k-Attractors: A Clustering Algorithm for Software Measurement Data Analysis

Yiannis Kanellopoulos[1,2], Panagiotis Antonellis[1], Christos Tjortjis[2] and Christos Makris[1]
1. University of Patras, Department of Computer Engineering and Informatics, Greece
2. The University of Manchester, School of Computer Science, U.K.
Yiannis.Kanellopoulos@postgrad.manchester.ac.uk, {adonel, makri}@ceid.upatras.gr, christos.tjortjis@manchester.ac.uk

Abstract

Clustering is particularly useful in problems where there is little prior information about the data under analysis. This is usually the case when attempting to evaluate a software system's maintainability, as many dimensions must be taken into account in order to reach a conclusion. On the other hand partitional clustering algorithms suffer from being sensitive to noise and to the initial partitioning. In this paper we propose a novel partitional clustering algorithm, k-Attractors. It employs the maximal frequent itemset discovery and partitioning in order to define the number of desired clusters and the initial cluster attractors. Then it utilizes a similarity measure which is adapted to the way initial attractors are determined. We apply the k-Attractors algorithm to two custom industrial systems and we compare it with WEKA's implementation of K-Means. We present preliminary results that show our approach is better in terms of clustering accuracy and speed.

1. Introduction

Software systems have become very complex and beset with problems concerning their maintenance and evolution. They have also become larger, and it is difficult for a single person to understand a system in its entirety. According to N.I.S.T. [1], software errors cost the U.S. economy $60 billion per year. Essentially, a correct and consistent behavior of a software system is a fundamental part of users' expectations. As a result, maintenance processes can be considered an area of competitive advantage. There are several studies on evaluating a system's maintainability and controlling the effort required to carry out maintenance activities. According to ISO/IEC-9126 [2] maintainability is the capability of a software product to be modified. Evaluating maintainability is difficult because many contradictory criteria must be considered in order to reach a decision. Data mining and its capacity to deal with large volumes of data and to uncover hidden patterns has been proposed as a means to support the evaluation and assessment of the maintainability of industrial scale software systems.

Furthermore, the use of metrics (measurement data) provides a systematic approach for maintainability evaluation. It also enables the engineers of a system to track status, identify potentially problematic areas, and make decisions related to their tasks.

We propose in this paper a novel partitional clustering algorithm, *k-Attractors*, which is tailored for the analysis of this type of data. The main characteristics of k-Attractors are:

- It defines the desired number of clusters (i.e. the number of k), without user intervention.
- It locates the initial attractors of cluster centers with great precision.
- It measures similarity based on a composite metric that combines the Hamming distance and the inner product of transactions and clusters' attractors.
- It can be used for large data sets.

The remaining of the paper is organized as follows: section 2 discusses the background on clustering an related problems, section 3 proposes our approach, section 4 details k-Attractors, section 5 presents experimental results and section 6 concludes with directions for future work.

2. Background

This section presents the main clustering problems that the proposed algorithm addresses and the main concepts we used for its development.

2.1. Clustering

Clustering is particularly useful in problems where there is little prior information available about the data, and the decision maker (a maintenance engineer in our case) must make as few assumptions about the data as possible. These restrictions make this methodology appropriate for the exploration of interrelationships among the data points to make an assessment concerning their structure [3].

Cluster creation can be performed in a number of ways. Hierarchical algorithms organize data into a hierarchical structure based on a proximity matrix. On the other hand, partitional algorithms identify the partition that optimizes, usually locally, a clustering criterion.

K-Means is one of the most popular partitional clustering algorithms. Each cluster is represented by the mean value of the objects in the cluster. As a result, cluster similarity is measured based on the distance between the object and the mean value of the input data in a cluster. It is an iterative algorithm in which objects are moved among clusters until a desired set is reached. Its main problems are that it is sensitive to noise and to the initial partitioning. As many possible initial partitions lead to many different results, the final clustering is influenced by the initial partition, which is indicated by the user input [4].

2.2. Frequent Itemsets

A frequent itemset is a set of items that appear together in more than a minimum fraction of the whole dataset. More specifically let J be a set of quantitative data. Any subset I of J is called an itemset. Let ($T = <t1,...,tn>$) be a sequence of itemsets called a transaction database. Its elements $t \in T$ will be called itemsets or transactions.

An itemset can be frequent if its support is greater than a minimum support threshold, denoted as *min_sup*. The support of an item X in T denoted as *min_sup T(X)* is the number of transactions in T that contain X. The term frequent item refers to an item that belongs to a frequent itemset. If now, an item X is frequent and no superset of X is frequent, then X is a maximally frequent itemset; and we denote the set of all maximally frequent itemsets by *MFI*. From these definitions it is easy to see that the following relationship holds $MFI \leq FI$. Association rule mining algorithms such as *Apriori* are used in order to discover frequent itemsets [5].

2.3. Related Work

Zhuang and Dai [6] present a new approach for the clustering of web documents, which is called Maximal Frequent Itemset Approach. Based on maximal frequent itemset discovery they propose an efficient way to precisely locate the initial points for the K-Means algorithm. Fung et al. [7] propose the Frequent Itemset-based Hierarchical Clustering (FIHC) for document clustering. The intuition of this algorithm is that there are some frequent itemsets for each cluster (topic) in the document set, and different clusters share few frequent itemsets. A frequent itemset is a set of words that occur together in some minimum fraction of documents in a cluster. Therefore, a frequent itemset describes something common to many documents in a cluster. Hence, frequent itemsets are used in order to construct clusters and to organize them into a topic hierarchy. Wang et al. [8] propose a similarity for a cluster of transactions based on the notion of large items. An item is large in a cluster of transactions if it is contained in a user-specified fraction of transactions in that cluster. According to this measure, a good clustering is one that there are many large items within a cluster and there is little overlapping of such items across clusters. Karypis et al. [9] introduced the Association Rule Hypergraph Partitioning (ARHP). It constructs a weighted hypergraph to represent the relationships among discovered frequent itemsets. A hypergraph $H = (X, E)$ is an extension of a normal graph where $X = (x_1, x_2, ...,x_n)$ is a finite set and $E = (E_i | i \in I)$ is a family of subsets of X. The elements $x_1, x_2, ...,x_n$ are called vertices and the sets $E_1, E_2,..., E_m$ are called hyperedges. Frequent itemsets are represented by hyperedges and the hyperedge weights are defined as the support of the frequent itemsets. The aim of the proposed algorithm is to find k partitions such that the vertices in each partition are highly related. Finally Kosters et al. [10] propose a method that employs association rules having high confidence to construct a hierarchical sequence of clusters. A specific metric is introduced for measuring the quality of the resulting clusters. Assuming that there is a space S then for subsets R and T of S it is defined $d(R, T) = |R\T|+|T\R| \setminus |RUT| + 1$ where \ denotes the set-theoretic difference, $(X\Y)$ consists of those elements from X that are not in Y and $|X|$ denotes the number of elements of X. Hence, the numerator is the number of elements in the symmetric difference of R and T.

A large amount of data is produced in software development and software organizations collect them in hope of extracting useful information. Clustering provides the capability to analyze and extract novel, interesting patterns from this type of data. Then the extracted information can be used to evaluate a software system and/or predict its behavior. A methodology that uses K-Means clustering for recovering the structure of a software artifact and assessing its maintainability is presented in [13]. It does that by creating an input model which considers as program's entities' attributes both metrics and elements from source code data (e.g. class name, method name, superclass etc.). On the other hand, [14] presents a methodology for knowledge acquisition from

k-Attractors Algorithm

/*Input Parameters*/
Support: s
Number of dimensions: n
Hamming distance power: h
Inner product power: i
Attractor similarity ratio: r
Outlier factor: d
Given a set of m data items $t_1, t_2, \ldots, t_m$

/*Initialization Phase*/
(1) Generate frequent itemsets using the APriori Algorithm;
(2) Construct the *itemset graph* and partition it using the confidence similarity criteria related to the support of these itemsets;
(3) Use the number of partitions as the final k;
(4) Select the maximal frequent itemset of every cluster in order to form a set of k initial attractors;

/*Main Phase*/
Repeat
(6) Assign each data item to the cluster that has the minimum $Score(C_i \rightarrow t_j)$;
(7) When all data items have been assigned, recalculate new attractors;
Until t_i don't_move
(8) Search all clusters to find outliers and group them in a new cluster

Figure 1 - k-Attractors overview

source code elements. A combination of clustering and association rules is applied on data extracted from object oriented code. K-Means clustering produces system overviews and deductions, which support further employment of an improved version of MMS Apriori to identify hidden relationships between classes, methods and member data.

3. Our approach

In this research work we propose a partitional algorithm that utilizes a preprocessing method for its initial partitioning and incorporates a similarity measure adapted on the rationale for this method.

More specifically, the k-Attractors algorithm employs the maximal frequent itemset discovery and partitioning, similarly to [6], [9] in order to define the number of desired clusters and the initial attractors of the centers of these clusters. What differentiates it at first, from the work presented above, is that it is used in a different (than document clustering) context. The intuition is that a frequent itemset in the case of software metrics is a set of measurements that occur together in a minimum part of a software system's classes. Classes with similar measurements are expected to be on the same cluster. The term attractor is used instead of centroid, as it is not determined randomly, but by its frequency in the whole population of a software system's classes.

The main contribution of k-Attractors is that it proposes a similarity measure which is adapted to the way initial attractors are determined by the preprocessing method. Hence, it is primarily based on the comparison of frequent itemsets. More specifically, a composite metric based on the Hamming distance and the dot (inner) product between each transaction and the attractors of each cluster is utilized.

Hamming distance is given by the number of positions that a pair of strings is different. Put another way, it measures the number of substitutions required to change one string to another. In our case a string is a set of data items and more specifically a vector containing software measurement data. Furthermore, the dot product of two vectors is a measure of the angle and the orthogonality of two vectors. It is used in order to compensate for the position of both vectors in Euclidean space.

4. Algorithm Description

This section details the basic steps of the k-Attractors algorithm along with some representative examples.

4.1. Overview

The two basic steps of the k-Attractors algorithm are:

- Initialization phase:
  - The first step of this phase is to generate frequent itemsets using the APriori algorithm. The derived frequent itemsets are used to construct the *itemset graph*, and a graph partitioning algorithm is used to find the number of the desired clusters and assign each frequent itemset into the appropriate cluster.
  - As soon as the number of the desired clusters (k) is determined, we select the maximal frequent itemsets of every cluster, forming a set of k frequent itemsets as the initial attractors.
- Main Phase:
  - As soon as the attractors have been found, we assign each transaction to the cluster that has the minimum $Score(C_i \to t_j)$ against its attractor.
  - When all transactions have been assigned to clusters we recalculate the attractors for each cluster in the same way as during the initialization phase.

Figure 1 illustrates an overview of the k-Attractors algorithm. We detail its two phases next.

4.2. Initialization Phase

The goal of the initialization phase is twofold: firstly to identify the most frequent itemsets of the input data and secondly to determine the number of clusters.

In order for the most frequent itemsets to be discovered, we apply the APriori algorithm against the input data file. The APriori algorithm takes as parameter the absolute support s of the required itemsets and returns all the one-dimensional and multi-dimensional itemsets with support greater than or equal to s.

Once the most frequent itemsets have been discovered, we form the *itemset graph*. Given the set of the most frequent itemsets $FI = \{fi_1, fi_2, \ldots fi_m\}$, the itemset graph is a graph $G(V,E)$, where $V = \{fi_i \in FI\}$ and $E = \{e_{ij} \,|\, fi_i \cap fi_j \neq \varnothing\}$. The intuition behind this graph is that if two itemsets have at least one common item, then they should possibly belong to the same cluster and thus we connect them with an edge in the itemset graph. For more accuracy, we could have weighted each edge with the number of common items between the two corresponding itemsets/vertices, but in order to keep the initialization phase as simple and fast as possible we decided not to weigh the edges.

Figure 2 demonstrates an example of the itemset graph's construction.

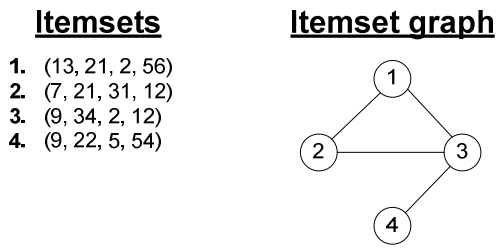

Figure 2 - Itemset graph example

The next step is to apply a graph partitioning algorithm to the itemset graph. In our case, we utilized the *kMetis* algorithm in order to partition the itemset graph [9]. The kMetis algorithm partitions the itemset graph into a number of distinct partitions and assigns each vertex of the graph (i.e. each itemset) into a single partition. The final number of the derived partitions is the number of clusters that we will use in the main phase of the k-Attractors algorithm.

The final step of the initialization phase is the attractors' discovery. During this step, every previously determined graph partition is examined. For each partition we find the maximal frequent itemset belonging to this partition, and check its dimensions' cardinality. If the number of dimensions is equal with the input data items' number of dimensions, *n*, then we assign the corresponding itemset as the attractor of the corresponding partition. However, in most cases, the cardinality of the maximal itemset is less than *n*. In such a case, we search for the next maximal frequent itemset in the corresponding partition and merge it with the previous itemset. Merging occurs only in dimensions that are absent from the first maximal itemset. We repeat this procedure until we have formed an itemset with cardinality equal to *d*, and assign the formed itemset as attractor of the corresponding partition.

In order to provide more flexibility, k-Attractors performs an extra post-processing step against the previously determined attractors. The algorithm takes as parameter an *attractor similarity ratio*, *r*. This parameter defines the maximum allowed similarity between two different attractors. The similarity between two attractors a_1, a_2 is defined as follows:

$$sim(a_1, a_2) = \frac{\#(a_1 \cap a_2)}{n} \quad (1)$$

In other words, the similarity between two attractors is the ratio of the number of common items per the number of total dimensions.

If the similarity of two attractors is more than *r,* then we randomly discard one of the two attractors; thus the number of total partitions is decreased by one.

4.3. Main Phase

The goal of k-Attractor's main phase is to assign each input data item to a cluster, using a partitional approach.

At first, we form a set of *k* empty clusters, where *k* is the number of distinct graph partitions discovered during the initialization phase. Every formed cluster is then assigned to the corresponding attractor calculated previously.

Once the clusters have been formed, the main phase of k-Attractors begins. This phase resembles a partitional algorithm, where every data item is assigned to a cluster according to a predefined distance metric and in every step the centers of every cluster are re-calculated until the algorithm converges to a stable state where the clusters' centers do not change.

The k-Attractors algorithm utilizes a hybrid similarity metric based on vector representation of both the data items and the cluster's attractors. The similarity of these vectors is measured employing the following composite metric:

$$Score\left(C_i \leftarrow t_j\right) = h \cdot H\left(a_i, t_j\right) + i \cdot \left(a_1 t_1 + \ldots a_n t_n\right) \quad (2)$$

In this formula, the first term is the Hamming distance between the attractor a_i and the data item t_j. It is given by the number of positions that pair of strings is different and is defined as follows:

$$H\left(a_i, t_j\right) = n - \#\left(a_i \cap t_j\right) \quad (3)$$

As our algorithm is primarily based on itemsets' similarity, we want to measure the number of substitutions required to change one into the other. The second term is the dot (inner) product between this data item and the attractor a_i. It is used in order to compensate for the position of both vectors in the Euclidean space. Because of the semantics of software measurement data, the usually utilized internal metrics (such as lines of code, coupling between objects, number of comments etc) have large positive integer values. Thus in order for the inner product distance to be more accurate, we firstly normalize all the values in the interval [-1, 1] and then apply the k-Attractors algorithm.

The multipliers h, i in equation 2 define the metric's sensitivity to Hamming distance and inner product respectively. For example, the $i = 0$ case indicates the composite metric is insensitive to the inner product between the data item and the cluster's centroid. Both h and i are taken as input parameters in our algorithm during its execution. Thus, k-Attractors provides the flexibility of changing the sensitivity of the composite distance metric to both Hamming distance and inner product, in correspondence with the each clustering scenario's semantics.

Utilizing this similarity metric, k-Attractors assigns every data item to a single cluster and recalculates the attractors of every cluster. Recalculation includes finding that data item of a cluster that minimizes the total sum of distances between the other data items belonging to that cluster. The above procedure is repeated until the attractors of all clusters do not change any further.

The final step of the k-Attractors' main phase is the outlier handling step. During this step, every cluster is checked for outliers, according to a parameter *d*. Specifically, a data item t, belonging to cluster *i*, is considered as an outlier if:

$$Score\left(a_i, t\right) >= d \cdot avg_i\left(Score\left(a_i, t_j\right)\right) \quad (4)$$

where $avg_i\left(Score\left(a_i, t_j\right)\right)$ is the average distance between the data items of cluster *i* and the attractor a_i.

Every discovered outlier is grouped into a new cluster, called *outliers cluster*, thus the total number of formed clusters is *k+1*.

5. Experimental Results

5.1. Datasets used

The evaluation of the proposed clustering algorithm involved the clustering of two sets of industrial software measurement data. For our experiments we have implemented k-Attractors in Java 1.4 and compared it with the k-Means implementation of *Weka 3*, the Java open source data mining tool. All the experiments were performed in a Pentium M 2.0 GHz machine with 1GByte RAM. For this work we combined two sets of metrics proposed by [11] and [12]. The derived set can be applied to OO programs and can be used as a predictor and evaluator of a system's maintenance effort. The following metrics were included and calculated for the systems' classes and were used as their clustering attributes:

- *Lines of Code (LOC),* which measures a class's number of lines of code including empty lines and comments.
- *Weighted Methods per Class (WMC)*, which is simply the sum of the complexities of its methods [11].
- *Coupling between Objects – Efferent Coupling (CBO)*, which represents the number of classes a given class, is coupled to [11].
- *Lack of Cohesion in Methods (LCOM),* which measures if a class has all its methods working together in order to achieve a single, well-defined purpose [11].
- *Number of Children (NOC)*, which measures the number of immediate descendants of the class [11].

- *Depth of Inheritance Tree (DIT)*, which provides for each class a measure of the inheritance levels from the object hierarchy top [11].
- *Data Access Metric (DAM),* which reflects how well the property of encapsulation is applied to a class [12].
- *Measure of Aggregation (MOA),* which measures the extent of the part-whole relationship realized by using attributes [12].
- *Number of Polymorphic Methods (NOP)* that is a measure of the overridden (or virtual) methods of an object oriented software system [12].
- *Number of Messages (NOM),* which is a measure of the services that a class provides [12].

Both datasets were derived by parsing a commercial software system and an analysis framework by calculating the above mentioned metrics for each system's class. The *c4t* tool was employed for this purpose [15]. The first data set was derived from parsing *System2,* a large logistics system implemented in Java (6782 classes). Thus the first dataset consists of 6782 10-dimensional data items. We used this first dataset in order to evaluate our algorithm's performance and compare it with k-Means. The second data set was derived from parsing a small fragment, only 50 classes, of the analysis framework. The resulted data items were at first grouped manually by a domain expert of the corresponding system, and then we applied k-Attractors against them. We used the domain expert's grouping in order to evaluate the quality of our algorithm by calculating some external clustering quality metrics such as precision and recall.

5.2. Performance evaluation

In order to evaluate the performance of k-Attractors and compare it with the performance of k-Means, we used a large dataset of software measurement data. This dataset consisted of 6782 10-dimensional data items. Each data item corresponded to a system's class and stored 10 software quality metrics for that class.

We applied both k-Attractors and k-Means against this dataset and measured the iterations performed by the two algorithms until they converged. Because every iteration in k-Attractors and k-Means has a complexity of $O(n)$, we can compare the performance of the two algorithms by comparing the number of iterations performed.

The values of input arguments to k-Attractors are presented in Table 1. The support factor *s* for the frequent itemsets discovery was chosen to be 0.1 (10%) because the value of each class metric has a large range, thus it is more difficult to find frequent itemsets with a greater support. The number of dimensions *n* is 10, because every data item contains 10 software metrics. Regarding the attractors similarity ratio *r*, we have decided not to merge any attractors so we have chosen *r* to be 1. Additionally we have chosen to give more weight to the inner product distance, in relation to the Hamming distance, because of the very different values of every class metric which result in a high Hamming distance for the most data items. Finally, we chose the outlier factor *d* to be 3, because in such a case the corresponding data item would have very irregural class metrics and thus it should be considered as an outlier.

The number of clusters in k-Attractors was derived by applying the graph partitioning algorithm (kMetis). In our case, kMetis partitioned the graph into 7 parts, thus the initial number of clusters was 7.

Table 1 - k-Attractors input arguments (performance evaluation)

Argument	Value
Support: s	*0.1*
Number of dimensions: n	*10*
Attractors similarity ratio: r	*1*
Hamming distance power: h	*1*
Inner product power: i	*3*
Outlier factor: d	*3*

Table 2 - Performance evaluation results

	k-Attractors	k-Means
Number of Iterations	**3**	**18**

However, as mentioned in Section 4.3, k-Attractors forms an extra cluster to group the outliers, thus the final number of formed clusters was 8. In order for the results to be comparable, we also applied k-Means for 8 clusters. Because of the random initialization of k-Means, we ran k-Means 10 times in order to get statistically significant results.

The number of iterations performed by the two clustering algorithms is presented in Table 2. The number of k-Means' iterations is the average number of iterations performed in the 10 executions of the algorithm. It is obvious that k-Attractors outperforms k-Means in terms of convergence speed as it requires only 16.6% of the iterations that k-Means requires until it converges. This is due to the initialization phase of the proposed algorithm, where the initial attractors of the clusters are chosen in such a way that they approximate the final attractors of the formed clusters. On the other hand, k-Means chooses randomly the initial cluster means, thus it requires more time until it converges to a steady state.

5.3. Quality evaluation

In order to evaluate the quality of k-Attractors and compare it with the quality of k-Means, we used a small

dataset consisting of the calculated metrics for 50 classes of an analysis application. Every data item contained only 4 metrics of the corresponding class (*LOC, CBO, DIT, WMC/LOC)*, thus the dataset consisted of 50 4-dimensional data items. The main reason for reducing the size of the example is that it is was easier for human experts to cluster manually a small subset of classes with low dimensions. Domain experts of the corresponding software system grouped manually the data items into 4 clusters. The utilized 4 metrics mentioned above were chosen by the domain expert as the most representative of the quality of each class. We then applied k-Attractors and k-Means against the data items and calculated the precision and recall of the formed clusters in both cases.

The values of input arguments to k-Attractors are presented in Table 3. The only differences between those input arguments and the input arguments of the previous experiment (performance evaluation) are related to the number of dimensions and the outlier factor. In this experiment we chose to relax the outlier factor and set it equal to 2, because of the small dataset and the small set of chosen metrics. The number of dimensions was set to 4 because we used only 4 metrics for every parsed system class.

Table 3 - k-Attractors input arguments (quality evaluation)

Argument	Value
Support: s	*0.1*
Number of dimensions: n	*4*
Attractors similarity ratio: r	*1*
Hamming distance power: h	*1*
Inner product power: i	*3*
Outlier factor: d	*2*

For every formed cluster by k-Attractors and k-Means, we calculated its precision and recall regarding the corresponding cluster formed by the domain expert.

The recall of a cluster c_j regarding the corresponding domain expert's group t_i is defined as:

$$\text{Recall}\left(c_j, t_i\right) = \frac{\#\left(c_j \cap t_i\right)}{\#t_i} \quad (5)$$

Similarly, the precision is defined as:

$$\text{Precision}\left(c_j, t_i\right) = \frac{\#\left(c_j \cap t_i\right)}{\#c_j} \quad (6)$$

The calculated precision and recall for every formed cluster by k-Attractors is presented in Table 4. The calculated precision and recall for every formed cluster by k-Means is presented in Table 5.

Table 4 - k-Attractors recall and precision

Cluster	Population	Precision	Recall
1	35	0.88	0.97
2	9	0.88	0.8
3	4	0.66	0.66
4	2	0.5	0.25

Table 5 - k-Means recall and precision

Cluster	Population	Precision	Recall
1	27	0.8	0.65
2	13	0.15	0.5
3	7	1.0	0.7
4	3	0.0	0.0

It is obvious from the above tables, that k-Attractors' clusters are closer to the domain expert's clusters. Especially, regarding the two largest clusters (cluster 1 and cluster 2), the corresponding calculated recall and precision are very high and better than the two corresponding k-Means clusters. Additionally, consider the smallest cluster in k-Attractors (cluster 4) and k-Means (cluster 4): those clusters correspond to the domain expert's outlier cluster. It is obvious that k-Attractors approximates the domain expert's cluster because of the application of the outlier handling phase. k-Means lacks such a phase, thus the recall and the precision of the corresponding cluster are both 0.

Hence, the experimental results show that k-Attractors forms more quality clusters than k-Means, approximating the domain expert's manually created clusters. This is due to the fact that k-Attractors is designed for the semantics of software measurement data, thus it outperforms k-Means both in performance and quality.

6. Conclusions and Future Work

The aim of this work was the development of a new partitional clustering algorithm, k-Attractors, which could be tailored for the analysis of software measurement data and overcome the weaknesses of other partitional algorithms.

The first step of the proposed algorithm is the application of a preprocessing step which calculates the initial partitions for the k-Attractors partitional algorithm. During this step, we discover the maximal frequent itemsets of the input software measurement data using the Apriori algorithm. After the frequent itemsets' discovery, we form the itemset graph and apply a graph partitioning algorithm, kMetis. The number of the resulted partitions defines the number of k-Attractors clusters. Thus defining the number of clusters, the common problem for all clustering algorithms, is resolved. Additionally, instead of randomly choosing the clusters' centers, for every formed partition, we merge its maximal frequent itemsets,

constructing that way the attractor of the corresponding cluster. As a result, the constructed initial attractors approximate the real clusters' attractors, improving that way the convergence speed of the proposed algorithm

The next step is the application of the k-Attractors main phase. During this phase, all the input software measurement data are clustered into appropriate clusters. As a distance metric we employ a composite metric consisting of the Hamming distance and the inner product. Thus, the employed metric is adapted to the way initial attractors are determined by the preprocessing step.

The last step deals with outliers. Handling outliers includes outlier discovery and grouping into a separate cluster. The discovery of outliers is based on the composite distance between a data item and its cluster's attractors. The corresponding distance is compared with the average distance inside this cluster and if it is greater than a factor, then the corresponding data item is considered as an outlier and is grouped into a designated cluster.

Preliminary results showed that the proposed clustering algorithm is about 600% faster than k-Means, the state-of-the-art partitional algorithm. Additionally, regarding software measurement data, k-Attractors appears to form better, in terms of quality, and more concrete clusters.

We intend to investigate ways to further evaluate and improve the k-Attractors algorithm. At first we are interested in conducting more experiments in order to see its effectiveness and clustering quality with other types of datasets. Another interesting topic is to improve the way the initial attractors are derived by employing more innovative versions of APriori.

Acknowledgements

This research work has been partially supported by the Greek General Secretariat for Research and Technology (GSRT) and Dynacomp S.A. within the program "P.E.P. of Western Greece Act 3.4". We would also like to thank Rob van der Leek and Patrick Duin from the Software Improvement Group for their valuable comments and feedback concerning our clustering results.

References

1. National Institute of Standards and Technology (NIST), "The Economic Impacts of Inadequate Infrastructure for Software Testing", Washington D.C. 2002.
2. ISO/IEC 9126, Software Engineering – Product Quality International Standard Quality Model, Geneva 2003.
3. A.K. Jain, M.N. Murty, and P.J. Flynn, "Data Clustering: A Review", *ACM Computing Surveys*, ACM, Vol. 31, No 3, September 1999, pp. 264-323.
4. J. Han and M. Kamber, Data Mining: Concepts and Techniques, (Academic Press 2001).
5. R. Agarwal, and R. Srikant, Fast Algorithms for Mining Association Rules in Large Databases, in: Proc. 20th Int'l Conf. VLDB (1994) 487-499
6. L. Zhuang, H. Dai, "A Maximal Frequent Itemset Approach for Web Document Clustering", Proc. of the 4th IEEE Int'l Conf. on Computer and Information Technology (IEEE CIT'04), 2004
7. B.C.M. Fung, K. Wang, Martin Ester, "Hierarchical Document Clustering Using Frequent Itemsets", Proc. of the 3rd SIAM Int'l Conf. on Data Mining, 2003
8. K. Wang, C. Xu, B. Liu, "Clustering Transactions Using Large Items", Proc. of the 8th ACM Int'l Conf. on Information and Knowledge Management, pp.483-490, 1999
9. E.H. Han, G. Karypis, V. Kumar, B. Mobasher, "Clustering Based on Association Rule Hypergraphs", In Research Issues on Data Mining and Knowledge Discovery, 1997
10. W.A. Kosters, E. Marchiori and A.A.J. Oerlemans, Advances in Intelligent Data Analysis, pp: 39-50, Lecture Notes in Computer Science 1642, Springer, 1999
11. S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):pp. 476–493, 1994
12. J. Bansiya, C.G Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment", IEEE Transactions on Software Engineering, 28: pp. 4—19, 2002.
13. Y. Kanellopoulos, T. Dimopoulos, C. Tjortjis and C. Makris, "Mining Source Code Elements for Comprehending Object-Oriented Systems and Evaluating Their Maintainability" ACM SIGKDD Explorations Vol.8 Issue 1, Special Issue on Successful Real-World Data Mining Applications, pp 33-40 June 2006.
14. Y. Kanellopoulos, C. Makris and C. Tjortjis, "An Improved Methodology on Information Distillation by Mining Program Source Code", Elsevier's Data & Knowledge Engineering, May 2007, Volume 61, Issue 2, pp. 359 - 383.
15. www.code4thought.org