Journal of Theoretical and Applied Information Technology
15th September 2016. Vol.91. No.1
© 2005 - 2016 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

IDENTIFYING SIMILAR BUSINESS PROCESS MODELS

[1]YEDILKHAN D., [2]BEKTEMYSSOVA G.U.
[1]Institute of information and computation technologies, Almaty, Kazakhstan
[2]International information technology university, Almaty, Kazakhstan
E-mail: [1]yedilkhan@gmail.com, [2]g.bektemisova@gmail.com

ABSTRACT

Large organizations experience shows that repositories of business process models contain lots amounts of duplication. For example, this duplication arises when the repository covers multiple variants of the same processes. This article studies the problem of identifying similar business process models. The article proposes technique for detecting clusters of approximate duplications based on well-known clustering algorithm, such as DBSCAN.

Key words: *Business Process Model, Duplication Models, Standardization, BPM*

1. INTRODUCTION

A business process is defined as a "group of tasks that together create a result of value to a customer" [1]. Its purpose is to offer each customer the right product or service, i.e., the right deliverable, with a high degree of performance measured against cost, longevity, service and quality [2]. Although increasing business process performance is an important topic in research and industry [3]. One of variants is the process of standardization by identifying similar process models. Detecting identical process model in Business Process Management systems allows modelers to identify opportunities for business process standardization. For example, consider the case of multiple variants of an insurance claims handling process, where each variant is captured as a separate process model. Given that disbursement of the insurance payout occurs in every variant (albeit differently depending on the type of claim), it is likely that these separate models will contain identical process models corresponding to disbursement activities. These models can potentially be standardized, i.e. replaced by a single model instance, and refactored as a shared subprocess. In this way, duplication is reduced and uniformity across process models is increased, to the benefit of model maintainability. Current conceptual BPMSs do not provide explicit tools for identifying duplications in repositories of process models and do not provide recommendations about how to work with them. Furthermore, such tools need to be integrated into the process execution and require continuous monitoring.

For a further understanding of the article is important to introduce two concepts. First one is a business process management system (BPMS). BPMS is a generic software system that is driven by explicit process representations to coordinate the enactment of business processes. Moreover, second one is the process model. The business process model consists of a set of activity models and execution constraints between them.

2. LITERATURE REVIEW

The problem of process models duplication detection has been widely studied in the field of software engineering, primarily in the context of source code duplication detection [4]. Duplicate code is a computer programming term for a sequence of source code that occurs more than once, either within a program or across different programs owned or maintained by the same entity. Duplicate code is generally considered undesirable for a number of reasons. [5] A minimum requirement is usually applied to the quantity of code that must appear in a sequence for it to be considered duplicate rather than coincidentally similar. Sequences of duplicate codes are sometimes known as code clones or just clones, the automated process of finding duplications in source code is called clone detection. A number of different algorithms has been proposed to detect duplicate codes such as Baker's algorithm [6], Rabin–Karp string search algorithm, Abstract Syntax Trees [7], Visual clone detection [8], Count Matrix Clone Detection [9, 10].

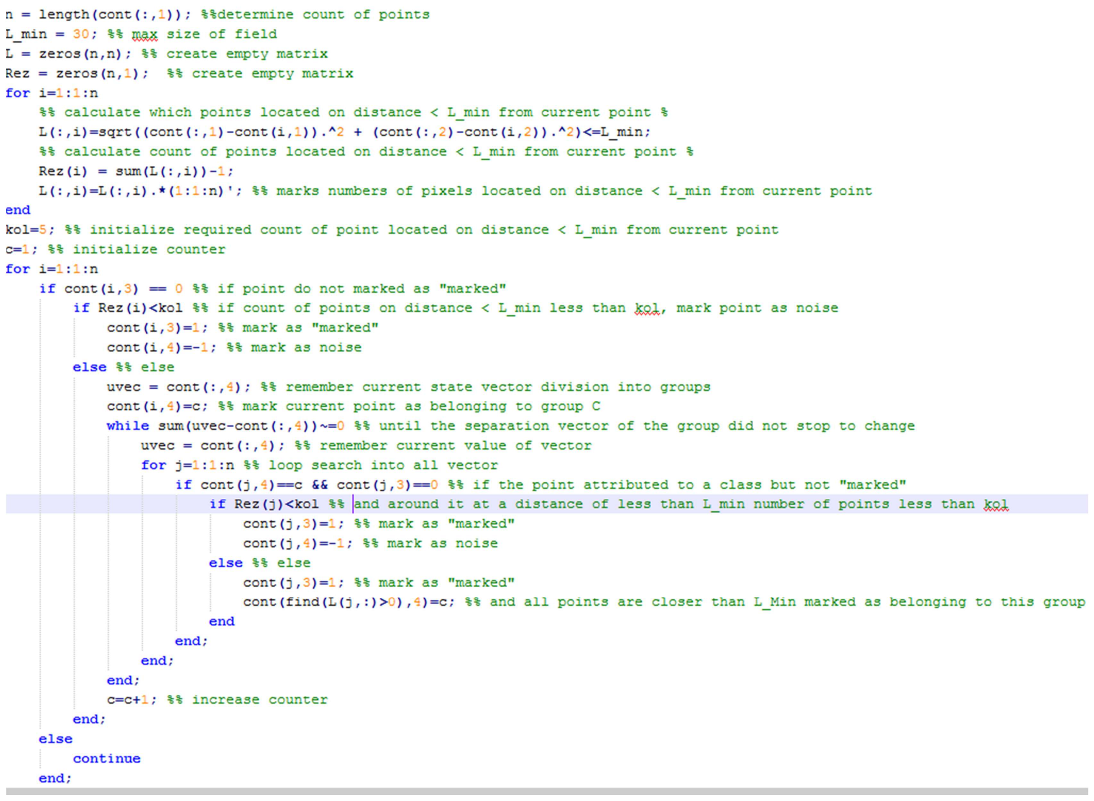
However, these techniques cannot detect process model duplications, which are arguably likely to emerge in process model repositories when process modelers copy and paste fragments across models – thus creating exact duplications – and later on these exact duplications evolve separately. Identifying identical business process models can be divided into two groups. First one is searching clones in repositories of process models, second one is discovering identical process model collections from event logs.

This article presents technique for identifying approximate duplications in repositories of process models for the purpose of standardizing. Approximate duplications is copied fragments with further modifications such as changed, added or removed model elements in addition to variations allowed in syntactically identical fragments. Technique for identifying approximate duplications taken from clustering analysis and refers to density-based clustering algorithms and shortly named DBSCAN.

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996 [11]. It is a density-based clustering algorithm that is for a given set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outlier points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature [12]. In 2014, the algorithm was awarded the test of time award (an award given to algorithms which have received substantial attention in theory and practice) at the leading data mining conference, KDD [13]. Main purpose of article is the task of grouping a set of process models in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other clusters).

3. DENSITY-BASED SPATIAL CLUSTERING OF APPLICATIONS WITH NOISE (DBSCAN)

Next part of article demonstrate realized simple example of using a clustering algorithm DBSCAN in MATLAB. The input information is a matrix *cont* with dimension *n x 4*. The first two columns are arranged to coordinate considered points, the other two columns are filled with 0. As a result of the algorithm filled columns 3 and 4 of the matrix *cont*: the third column represents information on whether the point has been processed; 4th column in charge of belonging to a particular class of 1 to-1. If the fourth column is written -1 it means that this point has been attributed to noise.



```
n = length(cont(:,1)); %%determine count of points
L_min = 30; %% max size of field
L = zeros(n,n); %% create empty matrix
Rez = zeros(n,1);  %% create empty matrix
for i=1:1:n
    %% calculate which points located on distance < L_min from current point %
    L(:,i)=sqrt((cont(:,1)-cont(i,1)).^2 + (cont(:,2)-cont(i,2)).^2)<=L_min;
    %% calculate count of points located on distance < L_min from current point %
    Rez(i) = sum(L(:,i))-1;
    L(:,i)=L(:,i).*(1:1:n)'; %% marks numbers of pixels located on distance < L_min from current point
end
kol=5; %% initialize required count of point located on distance < L_min from current point
c=1; %% initialize counter
for i=1:1:n
    if cont(i,3) == 0 %% if point do not marked as "marked"
        if Rez(i)<kol %% if count of points on distance < L_min less than kol, mark point as noise
            cont(i,3)=1; %% mark as "marked"
            cont(i,4)=-1; %% mark as noise
        else %% else
            uvec = cont(:,4); %% remember current state vector division into groups
            cont(i,4)=c; %% mark current point as belonging to group C
            while sum(uvec-cont(:,4))~=0 %% until the separation vector of the group did not stop to change
                uvec = cont(:,4); %% remember current value of vector
                for j=1:1:n %% loop search into all vector
                    if cont(j,4)==c && cont(j,3)==0 %% if the point attributed to a class but not "marked"
                        if Rez(j)<kol %% and around it at a distance of less than L_min number of points less than kol
                            cont(j,3)=1; %% mark as "marked"
                            cont(j,4)=-1; %% mark as noise
                        else %% else
                            cont(j,3)=1; %% mark as "marked"
                            cont(find(L(j,:)>0),4)=c; %% and all points are closer than L_Min marked as belonging to this group
                        end
                    end;
                end;
            end;
            c=c+1; %% increase counter
        end;
    else
        continue
    end;
```

Figure 1: Realization of simple program code in MATLAB

The initial information for the algorithm is a set of points (see. Figure below), which contains in addition to 3 clearly evolved groups, points to be attributed to noise.

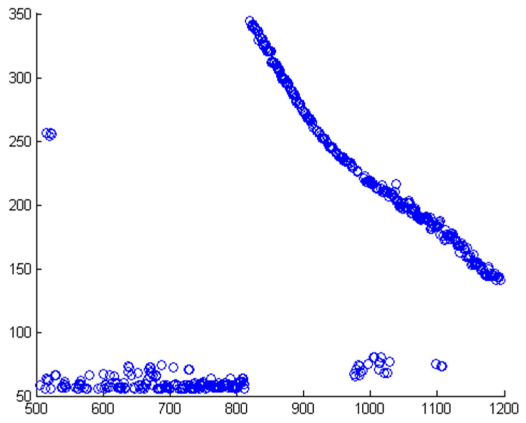


Figure 2: Input data under DBSCAN

After the algorithm we get the following picture of the cluster:

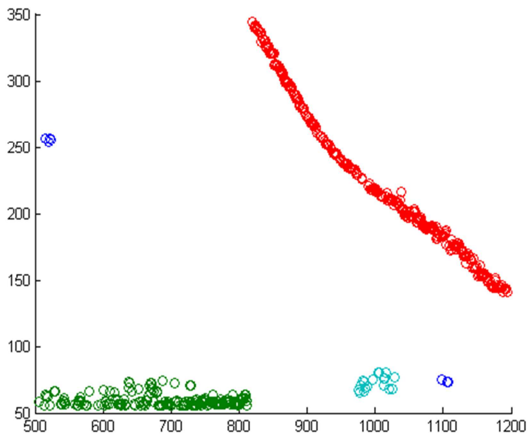


Figure 3: Input data after DBSCAN

As shown, the algorithm identified 3 groups and noise. Also we would like to draw on the fact that the algorithm is perfectly coped with the task allocation spaced points along a curve that allows you, if properly applied, use this algorithm for many other task.

4. DBSCAN FOR IDENTIFYING SIMILAR PROCESS MODELS

DBSCAN identifies all core objects of a given dataset and considers their neighborhoods as initial clusters. If two core objects are within each other's neighborhood, their neighborhoods are merged into a single cluster. On the other hand, if an object does not belong to the neighborhood of any core object, it is marked as noise. Adaptation of DBSCAN for business process models is described in **Algorithm 1**. Given the set of process fragments *G* extracted from the RPSDAG (an index structure designed for efficient and accurate identification of exact duplications in a collection of process models), the algorithm repeats the clustering process (Steps 2–14) until all fragments in *G* have been checked whether they are core objects. At the beginning of each iteration, a random fragment *f* is removed from *G* and marked as "processed". The neighborhood N_f of *f* is computed (Step 3), and if *f* is a core object the fragments in N_f are removed from *G* and from *Noise* (Step 5), and added to a new cluster *C* (Step 6). Otherwise *f* is treated as noise and another fragment is extracted from *G*. The algorithm then expands cluster *C* by checking whether there are core objects in *C* whose neighborhoods can be merged with *C*. This is done by iterating over all fragments in N_f except *f*, via a set M_C. For a fragment *m* in M_C that has not been processed, its neighborhood N_m is computed (Step 8) to determine whether *m* is itself a core object. If so, before merging its neighborhood with *C*, we check whether there is still a medoid *s* whose distance with all other fragments of the combined cluster is within τ (Step 10), otherwise we will create clusters whose fragments are far apart from each other to be standardized. In case of merging, the fragments in N_m are removed from *G* and added, except *m*, to M_C (Step 11), so that they can be checked whether they are core objects. If N_m cannot be merged with *C*, *m* is added back to *G* so that it can be eventually processed again (Step 12). In fact, N_m may form a cluster by itself or be merged with some other cluster.

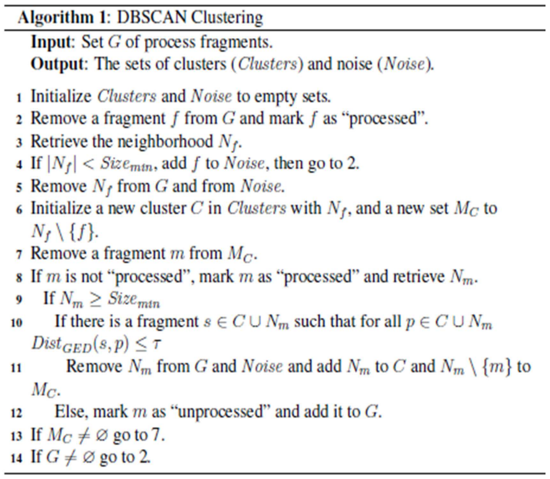
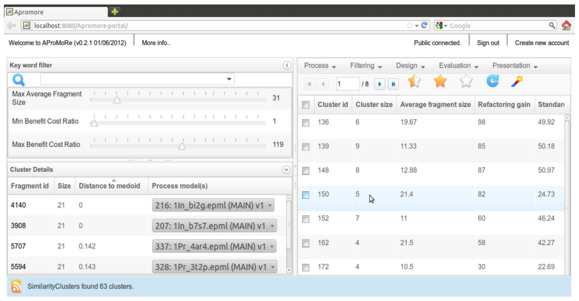

```
Algorithm 1: DBSCAN Clustering
   Input: Set $G$ of process fragments.
   Output: The sets of clusters ($Clusters$) and noise ($Noise$).
1  Initialize $Clusters$ and $Noise$ to empty sets.
2  Remove a fragment $f$ from $G$ and mark $f$ as "processed".
3  Retrieve the neighborhood $N_f$.
4  If $|N_f| < Size_{min}$, add $f$ to $Noise$, then go to 2.
5  Remove $N_f$ from $G$ and from $Noise$.
6  Initialize a new cluster $C$ in $Clusters$ with $N_f$, and a new set $M_C$ to
   $N_f \setminus \{f\}$.
7  Remove a fragment $m$ from $M_C$.
8  If $m$ is not "processed", mark $m$ as "processed" and retrieve $N_m$.
9    If $N_m \geq Size_{min}$
10     If there is a fragment $s \in C \cup N_m$ such that for all $p \in C \cup N_m$
   $Dist_{GED}(s, p) \leq \tau$
11       Remove $N_m$ from $G$ and $Noise$ and add $N_m$ to $C$ and $N_m \setminus \{m\}$ to
   $M_C$.
12     Else, mark $m$ as "unprocessed" and add it to $G$.
13 If $M_C \neq \varnothing$ go to 7.
14 If $G \neq \varnothing$ go to 2.
```

Figure 4: Algorithm of DBSCAN for process models

5. RESULTS AND CONCLUSION

The proposed technique that allows analysts to identify, cluster, analyze and visualize approximate clones by using DBSCAN algorithm realized as plugin of the "Apromore" advanced process model repository [14]. "Apromore" uses an internal process representation format named canonical process format, which captures common features of widely-used process modeling languages. The duplication detection plugin operates on this canonical format, thus it can detect approximate duplications in process models defined in different modeling languages such as BPMN and EPC for example.

The Web interface of the approximate clone detection plugin (shown in Fig. 3) provides features for creating, browsing and visualizing fragment clusters. Users can select one or more process models from the repository and kick off the clustering. Once the fragments included in the selected process models have been clustered, users can apply different filtering criteria (i.e. on the size of the clusters, on the average size of fragments) and browse the resulting clusters in a detailed list view. Another useful feature is the visualization of clusters in the 2D space. The visualization component (shown in Fig. 4) displays each fragment in a cluster as a point in the space and positions fragments within a cluster according to their distances to the medoid (distances being represented as edges between the points).


Apromore
localhost:8080/Apromore-portal/
Welcome to AProMoRe (v0.2.1 01/06/2012) | More info.. Public connected. | Sign out | Create new account

Key word filter

Max Average Fragment Size	31
Min Benefit Cost Ratio	1
Max Benefit Cost Ratio	119

Cluster Details

Fragment id	Size	Distance to medoid	Process model(s)
4140	21	0	216: 1In_bi2g.epml (MAIN) v1
3908	21	0	207: 1In_b7s7.epml (MAIN) v1
5707	21	0.142	337: 1Pr_4ar4.epml (MAIN) v1
5594	21	0.143	328: 1Pr_3t2p.epml (MAIN) v1

SimilarityClusters found 63 clusters.

Process Filtering Design Evaluation Presentation
1 /8

Cluster id	Cluster size	Average fragment size	Refactoring gain	Standar
136	6	19.67	98	49.92
139	9	11.33	85	50.18
148	8	12.88	87	50.97
150	5	21.4	82	24.73
152	7	11	60	46.24
162	4	21.5	58	42.27
172	4	10.5	30	22.69

Figure 5: Web interface of program

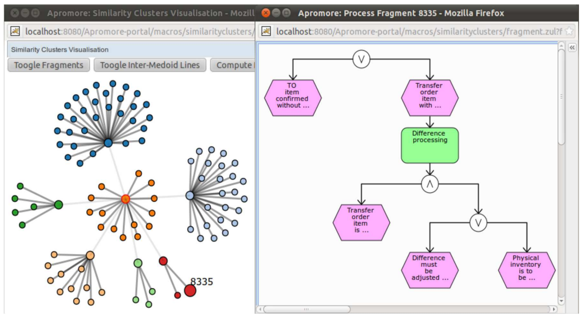

Apromore: Similarity Clusters Visualisation - Mozill
localhost:8080/Apromore-portal/macros/similarityclusters/
Similarity Clusters Visualisation
Toogle Fragments
Toogle Inter-Medoid Lines
Compute
8335
Apromore: Process Fragment 8335 - Mozilla Firefox
localhost:8080/Apromore-portal/macros/similarityclusters/fragment.zul?f
V
TO item confirmed without ...
Transfer order item with ...
Difference processing
Λ
Transfer order item is ...
V
Difference must be adjusted ...
Physical inventory is to be ...

Figure 6: Visualization options

For identifying similar process models we looked at dataset that is taken from an insurance company under condition of anonymity. It contains 363 models ranging from 4 to 461 nodes (average 27.12). The cluster computation is dominated by the computation of the distance matrix which took more than 2 hours for the insurance dataset. The longer time taken for the insurance dataset is justified by the size of its fragments (e.g. the largest fragment in the insurance dataset is a rigid with 461 nodes). For the insurance dataset we retrieved 243 clusters with DBSCAN (sizes between 2 and 6). This confirms the intuition that real-life process model repositories contain a large number of approximate clone clusters, and thus that copying/pasting of fragments across process models is a very common practice.

Hence, it can be concluded that the proposed technique provides a basis for identifying clusters of approximate duplications that are usable to standardization. One of direction of future work is how a detected clusters should be standardized into a single reference fragment in such a way that the stakeholders involved in the management and execution of the process are satisfied with the standardized process.

One of future research direction is how to work with identified similar process models. Interesting question is how process models can be standardized and how to use one variant of process model in different business processes. These changes mean increasing of business process management performance.

REFERENCES:

[1] Michael Hammer. Beyond Reengineering How the process-centered organization is changing our work and our lives. Harper Collins Publishers, 1996.
[2] Ivar Jacobson, Maria Ericson, and Agneta Jacobson. The Object Advantage Business Process Reengineering with Object Technology. ACM Press, Addison-Wesley Publishing, 1994.
[3] Fabio Casati. Industry trends in business process management getting ready for prime time. In 16th International Workshop on Database and Expert Systems Applications (DEXA 2005), First International Workshop on Business Process Monitoring and Performance Management (BPMPM 2005). IEEE Press, August 2005.
[4] C. K. Roy, J. R. Cordy, R. Koschke, Comparison and evaluation of code clone detection techniques and tools: A qualitative approach, Sci. Comput. Program. 74 (7) (2009) 470–495.
[5] Stefan Wagner, Asim Abdulkhaleq, Ivan Bogicevic, Jan-Peter Ostberg, Jasmin Ramadani. How are functionally similar code clones syntactically different? An empirical study and a benchmark PeerJ Computer Science 2:e49.doi:10.7717/peerj-cs.49.
[6] Brenda S. Baker. A Program for Identifying Duplicated Code. Computing Science and Statistics, 24:49–57, 1992.
[7] Ira D. Baxter, et al. Clone Detection Using Abstract Syntax Trees.
[8] Visual Detection of Duplicated Code by Matthias Rieger, Stephane Ducasse.
[9] Yuan, Y. and Guo, Y. CMCD: Count Matrix Based Code Clone Detection, in 2011 18th Asia-Pacific Software Engineering Conference. IEEE, Dec. 2011, pp. 250–257.
[10] Chen, X., Wang, A. Y., & Tempero, E. D. (2014). A Replication and Reproduction of Code Clone Detection Studies. In ACSC (pp. 105-114).
[11] Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei (1996). Simoudis, Evangelos; Han, Jiawei;

Fayyad, Usama M., eds. A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226–231. ISBN 1-57735-004-9. CiteSeerX: 10.1.1.121.9220.

[12] Most cited data mining articles according to Microsoft academic search; DBSCAN is on rank 24, when accessed on: 4/18/2010.

[13] "2014 SIGKDD Test of Time Award". ACM SIGKDD. 2014-08-18. Retrieved 2014-08-22.

[14] M. La Rosa, H. Reijers, W. Aalst, R. Dijkman, J. Mendling, M. Dumas, L. Garcıa-Banuelos, APROMORE: An Advanced Process Model Repository, Expert Systems With Applications 38 (6).