(m_j) \neq \emptyset \\ \infty & otherwise \end{cases}$$

where, for a given method $m \in \mathcal{M}$, $in(m)$ defines the set of methods and classes that invoke m, as expressed in Equation (2).

$$in(m) = \{m\} \cup \{m' \in \mathcal{M} \cup \mathcal{C} |\ m' \ \ invoke \ \ m\}. \tag{2}$$

- *Tangling* distance that captures the *tangling* symptom of crosscutting concerns:

$$d_T^P(m_i, m_j) = \begin{cases} 1 - \frac{|r(m_i) \cap r(m_j)|}{|r(m_i) \cup r(m_j)|} & if \ r(m_i) \cap r(m_j) \neq \emptyset \\ \infty & otherwise \end{cases} \tag{3}$$

  where, for a given method $m \in \mathcal{M}$, $in(m)$ is defined as in Equation (2), and $r(m)$ denotes the set of relevant properties for each invocation context $inv \in in(m)$.
- *Scattering-Tangling* distance that tries to capture both *scattering* and *tangling* symptoms:

$$d_{ST}^P(m_i, m_j) = min\{d_S(m_i, m_j), d_T(m_i, m_j).\} \tag{4}$$

3.4. **HACO.** This algorithm is based on the idea of hierarchical agglomerative clustering, and uses a heuristic for determining the number of clusters. In order to determine the number p of clusters, the focus is on determining p representative methods from the software system S. The method chosen as the first representative method is the most "distant" method from the set of all methods (the method that maximizes the sum of distances from all the other methods). At each step we select from the remaining methods the most distant method relative to the already chosen methods. If the selected method is close enough to the already chosen representative methods, then the process is stopped, otherwise the selected method is considered as a new representative method.

The distance metric used for computing the similarity between two methods is defined as follows:

$$d_{ST}^H(m_i, m_j) = \begin{cases} 0 & i = j \\ 1 - \frac{|Col(m_i) \cap Col(m_j)|}{|Col(m_i)| + |Col(m_j)|} & if \ Col(m_i) \cap Col(m_j) \neq \emptyset \\ \infty & otherwise \end{cases}, \tag{5}$$

where $Col(m)$ is a collection consisting of: the method itself, the class in which the method is defined, the classes and methods that invoke m and the classes in which the classes and methods that invoke m are contained. This distance function considers both the *scattering* and *tangling* symptoms.

4. Comparative Analysis

In this section we comparatively analyze the results obtained by the above presented clustering algorithms. The comparison is made considering how well did a clustering algorithm succeed in identifying clusters corresponding to the crosscutting concerns from the software system to be mined.

For this evaluation we have used two quality measures introduced in [11]:

- **DISP**. The dispersion degree of crosscutting concerns in clusters.
- **DIV**. The degree to which each cluster contains elements from different crosscutting concerns or elements from other concerns.

$DISP$ measure defines the dispersion degree of crosscutting concerns in clusters, considering, for each crosscutting concern, the number of clusters that contain elements belonging to the concern. DIV measure defines the degree to which each cluster contains elements from different crosscutting concerns or elements from other concerns.

For each measure the values are in the interval $[0, 1]$, the ideal value being 1 for both of them. Larger values for $DISP$ and DIV indicate better partitions with respect to set of the crosscutting concerns to be discovered, meaning that both measures have to be maximized. Theorem 3 that gives the neccesary and sufficient conditions for a partition to be an optimal partition was proven in [11]:

Theorem 3. *If $\mathcal{K}$ is a partition of the software system S and CCC is the set of crosscutting concerns in S, then $\mathcal{K}$ is an* ***optimal partition*** *iff* $DISP(CCC, \mathcal{K}) = 1$ *and* $DIV(CCC, \mathcal{K}) = 1$.

We will use these properties of the above presented quality measures in analyzing the results obtained by the different clustering algorithms used for this comparison.

4.1. **Case study.** We have considered a medium size software application, the open source JHotDraw version 5.4b1 case study [3]. It is a Java GUI framework for technical and structured graphics, developed by Erich Gamma and Thomas Eggenschwiler, as a design exercise for using design patterns. It consists of **396** classes and **3359** methods.

The set of crosscutting concerns used for the evaluation is: *Adapter*, *Command*, *Composite*, *Consistent behavior*, *Contract enforcement*, *Decorator*, *Exception handling*, *Observer*, *Persistence*, and *Undo*. The set of crosscutting concerns and their implementing methods was constructed using the results reported by Marin et al. and publicly available at [10].

We mention that the value of the threshold is 1 for all clustering algorithms.

Analyzing the results from Table 1 we can conclude the following:

Algorithm	**Distance function**	**DISP**	**DIV**
kAM	Euclidian	0.4005	0.9972
HAM	Euclidian	0.4005	0.9973
PACO	d_S^P	0.4444	0.9753
PACO	d_T^P	0.4433	0.8732
PACO	d_{ST}^P	0.4207	0.8798
HACO	d_{ST}^H	0.457	1

Table 1. Results for JHotDraw case study.

- From the analyzed clustering algorithms, the algorithm that provides the best results is $HACO$, as $DISP$ and DIV have the larger values. This means that in the partition obtained by $HACO$ algorithm, the methods from the crosscutting concerns were better grouped than in the partitions obtained by the other algorithms. $HACO$ also provides the maximum value for DIV measure, i.e. 1, meaning that for each crosscutting concern its elements are not mixed with elements from other (crosscutting) concerns.
- The elements of crosscutting concerns are spread in two or more clusters for all algorithms, as the values of the $DISP$ measure are less than 0.5 for all algorithms.
- For $PACO$ algorithm, the *scattering* distance has obtained the best results, for both $DISP$ and DIV measures.
- The vector space model approach has obtained better results for DIV measure than $PACO$, but it has obtained the worst results for $DISP$ measure.
- The hierarchical clustering approach seems to be more appropriate in aspect mining than partitional clustering.
- None of the clustering algorithms used has succeeded in obtaining an optimal partition of the software systems. The value of the $DISP$ measure is less than 1 for all algorithms, and the value of DIV measure is 1 only for $HACO$ algorithm.

As a conclusion, $DISP$ measure can be improved for $HACO$ algorithm by improving the distance semi-metric d_{ST}^H used for discriminating the methods from the software system in the clustering process.

5. Related Work

In this section we briefly present other existing clustering based approaches used for crosscutting concerns identification.

Shepherd and Pollock [15] use clustering to find methods with similar name as an indication of crosscuttingness. They perform agglomerative hierarchical clustering in order to group methods. The objects to be clustered are the names of methods from the software system under analysis. The distance function between two methods m_1 and m_2 is proportional with the common substring length. The authors have developed a tool, called **AMAV**, that helps users navigate and analyze the obtained clusters. The rest of the approach is just manual analysis of the obtained results using the tool.

He and Bai [4] have proposed an aspect mining technique based on dynamic analysis and clustering that also uses association rules. They first use clustering to obtain crosscutting concern candidates and then use association rules to determine the position of the source code belonging to a crosscutting concern in order to ease refactoring.

We did not provide a comparison of the clustering algorithms considered in this study with the two other existing clustering based aspect mining approaches for the following reasons:

- Shepherd and Pollock have proposed in [15] an aspect mining tool based on clustering, but it does not automatically identify the crosscutting concerns. The user of the tool has to manually analyze the obtained clusters in order to discover crosscutting concerns.
- The technique proposed by He and Bai cannot be reproduced, as they do not report neither the clustering algorithm used, nor the distance metric between the objects to be clustered. Also, the results obtained for the case study used by the authors for evaluation are not available. For these reasons, we cannot provide a comparison with this technique.

6. Conclusions and Further Work

In this paper we have provided a comparative analysis of four clustering algorithms that are used for crosscutting concerns identification: *kAM*, *HAM*, *PACO*, and *HACO*. For the evaluation we have used two quality measures that were previously introduced in the aspect mining literature.

In the future we plan to apply the clustering algorithms considered in this paper to other larger software systems. We will also consider to use other *unsupervised learning* tehniques (such as self-organizing maps [9], Hebbian learning [5]) in order to identify crosscutting concerns in existing software systems.

ACKNOWLEDGEMENT. This work was supported by research project ID_2268/2009, sponsored by the Romanian National University Research Council (CNCSIS).

References

[1] Gabriela Czibula, Grigoreta Sofia Cojocar, and Istvan Gergely Czibula. A Partitional Clustering Algorithm for Crosscutting Concerns Identification. In *Proceedings of the International Conference on Software Engineering, Parallel and Distributed Systems (SEPADS '09)*, pages 111–116, Cambridge, UK, February, 21-23 2009.

[2] Istvan Gergely Czibula, Gabriela Czibula, and Grigoreta Sofia Cojocar. Hierarchical Clustering for Identifying Crosscutting Concerns in Object Oriented Software Systems. In *Proceedings of the 4th Balkan Conference in Informatics (BCI'09)*, Thessaloniki, Greece, September, 17-19 2009, submitted.

[3] E. Gamma. JHotDraw Project. http://sourceforge.net/projects/jhotdraw.

[4] Lili He and Hongtao Bai. Aspect Mining using Clustering and Association Rule Method. *International Journal of Computer Science and Network Security*, 6(2):247–251, February 2006.

[5] John Hertz, Richard G. Palmer, and Anders S. Krogh. *Introduction to the Theory of Neural Computation.* Perseus Publishing, 1991.

[6] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, New Jersey, 1998.

[7] L. Kaufman and P. J. Rousseeuw. Clustering by means of Medoids. *Statistical Data Analysis Based on the L1-Norm and Related Methods*, pages 405–416, 1987.

[8] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *Proceedings European Conference on Object-Oriented Programming*, volume LNCS 1241, pages 220–242. Springer-Verlag, 1997.

[9] Teuvo Kohonen. *Self-Organizing Maps.* Springer-Verlag, Berlin, Heidelberg, New York, 2001.

[10] Marius Marin. Fan-in results.
http://swerl.tudelft.nl/bin/view/AMR/FanInAnalysisResults.

[11] Grigoreta Sofia Moldovan and Gabriela Serban. Clustering Based Aspect Mining Formalized. *WSEAS Transactions on Computers*, 6(2):199–206, 2007.

[12] David L. Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, December 1972.

[13] Gabriela Serban and Grigoreta Sofia Cojocar. A New Hierarchical Agglomerative Clustering Algorithm in Aspect Mining. In *Proceedings of 3rd Balkan Conference in Informatics (BCI'2007)*, pages 143–152, Sofia, Bulgaria, September, 27-29 2007.

[14] Gabriela Serban and Grigoreta Sofia Moldovan. A New k-means Based Clustering Algorithm in Aspect Mining. In *Proceedings of 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'06)*, pages 69–74, Timisoara, Romania, September, 26-29 2006. IEEE Computer Society.

[15] David Shepherd and Lori Pollock. Interfaces, Aspects, and Views. In *Proceedings of Linking Aspect Technology and Evolution Workshop(LATE 2005)*, March 2005.

Babeş-Bolyai University, Department of Computer Science, 1, M. Kogalniceanu Street, Cluj-Napoca, Romania

E-mail address: {grigo, gabis, istvanc}@cs.ubbcluj.ro