Exploratory Analysis of Co-Change Graphs for Code Refactoring

Hassan Khosravi and Recep Colak

School of Computing Science, Simon Fraser University Vancouver, Canada

Abstract. Version Control Systems (VCS) have always played an essential role for developing reliable software. Recently, many new ways of utilizing the information hidden in VCS have been discovered. Clustering layouts of software systems using VCS is one of them. It reveals groups of related artifacts of the software system, which can be visualized for easier exploration. In this paper we use an Expectation Maximization (EM) based probabilistic clustering algorithm and visualize the clustered modules using a compound node layout algorithm. Our experiments with repositories of two medium size software tools give promising results indicating improvements over many previous approaches.

Key words: Clustering, Software artifacts, Expectation Maximization,

1 Introduction

VCS are used to manage the multiple revisions of the same source of information and are vital for developing reliable software. They are mainly used in medium to large size projects so that many developers can simultaneously alter different parts of the code without interfering with each other's work. VCS repositories keep many versions of the code and has the ability to resolve conflicts on parts of the code that several people have changed simultaneously. On top of their regular operational duties, VCS contain very useful hidden information, which can be used to reverse engineer and refactor the codes. The underlying idea behind such methods is the fact that dependent software artifacts, which are usually elements of a submodule, co-occur in VCS transactions. Although this is the intended behavior, quality of code usually decays over time. In such cases, software artifacts of unrelated modules start to co-occur or co-change together. This is a strong indication of quality decay, which must be monitored and treated. Several approaches, most notably from the pattern mining community, have been introduced to attack this problem. Clustering is among the most famous approaches used.

The main contribution of this paper is using a probabilistic clustering algorithm on VCS repositories. Existing VCS repository mining algorithms mostly use hard clustering techniques, whereas we propose using a probabilistic soft clustering algorithm followed by a powerful compound graph based layout method to help the end user to easily comprehend the results

2 Method

In this paper we use the concept of co-change graphs as introduced in [2]. A co-change graph is an artificial graph constructed from VCS repository that abstracts the information in the repository. Let $G = (V, E)$ be a graph in which V is the set of software artifacts. An undirected edge $(v_1, v_2) \in E$ exists, if and only

if artifacts v_1 and v_2 co-occur in at least a predefined number of commit transactions. This graph is the input of the probabilistic graph clustering algorithm.

The exploited clustering algorithm which is based on Expectation Maximization (EM) framework, has only recently been applied to graph clustering. The intuition behind Expectation maximization is very similar to K-means, the most famous clustering technique [4]. Both EM and K-means algorithms have two steps; an expectation step followed by a maximization step. The first step is with respect to the unknown underlying model parameters using the current estimate of the parameters. The maximization step then provides a new estimate of the parameters. Each step assumes that the other step has been solved. Knowing the assignment of each data points, we can estimate the parameters of the cluster and the parameters of the distributions, assign each point to a cluster. Unlike kmeans, expectation maximization can use soft clustering in which variable are assigned to each cluster with a probability equal to the relative likelihood of that variable belonging to the class.

We first define the algorithm as a black box. It takes an adjacency graph A_{ij} of a graph G and the number of clusters c as input. The algorithm outputs the posterior probabilities q_{ir} denoting the probability of node v_i belonging to cluster r. Finally, assignment of nodes to clusters is done using these posterior probabilities.

Let Π_r denote the fraction of vertices in cluster r. We initialize the probability of each π_r with $(n/c+noise)$, as recommended by authors. Let θ_{ri} denote the probability that an edge from a particular node in group r connects to node i. This matrix shows how nodes in a certain cluster are connected to all the nodes of the graph. The aim of the algorithm is to represent a group of nodes for each cluster that all have similar patterns of connection to others. The parameters π_r , θ_{ri} and q_{ir} satisfy the normalization conditions:

$$\sum_{r=1}^{c} \pi_r = \qquad \sum_{i=1}^{n} \theta_{ri} = \qquad \sum_{r=1}^{c} q_{ir} = 1 \tag{1}$$

In the expectation (E) step of the algorithm the model parameter q_{ir} is updated assuming all the other model parameters is fixed.

$$q_{ir} = \frac{\pi_r \prod_j \theta_{rj}^{A_{ij}}}{\sum_s \pi_s \prod_j {\theta_{sj}}^{A_{ij}}} \tag{2}$$

In the Maximization (M) step of the algorithm, model parameters π_r and and θ_{rj} are updated assuming fixed posterior probabilities. The EM framework guarantees that the algorithm converges. The convergence happens when the log likelihood ceases to increase. Equations 2 and 3 show how the two steps are dependent on each other. To find the variable assignments we need to know the values of the model's parameters, and to update the parameters of the model the variable assignments are required.

$$\pi_r = \frac{1}{n} \sum_i q_{ir}, \qquad \theta_{rj} = \frac{\sum_i A_{ij} q_{ir}}{\sum_i k_i q_{ir}} \tag{3}$$

EM based clustering algorithm requires the number of cluster c to be given as input. This is a challenging problem. A practical solution is the Bayesian Information Criterion (BIC) which suits well with model based approaches Given

any two estimated models, the model with the lower value of BIC is preferred. In our problem definition, n is the number of software artifacts. The free parameters are θ and π. Formula 4 is used to compute the BIC score. By changing the value of c, different models and different scores are achieved. we select the c that gives the minimum BIC score.

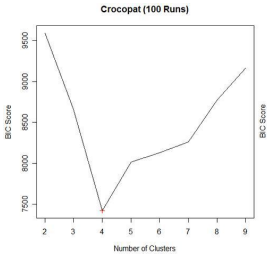
$$BIC = -2\sum_{i=1}^{n}[\ln(\pi_{g_i}) + \sum_{j=1}^{n} A_{ij} \ln_{\theta_{g_i j}}] + (c + c * n)n \tag{4}$$

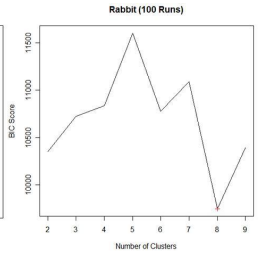
Finally, we use the compound layout algorithm as defined in [3] to visualize the result. We generate a compound node for every cluster and add all nodes to it as member nodes. Then we run the compound graph layout algorithm, which is an improved force directed layout algorithm tailored for nested graphs, using the Chisio [1] tool. We remove the compound nodes from the visualization Thus, compound nodes serves only for better layout and they are never shown to the end user.

3 Evaluation

We evaluate our clustering method by applying it to the VCS repositories of two software systems and comparing the results to authoritative decompositions. The clustering results are presented using compound node layouts that show software artifacts and their dependencies at class level. The two softwares have different sizes and different number of clusters. Because the evaluation requires the knowledge of authoritative decompositions, we chose systems that we are familiar with. We use Crocopat2.1 and Rabbit 2.1 as our examples and Figure 1 right presents some information about them

Crocopat 2.1 is an interpreter for the language RML(relational manipulation language). It takes as input an RML program and relations, and outputs resulting relations. We have experimented different number of clusters 100 time each. Figure 1 shows the average results. The results indicates 4 clusters must be used confirming the authoritative decomposition. According to the authoritative decomposition, Crocopat has four major subsystems. The output of the probabilistic graph clustering algorithm clustering for Crocopat is shown in Figure 2 left.





Project	CrocoPat 2.1	Rabbit 2.1
Line	14000	317000
Files	60	740
Changes	800	6300
Commits	140	1200
Users	1	9
Months	8	52

Fig. 1. BIC is used on Crocopat and Rabit in order to find the appropriate number of clusters to be fed to the EM algorithm. The table on the right presents some information about the Crocopat and Rabbit.

[1] Chisio: Compound or Hierarchical Graph Visualization Tool

Rabbit 2.1 is a model checking tool for modular timed automata. It is a command line program which takes a model and a specification file as input and writes out verification results. Figure 1 represents the average score for each number of clusters. Authoritative decomposition indicates that the number of clusters should be 6 whereas BIC score designate that 6 is the second best score and the best score is given by 8 clusters. We clustered the software using 6 clusters and the result achieved is illustrated in Figure 2. The clusters have been labeled on the Figure.

The result for clustering of both softwares were approved by the developers of both tools and have been found to be as good as, if not better than the existing approaches. Detailed analysis is omitted due to space considerations.

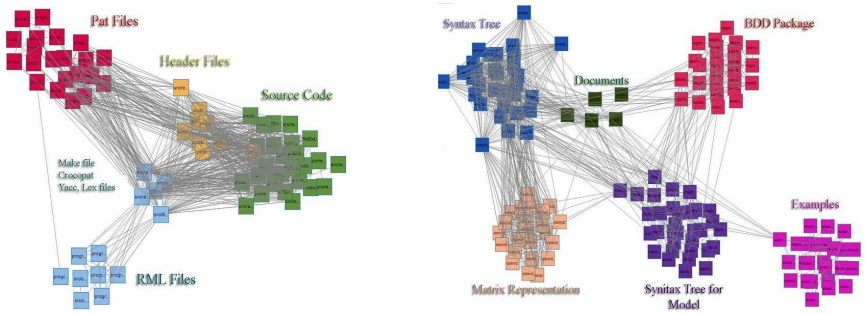


Fig. 2. Artifacts of Crocopat (one the right) and Rabbit(on the left) clustered using probabilistic fuzzy algorithm and visualized using compound nodes

4 Conclusion

In this work, we proposed a probabilistic clustering approach followed by a compound graph visualizing in order to reverse engineer a software project. The algorithm is independent of the platforms, languages and tools the project was built on. It relies only on VCS files and it has internal tools for estimating best number of components of the system . We have shown that our approach gives useful results in finding components and the relations between them, which can be used for quality monitoring and code refactoring.

Acknowledgments. This article is the result of a student project in the software-engineering course CMPT 745 in fall 2008 at Simon Fraser University, and we thank Dirk Beyer for interesting discussions.

References

1. M.Newman and E. Leicht, *Mixture models and exploratory analysis in networks*, PNAS 2007
2. Beyer D, Noack A. *Clustering Software Artifacts Based on Frequent Common Changes*. IWPC 2005.
3. Dogrusoz U, Giral E, Cetintas A, Civril A, Demir E. *A Compound Graph Layout Algorithm for Biological Pathways*, Lecture Notes in Computer Science, vol. 3383, pp. 442-447, 2005.
4. R.Dubes and A Jain, *Algorithms for Clustering Data* Prentice Hall 1998