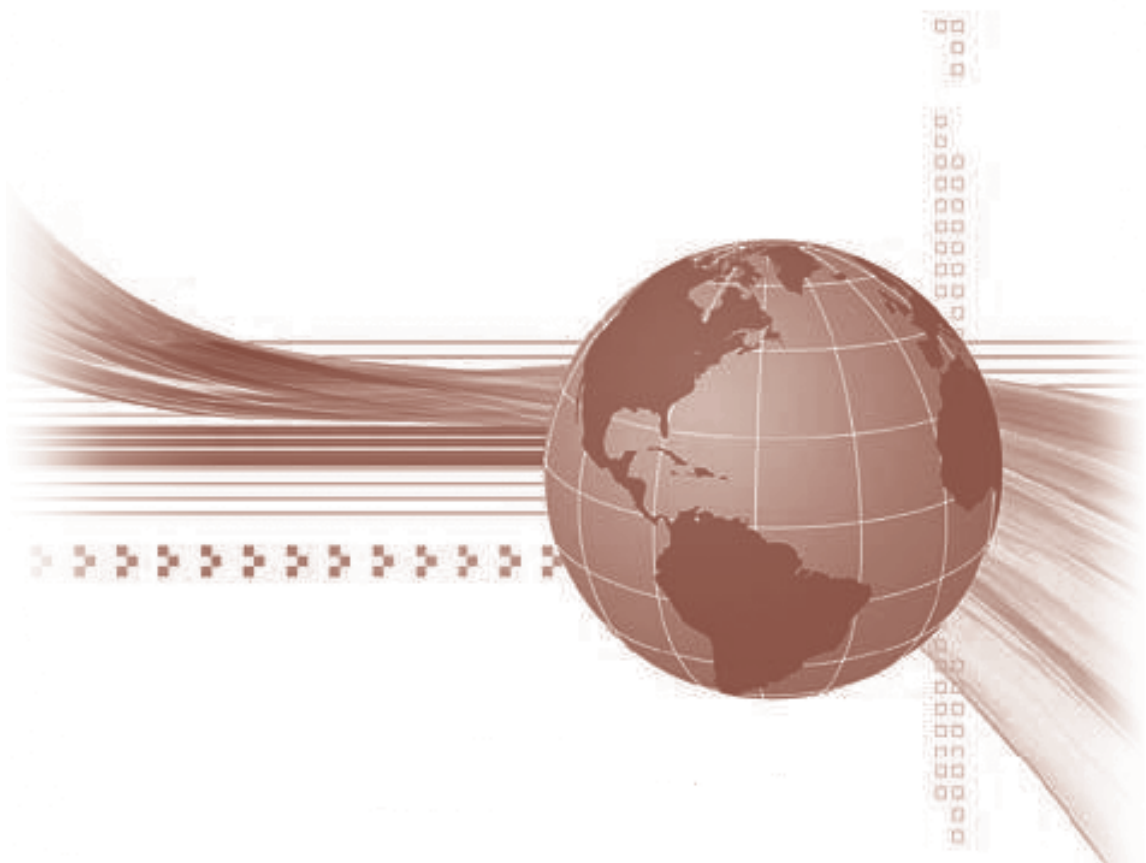
STUDIA UNIVERSITATIS
BABEŞ-BOLYAI



INFORMATICA

1/2017

STUDIA

UNIVERSITATIS BABEŞ-BOLYAI
INFORMATICA

No. 1/2017

January - June

EDITORIAL BOARD

YEAR **Volume 62 (LXII) 2017**
MONTH **JUNE**
ISSUE **1**

S T U D I A
UNIVERSITATIS BABEŞ-BOLYAI
INFORMATICA

1

EDITORIAL OFFICE: M. Kogălniceanu 1 • 400084 Cluj-Napoca • Tel: 0264.405300

SUMAR – CONTENTS – SOMMAIRE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 1, 2017
DOI: 10.24193/subbi.2017.1.01

THE USE OF SIMPLE CELLULAR AUTOMATA IN IMAGE PROCESSING

LAURA DIOSAN, ANCA ANDREICA, AND ALINA ENESCU

Abstract. Cellular Automata have been considered for a series of applications among which several image processing tasks. The goal of this paper is to investigate such existing methods, supporting the broader goal of identifying Cellular Automata rules able to automatically segment images. With the same broader goal in mind as future work, a detailed description of evaluation metrics used for image segmentation is also given in this paper.

1. Introduction

The one-dimensional binary-state Cellular Automata (CA) capable of performing computational tasks has been extensively studied in the literature [13, 34, 19, 23, 4]. Usually, a one-dimensional lattice of N two-state cells is used for representing the CA. The state of each cell changes according to a function depending on the current states in the neighbourhood. The neighbourhood of a cell is given by the cell itself and its r neighbours on both sides of the cell, where r represents the radius of the CA. The initial configuration of cell states (0s and 1s) for the lattice evolves in discrete time steps updating cells simultaneously according to the CA rule.

CAs have been considered for a series of applications like computer processors, cryptography, physical reality modelling, image processing and many others. Three-dimensional CAs have mainly been used within the framework of chemical systems for tasks like percolation description, dissociation of organic acid in solutions, bond interactions, simulation of diffusion controlled reaction kinetics, dissolution and many others [16].

In image processing for example, two-dimensional CAs are usually involved. The pixels of the image represent cells of the CA and they update their state

Received by the editors: November 5, 2016.

2010 *Mathematics Subject Classification.* 68Q80, 94A08.

1998 *CR Categories and Descriptors.* A.1 [**General Literature**]: INTRODUCTORY AND SURVEY; F.1.1 [**Theory of Computation**]: COMPUTATION BY ABSTRACT DEVICES – *Models of computation.*

Key words and phrases. cellular automata, image processing.

based on the states of the neighbouring cells (pixels). Multiple states of CA cells allow the processing of greyscale images or colour images. Identifying the rules that apply to cells in order to answer a certain request in image processing is nevertheless a nontrivial task.

Cellular Automata have been used for various image processing tasks among which: geometric transformations, noise filtering, feature detection, edge detection. Image segmentation was also approached by the means of Cellular Automata, but there are only few attempts in the literature.

Incorporating cellular automata into image segmentation brings several advantages:

- ease of implementation;
- parallel implementation;
- the number of classes does not need to be specified before segmentation is performed (both two-label and multi-label image segmentations are possible);
- extensibility (to various features extracted from images): currently, pixel intensity values have been used as state transition rules, but other image features such as texture or edges could be easily incorporated into the update mechanism;
- possibility to work with images of any dimension (the computational complexity of the segmentation process is not directly influenced by the image size or the number of image features).

The simplest use of CA for image processing is given by the application of specific rules for different tasks, for example totalistic rule [6, 8, 25], majority rule [38] or linear rule [20, 21].

Seed-based CAs represent another category of CA applied for image processing. One of the most popular approaches found in the literature in this sense is the GrowCut algorithm [37]. In [14] the authors show that the seeded GrowCut proposed by Vezhnevets[37] is essentially no different from the Ford-Bellman algorithm that computes shortest paths from a cell to all the other cells in the CA. An unsupervised version of GrowCut is proposed in [11]. Another version of GrowCut, that improves its ability to correctly detect the edges, is proposed in [1]. Other variants of GrowCut are proposed in [17, 12]. In [26] the authors propose an enhancement of GrowCut with automatic seed selection. In [2] the image noise is reduced (and therefore the GrowCut algorithm improved) by adding an anisotropic diffusion filter.

Another class of CAs applied to image processing involves methods for finding the optimal rule for a given task. A deterministic method based on a Hill-Climbing approach is proposed in [27]. There are also many heuristic

methods based on Genetic Algorithms [35, 24, 32, 33, 15], Particle Swarm Optimization [9], Genetic Programming [31, 30].

A CA based Level Set approach was proposed in [3], and continuous CAs have been applied for image processing tasks in [29, 28].

Due to the fact that beyond the goal of this paper, our final goal is to identify CA rules that are able to successfully segment images, we intend to study the application of CA rules for image processing tasks which are close to image segmentation, like edge detection. On this purpose, the aim of this paper is to describe in detail the first class of CAs applied for image processing, namely CAs that are using specific given rules, the class of so-called 'Simple CA'. From the same perspective of a final goal, a detailed description of the most popular performance measures used for evaluating the segmentation results is also given in this paper.

2. Simple Cellular Automata for Image Processing

2.1. **Totalistic rule.** A CA very similar to the Conway's Game of Life [10] is used in [6] in order to detect the edges of an object in an image. The authors of [6] apply this method for ultrasound kidney images. The greyscale images are binarized prior to the application of the CA based method. A black cell is called 'alive' and will have the value 1, while a white cell is called 'dead' and will have the value 0. The Moore neighbourhood gives the neighbours of a cell; therefore a cell has 9 neighbours, including the cell itself. In order to apply the rule, one has to compute first the sum of the neighbours values (including the cell itself) of each cell. The rule specifies those cells with 3 alive neighbours or less will die of loneliness, while cells with 7 neighbors or more will die of overpopulation. The cells with 5 neighbours will revive and the cells with 4 or 6 will keep their previous state. After one iteration of rule application, the boundaries are detected.

The same metaphor of the Game of Life can also be found in [8]. The authors work on binarized greyscale images, use the Moore neighbourhood and have the same cell state meaning similar to [6]. They apply different 'survival' rules and find, experimentally, that the best rule is given by the survival or the revival of the cells having 3, 4, 5, 6 and 7 alive neighbours. The results are presented for 3 real world images, the performance of the proposed method being only visually analyzed.

2.2. **Linear rule.** Due to the fact that the rule search space is significantly large (2^{2^9} possible rules for Moore neighbourhood) and an exhaustive search is therefore out of question, there are researchers that focused their investigation on linear rules. The linear rules are those that can be realized by EX-OR operation only, the search space being thus reduced to 512 rules. A

detailed presentation of theory and application of two-dimensional, null boundary, nine-neighbourhood cellular automata linear rules in given in [5]. There are 9 fundamental rules (1, 2, 4, 8, 16, 32, 64, 128, 256 - powers of 2), which are arranged in a certain order inside a 3x3 grid which resembles a Moore neighbourhood. Each of this 9 fundamental rules specifies which neighbour is considered when changing the state of the current cell, based on EX-OR operations. Adding these powers of 2 gives us other rule numbers that again represent the neighbours that contribute to the state of the current cell at the next iteration.

In [5] the 9 fundamental linear rules are applied for solving several image transformation tasks like translation, generation of multiples copies, zooming, thickening and thinning of symmetric images.

The authors of [25] apply all 512 linear rules to edge detection in one image only and identify three groups of rules: no edge detection rules, strong edge detection rules and weak edge detection rules. However, there is no strong evidence of the significance of these groups of rules since only one image has been used for testing purposes. Moreover, it is not clear how do the authors apply the linear rules for greyscale images, since supporting theory of linear rules deals only with binary images.

In [20], the authors show that there are 4 rules among the 512 linear rules described above that obtain best results for edge detection. Only two images are used in order to show the performance of these 4 rules, and one more image is used in order to provide comparisons with other existing methods for edge detection. However, the results are not conclusive since only 3 images are being used and only visual evidence of the rules performance is given. Moreover, the images are first binarized because these rules cannot be directly applied to greyscale images.

The linear rules described above are extended to a 25 neighbourhood (extended Moore neighbourhood) in [22]. Among all resulted linear rules, the authors find some optimal rules that can be applied to edge detection. These optimal rules are applied to 2 images (a priori binarized) and the results are only visually compared to the results obtained by other methods of edge detection. Moreover, no details of the method used for identifying the optimal rules are given in this paper.

3. Evaluation measures

In image segmentation, it is very important to establish how we define similar regions or segmentations. Segmented regions and their boundaries can be compact, discontinuous, smooth, etc. One of the most popular evaluation

		real segments	
		interest segment	background segment
computed segments	interest segment	TP	FN
	background segment	FN	TN

TABLE 1. Confusion Matrix

metrics (but not very reliable) is the **Dice coefficient** [7]. Dice computes the overlap between regions, quantifying the similarity of two segmentations.

Given two segmentations:

- reference segmentation (gold standard) S_r
- machine segmentation S_m

Each image point (pixel) can be classified as:

- true positive (TP): $S_r(x, y)$ is 1 $\wedge$ $S_m(x, y)$ is 1
- false positive (FP): $S_r(x, y)$ is 0 $\wedge$ $S_m(x, y)$ is 1
- true negative (TN): $S_r(x, y)$ is 0 $\wedge$ $S_m(x, y)$ is 0
- false negative (FN): $S_r(x, y)$ is 1 $\wedge$ $S_m(x, y)$ is 0

The Dice similarity coefficient is computed as the ratio between the number of pixels belonging to the intersection (of two possible segmentations) and the average of their sizes:

$$\text{Coeff}_{\text{Dice}}(S_m, S_r) = \frac{2\,|S_r \cap S_m|}{|S_r| + |S_m|} = \frac{2TP}{2TP + FP + FN} \tag{1}$$

For increased reliability, one has to also look at how the values of each pixel in the segmented image compare against some gold standard or ground truth.[1] The four basic cardinalities of the so–called *confusion matrix*, namely the true positives (TP), the false positives (FP), the true negatives (TN), and the false negatives (FN) are defined as follows:

Let $I(x, y) : R^2 \to R$ be a two-dimensional image and $S(I(x, y)) : R^2 \to \Omega$, $\Omega = \{0, 1, 2, \ldots, k-1\}$, be a k-ry decision segmentation of the image $I(x, y)$.

Each of these segmentations are composed by k segments, or regions, or classes (e.g. if $k = 2$, then the two segments are represented by the class of interest and the background; if $k = 3$, then two classes of interest and the background will represent possible segments). In the case of $k = 2$ segments, the confusion matrix can be represented as shown in Table 1.

[1] In order to call the reference segmentation *ground truth* we have to be certain that it is so. Manual reference segmentations drawn by experts normally approximate ground truth, in which case it can be used as gold standard, but not as the ground truth itself.

An alternative evaluation measure can be expressed as a percentage and its values range between 0 (no overlap) and 1 (perfect agreement) using the above values.

$$\mathrm{F}_\beta = \frac{(\beta^2 + 1) * \mathrm{Precision} * \mathrm{Recall}}{\beta^2 * \mathrm{Precision} + \mathrm{Recall}} \tag{2}$$

It is also called the *overlap index* and makes it possible to quantify reproducibility. An equivalent of the Dice coefficient is, therefore, the F_β measure with $\beta = 1$.

Precision is another measure that can be used to evaluate the quality of segmentation.

$$\frac{TP}{TP + FP} \tag{3}$$

Recall is computed as the ratio between the number of positive pixels in the reference image and the number of pixels identified as positive in the segmented image.

$$\mathrm{Recall} = \frac{TP}{TP + FN} \tag{4}$$

In conjunction with Precision, Recall is used in order to compute the F–measure.

Specificity is computed as the ration between the number of negative pixels in the reference image and the number of pixels identified as negative in the segmented image.

$$\mathrm{Specificity} = \frac{TN}{TN + FP} \tag{5}$$

Recall and Specificity depend on the size of segments.

There are two other measures that are related to these metrics, namely **Fallout** and the **false negative rate** (FNR). They are defined by:

$$\mathrm{Fallout} = \frac{FP}{FP + TN} = 1 - \mathrm{Specificity} \tag{6}$$

$$\mathrm{FNR} = \frac{FN}{FN + TP} = 1 - \mathrm{Recall} \tag{7}$$

Since the last two measures are equivalent to Specificity and Recall, only one pair ((Recall, Specificity) or (Fallout, False Negative rate)) should be used to evaluate the performance of segmentation.

Recall is also called Sensitivity or True Positive Rate. Specificity is also called True Negative Rate (TNR). Fallout is also called the false positive rate (FPR).

Another frequently used evaluation measure is the **Global Consistency Error** (GCE) [18]. An error-based measure is the complement to similarity measures, in that two segmentations are identical if an error–based measure is 0.

This measure is computed as an average over the error of pixels belonging to two segmentations. It compares partitions of the same image and it is tolerant to one partition refining the other (e.g. by splitting or merging regions). For an image I of n pixels ($n = |I|$) and a segmented region S, we denote the set of all neighbour pixels to pixel p which belong to the same segmentation region S by $R(S, p)$. For two segmentations, one computed S_c and one reference segmentation S_r, the asymmetric Local Refinement Error in [18] at pixel p, $\mathrm{LRE}(S_c, S_r, p)$ is defined as

$$\mathrm{LRE}(S_c, S_r, p) = \frac{|R(S_c, p) - R(S_r, p)|}{|R(S_c, p)|} \tag{8}$$

The GCE between segmentations can be defined as a mean over the error of all points (pixels):

$$\mathrm{GCE}(S_1, S_2) = \frac{1}{|I|} \min \left\{ \sum_{i=1}^{|I|} LRE(S_1, S_2, p), \sum_{i=1}^{|I|} LRE(S_2, S_1, p) \right\} \tag{9}$$

By using the cardinalities previously introduced, GCE can be expressed as follows:

$$GCE(S_c, S_r) = \frac{1}{|I|} \min \left\{ \frac{FN(FN + 2TP)}{TP + FN} + \frac{FP(FP + 2TN)}{TN + FP}, \frac{FP(FP + 2TP)}{TP + FP} + \frac{FN(FN + 2TN)}{TN + FN} \right\} \tag{10}$$

This measure is able to quantify the consistency between image segmentations of differing granularities. It has the advantage of being tolerant to (label) refinement. It makes most sense to use this measure when the two segmentations being compared have comparable numbers of segments [36].

4. Conclusions

This paper presents in detail a class of CAs applied for image processing tasks that are related to image segmentation, as well as a detailed description

of the most popular performance measures used for evaluating the segmentation results. As further work, an exhaustive description of CA based methods for image processing will be performed, followed by the proposal of competitive CA based methods for the task of image segmentation.

Acknowledgment

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS - UEFISCDI, project number PN-II-RU-TE-2014-4-1130.

References

[1] Ryan A Beasley. Semiautonomous medical image segmentation using seeded cellular automaton plus edge detector. *ISRN Signal Processing*, 2012:1–9, 2012.

[2] Lei Bi, Jinman Kim, Lingfeng Wen, A. Kumar, M. Fulham, and D.D. Feng. Cellular automata and anisotropic diffusion filter based interactive tumor segmentation for positron emission tomography. In *Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE*, pages 5453–5456, 2013.

[3] Yu Chen, Zhuangzhi Yan, and Yungao Chu. Cellular automata based level set method for image segmentation. In *Complex Medical Engineering, 2007. CME 2007. IEEE/ICME International Conference on*, pages 171–174, May 2007.

[4] C. Chira, A. Gog, R. I. Lung, and D. Iclanzan. Complex systems and cellular automata models in the study of complexity. *Studia Informatica series*, LV:33–49, 2010.

[5] Pabitra Pal Choudhury, Birendra Kumar Nayak, Sudhakar Sahoo, and Sunil Pankaj Rath. Theory and applications of two-dimensional, null-boundary, nine-neighborhood, cellular automata linear rules. *CoRR*, abs/0804.2346, 2008.

[6] C. Callins Christiyana, V. Rajamani, and A. Usha Devi. Article: Ultra sound kidney image retrieval using time efficient one dimensional glcm texture feature. *IJCA Special Issue on Advanced Computing and Communication Technologies for HPC Applications*, ACCTHPCA(4):12–17, July 2012. Full text available.

[7] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[8] Manoj Diwakar, Pawan Kumar Patel, and Kunal Gupta. Cellular automata based edge-detection for brain tumor. In *ICACCI*, pages 53–59. IEEE, 2013.

[9] Safia Djemame and Mohamed Batouche. Combining cellular automata and particle swarm optimization for edge detection. (14/):16–22, 2012.

[10] Martin Gardner. The fantastic combinations of john conway's new solitaire game "life". *Scientific American*, 223(10):120–123, October 1970.

[11] Payel Ghosh, Sameer Antani, L. Rodney Long, and George R. Thoma. Unsupervised grow-cut: Cellular automata-based medical image segmentation. In *HISB*, pages 40–47. IEEE, 2011.

[12] Andac Hamamci, Nadir Kucuk, Kutlay Karaman, Kayihan Engin, and Gözde B. Ünal. Tumor-cut: Segmentation of brain tumors on contrast enhanced MR images for radiosurgery applications. *IEEE Trans. Med. Imaging*, 31(3):790–804, 2012.

[13] Hugues Juille and Jordan B. Pollack. Coevolving the ideal trainer: Application to the discovery of cellular automata rules. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 519–527. Morgan Kaufmann, 22-25 July 1998.

[14] Claude Kauffmann and Nicolas Piche. Seeded nd medical image segmentation by cellular automaton on gpu. *Int. J. Computer Assisted Radiology and Surgery*, 5(3):251–262, 2010.

[15] Okba KAZAR and Sihem SLATNIA. Evolutionary cellular automata for image segmentation and noise filtering using genetic algorithms. *Journal of Applied Computer Science & Mathematics*, 11(5):33–40, 2011.

[16] A.C.J. Korte and H.J.H. Brouwers. A cellular automata approach to chemical reactions; 1 reaction controlled systems. *Chemical Engineering Journal*, 228:172–178, 2013.

[17] Yan Liu, H. D. Cheng, Jianhua Huang, Yingtao Zhang, and Xianglong Tang. An effective approach of lesion segmentation within the breast ultrasound image based on the cellular automata principle. *J. Digital Imaging*, 25(5):580–590, 2012.

[18] D. R. Martin, C. C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *ICCV*, pages II: 416–423, 2001.

[19] Melanie Mitchell, Michael D. Thomure, and Nathan L. Williams. The role of space in the success of coevolutionary learning. In *Artificial Life X: Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*, pages 118–124. MIT Press, 2006.

[20] Jahangir Mohammed and Deepak Ranjan Nayak. An efficient edge detection technique by two dimensional rectangular cellular automata. *CoRR*, abs/1312.6370, 2013.

[21] Deepak Ranjan Nayak, Prashanta Kumar Patra, and Amitav Mahapatra. A survey on two dimensional cellular automata and its application in image processing. *IJCA Proceedings on International Conference on Emergent Trends in Computing and Communication (ETCC-2014)*, ETCC(1):78–87, 2014.

[22] Deepak Ranjan Nayak, Sumit Kumar Sahu, and Jahangir Mohammed. A cellular automata based optimal edge detection technique using twenty-five neighborhood model. *CoRR*, abs/1402.1348, 2014.

[23] Gina M. B. Oliveira, Luiz G. A. Martins, Laura B. de Carvalho, and Enrique Fynn. Some investigations about synchronization and density classification tasks in one-dimensional and two-dimensional cellular automata rule spaces. *Electr. Notes Theor. Comput. Sci.*, 252:121–142, 2009.

[24] Blanca Priego, Daniel Souto, Francisco Bellas, and Richard J. Duro. Hyperspectral image segmentation through evolved cellular automata. *Pattern Recognition Letters*, 34(14):1648–1658, 2013.

[25] Fasel Qadir, Peer M. A., and Khan K. A. Efficient edge detection methods for diagnosis of lung cancer based on two dimensional cellular automata. *Advances in Applied Science Research*, 4(3):2050–2058, 2012.

[26] R. S. RajKumar and G. Niranjana. Image segmentation and classification of mri brain tumor based on cellular automata and neural networks. *International Journal of Research in Engineering & Advanced Technology*, 1(1):1–7, 2013.

[27] P. L. Rosin. Training cellular automata for image processing. In *SCIA*, pages 195–204, 2005.

[28] D. Safia and B.M. Chawki. Image segmentation using an emergent complex system: Cellular automata. In *Systems, Signal Processing and their Applications (WOSSPA), 2011 7th International Workshop on*, pages 207–210, May 2011.

[29] D. Safia, D. Oussama, and B.M. Chawki. Image segmentation using continuous cellular automata. In *Programming and Systems (ISPS), 2011 10th International Symposium on*, pages 94–99, April 2011.

[30] Mohamed Sandeli and Mohamed Batouche. Multilevel thresholding for image segmentation based on parallel distributed optimization. In *SoCPaR*, pages 134–139. IEEE, 2014.

[31] S. Sato and H. Kanoh. Evolutionary design of edge detector using rule-changing cellular automata. In *Nature and Biologically Inspired Computing (NaBIC), 2010 Second World Congress on*, pages 60–65, Dec 2010.

[32] Sihem Slatnia, Mohamed Batouche, and Kamal E. Melkemi. Evolutionary cellular automata based-approach for edge detection. In Francesco Masulli, Sushmita Mitra, and Gabriella Pasi, editors, *Applications of Fuzzy Sets Theory, 7th International Workshop on Fuzzy Logic and Applications, WILF 2007, Camogli, Italy, July 7-10, 2007, Proceedings*, volume 4578 of *Lecture Notes in Computer Science*, pages 404–411. Springer, 2007.

[33] Sihem Slatnia and Okba Kazar. Evolutionary cellular automata based-approach for region detection. http://www.researchgate.net/publication/229050070, 2015.

[34] Marco Tomassini and Mattias Venzi. Evolution of asynchronous cellular automata for the density task. In Juan J. Merelo Guervs, Panagiotis Adamidis, Hans-Georg Beyer, Jos Luis Fernndez-Villacaas Martn, and Hans-Paul Schwefel, editors, *PPSN*, volume 2439 of *Lecture Notes in Computer Science*, pages 934–944. Springer, 2002.

[35] Blanca Maria Priego Torres, Daniel Souto, Francisco Bellas, and Richard J. Duro. Unsupervised segmentation of hyperspectral images through evolved cellular automata. In Manuel Graa, Carlos Toro, Jorge Posada, Robert J. Howlett, and Lakhmi C. Jain, editors, *KES*, volume 243 of *Frontiers in Artificial Intelligence and Applications*, pages 2160–2169. IOS Press, 2012.

[36] R. Unnikrishnan, C. Pantofaru, and M. Hebert. Toward objective evaluation of image segmentation algorithms. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 29(6):929–944, June 2007.

[37] Vladimir Vezhnevets and Vadim Konouchine. Growcut" - interactive multi-label N-D image segmentation by cellular automata. pages 1–7. Russian Academy of Sciences, 2005.

[38] Sartra Wongthanavasu. *Cellular Automata for Medical Image Processing*. Cellular Automata - Innovative Modelling for Science and Engineering. INTECH Open Access Publisher, unknown 2011.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: {lauras, anca, aenescu}@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 1, 2017
DOI: 10.24193/subbi.2017.1.02

THIRD CASE STUDY FOR THE DYNAMIC MULTILEVEL COMPONENT SELECTION

ANDREEA VESCAN

Abstract. The architecture of a system changes after the deployment phase due to new requirements from the stakeholders. The software architect must make decisions about the selection of the right software components out of a range of choices to satisfy a set of requirements. This paper deals with the component selection problem with a multilevel system view in a dynamic environment.

To validate our approach we have used the case study method. Three different case studies were performed but only one is presented in the current paper. The research design was conducted using a research question, propositions and for interpreting the study's findings we have used the Wilcoxon signed ranks statistical test. The tests performed show the potential of evolutionary algorithms for the dynamic multilevel component selection problem.

1. Introduction

The problems of identification and selection the right software components out of a range of choices to satisfy a set of requirements have received considerable attention in the field of component-based software engineering during the last two decades [9, 10].

Identification of a software architecture for a given system may be achieved in two steps: (1) Component Identification and (2) Component Selection. Component Identification has the scope to partition functionalities of a given system into non-intersecting logical components to provide the starting points for designing the architecture. The aim of Component Selection methods is to find suitable components from a repository to satisfy a set of requirements under various constraints/criteria (i.e. cost, number of used components, etc.).

Received by the editors: January 25, 2017.

2010 *Mathematics Subject Classification.* 68N01, 68T20.

1998 *CR Categories and Descriptors.* D.2.2 [**Design Tools and Techniques**]: *Object-oriented design methods, Software libraries.*; I.2.8 [**Problem Solving, Control Methods, and Search**]: *Heuristic methods.*

Key words and phrases. Case study, Research design, Component selection, Dynamic, Multilevel, Multiobjective optimization.

This paper has focused on the component selection process, the goal being to provide the suitable existing components matching software requirements.

The **contribution** of this paper is **the use of the case study method and the research design** from the book of Yin [5] to validate our research proposal for the **Dynamic Multilevel Component Selection Problem** [2]. A research question and propositions are used to conduct research design. For interpreting the study's findings we used the Wilcoxon signed ranks statistical test. We have conducted three different experiments. The first one was reported in [1], and the second one in [2]. The current paper **reports the third experiment**. For each case study we have specified the component selection problem. After that, the experimental studies were followed for each considered case study: the two perspectives, changing requirements and changing component repository. Following the replication approach to multiple-case studies [5], each individual case study was finalized by an individual case report that will be next considered to be part of a summary report, i.e. a cross-case conclusion. Thus, in our case the results obtained are reported and conclusions about the potential of evolutionary algorithms for the dynamic multiobjective multilevel component selection problem are drawn.

The paper is organized as follows: Section 2 contains configuration and reconfiguration description problems, the description of the optimization process, and the proposed evolutionary-based algorithm approach. Section 3 presents the reasons for using case study method and the research design, the criteria used to interpret the findings of the results. The evaluation is presented in Section 4. In Section 5 we apply the approach to one example to validate the proposed approach. Some experiments are performed considering two dynamics: requirements changes over time and component repository varies over time. Section 6 introduces the current state of art regarding the component selection problem and analysis the differences compared with our present approach. We conclude our paper in Section 7.

2. Dynamic Multilevel Component Selection Problem

2.1. **Component Systems, Configurations and Reconfigurations.** A component [23] is an independent software package that provides functionality via defined interfaces. The interface may be an export interface through which a component provides functionality to other components or an import interface through which a component gains services from other components.

A configuration [24] of a component system is described as the structural relationship between components, indicated by the layout of components and connectors. Reconfiguration means modifying the structure of a component

system in terms of additions, deletion, and replacement of components and/or connectors. While the reconfiguration transforms the structural view of a component system, it changes the system's functionality and service specifications. From another aspect, a reconfiguration may consist of several individual updates or changes to components and/or connectors.

There are two type of components: *simple component* - is specified by the inports (the set of input variables/parameters), outports (the set of output variables/parameters) and a function (the computation function of the component) and *compound component* - is a group of connected components in which the output of a component is used as input by another component from the group. For details about the component model please refer to [25].

2.2. **Dynamic Multilevel Component Selection Problem Formulation.** An informal specification of the *configuration problem* is described in the following. It is needed to construct a final system specified by input data and output data. We can see the final system as a compound component and thus the input data becomes the required interfaces of the component and the output data becomes the provided interfaces, and in this context we have the required interfaces as provided and we need to provide the internal structure of the final compound component by offering the provided interfaces.

A formal definition of the *configuration problem* [25] (seen as a compound component) is as follows. Consider SR the set of final system requirements (the provided functionalities of the final compound component) as $SR = \{r_1, r_2, ..., r_n\}$ and SC the set of components (the repository) available for selection as $SC = \{c_1, c_2, ..., c_m\}$. Each component c_i can satisfy a subset of the requirements from SR (the provided functionalities) denoted $SP_{c_i} = \{p_{i_1}, p_{i_2}, ..., p_{i_k}\}$ and has a set of requirements denoted $SR_{c_i} = \{r_{i_1}, r_{i_2}, ..., r_{i_h}\}$. The goal is to find a set of components Sol in such a way that every requirement r_j ($j = \overline{1,n}$) from the set SR can be assigned a component c_i from Sol where r_j is in SP_{c_i} ($i = \overline{1,m}$), while minimizing the number of used components and the total cost of assembly. All the requirements of the selected components must be satisfied by the components in the solution. If a selected component is a compound component, the internal structure is also provided. All the levels of the system are constructed.

The *reconfiguration problem* [24] is defined similarly to the *configuration problem* but considering the dynamical changes of either requirements or component. Regarding the *reconfiguration problem* [15], the dynamics of the component selection problem can be viewed in two ways: the system requirements or the repository containing the components varies over time.

2.3. **Dynamic Multilevel Component Selection Optimisation Process.** Our approach starts by considering a set of components (the repository) available for selection and the specification of a final system (input and output). The optimisation process begins with the *Dynamic Multilevel Component Selection Problem Formulation* (see Figure 1 for details). The result of this step is the transformation of the final system specification as the set of required interfaces (and the set of provided interfaces).

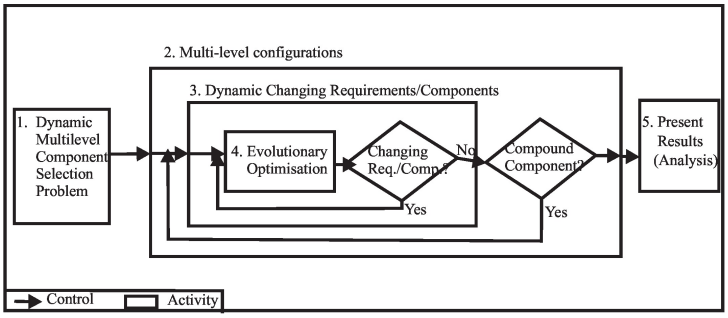


FIGURE 1. Dynamic Multilevel Component Selection Optimisation Process.

In the second step, the construction of the multilevel configurations is done by applying the evolutionary optimisation algorithm (fourth step, in Figure 1) for each time steps (from the Dynamic Changing Requirements or Dynamic Changing Components step). The evolutionary optimisation algorithm is applied for each time steps (i.e. if there are still changing requirements or components) and for each compound component from each level. The solution with best fitness value is selected at each level. The fifth step presents the results. The problem is formulated as a multiple objective optimization problem having 5 objectives: the number of used components, the number of new requirements, the number of provided interfaces, the number of the initial requirements that are not in solution, and the cost of a component (group of components). All objectives are to be minimized.

Remark. Detailed information about the optimisation process can be found in paper [1]. Since the current paper focuses on the research design and case study method we give only the reference for the proposed approach [1] and second case study paper [2] such that the reader can find the information.

3. Case study method and research design

The book of Yin [5] provided us with a strategy of identifying the method for our research project, showing when to choose the case study method and how to do research design. Defining the *Research Questions* is the most important step to consider in a research study. In general, case studies are the preferred strategy when "how" or "why" questions are being posed.

Our Research Question: How and Why do Search-based Algorithms (in our case a Genetic Algorithm and a Random Search Algorithm) provide different results for the Dynamic Multilevel Component Selection Problem?

Another component of the research design are the *Propositions* that direct attention to something that should be examined within the scope of study. These propositions begin to tell you where to look for relevant evidence.

Our Proposition: The Search-based Algorithms (in our case a Genetic Algorithm and a Random Search Algorithm) provide different results for the Dynamic Multilevel Component Selection Problem because in the case of Genetic Algorithm the fitness of a solution is reevaluated.

Criteria for interpreting a study's findings represents the third component of the research design of a case study according to [5]. Statistical analysis offer some explicit criteria for such interpretation.

Our Criteria for interpreting the study is based on the Wilcoxon signed ranks statistical test that aims to detect significant differences between two sample means, that is, the behavior of the two algorithms. For more information regarding the Wilcoxon statistical test see Section 4.

After covering these components of research designs, the construction of a theory related to our topic of study will follow. *Our Theory development*: The case study will show why the Genetic Algorithm performs better than the Random Search Algorithm.

An issue related to case studies is referring to the generalization from a case study to theory. According to [5] statistical generalization is the common way when doing surveys, but in doing case studies the analytic generalization should be used. Multiple cases resemble multiple experiments and under these circumstances the mode of generalization is analytic. If two or more cases are shown to support the same theory, replication [5] may be claimed. The replication logic is analogous to that used in multiple experiments. Some of the replications might attempt to duplicate the exact conditions of the original experiment. Other replications might alter one or two experimental conditions considered unimportant to the original findings, to see whether the findings could still be duplicated. Only with such replications would the original finding be considered robust.

Our Replication strategy: The time steps for the dynamic changing requirements (Level 1) and for the dynamic changing components (Level 1, 2, and 3) were modified for Case study 2 [1] and for Case Study 3 [2]. Additional information relating to the replication strategy is presented in paper [2].

Thus, we have selected the case study method and conducted three case studies, the first one is a real case study for constructing a *Reservation System* (reported in [1]) and the last two of them are constructed using artificial data (the second experiment is reported in [2] and the current paper reports the third experiment). Each individual case study was finalized by an individual case report that will be next considered to be part of a summary report, i.e. a cross-case conclusion. Thus, in our case the results obtained are reported and conclusions about the potential of evolutionary algorithms for the dynamic multiobjective multilevel component selection problem are drawn.

4. Interpreting the study

When comparing [8] two algorithms, the best fitness values obtained by the searches concerned are an obvious indicator to how well the optimisation process performed. Inferential statistics may be applied to discern whether one set of experiments are significantly different in some aspect from another. Usually we wish to be in a position to make a claim that we have evidence that suggests that Algorithm A (Genetic Algorithm) is better than Algorithm B (Random Search). The Wilcoxon signed ranks test [6] is used for answering the following question: do two samples represent two different populations? It is a nonparametric procedure employed in hypothesis testing situations, involving a design with two samples. It is a pairwise test that aims to detect significant differences between two sample medians, that is, the behavior of two algorithms. The best fitness value (from the entire population) was used for comparing the two algorithms.

The Wilcoxon signed ranks test has two hypothesis:

(1) Null hypothesis H_0: The median difference is zero versus.
(2) Research hypothesis H_1: The median difference is not zero, $\alpha = 0.05$.

Steps of the Wilcoxon signed ranks test: Compute W_- and W_+, Check if $W_- + W_+ = \mathrm{n(n+1)}/2$, Select the test statistic (for the two tailed test the test statistic is the smaller of W_- and W_+), We must determine whether the observed test statistic W_t supports the H_0 or H_1, i.e. we determine a critical value of W_c such that if the observed value of W_t is less or equal to critical value W_c, we reject H_0 in favor to H_1.

Due to stochastic nature of optimisation algorithms, searches must be repeated several times in order to mitigate against the effect of random variation. How many runs do we need when we analyze and compare algorithms?

In many fields of science (i.e. medicine and behaviour science) a common rule of thumb [7] is to use at least $N = 30$ observations. We have also used in our evaluation 30 executions for each algorithm.

Remark. One of the main reasons why we have used Wilcoxon signed rank test is that the area of the study (i.e. comparing the best solutions obtained by the two algorithms) is better represented by the median, thus we compared the best chromosomes obtained by the two methods and not the mean of all the chromosomes for each execution. Another reason of using non-parametric test could be a very small sample size, but if the median better represents the centre of the distribution, the indication is to consider the nonparametric test even when we have a larger sample.

5. Experimental results

In this section, the proposed approach is evaluated and the results are reported. According to the book of Yin [5] and as stated in Section 3 we have conducted three different experiments. The first one was reported in [1], and the second one in [2]. The current paper reports the third experiment.

5.1. **Third case study. Component Selection Problem formulation.**

The third case study uses a set of 90 available components. The final system has one input data and two output data, the goal is to find a subset of the given components such that all the requirements are satisfied considering the optimisation criteria specified above. The set of requirements $SR = \{r_1,\ r_2\}$ (view as provided interfaces $\{p_1, p_2\}$) and the set of components $SC = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, ..., c_{90}\}$ are given. The final system has as input data (transformed in required interfaces) the set $\{r_3\}$.

Remark. Due to lack of space the component repository is not described in this paper but may be found at [16].

5.1.1. *Experimental studies - Case 1: Dynamically changing requirements.* As in the case of the first case study, we consider two types of dynamics and, consequently two experiments corresponding to each of them: the requirements of the problem change over time, and the components available at a certain time step change.

The algorithm was run 30 times and the number of nondominated solutions and the number of distinct nondominated solutions were recorded for all situations. Also, the cost and the number of distinct used components in a solution were logged. Also, the best, worse and average fitness values were recorded for all situations.

TABLE 1. Wilcoxon statistical test for Case Study 1 - *changing requirements* experiment.

L-T	W_-	W_+	W_{test}	N	W_{critic}	H_0	H_1	p-value
L1-T0	-46	254	46	24	81	×	✓	0.00298
L1-T1	-1	464	1	30	137	×	✓	0.00000
L1-T2	-46	389	46	29	126	×	✓	0.00020
L2-T0	-172	38	38	20	52	×	✓	0.01242
L3-T0	-269.5	55.5	55.5	25	89	×	✓	0.00560

Three different time steps are built using artificially generated data and the dynamics at each of these steps are: T0=The initial requirements, T1=Add one new requirement, T2=Add one new requirement.

Performed experiments.

The role of the performed test was to see if the number of iterations and the population size play a role in finding the Pareto solutions. The conclusions about the findings of this type of experiments are given in Section 5.1.3.

Multilevel configurations. The compound components from level 1 are constructed by applying the same algorithm but with different requirements and input data. For the Second Level of the system the set of required interfaces is $\{r_{11},\ r_{15}\}$ and the set of provided interfaces is $\{p_2\}$. For the Third Level of the system the set of required interfaces is $\{r_{15}\}$ and the set of provided interfaces is $\{p_{17},\ p_{18}\}$. The conclusions about the findings of this type of experiments are given in Section 5.1.3.

Remark. We have not presented the charts regarding the influence of population size or iteration number in finding the Pareto solutions, because we have concentrated our findings in comparing the algorithms using the Wilcoxon statistical test.

Wilcoxon statistical test.

In Section 4 we have described in details the Wilcoxon statistical test that we have used to compare our Genetic Algorithm with the Random Search Algorithm. In Table 1 the test results for the Case Study 1 - Dynamically Changing Requirements are shown.

The Wilcoxon statistical test (see Table 1) shows that we have statistically significant evidence at $\alpha = 0.05$ to show that the median is positive, i.e. the H_0 Null-Hypothesis is rejected in favor of H_1 for all levels and for all time steps.

5.1.2. *Experimental studies - Case 2: Dynamically changing components.* As in the first case study, the repository containing components changes over time.

This modification of the available components may be seen as an update of the COTS market, new components being available or other being withdrawn from the market.

Four different time steps are built using artificially generated data and the dynamics at each of these steps are: T0= The initial components, T1= Add two new components, T2= Remove one component and add one new component, T3= Remove one component.

Performed experiments.

The aim of the performed tests is the same as in the first case study: to see if the number of iterations and the population size play a role in finding the Pareto solutions. The conclusions about the findings of this type of experiments are given in Section 5.1.3.

Multilevel configurations. The compound components are next constructed by applying the same algorithm but with different requirements and input data. For the second level of the system the set of required interfaces is $\{r_{11},\ r_{15}\}$ and the set of provided interfaces is $\{p_2\}$. For the second level we have two time steps: T1= No modifications of the component repository, T2= Adding two new components.

For the third level of the system the set of required interfaces is $\{r_{15}\}$ and the set of provided interfaces is $\{p_{17},\ p_{18}\}$. The conclusions about the findings of this type of experiments are given in Section 5.1.3. For the third level we have three time steps: T1= No modifications of the component repository , T2= Adding two new components and removing two old components, T3=Removing three old components and adding one new component.

Remark. We have not presented the charts regarding the influence of population size or iteration number in finding the Pareto solutions, because we have concentrated our findings in comparing the algorithm using the Wilcoxon statistical test.

Wilcoxon statistical test.

In Section 4 we have described in details the Wilcoxon statistical test that we have use to compare our Genetic Algorithm with the Random Search Algorithm. In Table 2 we have the test results for the Case Study 2 - Dynamic Changing Components.

The Wilcoxon statistical test (see Table 2) shows that we have statistically significant evidence at $\alpha = 0.05$ to show that the median is positive, i.e. the H_0 Null-Hypothesis is rejected in favor of H_1 for all levels and for all time steps.

5.1.3. *Case Report 3.* The role of the conducted experiments had two directions: the first one was to see if the number of iterations and the population size play a role in finding the Pareto solutions. The second direction was

TABLE 2. Wilcoxon statistical test for Case Study 2 - *changing components* experiment.

L-T	W_-	W_+	W_{test}	N	W_{critic}	H_0	H_1	p-value
L1-T0	-110	325	110	29	126	×	✓	0.02034
L1-T1	-53	298	53	26	98	×	✓	0.00188
L1-T2	-65	286	65	26	98	×	✓	0.01468
L1-T3	-56	350	56	28	116	×	✓	0.00800
L2-T0	-152.5	39	39	19	46	×	✓	0.02088
L2-T1	-197	56	56	22	65	×	✓	0.02202
L3-T0	0	465	0	30	137	×	✓	0.00000
L3-T1	0	78	0	12	13	×	✓	0.00222
L3-T2	0	120	120	30	137	×	✓	0.00064

to find out how and why do Search-based Algorithms (in our case a Genetic Algorithm and a Random Search Algorithm) provide different results for the Dynamic Multilevel Component Selection Problem.

According to the experiment presented in Section 5, subsection 5.1.1 and 5.1.2, outline of the results (regarding the influence of iterations number and population size in finding the Pareto solutions) are the same as in the first case study.

According to the Wilcoxon statistical test values presented in Section 5, subsection 5.1.1 and 5.1.2, the outline of the results (regarding the different results obtained by the Genetic Algorithm and by the Random Search Algorithm) is as follows:

- We have statistically significant evidence at $\alpha = 0.05$, to show that the median difference is positive, i.e. the Null-Hypothesis is rejected, in favor to H_1.

5.2. **Summary Report.** As stated in Section 3 to generalize from a case study to theory, the analytic generalization should be used. If two or more cases are shown to support the same theory, replication [5] may be claimed. Our replication strategy used time steps for the the dynamic changing requirements (Level 1) and for the dynamic changing components (Level 1, 2, and 3).

The time steps were modified according to Table 3: the number of time steps were modified for each case study, and also the number or requirements/components modified for each time step.

In first case study we have used 4 time steps for the changing requirements experiment and 3 for the second case study, respectively 5 time steps for

TABLE 3. Replication strategy - time steps for each conducted case study

Case study 1 [1]	Case study 2 [2]	Case study 3 (current paper)
Changing requirements		
T1=init req. T2= +1R T3= -1R and +1R T4= +1R T5=	T1=init req. T2=+1R T3=+1R T4= T5=	T1=init req. T2=+1R T3=+1R T4=-1R T5=+1R
Changing components		
Level 1 T1=init comp. T2= +2C T3= -1C T4= -1C and +1C T5= +3C	Level 1 T1=init comp. T2=+2C T3=-1C and +1C T4=-1C T5=	Level 1 T1=init comp. T2=+2C T3= T4= T5=
Level 2 T1=init comp. T2= +2C T3= +3C and -1C T4=	Level 2 T1=init comp. T2=+2C T3= T4=	Level 2 T1=init comp. T2=+3C and -3C T3= +2C T4= +1C and -1C
Level 3 T1=init comp. T2= +2C T3= -2C T4= +2C and -1C	Level 3 T1=init comp. T2=+2C and -2C T3= -3C and +1 C T4=	Level 3 T1=init comp. T2=+2C and -2C T3= +1C and -1C T4= +1C and -2C

the third case study. Regarding the changing components experiment we have used different time steps for all three levels of the case studies: 3 to 5 time steps for the first case study, 2 to 4 time steps for the second case study, and 2 to 4 time steps for the third case study. Also, the number of requirements/components to be changed were considered for variation: either adding or removing 1 to 3 elements. Thus, our replications altered one or two experimental conditions considered unimportant to the original findings, to see whether the findings could still be duplicated. With such replications [5] the original finding should be considered robust.

Each case's conclusions are next considered for the summary report.

For each individual case, the report ([1, 2] and current paper, Section 5.1) indicated that the proposition from Section 5 was demonstrated, i.e. "The Search-based Algorithms (in our case a Genetic Algorithm and a Random Search Algorithm) provided different results for the Dynamic Multilevel Component Selection Problem because in the case of Genetic Algorithm the fitness of a solution is reevaluated."

Regarding the research question from Section 5, i.e. "How and Why do Search-based Algorithms (in our case a Genetic Algorithm and a Random Search Algorithm) provide different results for the Dynamic Multilevel Component Selection Problem?", the conclusions sustained by the conducted case studies is that the Genetic Algorithm provides better results than the Random Search Algorithm for the Dynamic Multilevel Component Selection Problem and that we have statistically significant evidence at $\alpha = 0.05$.

6. Related work analysis and discussion

This section presents the current state of art regarding the component selection problem and analyzes the differences compared with our present approach. Component selection methods are traditionally done in an architecture-centric manner. One approach was proposed [20], where the authors present a method for simultaneously defining software architecture and selecting off-the-shelf components.

Another type of component selection approach is built around the relationship between requirements and components available for use. Paper [19] proposes a comparison between a Greedy algorithm and a Genetic Algorithm. The discussed problem considers a realistic case in which cost of components may be different. *In relation to existing component selection methods, our approach aims to achieve goals similar to* [18, 17]. The [18] approach considers selecting the component with the maximal number of provided operations. The algorithm in [17] considers all the components to be previous sorted according to their weight value. Then all components with the highest weight are included in the solution until the budget bound has been reached.

All the above approaches did not considered the multilevel structure of a component-based system. They all constructed the final solution as a one level system. Our previous research has studied the problem of multilevel component selection considering multilevel configuration [25]. The proposed evolutionary multiobjective approach provided a way of finding the "best" solution out of a set of solutions.

Various genetic algorithms representations were proposed in [26, 27]. The authors proposed an optimization model of software components selection for

CBSS development. The proposed methodology involves some subjective judgments from software development team, such as determination of the score of interactions and the function ratings. *We argue that our model differs by the fact that components interactions are computed automatically* based on required and provided component interface specification. Also, regarding the function ratings (that describe the degrees of functional contributions of the software components towards the software modules), *our approach discovers automatically the constituent components for each module of the final system.*

In [12] a hybrid approach for multi-attribute QoS optimization of component-based software systems has been proposed. The approach is able to exploit the approximated analytical Pareto front providing a larger number of solutions with a more accurate estimate of performance and availability metrics. *In relation to this existing approach, ours aims to achieve similar goals, being capable of obtaining multiple solutions in a single run and it can be scaled to any number of components and requirements.*

Another perspective refers to updating/adding/removing one/many requirements (components) from an already constructed system [24]. Our previous research regarding this perspective was proposed in [15]. How to deal efficiently with the design of systems that are able to evolve overtime and adapt to rapid changes of their requirements was investigated in [14]. They proposed some metrics definitions that are able to quantify and evaluate such software adaptability at the architectural level. *Our current approach considers dynamic modifications of the requirements of the final system, investigating different ways of modifying the requirements, by adding new requirements or deleting existing ones.*

A similar approach considering evolution of software architecture was proposed by [4]. It suggests the best actions to be taken according to a set of new requirements. They associate to the evolution of a new requirement a set of plans to be applied. The model select the best available evolution plan such that it minimizes the evolution cost under reliability and performance constraints. In relation to this approach, *our current approach also discovers the optimal solution minimizing the final cost when new requirements are needed, and it also considers the case that the component repository changes over time (that was not included in the [4] study.)*

In the course of experiments during the evaluation, a number of limitations of the dynamic multilevel component selection algorithm became apparent. First, it does not take into consideration many factors, which are effective in component selection, such as performance and reliability. However, it is possible to extend the algorithm to consider other factors. The quality factors (other than cost), mapped to some existing component-based metrics may be

included in the fitness function of the genetic algorithm. Or the values of the metrics may assess the final obtained solutions, offering the architect a range of solutions with various measurements of a quality attribute or for many quality attributes.

The second limitation refers to the NP-complete problem of the component selection problem, therefore, like other existing methods, the proposed algorithm cannot guarantee to achieve an optimal solution (using the Pareto dominance principle it might be difficult to always find solutions which are better than the ones already found). However, we have introduced a supplementary condition for comparing two solutions which are nondominated among them (will prefer the one for which the aggregation of all of the objectives values lower).

7. Conclusion

The contribution of this paper is the use of the case study method and the research design from the book of Yin [5] to validate our research proposal for the **Dynamic Multilevel Component Selection Problem** [2].

We have conducted three different experiments. The first one was reported in [1], and the second one in [2]. The current paper **reports the third experiment**. Following the replication approach to multiple-case studies [5], each individual case study was finalized by an individual case report that will be next considered to be part of a summary report, i.e. a cross-case conclusions. Thus, in our case the results obtained are reported and conclusions about the potential of evolutionary algorithms for the dynamic multiobjective multilevel component selection problem are drawn.

The Wilcoxon statistical test was used to compare our Genetic Algorithm approach with a Random Search Algorithm: we have statistically significant evidence at $\alpha = 0.05$ to show that the median is positive, i.e. we obtain better results with our approach. The tests performed show the potential of evolutionary algorithms for this particular problem and for other similar ones.

Work-in-progress is devoted to improve the proposed approach in various ways. First, we are integrating the approach with the approach from [13]: to use fuzzy clustering to group similar components in order to select the best candidate.

Ongoing work focuses on the integration of metrics to asses the quality of the obtained solutions. We plan to integrate the evaluation of the final level configurations (an approach that is published in [3]) in the current approach by computing metrics values and assess and select the best solution architecture.

References

[1] A. Vescan, *An Evolutionary Multiobjective Approach for the Dynamic Multilevel Component Selection Problem*, The First International Workshop on Big Data Services and Computational Intelligence, in conjunction with ICSOC, 193–204, 2016.

[2] A. Vescan, *Case Study Method and Research Design for the Dynamic Multilevel Component Selection Problem*, The First International Workshop on Big Data Services and Computational Intelligence, in conjunction with ICSOC, 130–141, 2016.

[3] A. Vescan, C. Serban, *Multilevel component selection optimisation towards an optimal architecture*, Soft Computing Journal (accepted December 2016).

[4] V. Cortellessa and R. Mirandola and P. Potena *Managing the evolution of a software architecture at minimal cost under performance and reliability constraints*, Science of Computer Programming, no. 98, pp. 439–463, 2015.

[5] Robert K. Yin *Case Study Research: Design and Methods*, SAGE Publications, 2009.

[6] J. Derrac and S. Garcia and D. Molina and F. Herrera, *A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms*, Swarm and Evolutionary Computation, no. 1, pp. 3–18, 2011.

[7] Arcuri and L. Briand, *A practical guide for using statistical tests to assess randomized algorithms in software engineering*, The 33rd International Conference on Software Engineering, 1–10, 2011.

[8] M. Harman and P. McMinn and J. Teixeira de Souza and S. Yoo *Search Based Software Engineering: Techniques, Taxonomy, Tutorial*, Empirical Software Engineering and Verification, no. 7007, pp. 1–59, 2012.

[9] L. Iribarne and J.M. Troya and A. Vallecillo, *Selecting Software Components with Multiple Interfaces*, The 28th EUROMICRO Conference Component-Based Software Engineering, 26–32, 2002.

[10] Christoph Becker and Andreas Rauber *Improving component selection and monitoring with controlled experimentation and automated measurements*, Information and Software Technology, no. 6, pp. 641–655, 2010.

[11] M. A. Khan and S. Mahmood *A graph-based requirements clustering approach for component selection*, Adv. Eng. Software, no. 54, pp. 1–16, 2012.

[12] A. Martens and R.Mirandola and D. Ardagna and R. Reussner and H. Koziolek, *A Hybrid Approach for Multi-Attribute QoS Optimisation in Component Based Software Systems*, Proc. of the QoSA, 84–101, 2010.

[13] A. Vescan, C. Grosan, *A new Component Selection Algorithm Based on Metrics and Fuzzy Clustering Analysis*, Proceedings of the 4th International Conference on Hybrid Artificial Intelligence Systems, pp. 621–628, 2009.

[14] D. P. Palacin and R. Mirandola and J. Merseguer *Software Architecture Adaptability Metrics for QoS-based Self-Adaptation*, Proc. of the QoSA, pp. 171–176, 2011.

[15] A. Vescan .and Grosan, C. and Shengxiang Yang *A hybrid evolutionary multiobjective approach for the dynamic component selection problem* , Proc. of the 11th International Conference on Hybrid Intelligent Systems (HIS), pp. 714–721, 2011.

[16] A. Vescan and C. Serban *etails on case study for the dynamic multilevel component selection optimisation approach*, http://www.cs.ubbcluj.ro/˜avescan/?q=node/178, 2016.

[17] P. Baker and M. Harman and K. Steinhofel and A. Skaliotis *Search Based Approaches to Component Selection and Prioritization for the Next Release Problem*, Software Maintenance, The 22nd IEEE International Conference on, pp. 176–185, 2006.

[18] M. R. Fox and D. C. Brogan and P. F. Reynolds *Approximating Component Selection*, Software Maintenance, Proceedings of the 36th Conference on Winter Simulation, pp. 429–434, 2004.

[19] N. Haghpanah, S. Moaven, J. Habibi, M. Kargar, S. H. Yeganeh, *Approximation Algorithms for Software Component Selection Problem*, APSEC conference, pp. 159–166, 2007.

[20] E. Mancebo, A. Andrews, *A strategy for selecting multiple components*, SAC '05: Proceedings of the 2005 ACM symposium on Applied computing, pp. 1505–1510, 2005.

[21] A. Abraham and L. Jain and R. Goldberg *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*, Springer Verlag, 2005.

[22] C. Grosan *A comparison of several evolutionary models and representations for multiobjective optimization*, ISE Book Series on Real Word Multi-Objective System Engineering, chapter 3, Nova Science, 2005.

[23] I. Crnkovic, M. Larsson, *Building Reliable Component-Based Software Systems*, Artech House publisher, 2002.

[24] L. Wei *QoS Assurance for Dynamic Reconfiguration of Component-Based Software Systems*, IEEE Transactions on Software Engineering, no. 38(3), pp. 658–676, 2012.

[25] A. Vescan and C. Grosan *Evolutionary multiobjective approach for multilevel component composition*, Studia Univ. Babes-Bolyai, Informatica,

no. LV(4), pp. 18–32, 2010.

[26] C.K. Kwong and L.F. Mu and J.F. Tang and X.G. Luo *Optimization of software components selection for component-based software system development*, Computers and Industrial Engineering, no. 58(1), pp. 618–624, 2010.

[27] P.C. Jhaa and V. Balib and S. Narulaa and M. Kalra *Optimal component selection based on cohesion and coupling for component based software system under build-or-buy scheme*, Journal of Computational Science, no. 5(2), pp. 233–242, 2014.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: avescan@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 1, 2017
DOI: 10.24193/subbi.2017.1.03

A FILTER-BASED DYNAMIC RESOURCE MANAGEMENT FRAMEWORK FOR VIRTUALIZED DATA CENTERS

CORA CRĂCIUN AND IOAN SALOMIE

Abstract. Data centers adapt their operation to changing run-time conditions using energy-aware and SLA-compliant resource management techniques. In this context, current paper presents a novel filter-based dynamic resource management framework for virtualized data centers. By choosing and combining properly software filters performing the scheduling and resource management operations, the framework may be used in what-if analysis. The framework is evaluated by simulation for deploying batch best-effort jobs with time-varying CPU requirements.

1. Introduction

High energy consumption and low Quality of service (QoS) are the main problems in data centers. Service providers aim to reduce the energy consumption, while the users demand high performance at low cost. Moreover, data centers are dynamic systems in which the users' requests and resource availability are time-varying. Therefore, data centers must adapt their operation to run-time conditions using appropriate scheduling and resource management strategies [22].

In this context, we present a novel filter-based dynamic resource management framework for virtualized data centers. The framework extends the Haizea lease scheduler (version 1.1) [15, 24, 26] and may assess the energy and performance efficiency of different resource management techniques. The framework uses software filters to perform the job scheduling, resource allocation, and virtual machines' migration operations in data centers. A data

Received by the editors: February 1, 2017.

2010 *Mathematics Subject Classification.* 68M14, 68M20.

1998 *CR Categories and Descriptors.* C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *Network operating systems*; C.4 [**Computer Systems Organization**]: Performance of Systems – *Design studies*.

Key words and phrases. Filter-based framework, Dynamic resource management, Virtualization.

center may adapt to run-time conditions by properly replacing and combining the filters. Current work uses CPU-related filters, but similar software components may be defined for other physical resources.

Presently, the framework has some limitations. For instance, it does not take into account the performance degradation when more virtual machines (VMs) are migrated simultaneously from or to the same physical machine. In addition, the framework considers the following simplified resource management scenarios and data center configurations: resource allocation policies with job queuing in which the jobs wait until they receive all required resources, single-core processing units, overloaded but not underused host management, small size homogeneous data centers, a single constraining physical resource (CPU), simultaneously arrived jobs at the data center, periodic change of the VMs' resource requirements, off-line VM migration as implemented in Haizea. The framework, however, is extensible with other filters, algorithms, or resource allocation policies.

The paper is structured as follows. Next section presents other investigations related to our dynamic resource management approach. Sect. 3 presents the filter-based framework and the scheduling, resource allocation, and host management filters. The framework is evaluated by simulation in Sect. 4, for different resource allocation policies including First Fit (FF), Best Fit (BF) and their decreasing forms, and the Gaussian-type policies defined in reference [7]. Final section summarizes current work.

2. Related work

Different resource management frameworks for clusters, grids, or cloud environments have been developed in the last years. Many of them use virtualization for dynamic resource provisioning. pMapper is a framework for virtualized heterogeneous clusters, which minimizes the electrical power and the VMs migration costs with performance guarantees [27]. The framework uses different VM-to-host mapping algorithms based on variants of the FFD heuristic. Unlike pMapper, our framework has been evaluated only for best-effort VMs, and for this case, it does not offer performance guarantees. Entropy consolidates the VMs in homogeneous clusters using constraint programming [16]. Compared to our work, this framework considers both CPU and memory as constraining resources, manages not only the overloaded but also the underused hosts, and has been evaluated in real environments. The framework presented in current paper, on the other hand, provides scheduling facilities, by means of Haizea, and queuing options. The GREEN-NET framework reduces the energy consumption in large scale distributed systems by switching

off the unused physical resources, by aggregating the reservations, or by predicting next reservations based on history [6]. OpenNebula is an open-source virtual infrastructure manager for private or public IaaS clouds [20, 24]. Two extensions of this framework address issues such as energy consumption (the Green Cloud Scheduler [14]) or advance reservation (the Haizea scheduler [15]) in data centers. Haizea is a resource manager used either as a simulator or as a backend VM scheduler for OpenNebula [15, 23, 24, 25, 26]. Haizea defines different types of leases implemented as virtual machines [26], uses off-line VMs migration, and considers the VM management time overheads. Our filter-based framework extends Haizea with new VM scheduling, resource allocation, and host management policies, and computes energy and performance-related quantities for virtualized data centers. CloudSim is an event-based simulation toolkit for private and hybrid virtualized cloud systems [4, 5]. The Cloudsim's core has been extended with a power package [1, 2, 3] and an interconnection network model of a cloud data center [11]. The power package contains different energy-aware VM placement algorithms and uses power models for specific server architectures. Our modular approach to solving the resource management problem in data centers is close to the work presented in [2] using CloudSim. OpenStack is a cloud operating system providing virtualized computing resources to the users [21]. The resource allocation method presented in current paper has similarities with the OpenStack's filtering procedure. This procedure selects the eligible hosts with the largest weighted cost scores for allocating the VMs.

3. The filter-based framework

3.1. **Framework design.** The filter-based dynamic resource management framework (Fig. 1) uses software filters to perform the VM scheduling, resource allocation, and host management activities. We assume that a virtual machine has already been provisioned to each job arrived at the data center. A job is a request addressed to a batch application or a workflow to a web service. All jobs arrive at the data center simultaneously. The jobs' CPU requirements are time-varying and not known in advance. The job-to-VM mapping is one-to-one and the resource requirements of the VM and the job coincide. The virtualized jobs have been assimilated to the Haizea's best-effort leases. Henceforth, we mainly refer to the virtual machines instead of jobs.

A virtual machine is mapped on a physical machine if it receives all resources required at mapping time. The CPU capacity needed for a specified duration is the single constraining resource for the VM-to-host mapping. More VMs may simultaneously share a physical machine if their cumulated resource

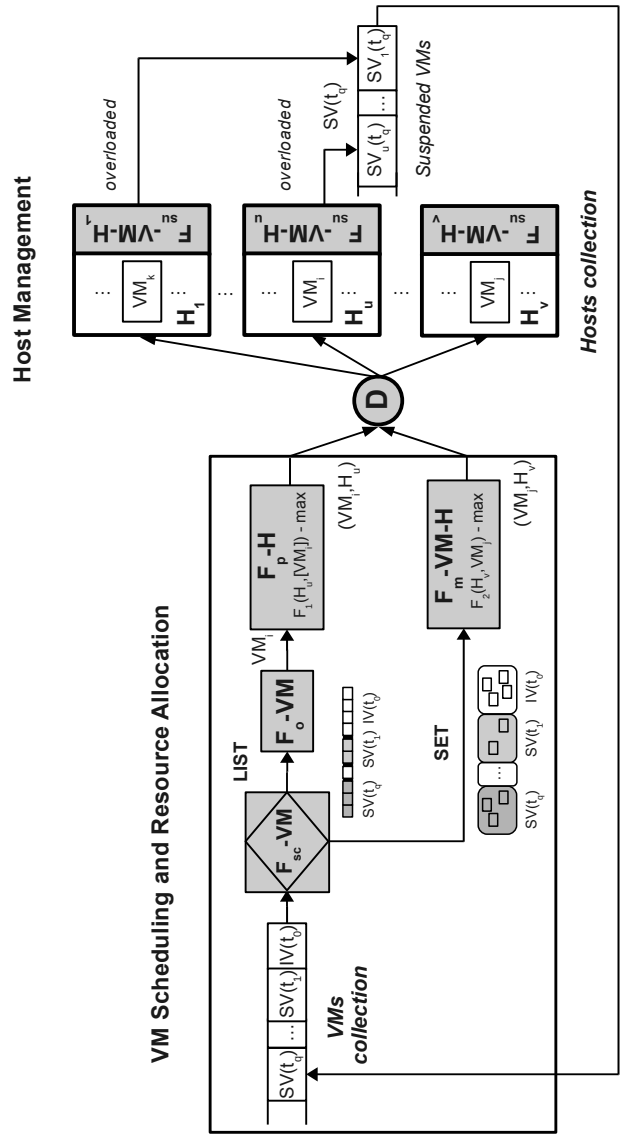


FIGURE 1. The filter-based dynamic resource management framework. Notations: $IV(t_0)$ - the subcollection of initial VMs; $SV(t_1), \ldots, SV(t_q)$ - the subcollections of suspended VMs; H_1, H_u, H_v - physical machines (hosts); VM_k, VM_i, VM_j - virtual machines; $\mathbf{F_{sc}}$**-VM** - the VM scheduling filter, $\mathbf{F_o}$**-VM** - the VM ordering filter, $\mathbf{F_p}$**-H** - the host provisioning filter, $\mathbf{F_m}$**-VM-H** - the VM-to-host mapping filter, $\mathbf{F_{su}}$**-VM-H$_u$** - the VM suspending filter for host H_u; F_1 and F_2 - resource allocation functions; **D** - dispatcher

requirements do not exceed the available resources. At run-time, selected VMs from the overloaded hosts are suspended for rescheduling.

The framework gathers all virtual resources in a *VMs* collection, $\{VM_j\}_{j=1,\ldots,N_V}$, and the physical resources in a *Hosts* collection, $\{H_u\}_{u=1,\ldots,N_H}$ (Fig. 1). The individual VMs and hosts have unique identifiers (ID). The *VMs* collection is structured in time-labeled subcollections, which are scheduled in the increasing order of their time label. The first subcollection, $IV(t_0)$, contains the VMs provisioned to the jobs arrived at the data center at the initial time t_0. Next subcollections, $SV(t_q)$ $(q = 1, 2, \ldots)$, contain the VMs suspended at times t_q from all overloaded hosts.

The filters (Fig. 1) perform constraint or policy-based operations. The VM scheduling filter, $\mathbf{F_{sc}}$**-VM**, decides whether the VM subcollections are represented as lists or as sets. The VM ordering filter, $\mathbf{F_o}$**-VM**, optionally sorts the VM lists by a specified criterion. The difference between the VM lists and VM sets is that the VM lists are already ordered at scheduling time, while the VM sets are not. The VMs from VM lists are scheduled, if possible, in their queuing order. The VMs from VM sets, on the other hand, are mapped on hosts in an order depending on the required and the available resources. The host provisioning filter, $\mathbf{F_p}$**-H**, allocates resources to the VMs from VM lists, and the VM-to-host mapping filter, $\mathbf{F_m}$**-VM-H**, allocates resources to the VMs from VM sets. The $\mathbf{F_p}$**-H** filter selects the destination host for an already chosen VM, by maximizing a resource allocation function F_1 depending only on the host's usage. The $\mathbf{F_m}$**-VM-H** filter, on the other hand, selects the $(VM, host)$ pairs from a pool of VMs and a pool of hosts, such that to maximize a resource allocation function F_2. This two-dimensional function depends both on the VMs' required and the hosts' available resources. Finally, the VM suspending filter, $\mathbf{F_{su}}$**-VM-H**$_\mathbf{u}$, suspends VMs from the overloaded host H_u.

The framework executes the resource management activities in a repeated way. The timing for framework operation is presented in Table 1. Next sections describe the main activities of the filter-based framework.

3.2. **VMs scheduling.** At each scheduling time τ_s (Table 1), the framework tries to schedule the virtual machines present in the *VMs* collection. The scheduling policy is different for VM lists and VM sets.

3.2.1. *Scheduling of VM lists.* The VM lists are optionally sorted by the VM ordering filter $\mathbf{F_o}$**-VM** (Fig. 1). Examples of sorting the lists decreasingly by the VMs' CPU request are presented in the framework evaluation section. If the initial VM list $IV(t_0)$ is ordered, then its VMs having the same property according to the ordering criterion are sorted increasingly by their ID. If the

TABLE 1. The timing for framework operation

Time	Description
t_0	the arrival time for all jobs
$\{t_p\}_{p=1,2,...}$	the times when the VMs' CPU requirements are changed (periodic with time period T_0: $t_p = t_0 + pT_0$) and when the host management is performed
$t_d = nT_0$, $n \in \mathbb{Z}^+$	the CPU time required by each VM
$\{t_q\}_{q=1,2,...}$	the times when overloaded hosts are detected; $\{t_q\}_{q=1,2,...}$ is a subset of $\{t_p\}_{p=1,2,...}$
$\{\tau_s\}_{s=1,2,...}$	the VM scheduling times (periodic with time period $T_0/2$ or triggered by any event[1] in data center); include $\{t_p\}_{p=1,2,...}$

[1] Examples of events: VMs' suspension, migration, resumption, completion

suspended VM lists are ordered, on the other hand, then the ties are broken by sorting the VMs increasingly, first by their last host ID and then by the VMs' own ID value, in case of common host.

The framework uses two scheduling policies for VM lists, *FLIST* and *NLIST*. These policies behave differently when the next VM in the *VMs* collection can not be scheduled due to lack of physical resources. At each scheduling time τ_s, the framework using the *FLIST* policy schedules the VMs one by one, until it encounters a VM that can not be scheduled. In this case, the *FLIST* policy postpones the entire scheduling process for the next scheduling time. *FLIST* resembles the First-Come First-Served (FCFS) scheduling policy. However, for *FLIST*, the VMs from the same subcollection have a common time label and are optionally ordered, while FCFS considers the VMs in the order of their arrival time. The *FLIST* policy is useful when strict ordering is compulsory, for example when the VMs encapsulate the web services of a business process. For unrelated VMs, *FLIST* has the same drawbacks as FCFS: (a) the VMs with low resource requirements may be delayed by VMs with high requirements and (b) the VMs that need more processors may cause resource fragmentation [8, 18].

Unlike *FLIST*, when the next VM can not be scheduled, the *NLIST* policy tries to schedule the other VMs from the same subcollection as the first one. The process continues until the subcollection is completely scheduled and then is repeated for the next subcollections. The *NLIST* policy uses a list scanning procedure that resembles the List Scheduling algorithms for independent tasks, with no imposed partial ordering [13]. Nevertheless, in case of *NLIST*, the

VMs' ordering at subcollection level does not depend on time as it does for List Scheduling.

3.2.2. *Scheduling of VM sets.* At scheduling level, no ordering is imposed on VM sets. The VM-to-host mapping module presented in the next section decides the VMs' ordering in this case. As for VM lists, the VM sets are scheduled completely in the increasing order of their time label. The VM sets may model, for example, bag-of-tasks applications with independent tasks executed in parallel.

3.3. **Resource allocation.** The framework uses different policies (presented in Sect. 3.3.2) to provision physical resources to the virtual machines. For VM lists, the scheduling and ordering filters have already decided the order in which the VMs are mapped on hosts. On contrary, for VM sets, the order depends both on the VMs' requirements and the hosts' available resources, and is decided by the resource allocation filters.

3.3.1. *Resource allocation filters.* The host provisioning filter, $\mathbf{F_p}$**-H** (Fig. 1), allocates physical resources to the VMs from VM lists. This filter maximizes a resource allocation function $F_1(H_v, [VM_i])$, $v = 1, \ldots, N_H$, to find the destination host H_v for an already selected virtual machine VM_i. The VM-to-host mapping filter, $\mathbf{F_m}$**-VM-H**, on the other hand, provisions physical resources to the VMs from VM sets. For each VM set, the mapping filter chooses iteratively the $(VM, host)$ pairs that maximize a resource allocation function $F_2(H_v, VM_j)$, $v = 1, \ldots, N_H$ and $j = 1, \ldots, n$ (n is the set size). The searching process is exhaustive and finds a global solution to the optimization problem.

3.3.2. *Resource allocation policies.* This section presents the resource allocation policies currently used by the framework: (a) the reference First Fit and Best Fit policies and their decreasing forms (FFD and BFD), and (b) the Gaussian-type policies G1 and G2 defined in [7]. All policies may be used for VM lists. A variant of the BFD policy is also defined for VM sets and the G2 policy is mainly used for this type of VM subcollections.

Let denote by R_{CPU} a VM's required CPU share and by T_{CPU}, U_{CPU}, and A_{CPU} a host's total, used, and respectively available CPU resources ($T_{\text{CPU}} = U_{\text{CPU}} + A_{\text{CPU}}$). All hosts are identical and have a unique processor with $T_{\text{CPU}} = 100\%$ (the CPU quantities are considered in percents, which is numerically more efficient). A VM may be assigned only to a feasible host (a host with $A_{\text{CPU}} \geq R_{\text{CPU}}$) and only if it receives all required resources. If no such host exists, the VM remains in the waiting queue until a feasible host is found.

The First Fit policy assigns each VM from a list to the lowest-indexed feasible physical machine from the *Hosts* collection [12]. This policy uses an F_1-type resource allocation function, which takes the constant value 1 for any feasible host and the value 0 for the other hosts. The FFD policy uses the same resource allocation function as FF, but the VM lists are sorted decreasingly by the VMs' CPU request. The VM ordering filter, $\mathbf{F_o}$**-VM**, performs this sorting operation at scheduling level, by maximizing a function equal to R_{CPU}.

The Best Fit policy maps each VM from a list to the host with the minimal remained unused resources after allocation [12]. Since the current VM to be mapped on hosts has been chosen at scheduling time (its R_{CPU} value is known), we define the F_1-type BF resource allocation function as $1/A_{\text{CPU}}$ for the feasible hosts and 0 for the other hosts. The BFD and BF policies use the same resource allocation function for VM lists, but the lists are additionally sorted decreasingly for BFD. At its turn, the BFD policy for VM sets uses an F_2-type resource allocation function, which is $R_{\text{CPU}}/A_{\text{CPU}}$ for the feasible hosts and 0 for the other hosts. This function resembles the weight factor defined for some FFD and BFD-type heuristics proposed in the multi-dimensional Vector Bin Packing context [10, 17].

Two Gaussian-type resource allocation policies, G1 and G2, have been defined in reference [7]. These policies consolidate the VMs on physical resources in a less greedy way than FF and BF, but more tightly than load balancing methods. The G1 resource allocation policy is used by the host provisioning filter $\mathbf{F_p}$**-H** (Fig. 1) for VM lists. A given VM is assigned to the feasible host which maximizes the $G_1(U_{\text{CPU}})$ Gaussian function. This function depends only on the host usage and has adjustable location and width [7]. On contrary, the G2 policy may be used either by the host provisioning filter $\mathbf{F_p}$**-H**, for VM lists, or by the VM-to-host mapping filter $\mathbf{F_m}$**-VM-H**, for VM sets. In case of VM lists, the G2 policy finds the destination host for already selected VMs, by maximizing the F_1-type function $G_2([R_{\text{CPU}}], A_{\text{CPU}})$, with fixed R_{CPU} and variable A_{CPU}. In case of VM sets, the G2 policy selects the feasible $(VM, host)$ pairs that maximize the F_2-type two-variable function $G_2(R_{\text{CPU}}, A_{\text{CPU}})$ [7].

3.4. **Host management.** The resource requirements of the VMs deployed on physical resources are time-varying. Therefore, at run-time, some physical machines may be underused and others overloaded. VMs migrations have proved efficient for host management in both cases [2]. Currently, the framework considers only the case of overloaded hosts; the case of underused hosts remains as future work. A host is overloaded when the total CPU requirements of its VMs exceed the host's CPU resources ($U_{\text{CPU}} > T_{\text{CPU}}$). In real conditions,

however, the lower limit for hosts' overloading may be some percent from the total CPU capacity, such as $80\%T_{\text{CPU}}$.

The framework performs the host management at each time t_p (Table 1), when the VMs' CPU requirements are changed. The hosts are verified for possible overloading in increasing order of their ID. Then, the VM suspending filters (Fig. 1) select the VMs to be suspended from each identified overloaded host. For example, if the host H_u is overloaded at time t_q (Table 1), then the $\mathbf{F_{su}}$**-VM-H**$_u$ filter suspends a subset $SV_u(t_q)$ from its VMs. The VMs suspended from all overloaded hosts are collected into the $SV(t_q)$ subcollection. This subcollection is then appended to the *VMs* collection. When the suspended VMs are rescheduled, they are either resumed on the same hosts or are migrated to other hosts, depending on the result of the resource allocation process.

Haizea, the underlying scheduler of the filter-based framework, uses "cold" (off-line) VM migration. The VMs to be relocated are first suspended on the initial hosts, then migrated, and finally resumed on the new hosts. The applications encapsulated in the migrating VMs are completely stopped and restarted at the new location. The migration time is calculated as the ratio between the size of the VM's memory image and the network bandwidth.

The VM suspending filters may use different policies. In our previous work [7], we have evaluated the policy suspending the VMs with the lowest CPU requests from the overloaded hosts. This policy was combined with the FF, BF, and Gaussian-type resource allocation methods. Here we test two other policies, denoted by H and L. The H-policy suspends the VMs of an overloaded host in the decreasing order of their CPU required share. This policy reduces the number of VMs migrations, but the migrated VMs have high resource requirements at the destination hosts. Algorithms migrating VMs with high resource requirements or with high values of some metrics have been presented for example in references [2, 28, 29]. At its turn, the L-policy suspends some VMs with low CPU requirements, but not necessarily the lowest. To our knowledge, the L-policy has not been previously used in this form, but related variants exist in the literature, for example the iFFD algorithm in [27] or the HPG algorithm in [2].

The steps of the host management process at any time t_p, $p = 1, 2, \ldots$, are presented in Algorithm 1. As in reference [2], for both VM suspending policies, the algorithm tries first to suspend a single VM from each overloaded host (Algorithm 1, line 5), in order to minimize the VM migration number. Only the feasible VMs (the VMs not finishing their work in the next time period T_0) are potential candidates for suspension (line 4). The H-policy selects the VM with the highest CPU request and the L-policy the VM with

the lowest one, but greater than the overload. In each case, the ties are broken by selecting the VM with the smallest ID.

If a single VM is not able to eliminate the host overload (SV_u at line 5 in Algorithm 1 is the emptyset), then more VMs are suspended from that host. The H-policy sorts the host's VMs decreasingly by their CPU request (line 8, with $SUSP = H$) and selects as many VMs as necessary to eliminate the overload. On contrary, the L-policy sorts the VMs increasingly by their CPU request (line 8, with $SUSP = L$) and then uses two steps for VMs suspension. First, the policy selects the VMs in order until their cumulated CPU request exceeds the host's overload (lines 9–16). This means that by removing the selected VMs, A_{CPU} becomes greater than or equal to zero. Second, the list of selected VMs is scanned backwards and the VMs which are still not causing the host overload are restored (lines 17–28). The VMs suspended from each overloaded host are appended to the SV list (line 30). Finally, this list is sorted (lines 33–37) as explained in Sect. 3.2.1.

3.5. **Resource management compound filters.** A compound filter is a chain of filters able to perform all resource management operations for VMs deployment on physical resources. The compound filters evaluated in this paper are presented in Table 2. Their VM ordering, host provisioning, and VM-to-host mapping filters maximize the indicated objective functions, while the VM scheduling and VM suspending filters use the specified policies. The FF, BF, G1, and G2 compound filters for VM lists assign the VMs to the hosts by using the host provisioning filter $\mathbf{F_p}$**-H**. The FFD and BFD compound filters additionally sort the VM lists before resource allocation, with the VM ordering filter $\mathbf{F_o}$**-VM**. The BFD and G2 compound filters for VM sets allocate physical resources using the VM-to-host mapping filter $\mathbf{F_m}$**-VM-H**.

4. Framework evaluation

In this section, we present the results of evaluating the framework by simulation, for a set of batch jobs arrived simultaneously at the data center. A virtual machine was provisioned to each job. Simulation experiments consisted in processing 40 VMs with time-varying CPU requirements in two environments: one with sufficient physical resources (20 hosts) and the other with insufficient resources (8 hosts). In the (40VM,20H) case, the *NLIST* and *FLIST* scheduling policies were equivalent.

A trace of CPU requests lasting for $t_d = 500$ min was generated for each VM. The CPU requirements of the active VMs were changed periodically based on their trace, at times $t_p = t_0 + pT_0$, for $p = 1, 2, \ldots$ and $T_0 = 2$ min (Table 1). The active VMs were the ones deployed on physical machines and not waiting in queue for free resources. The VMs' CPU required shares (in percents) were

Algorithm 1 Management of overloaded hosts

Input: *Hosts* - the host collection, $\{H_u\}_{u=1,\ldots,N_H}$
VMs - the VM collection, $\{VM_j\}_{j=1,\ldots,N_V}$
ORDER - ordering option (Decreasing, None) for $\mathbf{F_o}$**-VM**
SUSP - suspending policy (L, H) for $\mathbf{F_{su}}$**-VM-H**
Output: *SV* - suspended VMs from all overloaded hosts

$SV \leftarrow \emptyset$
for all $H_u \in Hosts$ **do**
  if H_u.availableCPU < 0 **then**
    $AV_u \leftarrow H_u$.feasibleActiveVMs
    $SV_u \leftarrow$ choseOneVM(AV_u, *SUSP*)
    if $SV_u = \emptyset$ **then**
      sortIncreasingByVmID(AV_u)
      sortBySuspensionOption(AV_u,*SUSP*)
      for all $VM_i \in AV_u$ **do**
        if H_u.availableCPU < 0 **then**
          appendVM(SV_u, VM_i)
          H_u.availableCPU $\leftarrow H_u$.availableCPU $+ VM_i$.requiredCPU
        else
          break
        end if
      end for
      if $SUSP = L$ **then**
        $i \leftarrow SV_u$.size
        while $i \geq 1$ **do**
          $VM_i \leftarrow$ getVmByIndex(SV_u,i)
          $temp \leftarrow H_u$.availableCPU $- VM_i$.requiredCPU
          if $temp \geq 0$ **then**
            H_u.availableCPU $\leftarrow temp$
            removeVM(SV_u, VM_i)
          end if
          $i \leftarrow i - 1$
        end while
      end if
    end if
    appendVmList(SV,SV_u)
  end if
end for
sortIncreasingByVmID(SV)
sortIncreasingByHostID(SV)
if *ORDER* = Decreasing **then**
  sortDecreasingByVmRequiredCPU(SV)
end if
return SV

TABLE 2. Resource management compound filters[1]

Compound filter \ Filter	Scheduling		Resource allocation		Host management
	F_{sc}-VM	F_o-VM	F_m-VM-H	F_p-H	F_{su}-VM-H
FF-fZ	*FLIST*	-	-	1	Z = *L* or *H*
FF-nZ	*NLIST*	-	-	1	Z = *L* or *H*
FFD-fZ	*FLIST*	R_{CPU}	-	1	Z = *L* or *H*
FFD-nZ	*NLIST*	R_{CPU}	-	1	Z = *L* or *H*
BF-fZ	*FLIST*	-	-	$1/A_{CPU}$	Z = *L* or *H*
BF-nZ	*NLIST*	-	-	$1/A_{CPU}$	Z = *L* or *H*
BFD-fZ	*FLIST*	R_{CPU}	-	$1/A_{CPU}$	Z = *L* or *H*
BFD-nZ	*NLIST*	R_{CPU}	-	$1/A_{CPU}$	Z = *L* or *H*
BFD-sZ	*SET*	-	R_{CPU}/A_{CPU}	-	Z = *L* or *H*
G1-fZ	*FLIST*	-	-	G_1	Z = *L* or *H*
G1-nZ	*NLIST*	-	-	G_1	Z = *L* or *H*
G2-fZ	*FLIST*	-	-	G_2	Z = *L* or *H*
G2-nZ	*NLIST*	-	-	G_2	Z = *L* or *H*
G2-sZ	*SET*	-	G_2	-	Z = *L* or *H*

[1] FF - First Fit, FFD - First Fit Decreasing, BF - Best Fit, BFD - Best Fit Decreasing, G1 and G2 - Gaussian-type filters (the G1 filter uses the G_1 resource allocation function and G2 uses G_2); F_{sc}-VM - the VM scheduling filter (uses the *FLIST*, *NLIST*, or *SET* policy), F_o-VM - the VM ordering filter (optionally orders the VMs decreasingly by their R_{CPU} value), F_m-VM-H - the VM-to-host mapping filter (uses an F_2-type resource allocation function), F_p-H - the host provisioning filter (uses an F_1-type resource allocation function), F_{su}-VM-H - the VM suspending filter (uses the *L* or *H* policy); R_{CPU} - the VM CPU request, A_{CPU} - the host available CPU.

random numbers generated uniformly between 10 and 40 and then rounded up to the closest integer value. This range of values ensured some variation among the VMs' CPU traces and favored the hosts' overloading, of interest for our study. In the migration process, the VM memory image was off-line migrated, with no disk image transfer. The VMs were suspended by saving their memory state on the filesystem at a rate of 32 MB/s [25, 15]. The VMs' resumptions were performed at the same rate. The suspension and resumption needed 4 s each and the copy of the VM memory image other 11 s. In simulations, the same total delay of 19 s was considered for all suspended VMs, either migrated or later resumed on the same hosts. The Haizea's restrictions have been relaxed, more VMs being allowed to migrate simultaneously from or to the same physical machine, with no performance overhead. Simulation experiments have been repeated 100 times. All compound filters used identical environment conditions and CPU traces within the same experiment.

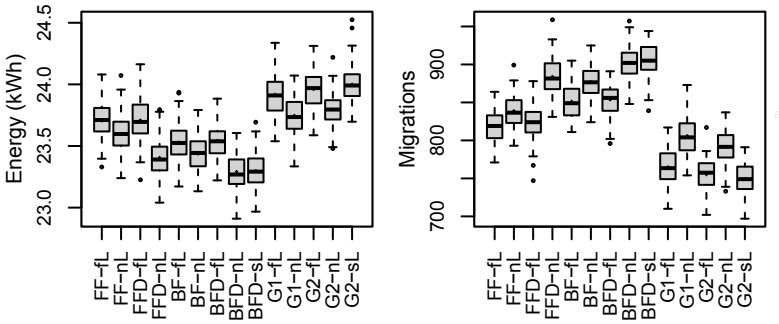


FIGURE 2. Boxplot representation of the consumed energy and the VM migration number for the (40VM,8H) configuration, *L*-suspending policy, and *FLIST*, *NLIST* and *SET* scheduling policies, in 100 simulation experiments.

The resource management policies presented in this paper were compared using the following metrics: the energy consumed by the physical machines, the VMs' total flow time (the sum of all VMs' processing times), the number of VMs migrations, and the mean number of active hosts for the entire makespan. The VMs migrations and the VMs suspensions with resumption on the same hosts were counted independently. We considered that the total electrical power of each physical machine depended linearly on its CPU usage [9]. Moreover, the idle power represented 70% of the total power of 250 W at full CPU utilization [1]. We assumed that the hosts were switched off when they became idle. In all experiments, the G_1 resource allocation function used the parameters $Thr_{\mathrm{L}} = 40$, $Thr_{\mathrm{H}} = 80$, and $a = 0.8$, as defined in reference [7], and the G_2 function used the parameters $\alpha = 0.5$ and $r = 0.001$.

Simulation results are presented in Figures 2 and 3. The boxplots contain boxes from the first to the third quartile of data, whiskers extending to the most extreme data point, but not further than 1.5 times the interquartile range [19], the data's median value (the horizontal line), the mean value (the full knot, possibly overlapped by the median's line), and outliers (the open knots). Based on these results, we made the following observations:

(a) *VM scheduling policy.* In the (40VM,8H) configuration, for the same type of compound filter, the average energy consumption was slightly smaller for the *NLIST* scheduling policy than for the *FLIST* policy, but with a slightly higher average VM migration number (Fig. 2). The *SET* scheduling policy was closer in outcome either to *NLIST* (for BFD) or to *FLIST* (for G2). For example, the median energy consumption and VM migration number were

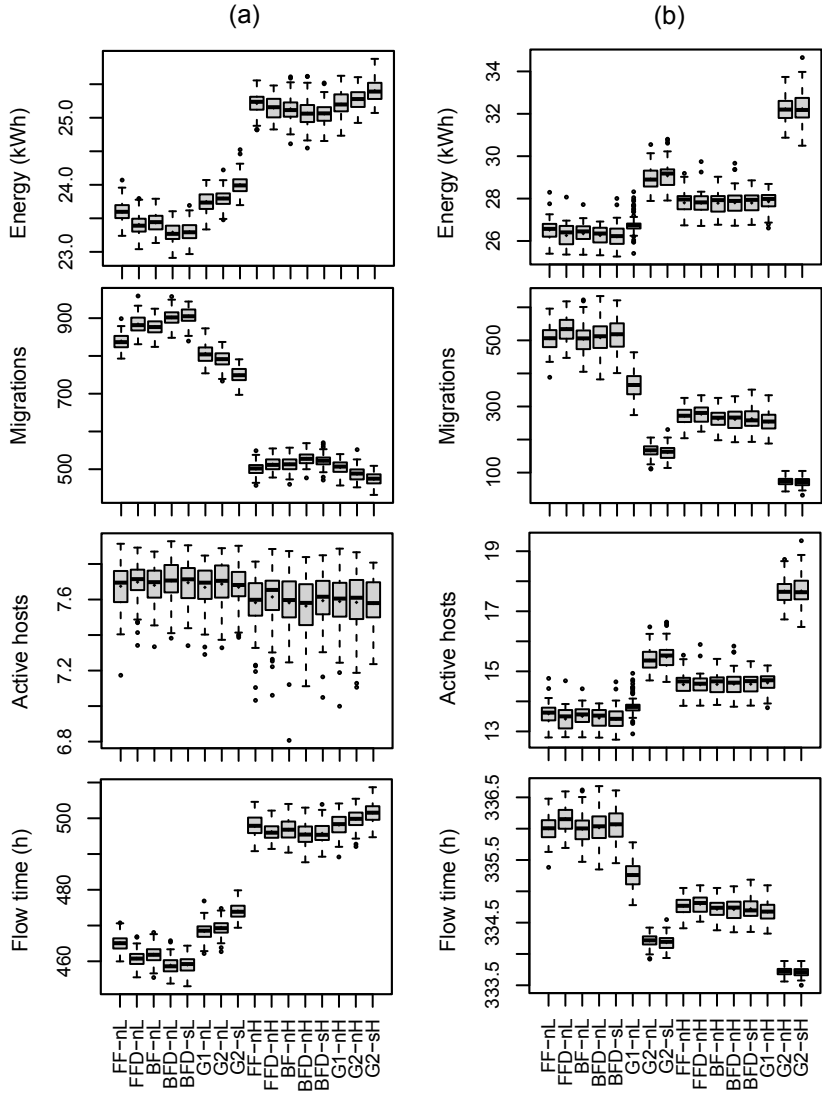


FIGURE 3. Boxplot representation of the energy and performance metrics for the (a) (40VM,8H) and (b) (40VM,20H) configurations, L and H suspending policies, and *NLIST* and *SET* scheduling policies, in 100 simulation experiments.

23.54 kWh and 856 for BFD-fL, 23.27 kWh and 902 for BFD-nL, and 23.29 kWh and 905 for BFD-sL.

(b) *Metrics.* The average VM migration number and the average energy consumption had opposite behavior. A more energy efficient compound filter was less efficient regarding the number of VMs migrations. Moreover, the relative average behavior of the compound filters was similar for the consumed energy and flow time in the (40VM,8H) configuration (Fig. 3a), but similar for the VM migration number and flow time in the (40VM,20H) configuration (Fig. 3b).

(c) *Compound filters.* The Gaussian-type compound filters were less energy-efficient than the FF, FFD, BF, and BFD compound filters, but generated a lower number of VMs migrations (Fig. 3). For example, in the (40VM,20H) configuration, the median energy consumption and VM migration number were 26.35 kWh and 513 for BFD-nL, 26.74 kWh and 365 for G1-nL, and 28.90 kWh and 168 for G2-nL.

(d) *Suspended VMs.* In the (40VM,8H) configuration, 11–14% from the total number of VMs suspensions were resumed later on the same hosts, without relocation. However, the compound filters' relative average behavior was not much affected qualitatively when considering all suspended VMs instead of only migrated ones.

(e) *VM suspending policy.* In both configurations and for all compound filters, the H-suspending policy caused a higher energy consumption and fewer VMs migrations than the L-suspending policy (Fig. 3). In the (40VM,20H) configuration, for instance, the median energy consumption and VM migration number were 26.35 kWh and 513 for BFD-nL, but 27.89 kWh and 267 for BFD-nH.

5. Conclusions

This paper has presented a filter-based dynamic resource management framework for virtualized environments. The framework uses resource management filters to perform the VM scheduling, resource allocation, and host management operations in data centers. The framework may be used to assess the energy and performance efficiency of different resource management techniques. The framework has been evaluated by simulation, for deploying virtual machines with time-varying CPU requirements, in small size environments with sufficient and insufficient physical resources. Since the resource management filters may be combined in the desired way, the framework may be included in autonomous systems and may be used in what-if analysis.

Acknowledgments

The authors would like to thank the anonymous reviewers for their suggestions and comments, which improved the paper.

References

[1] A. Beloglazov, R. Buyya, *Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers*, in Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science (MGC'10), 2010, pp. 4:1-4:6.

[2] A. Beloglazov, J. Abawajy, R. Buyya, *Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing*, Future Gener. Comput. Syst., 28 (2012), pp. 755-768.

[3] A. Beloglazov, R. Buyya, *Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers*, Concurr. Comput.: Pract. Exper., 24 (2012), pp. 1397-1420.

[4] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya, *CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*, Software: Practice and Experience, 41 (2011), pp. 23-50.

[5] CloudSim. `http://www.cloudbus.org/cloudsim/`

[6] G. Da Costa, J.-P. Gelas, Y. Georgiou, L. Lefevre, A.-C. Orgerie, J.-M. Pierson, O. Richard, K. Sharma, *The GREEN-NET framework: Energy efficiency in large scale distributed systems*, in Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing (IPDPS'09), 2009, pp. 1-8.

[7] C. Crăciun, I. Salomie, *Gaussian-type resource allocation policies for virtualized data centers*, Studia Univ. Babeş-Bolyai, Informatica, LXI(2) (2016), pp. 94-109.

[8] L. Eyraud-Dubois, G. Mounié, D. Trystram, *Analysis of scheduling algorithms with reservations*, in Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'07), 2007, pp. 1-8.

[9] X. Fan, W.-D. Weber, L. A. Barroso, *Power provisioning for a warehouse-sized computer*, in Proceedings of the 34th annual International Symposium on Computer architecture (ISCA'07), 2007, pp. 13–23.

[10] M. Gabay, S. Zaourar, *Variable size vector bin packing heuristics - Application to the machine reassignment problem*, Inria, TechReport hal-00868016 (OSP. 2013). Available online: `http://hal.archives-ouvertes.fr/hal-00868016`.

[11] S. K. Garg, R. Buyya, *NetworkCloudSim: Modelling parallel applications in cloud simulations*, in Proceedings of the 2011 4th IEEE International Conference on Utility and Cloud Computing (UCC'11), 2011, pp. 105-113.

[12] M. R. Garey, R. L. Graham, J. D. Ullman, *An analysis of some packing algorithms*, R. Rustin, ed., Combinatorial Algorithms, Algorithmics Press, New York, 1973, pp. 39-47.

[13] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Technical Journal, 45 (1966), pp. 1563-1581.

[14] Green Cloud Scheduler. `http://coned.utcluj.ro/GreenCloudScheduler/`

[15] Haizea. `http://haizea.cs.uchicago.edu/`

[16] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, J. Lawall, *Entropy: a consolidation manager for clusters*, in Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual execution environments (VEE'09), 2009, pp. 41-50.

[17] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, U. Wieder, *Validating heuristics for virtual machines consolidation*, Microsoft Research, TechReport MSR-TR-2011-9, Jan 2011. Available online: `http://research.microsoft.com/pubs/144571/virtualization.pdf`

[18] A. W. Mu'alem, D. G. Feitelson, *Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling*, IEEE Trans. Parallel Distrib. Syst., 12 (2001), pp. 529-543.
[19] The R Project for Statistical Computing. `http://www.r-project.org/`
[20] OpenNebula. `http://www.opennebula.org/`
[21] OpenStack. `http://www.openstack.org/`
[22] I. Salomie, T. Cioara, I. Anghel, D. Moldovan, G. Copil, P. Plebani, *An energy aware context model for green IT service centers*, Service-Oriented Computing. Lecture Notes in Computer Science 6568, Springer, Berlin, 2011, pp. 169-180.
[23] B. Sotomayor, K. Keahey, I. Foster, *Combining batch execution and leasing using virtual machines*, in Proceedings of the 17th International Symposium on High performance distributed computing (HPDC'08), 2008, pp. 87-96.
[24] B. Sotomayor, R. S. Montero, I. M. Llorente, I. Foster, *An open source solution for virtual infrastructure management in private and hybrid clouds*, IEEE Internet Computing, Special Issue on Cloud Computing, 2009.
[25] B. Sotomayor, R. S. Montero, I. M. Llorente, I. Foster, *Resource leasing and the art of suspending virtual machines*, in Proceedings of the 11th IEEE International Conference on High Performance Computing and Communications (HPCC-09), 2009, pp. 59-68.
[26] B. Sotomayor Basilio, *Provisioning computational resources using virtual machines and leases*, PhD Dissertation, Univ. of Chicago, Illinois, USA, 2010.
[27] A. Verma, P. Ahuja, A. Neogi, *pMapper: power and migration cost aware application placement in virtualized systems*, in Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware'08), 2008, pp. 243-264.
[28] T. Wood, P. Shenoy, A. Venkataramani, M. Yousif, *Sandpiper: Black-box and gray-box resource management for virtual machines*, Comput. Netw., 53 (2009), pp. 2923-2938.
[29] H. Zhang, K. Yoshihira, Y.-Y. Su, G. Jiang, M. Chen, X. Wang, *iPOEM: A GPS tool for integrated management in virtualized data centers*, in Proceedings of the 8th IEEE/ACM International Conference on Autonomic Computing (ICAC'11), 2011, pp. 41-50.

Department of Computer Science, Technical University of Cluj-Napoca, Romania; Faculty of Physics, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: `cora.craciun@phys.ubbcluj.ro`

Department of Computer Science, Technical University of Cluj-Napoca, Romania
E-mail address: `Ioan.Salomie@cs.utcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 1, 2017
DOI: 10.24193/subbi.2017.1.04

MACHINE LEARNING TECHNIQUES FOR DETECTING FALSE SIGNATURES

MIHAI TELETIN

Abstract. Deciding whether a handwritten signature is legit or it has been falsified is a very complex task. Several methods have been tried out by the graphology experts in order to detect such fraud. However, it is obvious that it is very hard to perform such a classification. In this paper we investigate the possibility to use some supervised learning techniques in order to build models capable to accurately perform such an analysis. The results reported during the testing phase of the obtained model are encouraging for further work.

1. Introduction

Determining the authenticity of signatures is quite an old task. From an algorithmic point of view, even though the solution for this problem was researched for a long time, a state of the art solution does not exist. Furthermore, designing a computational algorithm to cover the experience of a trained eye of a graphology expert may seem impossible.

Obviously, in some cases this problem may be considered a very critical one. For examples, high costly frauds can be avoided if it can be proven that they are based on forger signatures.

Considering that this task is a very complex one and that an outstanding experience of an expert is needed in order to perform such a discrimination, we can say that the problem can be tackled using learning methodologies. Designing such a complex algorithm from scratch may be much harder and may not manage to highlight all the corner cases. However, *machine learning* approaches are known to be very easily adaptable to changes [10] and are relatively easier to be used when solving complex problems.

Received by the editors: April 26, 2017.

2010 *Mathematics Subject Classification.* 68T05, 62M45.

1998 *CR Categories and Descriptors.* I.2.6 [**Computing Methodologies**]: Artificial Intelligence – *Learning*; I.2.8 [**Computing Methodologies**]: Artificial Intelligence – *Problem solving*.

Key words and phrases. Machine learning, Convolutional neural networks, Support vector machines, Classification, Signature verification.

The aim of this paper is to present a *machine learning* approach proposed for this binary classification, to highlight its performance by testing it against new, unseen data. We consider that *supervised learning* approaches can contribute very well in the area of research focused on this task. Furthermore, recent research has shown that deep learning methods can be successfully applied in this area of research [13][4]. Thus, we propose a method to solve the signature verification problem by using a powerful existing feature extractor which is based on *deep convolutional networks*. The method proposed in this paper is novel since it combines a powerful trainable classifier that is the *support vector machine* with the well known convolutional pretrained model.

The remaining of the paper is structured as follow: in Section 2 we are going to present the problem from a machine learning perspective and to highlight its difficulty. In Section 3 we briefly describe the *models* used for the task, the *support vector machine* on top of the *convolutional network based feature extractor*. A short overview of the related work is also presented in Section 3.2. Our approach is then presented in Section 4. We will explain our *training* and *testing* methodologies. The performance of the model is then analysed in Section 5. Finally, the conclusions of the work are outlined in Section 6.

2. Problem Relevance and Difficulty

The specialists in the domain of graphology have developed an amazing sense of truth when dealing with an amazing diversity of signatures. While some of them can easily be classified by any amateur, there are lots of amazingly imitated signatures that can only be classified as forgery by a trained eye. Also, we can also have authentic signatures that look strange and may fool even an expert to decide that it looks like a false one.

Even though the graphologists are recognized for performing really well this classification, computer scientists have tried to come up with algorithmic approaches that can assist the experts for taking this decision. However, trying to define a set of rules capable to reproduce the natural behaviour of experts is a very challenging task.

Nevertheless, a proper solution for defining an algorithm to recognize such signatures can be expressed by *Machine learning* approaches. We are going to highlight that *supervised learning* approaches such as *support vector machines* combined with *convolutional neural networks* can prove that this problem is solvable obtaining some good results. In this *supervised learning* scenario, the model will improve its performance using a training set consisting of already classified images of signature. This problem will be viewed from a *machine learning* perspective as a *binary classification problem*.

Classification is one type of problem which can be solved in a supervised manner. The aim is to learn to develop approximation functions for an unknown function called *target function*. Classification deals with functions that produce discrete output (a finite set of values, in our case the true classes: *legit* and *forgery*).

For the presented *classification* problem, we are going to learn to directly classify images of signatures. For doing so we will use a pretrained feature extractor that is capable to determine a set of relevant features by processing the images. Using this set of features, we can train our own model capable to solve the discussed problem.

3. Background

3.1. **Machine learning models used.** In this section we are going to briefly present the proposed *machine learning* models used in order to perform the classification, the *support vector machine* and the *convolutional neural network*. Furthermore we are going to summarise some of the related work.

3.1.1. *Convolutional neural networks.* Research results have shown that the full automatic models such as *convolutional neural networks* give much more better performance than performing manual feature extraction [9]. In the classical approach the system was split in two subsystems:

- *feature extraction module* - a module which processes the given shape (the raw input), performs diverse *heuristics* and produces the feature vector which is considered to describe best the image;
- *trainable classifier module* - a module which learns to classify the data using the feature vector as input.

The main problem of this approach is that its performance is directly decided by the ability of the *extraction module* to come up with relevant and correct sets of features [11][9]. Moreover, this module is usually implemented from scratch, making the task of feature extraction very complicated.

One of the main reason for which this approach was no longer considered was the development of more powerful machine learning models which could easily handle high-dimensional input values [9]. It is now preferred to build a full model which extracts features by itself from the data and learns to predict the desired class. However, data preprocessing process specific to the problem may always be needed in order to increase performance [9].

A much better approach is a model capable to associate proper classes given an almost raw input (e.g. an image). Basically, this system will have to learn by itself to perform the feature engineering.

Convolutional neural networks are special types of neural networks mainly used for problems which require working with huge number of features. Unlike

simple neural networks, they tend to manage well feature sets that are correlated. One of their main possible applications is image classification. This is due to the fact that the images are made of several hundred or thousands pixels, having all the pixels in a neighbourhood highly correlated [9].

A convolutional architecture contains some special types of layers that are capable to process complex input space: *convolutional layers* and *subsampling layers* [9]. Each of these layers are composed of several feature maps capable to learn to extract different relevant features [9]. The main advantage of these networks is that they can easily adapt to several different problems. Thus, one of the domains were they are successfully applied is *computer vision* [9][15][11][13].

3.1.2. *Support vector machine.* The support vector machine (SVM) is a supervised learner introduced by Cortes and Vapnik [2]. The original model was intended to be used for binary classification. The main goal of the SVM is to search for the optimal separating hyperplane among data in order to perform the classification. By finding the optimal separating hyperplane, the model is minimizing the risk to misclassify new, unseen data.

In most cases the data on which the model is trained is not linearly separable. In this case the SVM model is extended in order to support soft margins [10]. By doing so we introduce a mapping function, named *kernel*. The kernel function is used to map the input space into a higher dimensional space, where a linear separating hyperplane may be computed. The linear separation in the high dimensional space will lead to a non-linear decision boundary in the (lower dimensional) input space [10]. Several kernel functions are available in the literature and usually used for non-linear SVMs: *linear* kernel, *polynomial* kernel, *sigmoid* kernel, *radial basis function* (RBF) kernel. Furthermore, the SVM is enhanced with a new hyperparameter, C, the *misclassification parameter*. This hyperparameter lets SVM intentionally misclassify some training data in order to improve the performance on testing [10].

3.2. **Related work.** One of the first approaches tested on the dataset that we are using for our models was introduced in [6]. It represents a method based on *statistical analysis* of the features expressed by the signatures. The approach is intended to extract multiple features from images and to compute the probability of belonging in the two discussed classes. The approach was further discussed and extended in [5].

Classification using neural networks and different feature engineering techniques are presented in [8]. The authors try to identify and to eliminate the weak points of the process of analyzing and classifying signatures.

Machine learning approaches are discussed in [14]. The approach uses *statistical learning* methods in order to learn to predict the class based on the similarity of the features between the known samples and the new ones.

Deep learning approaches including *Restricted Boltzman Machines* were used in [13]. The authors have developed a verification system for this task built on a two-step hybrid classifier system. They have proven that deep learning methods are capable to learn very well to extract relevant features with very limited prior knowledge.

4. The proposed approach

We consider that this problem is solvable in a supervised manner since we can use already annotated datasets of images of signatures. In this learning scenario, the model will learn to detect whether an image contains a legit signature or a false one, by analyzing such already annotated examples.

Our approach consists of three steps. First, a *feature extraction* step is applied on the input data. For this step we are using the *Tensor flow inception graph* pretrained model [15]. This model was developed by *Google* and it represents a very complex convolutional neural network which is composed of 59 layers. The model was trained on a considerable set of images and was capable to obtain state of the art accuracies on very complex problems, such as *ImageNet* classification [15]. So, instead of training a new convolutional neural network, we intend to use this pretrained model as feature extractor.

The next step consists of training a classifier on the pre-processed data. More specifically, using the features extracted from our dataset of signatures we aim to build a classifier that will learn to identify the forger signatures based on such input data. A *Support Vector Machine* classifier will be used for discriminating between original and false signatures. The trained *SVM* will be then *tested* in order to evaluate its performance.

In the following we will detail the steps of our approach.

4.1. **Feature extraction.**

Our dataset consists of annotated images of signatures. From a mathematical point of view, a colored image can be viewed as a 3 dimensional matrix, containing the values of the pixels in the *RGB* code. Obviously, the dimensionality of such data is huge. Thus, directly applying a learner such as *SVM* may be impossible.

Feature extractors are intended to analyze such huge input spaces and come up with a drastically lower set of features which are as representative as possible for the original input space.

We intend to integrate the previously highlighted model, *inception* as feature extractor. By doing so, we use the model in order to extract the desired

set of features, named *bottleneck* features. This name suggests that the features are coming from the latter layers of the model, making them as abstract as possible.

4.2. **Training.** On the set of extracted features, a *SVM* is trained. In order to do so, we take all the available samples in the dataset and we apply the feature extractor. Furthermore, the obtained set of instances (vectors of features) is split in 2 sets: training and testing.

In order to train the model, several hyperparameters are used, such as C, the kernel function, the parameters of the kernel (e.g. γ for the RBF kernel). For optimizing the hyperparameters, a grid search is performed in order to find the best suited ones on a 10-fold cross validation approach. The grid search performs repeated trials for each parameter across a specified interval using geometric steps. The quality of a combination is computed as the average of the accuracy rates estimated for each of the 10 divisions of the dataset.

4.3. **Testing.** The performance of the trained *SVM* model will be tested on a testing set completely disjoint from the training dataset. The testing phase will be performed on unseen data.

Since the considered problem is a binary classification one, the *confusion matrix* will be computed. For building the confusion matrix and computing the measures, we consider that the *forger* signatures are representing the *positive* class while the *negative* class is represented by the *original* ones. A large number of different performance metrics can be computed from the confusion matrix. The *accuracy* (Acc) (Formula 1) is often used, but it is not suitable in the case of imbalanced datasets.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

For better evaluating the performance in case of imbalanced data, the *Area under the ROC curve* (AUC) measure [3] is used in the literature as a more relevant evaluation measure. For our classifier, the output is directly the class label, thus on the ROC curve there is one single (Pf, Pd) point that can be linked to the $(0, 0)$ and $(1, 1)$ points and the area under this curve can be computed using Formula 2.

$$AUC = (1 - Pf) * Recall + \frac{Pf * Recall}{2} + \frac{(1 - Pf) * (1 - Recall)}{2} \tag{2}$$

In Formula (2), the *recall* also known as *probability of detection* (Pd) is computed as $Recall = \frac{TP}{TP+FN}$ and the *probability of false alarm* (Pf) is $Pf = \frac{FP}{FP+TN}$. *F-measure* will also be reported as an evaluation measure for

the classification task. It is computed as the harmonic mean of *precision* and *recall*, as shown in Formula (3). The *precision* of the classification is expressed as $Precision = \frac{TP}{TP+FP}$.

$$F - measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{3}$$

We report both the *accuracy* and the *confusion matrix* related measures because by doing so we can easier interpret the performance of the model. Moreover, since the testing set is imbalanced, these measures can be considered very important.

5. Results and discussion

In this section we start by presenting the experimental results obtained by applying the approach introduced in Section 4 on a publicly available dataset of images representing signatures.

5.1. **Dataset and parameters setting.** The dataset used in our experiments is free and publicly available [7]. It consists of 4000 annotated samples from which 800 are forgeries. In order to construct the dataset, several persons were asked to write down their own signature. Furthermore, another person was asked to try to replicate the original signature.

In the dataset we have multiple signers each of them having the original signature and some forgeries. We intend to train our model in order to distinguish between the two signature types, forgery and original in an offline manner [6]. The available set can help the model to generalize since it consists of both forgeries and original samples coming from several persons.

The feature extraction step (Section 4.1) is first applied. The set of *bottleneck* features extracted from our images consists of 2048 positive real numbers, lower than 1. The pre-processed dataset will be then used for training the SVM.

The following sequences are used for optimizing the hyperparameters C and γ: $C \in \{1, 5, 10, 100, 1000\}$ and $\gamma \in \{1e-1, 1e-2, 1e-3, 1e-4, 1e-5\}$. The following kernels are candidates for the grid search: linear, polynomial, sigmoid and RBF.

The best hyperparameters which were chosen by analysing the results from the *grid search* are:

- kernel: *RBF*
- C=5
- γ=1e-2

5.2. **Results.** For our experiments we have used the *scikit-learn* implementation of *SVM* [12]. 80% of the dataset was reserved for training and on these instances we performed a training methodology which mainly consisted of a *SVM* grid search over the training set. The testing methodology described in Section 4.3 was applied on the trained *SVM* using the rest of the dataset. The obtained *accuracy* (*Acc*) was **95.1%**. For the obtained accuracy we compute the 95% Confidence Interval (CI) [1] as given in Formula (4).

$$CI(Acc) = 1.96 \cdot \sqrt{\frac{Acc \cdot (1 - Acc)}{n}} \tag{4}$$

where n represents the number of samples in the testing set. Accordingly, the 95% confidence interval is computed as follow: $[Acc - CI(Acc), Acc + CI(Acc)]$. For our experiment, the reported 95% CI for the *accuracy* on the testing set is **[0.935, 0.966]**. Thus, there is a 95% confidence that the accuracy of our classifier ranges in the confidence interval.

The confusion matrix from Table 1 provides a better overview on the performance of the proposed model.

		Forgery	Original	Total
Signatures	Forgery	132	23	155
	Original	12	548	560
	Total	144	571	715

TABLE 1. Confusion matrix

The AUC measure computed for our classifier is **0.92** and the *F-measure* is **0.88**. These values express a very good performance for the proposed classification model.

The dataset was reshuffled in order to repeat the random split for the training and testing sets. The proposed experiment was repeated 20 times in order to analyze the evolution of the AUC measure. We observe in Table 2 a low value for the standard deviation, as well as close AUC values for the minimum and maximum AUC reported during the 20 runs. Some of the ROC curve outcomes can be visualised in Figure 1. The random classifier (having an AUC of 0.5) is represented in Figure 1 by the dotted red line.

	min	max	median	mode	mean	stdev
AUC	0.88	0.94	0.90	0.91	0.90	0.0148

TABLE 2. Experimental results for 20 experiments

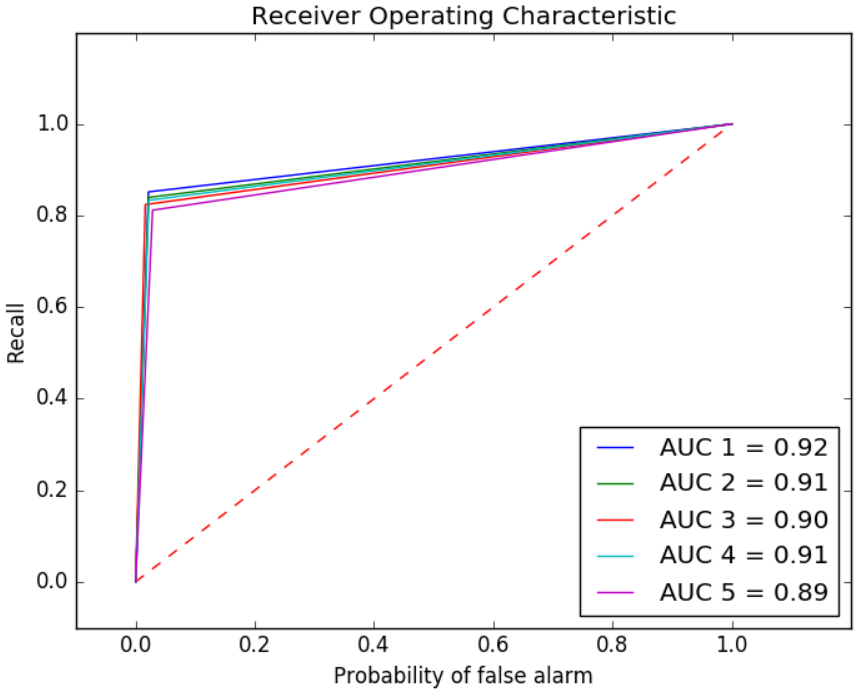


FIGURE 1. ROC curve outcomes for multiple experiments

We illustrate in Table 3 a brief comparison between our approach introduced in Section 4 and the similar related work described in Section 3.2. An exact comparison can be made only with the approaches from Kovari and Charaf [6][5], since they use for evaluation the same dataset as our case study. The other two approaches from [14] and [13] report results on other datasets, thus the comparison is not entirely relevant.

#	**Approach**	**Performance**	**Our approach**
1	Statistical analysis [6][5]	89%	$95\% \pm 0.015$
2	Statistical learning[14]	84%	–
3	Deep learning[13]	$85.03\% \pm 14.25$	–

TABLE 3. Comparison to related work based on the accuracy evaluation measure.

If we look only to the performance measure of the approaches described in Table 3, we observe that our approach is comparable to the related work.

Moreover, the 95% CI obtained by our approach is very small, compared to the one from [13], and this proves again the performance of our model.

6. Conclusions and further work

In this paper we have presented a *machine learning* method based on a feature extractor that can be successfully used in solving the signature verification problem. Considering the good results, we may say that we have confirmed again that this complex task is suitable for *machine learning* solving.

Further work consists in extending the experiment on multiple benchmark datasets in order to have a better overview of the capability of the proposed method. Building a convolutional neural network from scratch will be also considered.

References

[1] L.D. Brown, T.T. Cai, and A. DasGupta. Interval Estimation for a Binomial Proportion (with discussion). *Statistical Science*, 16(2):101–133, 2001.

[2] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[3] Tom Fawcett. An introduction to ROC analysis. *Pattern Recogn. Lett.*, 27(8):861–874, 2006.

[4] Mohsen Fayyaz, Mohammad Hajizadeh Saffar, Mohammad Sabokrou, and Mahmood Fathy. Feature representation for online signature verification. *CoRR*, abs/1505.08153, 2015.

[5] Bence Kovari and Hassan Charaf. Analysis of intra-person variability of features for off-line signature verification. *W. Trans. on Comp.*, 9(11):1359–1368, November 2010.

[6] Bence Kovari and Hassan Charaf. Statistical analysis of signature features with respect to applicability in off-line signature verification. In *Proceedings of the 14th WSEAS International Conference on Computers: Part of the 14th WSEAS CSCC Multiconference - Volume II*, ICCOMP'10, pages 473–478, Stevens Point, Wisconsin, USA, 2010. World Scientific and Engineering Academy and Society (WSEAS).

[7] Bence Kovari and Hassan Charaf. A study on the consistency and significance of local features in off-line signature verification. *Pattern Recognition Letters*, https://www.aut.bme.hu/Pages/Research/Signature/Resources, 2013.

[8] Bence Kovari, Benedek Toth, and Hassan Charaf. Classification approaches in off-line handwritten signature verification. *WSEAS TRANSACTIONS on MATHEMATICS Issue 9*, 2009.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[10] Thomas M. Mitchell. *Machine learning*. McGraw-Hill, Inc. New York, USA, 1997.

[11] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, USA, January 2016.

[12] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, November 2011.

[13] Bernardete Ribeiro, Ivo Gonçalves, Sérgio Santos, and Alexander Kovacec. *Deep Learning Networks for Off-Line Handwritten Signature Recognition*, pages 523–532. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
[14] Harish Srinivasan, Sargur N. Srihari, and Matthew J. Beal. *Machine Learning for Signature Verification*, pages 761–775. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
[15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: tmic1334@scs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 1, 2017
DOI: 10.24193/subbi.2017.1.05

PRELIMINARY MEASUREMENTS IN IDENTIFYING DESIGN FLAWS

CAMELIA ŞERBAN, ANDREEA VESCAN, AND HORIA F. POP

Abstract. Software metrics are of great importance in object-oriented design assessment. They quantify various aspects of design entities and play an important role in predicting design quality. Despite the fact that software metrics have become increasingly useful, they raise several issues. Among them, relevant to our research are the issue of setting threshold values and the problem of measurement results interpretation. Fuzzy clustering analysis is used to overcome the limitations of the existing approaches that are using threshold values for metrics and to provide a better interpretation of the obtained measurement results.

This paper focuses on metrics-based design flaw detection in object-oriented design. A new metric, Design Flaw Entropy which measures the distribution of design flaws into the analyzed system is introduced. To validate the proposed approach, a case study was also proposed.

1. INTRODUCTION

Over an extended period of time software systems are often subject to a process of evolution, applications becoming very large and complex. They undergo repeated modifications in order to satisfy any requirement regarding a business change. The result is that the code deviates from its original design and the system becomes unmanageable. A minor change in one of its parts may have unpredictable effects in completely other parts [1]. To avoid such risk a high quality design should be preserved throughout the system life cycle. This can be achieved by repeatedly assessing the system design, aiming to identify in due course those design entities that do not comply with the rules, principles and practices of a good design, and suggesting possible refactorings or improvements to be performed.

Received by the editors: April 25, 2017.

2000 *Mathematics Subject Classification.* 68N30, 68T37.

1998 *CR Categories and Descriptors.* code D.2.8 [**Software Enginnering**]: *Metrics – Product metrics*; code D.1.5 [**Pattern recognition**]: *Clustering – Algorithms*.

Key words and phrases. Software metrics, object oriented design, fuzzy clustering.

As a consequence of the above detailing, our previous work [10] was focused on developing a methodology for quantitative evaluation of object-oriented design. The proposed methodology is based on static analysis of the source code and is described by a framework of four abstraction layers. A new method for measurements results interpretation, based on fuzzy clustering technique, is also contained in this framework.

The above mentioned proposed methodology for design assessment is completed in this paper with a new metric, Design Flaw Entropy (DFE), which offers preliminary measurements in identifying those parts of the system design that suffers from degradation. In other words, the DFE metric measures the dispersion or the distribution of a specified design flaw among the analyzed design entities. The proposed metric is based on fuzzy clustering analysis method which aims to overcome the limitations of existing approaches that use thresholds values for metrics used.

The rest of the paper is organized as follows. The description and relevance of the problem of object-oriented design assessment briefly emphasizing the main layers of the above mentioned framework is presented in Section 2. The Design Flaw Entropy metrics is introduced in Section 3, by giving the definition, representative examples and the properties. To validate this metric in Section 4 a case study that aims to identify those classes from a software system that are affected by "God Class" design flaw is presented. Section 5 presents some metric-based related approaches for solving the object-oriented assessment problem and also discuses their limitations. Finally, Section 6 summarizes the contributions of this work and outlines directions for further research.

2. OBJECT-ORIENTED DESIGN ASSESSMENT PROCESS

In this section we aim at presenting the problem of object-oriented assessment and its relevance, as well as the main steps of the software assessment process.

The main steps needed to be performed in any software assessment process require a clear specification of *entities that are evaluated* (the assessment domain) and of *the assessment objectives*, as well as the identification of *methods* and *techniques* that offer a *relevant interpretation of the assessment results obtained* (a thorny issue, insufficiently explored so far in the literature).

All the above mentioned elements are described in a formal manner in our previous proposed methodology for object-oriented design assessment [10], defining the contextual background for the proposed metric, metric that completes our previous work and adds more relevance to the interpretation of the assessment results.

Therefore, in what follows we briefly describe these steps of the software assessment process:

(1) Domain Assessment identification.
(2) Setting the assessment objectives.
(3) Computing the metrics values. Fuzzy partition determination. Assessment results analysis.

2.1. **Domain Assessment identification.** The proposed methodology for a quantitative assessment of object oriented design [10] uses static analysis of the source code. Therefore, the assessed domain should capture only those elements that define the structure of an object-oriented system: the *design entities* (e.g. classes, packages) that are relevant for the analysis, together with their *properties* (e.g. the visibility level of attributes) and the *relationships* (e.g. methods access attributes) that exist between them. Marinescu [1] gathers these elements into a model for object-oriented design.

Our previous work [10] has defined in a formal manner, using terms of algebraic sets and relations, the above mentioned elements, introducing a new background used to formally define metrics and to establish the assessment objectives.

In what follows, the 3-tuple:

$$D(S) = (E, Prop(E), Rel(E))$$

represents the assessment domain corresponding to a software system S, where: E represents the design entities set of S; $Prop(E)$ defines the properties of the elements from E, and $Rel(E)$ represents the relations between the design entities of the set E.

2.2. **Setting the assessment objectives.** The main objective of an object-oriented software assessment is that of verifying whether the built system meets quality factors such as *maintainability, extensibility, scalability and reusability*. Fenton's axiom [15] states that good internal structure should provide good external quality. Consequently, the assessment objectives are reduced to verifying if there is conformity between the software system internal structure and the principles and heuristics of good design. According to Marinescu [1] these principles and heuristics of good design are related with the internal quality attributes such as coupling, cohesion, complexity and data abstraction.

A design feature that indicates deviations from good design principles is named "design flaw" [1]. In recent years, the literature displays various forms of descriptions for bad or flawed design such as bad-smells [12]. The community of researchers [1, 12, 14, 5] was interested in setting a relation between the principles of good design with the design flaws. They wanted to seek what

the violated principles or rules were for a certain design flaw or vice versa, what were the design flaws that could propagate in code if a design principle was violated. These design flaws or principles are than correlated with metrics to quantify these aspects and to automate the assessment process. According to these, the assessment objectives are reduced at identifying a list of design entities, called "suspect" which are affected by a specified design flaw. More detailed, being given a list of design entities AE_p that are evaluated with respect to a specified design flaw p, we have to establish its corresponding design principles and for each of these principles some relevant software metrics M_p. Based on the metrics values, computed on a given design entities set, we aim to identify the suspect entities.

2.3. **Computing the metrics values. Fuzzy partition determination.** After establishing the assessment objectives, the next step is metrics computation. In order to automatically compute these metric and to obtain the fuzzy partition, we have developed a tool, called *Metrics* written in C#. Metrics is divided in five components, Metrics Worker, Parser, Design Entities Model, Metric Definitions and Fuzzy Analysis.

The results of the assessment, done in an automatically manner, are therefore, the values of selected metrics
$M_p = \{m_1, m_2, ..., m_{noM_p}\}$, computed on each design entity from the assessed design entities set AE_p.

To overcome the limitation encountered when a metric based approach is used, that of setting the thresholds for the metrics values, the fuzzy clustering analysis is used. Thus, an entity may be placed in more that one group, having different membership degree, obtaining a fuzzy partition defined as in Definition 1:

Definition 1. *([10]) **Fuzzy partition of the design entities.***
A set $U_{AE_p,M_p} = \{U_1, U_2, ..., U_c\}$ is called a fuzzy partition of the design entities set $AE_p = \{e_1, e_2, ..., e_n\}$, entities characterized by the values of metrics the $M_p = \{m_1, m_2, ..., m_{noM_p}\}$ iff:

- *$U_i = (u_{i1}, u_{i2}, ..., u_{in})$, $1 \leq i \leq c$;*
- *$u_{ij} \in [0..1]$, $1 \leq i \leq c$, $1 \leq j \leq n$, u_{ij} – represents the membership degree of the design entity e_j to cluster i;*
- *$\sum_{i=1}^{c} u_{ij} = 1$, $1 \leq j \leq n$ – the sum of each column of U is constrained to the value 1.*

If $c = 2$ then U is called a *binary fuzzy partition.*

The fuzzy partition U best represents the cluster substructure of the data set AE_p, i.e. objects of the same class should be as similar as possible (the

difference between any two metrics values of these objects is very close to 0 value), and objects of different classes should be as dissimilar as possible. The measure used for discriminating objects (classes) can be any *metric* or *semimetric* function (d). In our approach we have used the *Euclidian distance* metric.

Fuzzy Divisive Hierarchic Clustering (FDHC) algorithm [11] was applied to determine a fuzzy partition. The FDHC algorithm produce a binary tree hierarchy that provides an in-depth analysis of the data set, by deciding on the optimal number of clusters and the optimal cluster substructure of the data set. The leaves of the binary tree hierarchy determine an optimal fuzzy partition of the assessed entities.

Based on previously obtained optimal fuzzy partition, we have to decide which design clusters contain suspect entities and which of them require further investigation. This decision is influenced by the distribution of entities per clusters (how many entities have dominant membership degree in that cluster), being also the distribution of the analyzed design flaw into the system. It is obvious the fact that a uniform distribution of entities per cluster highlights a difficult re-factorization. For example, if we have a system with sixty design entities divided into two clusters with thirty entities on each of them, it will be hard to make a decision to redesign thirty entities out of sixty. Conversely, if the two clusters contain eight and fifty two entities, it is much easier to take a decision. In the following, we'll introduce a metric to provide us with information on defect design entities distribution. This metric measures also the effort needed in order to restructure the system design. A high value of this metric suggesting that the system is compromised and would require a redesign from scratch.

The next section introduces this metric, discusses some representative examples and identifies its main properties.

3. Design Flaw Entropy Metric

As we have mentioned before, our goal is to define a metric (Design Flaw Entropy - DFE) which could provide an in-depth analysis regarding the distribution of an analyzed design flaw (the degree of its spread into the system) in order to converge through an optimal decision regarding the set of "suspect" design entities.

DFE is defined considering the notion of *entropy* adapted from communication information theory of Shannon [23]. Starting from this concept many researchers [19, 20, 21, 22] have developed new measures for the assessment of software products.

3.1. **Design Flaw Entropy metric definition.** Let us consider a fuzzy partition $U_{AE_p,M_p} = \{U_1, U_2, \ldots, U_c\}$ of design entities $AE_p = \{e_1, e_2, ..., e_n\}$, entities characterized by the values of metrics $M_p = \{m_1, m_2, \ldots, m_{noM_p}\}$, metrics selected in order to quantify a specified design flaw p.

Definition 2. *We say that an entity $e_j \in AE_p$, $1 \leq j \leq n$, have dominant membership degree to cluster U_i, $1 \leq i \leq c$, if $u_{ij} = \max\{u_{rj} | r = \overline{1,c}\}$.*

Definition 3. *The relative frequency of occurrence or the probability of a cluster $U_i \in U_{AE_p,M_p}$, denoted by $p(U_i)$, represents the ratio between the number of entities from AE_p that have dominant membership degree to cluster U_i and the total number of entities from AE_p.*

We will denote by $P_{U_{AE_p,M_p}} = \{p(U_1), p(U_2), \ldots, p(U_c)\}$ the probability distribution per clusters of the partition U_{AE_p,M_p}.

Definition 4. *A measure of the information (self-information) contained in a cluster $U_i \in U_{AE_p,M_p}$ is defined as $I(U_i) = -\log_2 p(U_i)$.*

In the context of the previous definitions and notations, we can now introduce the definition of the proposed metric.

Definition 5. *Design Flaw Entropy (DFE) corresponding to fuzzy partition U_{AE_p,M_p} is defined as the average of the self-information associated to each cluster $U_i \in U_{AE_p,M_p}$. Formally:*

$$DFE : FP(AE_p, M_p) \to [0..\infty],$$

$$DFE(U_{AE_p,M_p}) = \sum_{i=1}^{c} p(U_i) \cdot I(U_i)$$

where $FP(AE_p, M_p)$ is the set of all fuzzy partitions of the design entities set AE_p, entities characterized by the values of metrics $M_p = \{m_1, m_2, \ldots, m_{noM_p}\}$, metrics selected in order to quantify a specified design principle or design flaw p.

The definition of this metric can be shortly described as follows: for each cluster c of the analyzed fuzzy partition, we compute its probability distribution as a ratio between the number of entities that have dominant membership degree to that cluster and the total number of analyzed entities. We also compute a measure of the information (self-information) contained in that cluster that is next used in the definition of the DEF metric: the average of the self-information associated to each cluster.

3.2. **Further analysis of DFE metric. Representative examples.** For a given set of entities, the value of DFE metric depends on the number of clusters of the obtained fuzzy partition and on the entities distribution per clusters. In order to identify the properties of this metric and to better emphasize its meaning, several representative examples are discussed in what follows. We compute the value of DFE metric for nine different fuzzy partitions on a set of sixty design entities. The meaning of these values are also discussed.

The first considered partition $U_1 = \{U_{1,1}\}$ has one cluster ($c = 1$) with the probability distribution per clusters $P_{U_1} = (60/60)$, all design entities having dominant membership degree on the same cluster. The DFE metric value being in this case:

$$DFE(U_1) = -p(U_{1,1}) \cdot \log_2 p(U_{1,1}) = -1 \cdot 0 = 0. \tag{1}$$

The meaning of such a situation is that all entities are equally affected by the analyzed design flaw, a case almost impossible to meet.

The second partition $U_2 = \{U_{2,1}, U_{2,2}\}$ has two clusters ($c = 2$) with the probability distribution per clusters $P_{U_2} = (1/60, 59/60)$, one design entity having dominant membership degree on the first cluster and 59 entities on the second one. The DFE metric value being:

$$\begin{aligned} DFE(U_2) &= -(p(U_{2,1}) \cdot \log_2 p(U_{2,1}) + p(U_{2,2}) \cdot \log_2 p(U_{2,2})) \\ &= -(1/60 \cdot \log_2 1/60 + 59/60 \cdot \log_2 59/60) = 0.12. \end{aligned} \tag{2}$$

In this example one entity need to be reviewed.

The third partition $U_3 = \{U_{3,1}, U_{3,2}\}$ has two clusters ($c = 2$) with the probability distribution per clusters $P_{U_3} = (2/60, 58/60)$, two design entities having dominant membership degree on the first cluster and 58 entities on the second one. The DFE metric value being in this case:

$$\begin{aligned} DFE(U_3) &= -(p(U_{2,1}) \cdot \log_2 p(U_{2,1}) + p(U_{2,2}) \cdot \log_2 p(U_{2,2})) \\ &= -(2/60 \cdot \log_2 2/60 + 58/60 \cdot \log_2 58/60) = 0.21. \end{aligned} \tag{3}$$

The U_3 partition is very similar with U_2, identifying two entities to be reviewed.

The fourth partition $U_4 = \{U_{4,1}, U_{4,2}\}$ has two clusters ($c = 2$) with the probability distribution per clusters $P_{U_4} = (10/60, 50/60)$, 10 design entities having dominant membership degree on the first cluster and 50 entities on the second one. The DFE metric value being in this case:

$$DFE(U_4) = -(10/60 \cdot \log_2 10/60 + 50/60 \cdot \log_2 50/60) = 0.65. \tag{4}$$

Now, we can observe that once the entities distribution per clusters tends to be uniform, the DFE metric value increases. This means that the number of entities that need to be reviewed is higher.

The fifth partition, $U_5 = \{U_{5,1}, U_{5,2}\}$ has two clusters ($c = 2$) with an equiprobable distribution per clusters $P_{U_5} = (30/60, 30/60)$. The DFE metric value being in this case:

$$DFE(U_5) = -(30/60 \cdot \log_2 30/60 + 30/60 \cdot \log_2 30/60) = 1. \tag{5}$$

In such a case the analyzed design flaw is spread on half of the system design or even more, at least fifty percents of design entities are affected.

The sixth partition, $U_6 = \{U_{6,1}, U_{6,2}, U_{6,3}\}$ has three clusters ($c = 3$) with the probability distribution per clusters $P_{U_5} = (1/60, 1/60, 58/60)$, one design entity having dominant membership degree on the first cluster, one have on the second cluster and 58 entities on the third one. The DFE metric value being in this case:

$$\begin{aligned} DFE(U_6) = -(&1/60 \cdot \log_2 1/60 + 1/60 \cdot \log_2 1/60, \\ &58/60 \cdot \log_2 58/60) = 0.45. \end{aligned} \tag{6}$$

A case very similar with the second one, but with three clusters.

The seventh partition, $U_7 = \{U_{7,1}, U_{7,2}, U_{7,3}\}$ has three clusters ($c = 3$) with the probability distribution per clusters $P_{U_7} = (1/60, 2/60, 57/60)$, one design entity having dominant membership degree on the first cluster, two have on the second cluster and 57 entities on the third one. The DFE metric value being in this case:

$$\begin{aligned} DFE(U_7) = -(&1/60 \cdot \log_2 1/60 + 2/60 \cdot \log_2 2/60, \\ &57/60 \cdot \log_2 57/60) = 0.53. \end{aligned} \tag{7}$$

The eighth partition, $U_8 = \{U_{8,1}, U_{8,2}, U_{8,3}\}$ has three clusters ($c = 3$) with the probability distribution per clusters $P_{U_8} = (10/60, 20/60, 30/60)$, 10 design entities having dominant membership degree on the first cluster, 20 have on the second cluster and 30 entities on the third one. The DFE metric value being in this case:

$$\begin{aligned} DFE(U_8) = -(&10/60 \cdot \log_2 10/60 + 20/60 \cdot \log_2 20/60, \\ &30/60 \cdot \log_2 30/60) = 1.45. \end{aligned} \tag{8}$$

Here, we can observe again that, once the probability distribution per clusters tends to be equiprobable (uniformity) the analyzed design flaw is spread almost over the entire system.

TABLE 1. Representative examples of DFE metric computed on 9 partitions of 60 design entities

U_k	c	$P_{U_k} = (p(U_{k,1}), p(U_{k,2}), ..., p(U_{k,c}))$	$DFE(U_k)$
$U_1 = \{U_{1,1}\}$	1	$(60/60)$	0
$U_2 = \{U_{2,1}, U_{2,2}\}$	2	$(1/60, 59/60)$	0.12
$U_3 = \{U_{3,1}, U_{3,2}\}$	2	$(2/60, 58/60)$	0.21
$U_4 = \{U_{4,1}, U_{4,2}\}$	2	$(10/60, 50/60)$	0.65
$U_5 = \{U_{5,1}, U_{5,2}\}$	2	$(30/60, 30/60)$	1
$U_6 = \{U_{6,1}, U_{6,2}, U_{6,3}\}$	3	$(1/60, 1/60, 48/60)$	0.45
$U_7 = \{U_{7,1}, U_{7,2}, U_{7,3}\}$	3	$(1/60, 2/60, 47/60)$	0.53
$U_8 = \{U_{8,1}, U_{8,2}, U_{8,3}\}$	3	$(10/60, 20/60, 30/60)$	1.45
$U_9 = \{U_{9,1}, U_{9,2}, U_{9,3}\}$	3	$(20/60, 20/60, 20/60)$	1.58

The ninth partition $U_9 = \{U_{9,1}, U_{9,2}, U_{9,3}\}$ has three clusters, $(c = 3)$ with an equiprobable distribution per clusters $P_{U_9} = (20/60, 20/60, 20/60)$. The DFE metric value being in this case:

$$DFE(U_9) = -(20/60 \cdot \log_2 20/60 + 20/60 \cdot \log_2 20/60, \\ 20/60 \cdot \log_2 20/60) = 1.58. \tag{9}$$

Having all these representative examples into account, we can identify the properties of the proposed metric. They are very important in order to draw a conclusion regarding the meaning of DFE metric value. The next section describes these properties.

3.3. **Properties of the Design Flaw Entropy metric.** Analyzing the information contained in Table 1 (that contains the above presented representative examples) we can conclude the following properties of DFE metric:

(1) $DFE(U_{AE_p,M_p}) = 0 \Leftrightarrow c = 1,\ \forall U_{AE_p,M_p} \in FP(U_{AE_p,M_p})$; This property states that design flaw entropy is zero if and only if all entities are placed on the same cluster. In this case the variety of the analyzed design flaw p is *minimal*, meaning that all entities are affected on the same measure by the analyzed design flaw.

On the other hand, the possible *maximum entropy* occurs when each entity is placed in a separate cluster. In such a case the number of clusters equals to the number of entities:

$$DFE(U_{AE,M}) = -\log_2 1/n \Leftrightarrow c = n \wedge p(U_i) = 1/n.$$

(2) Let us consider the set of all partitions from $FP(AE_p, M_p)$ which have the number of clusters equals to c,

$$FP_c(AE_p, M_p) = \{U_{AE_p,M_p} \in FP(AE_p, M_p)$$

$$|U_{AE_p,M_p} = \{U_1, U_2, ..., U_c\}\}.$$

In this case two values are important to be discussed: the *minimum* and the *maximum* values of DFE metric:

- A *minimum value* of DFE metric is attained for the following distribution of entities per clusters: $(1/n, ..., 1/n, (n-c)/n)$, where n is the number of entities. In what follows we denote these values by $MinDFE(n, c)$.
- On the other extreme, as the probabilities associated with each cluster will have values closer together, the entropy will have a higher value. At the limit, DFE reaches a *maximum value* for an equiprobable distribution of elements per clusters: $U_{AE_p,M_p} = \{U_1, U_2, ..., U_c\}$, $p(U_i) = \frac{1}{c}, 1 \leq i \leq c$ we have:
  $DFE(V_{AE_p,M_p)}) \leq DFE(U_{AE_p,M_p})$,
  $(\forall) V_{AE_p,M_p} \in FP(AE_p, M_p)$,
  $V_{AE_p,M_p} = \{V_1, V_2, ..., V_c\}$ we denote these values by $Max\ DFE(n, c)$.
  In this case, the higher the entropy becomes, the more difficult is to identify those design fragments that need to be reviewed.

(3) For an equiprobable distribution of elements per clusters, once the number of clusters increases, the entropy will have a higher value:
$(\forall)\ U_{AE_p,M_p}, V_{AE_p,M_p} \in FP(AE_p, M_p)$, $U_{AE_p,M_p} = \{U_1, U_2, ..., U_c\}$, $V_{AE_p,M_p} = \{V_1, V_2, ..., V_c, V_{c+1}\}$ such that $p(U_i) = \frac{1}{c}, p(V_j) = \frac{1}{c+1}, (1 \leq i \leq c), (1 \leq j \leq c+1)$ we have:

$$DFE(U_{AE,M}) \leq DFE(V_{AE,M}).$$

4. Experimental evaluation

In this section we present an empirical validation of our proposed metric. This validation is based on a case study which aims to evaluate the design of an open source object-oriented software system, called log4net [13]. It consists of 214 classes grouped in 10 packages.

4.1. **God Class suspect identification - a fuzzy based approach.** The objective of the proposed assessment is to identify those design entities affected by "God Class" [12] design flaw. An instance of God Class does most of the operation tasks, leaving only minor details to a series of trivial classes; it

TABLE 2. The distribution per clusters of the class design entities

Cluster	No. of members with dominant membership degree
1.1.1	19
1.1.2	23
1.2.1	42
1.2.2	106
2.1	14
2.2	10
Total number of entities = 214	

also uses the data from other classes. Briefly, *God Class* design flaw refers to those classes "which tend to centralize the intelligence of the system" [1]. As a consequence, the principle of manageable complexity is violated, as god classes tend to capture more than one abstraction. Another shortcoming of these pathological classes is their tendency towards non-cohesion. If we consider the quality attributes, god classes also have a negative impact on the reusability and understandability of that part of the system they belong to.

Marinescu [1] correlates this design flaw with the metrics: Weighted Methods per Class (WMC) [6], Tight Class Cohesion (TCC) [7], Access to Foreign Data (ATFD) [1]. Analyzing the definitions of these metrics, we can conclude that *a possible "God Class" suspect will have high WMC and ATFD metric values and low TCC metric values.* As we mentioned earlier, due to the fact that is hard to establish a threshold for metrics values, we have proposed a new approach based on fuzzy clustering analysis.

The assessed entities, $AE_{GodClass}$, are the set of classes from our "log4net" application. After computing the metrics values for each class design entity, we apply the $FDHC$ algorithm in order to obtain the optimal fuzzy partition, denoted as $U_{AE_{God\ Class},M_{God\ Class}} = \{1.1.1, 1.1.2, 1.2.1, 1.2.2, 2.1, 2.2\}$. Table 2 contains the distribution of entities per clusters. This is a filtered information needed for DFE metric computation. However, the complete data of this partition is required for further analysis in order to establish the final list of suspect entities.

An important aspect regarding this partition, is that of isolated data points. These entities defined a new cluster for the partition denoted by $U_{AE_{God\ Class},M_{God\ Class}}$.

In order to study the distribution of God Class design flaw into our system, we compute the DFE metric, introduced in Section 3. The obtained value of DFE metric, $DFE(U_{AE_{God\ Class},M_{God\ Class}}) = 2.22$. is then compared with the minimum (MinDFE(214,7)=0.32) and the maximum (MaxDFE(214,7)=2.81) values of this metric, computed on a set of 214 entities distributed in 7 clusters.

The probability distribution of the corresponding partition for minimum and maximum values of DFE metric being:

$$(1/214, 1/214, 1/214, 1/214, 1/214, 1/214, 198/214),$$

$$(30/214, 30/214, 30/214, 30/214, 30/214, 30/214, 34/214).$$

By analyzing the value of the DFE metric, we can observe that the values are very close to the maximum value, "saying" that many parts of the software systems must be revised. The computation of DFE metric gives us preliminary information regarding the extent to which our system design is affected by God Class flaw. Further analysis is needed in order to identify those classes that need to be refactored.

5. Related approaches

This section presents the current state of art regarding the entropy metric as a measure of object-oriented design quality and analyzes the differences compared with our present approach.

Object-oriented design assessment are traditionally done in metrics-centric manner. Using the notion of entropy from communication information theory, new measures [19] were developed for the assessment of software designs. The metric is computed using information available in class definitions. The new complexity measure of classes is correlated with traditional complexity measures such as McCabe's cyclomatic metric and the number-of-defects metric. The defined entropy metric is shown to be a reliable measure in predicting the implementation complexity of classes, given that the design of the classes does not change substantially during implementation. *In relation to this existing approach, our proposal uses the same "initial" definition of entropy but applied to measure design flaw in an already developed object-oriented system.*

An approach was proposed in [9]. The authors propose the use of entropy as defined in Information Theory [23], to evaluate the initial status of an object-oriented design as well as its status after the addition of new functionality. The difference between the entropy of the two systems provides insight to the quality of the design in terms of how flexible it has been during the enhancement of its functionality. *Our approach aims to achieve goals similar to* [9]*, to differentiate "good"from "bad" designs by the use of an information theoretic entropy metric. We argue that our model differs by the fact that DFE metric can be applied for any design alteration type not only for the extension of the system's functionality.*

Another type of metrics based assessment of an object-oriented system is defined using only the class definition [20] and not information from the class implementation. The proposed metric is a true design metric that can be

calculated during the design phase of the software maintenance or development life cycles, before any code has been implemented. The metric could provide a better early indication (in the design phase rather than in the implementation phase) of the design complexity than has been available before. *Our model retrieves information for metric computation by parsing the source but also it could be applied at the design phase, but the amount of information beeng limited at this stage (ex. methods' complexity).*

By replicating and extending previous studies on the entropy of software systems, in [24] the authors defined three entropies as global metrics at the system level, in terms of three CK metrics computed at the class level. They extended the empirical analysis also to RFC and CBO since these CK metrics have been shown to be correlated with fault-proneness of OO class. With the aim of finding a global metric for describing software quality in terms of code degradation and reduced maintainability during time, they correlated such metrics to the variation in time of the total number of defects of the system a good measure of defect proneness. *Our current approach considers a fuzzy partition obtained starting from any object oriented metric that is selected to quantify features related to asessed design entities.*

As a conclusion, in relation to this existing approach, our also uses the notion of entropy from communication information theory [23], but applied to measure design flaw in an object-oriented system, either for an already developed object-oriented system or for an extension of the systems functionality. The information for metric computation is obtained by parsing the source code, and regarding the threshold criteria our approach overcomes the limitation impose by it, using a fuzzy clustering method.

6. Conclusion and Future Work

Software metrics are considered of great importance in software quality assurance. In spite of this fact, there is a gap between the things measured and the ones really important in terms of quality characteristics. This discontinuity is due mainly to the fact that the methods currently used for interpreting metrics results are at a low level of abstraction, incomplete and sometimes irrelevant. The current paper proposes a new metric - Design Flaw Entropy (DFE) - which completes our previous framework for object oriented design assessment, being very useful in measurements results interpretation.

To highlight the relevance of our proposed approach, a case-study has been used which aims to identify those classes from a software system that are affected by "God Class" design flaw. The source code of the open source object-oriented software system, called log4net [13] was used.

For future work, we intend to focus our research in the following directions:

- to apply the proposed evaluation framework on more case studies;
- to repeat the evaluation for a new version of the software system, after some suggested refactorings were applied;
- to automate the task of establishing the list of suspect entities and the list of refactorings that could be applied.

References

[1] R. Marinescu. *Measurement and quality in object-oriented design*, Ph.D. thesis, Faculty of Automatics and Computer Science, Politehnica University of Timisoara, 2003.

[2] S. Mazeiar, Li. Shimin, and T. Ladan. *A Metric-Based Heuristic Framework to Detect Object-Oriented Design Flaws* Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC06), 2006.

[3] P.F. Mihancea, and R. Marinescu. *Towards the optimization of automatic detection of design flaws in object-oriented software systems*, In Proc. of the 9th European Conf. on Software Maintenance and Reengineering, 92-101, 2005.

[4] L. Tahvildari, and K. Kontogiannis. *Improving design quality using meta-pattern transformations: A metric-based approach*, Journal of Software Maintenance and Evolution: Research and Practice, 16, 331-361, 2004.

[5] A.J. Riel. *Object-Oriented Design Heuristics*, Addison-Wesley, 1996.

[6] S. Chidamber, and C. Kemerer. *A metric suite for object-oriented design*, IEEE Transactions on Software Engineering, 20(6), 476–493, 1994.

[7] J.M. Bieman, and B.K. Kang. *Cohesion and Reuse in an Object-Oriented System*, ACM Symposium on Software Reusability, 1995.

[8] M. O'Keeffe, and M.Ó. Cinnéide. Search-based refactoring: an empirical study, *Journal of Software Maintenance and Evolution: Research and Practice*, 20, 345–364, 2008.

[9] A. Chatzigeorgiou, and G. Stephanides. Entropy as a Measure of Object-Oriented Design Quality, *1st Balkan Conference on Informatics (BCI'2003)*, 21–23, 2003.

[10] C. Serban. A Conceptual Framework for Object-oriented Design Assessment. *Computer Modeling and Simulation, UKSim Fourth European Modelling Symposium on Computer Modelling and Simulation*, 90–95, 2010.

[11] D. Dumitrescu. Hierarchical pattern classification, *Fuzzy Sets and Systems* 28, 145–162, 1988.

[12] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.

[13] Open source project: log4net, http://logging.apache.org/log4net.

[14] R. Martin. Design Principles and Patterns: http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf, 2006.

[15] N. Fenton. *Software measurement: A necessary scientific base*, IEEE Transactions on Softw. Engineering, 20(3), 1994.

[16] J. Han, and M. Kamber. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, (2001).

[17] A. Jain, and R. Dubes. Algorithms for Clustering Data. Prentice Hall, Englewood Cliffs, New Jersey, 1998.

[18] A. Jain, M.N. Murty, and P. Flynn. Data clustering: A review. ACM Computing Surveys, 31(3):264–323, (1999).

[19] J. Bansiyav, C. Davis, L. Etzkorn. An entropy-based complexity measure for object-oriented designs. Theory an Practice of Object Systems. 5(2):111–118, 1999.
[20] L. Etzkorn, S. Gholston, and W.E. Hughes. A semantic entropy metric. Journal of Software Maintenance: Research and Practice. 14(4):293–310, 2002.
[21] A. Marcus, M. Boxall, and S. Araban. Interface Metrics for Reusability Analysis of Components. Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04), 2004.
[22] K. Kim, Y. Shin, and C. Wu. Complexity Measures for Object-Oriented Program Based on the Entropy. In Proceedings of the Second Asia Pacific Software Engineering Conference, 1995.
[23] C.E. Shannon, and W. Weaver. The Mathematical Theory of Communication. Urbana, IL, University of Illinois Press, 1949.
[24] I. Turnu, G. Concas, M. Marchesi, and R. Tonelli. Entropy of some CK metrics to Assess Object-Oriented Software Quality. International Journal of Software Engineering and Knowledge Engineering, 23(3), 2013.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: {camelia,avescan,hfpop}@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 1, 2017
DOI: 10.24193/subbi.2017.1.06

A BIG DATA APPROACH IN MUTATION ANALYSIS AND PREDICTION

SILVANA ALBERT

Abstract. Although the technology advancement in the last few years has been exponentially growing, there are still a lot of medical problems that don't have an accessible solution. One of these problems is the one that genetics is facing: the absence of a solution for inspecting the previously reported genetic mutations. In order to confirm a mutation, the specialists need to narrow it down based on their experience and, if present, the few documented precedent cases. This paper focuses on presenting a solution for analyzing big amounts of historical genetic data in an efficient, fast and user-friendly way. As a proof of concept, it demonstrates the huge role that Big Data has in genetic mutations aggregation and it can be considered a starting point for similar solutions that aim to continuously innovate genetics. The effectiveness of our proposal is highlighted by comparing it with similar existing solutions.

1. Introduction

The volume of aggregated medical information has increased exponentially in the last few years and it will keep increasing. DNA Sequencing is just a few years away from becoming affordable. When that happens, the technology has to be prepared to analyze it, extract patterns, prevent anomalies and provide accurate predictions, which will rely heavily on using big data [21]. In our opinion, the genetic advancement should go hand in hand with the technological advancements in order to exploit the full potential of both branches.

The need for interdisciplinary collaboration between the genetic specialists and software engineers has been recently identified and it is proved to be effective [22]. This paper is the result of a brief collaboration that started by putting the needs of the medical community on the first place and the concern of how to develop the solution on second.

Received by the editors: April 24, 2017.

2010 *Mathematics Subject Classification.* 68N01, 68T05.

1998 *CR Categories and Descriptors.* D.2.11 [**Software**]: Software engineering – *Software Architectures*; I.2.6[**Computing Methodologies**]: Artificial Intelligence – *Learning*.

Key words and phrases. Big data, genetics, software, machine learning.

Genetic mutation aggregation refers to the process of gathering any relevant information about genetic mutations and storing it for future use in analysis and visualization.

The contribution of our paper is twofold and is summarized in the following. First, we are proposing a solution for analyzing big amounts of historical genetic data in a very efficient and fast way, using a big data approach. The proposed solution demonstrates the huge role that Big Data [6, 14] has in genetic mutations aggregation and it can be considered a starting point for similar solutions that aim to continuously innovate genetics [5]. Our second aim is to highlight the potential of using supervised *machine learning* [16] models in predicting future genetic mutations. The overall purpose of the paper is to provide demographics and metrics regarding prophylaxis, and diagnosis of different genetic disorders and to offer a solution that allows the medical personnel to access a comprehensive history of patients screened/diagnosed with certain chromosome anomalies or gene mutations.

The rest of the paper is structured as follows. Section 2 presents the fundamental background concepts related to genetics, as well as the applicability of Big Data in mutation analysis. Our approach in mutation analysis and prediction using a Big Data approach is introduced in Section 3. Section 3.2 details the prediction component of the proposed solution and provides several experimental results. An analysis of our proposal and comparison with existing similar work is given in Section 4. Section 5 presents the conclusions of our paper and outlines directions for further improvement and extension.

2. Background

We are presenting in the following section the main concepts involved in our approach.

2.1. **Genetic background.**

DNA, short from deoxyribonucleic acid is a molecule present in all living things that contains instructions needed by organisms in order to develop and reproduce. RNA, abbreviated from Ribonucleic acid is a molecule which plays an important role in creating proteins from DNA.

A *mutation* is a natural process that occurs when a cell copies the DNA before dividing and that changes the DNA sequence.

Mutations are unavoidable; most of them arise along with the natural process of DNA transcription and therefore corrected by efficient DNA repair mechanisms. Usually, mutations are perceived as something bad that happened or that something got broken but there are a lot of cases when it has no impact (the changes are in the areas of the genome that is between the genes).

Nucleotides are structural components of the DNA and RNA. There are approximately 3.000.000.000 nucleotides in a human genome. When a cell

divides, it is supposed to make a copy of its own DNA but sometimes, something bad happens and the result is a similar sequence that has one different nucleotide. That small difference is called a mutation and it is depicted in Figure 1.

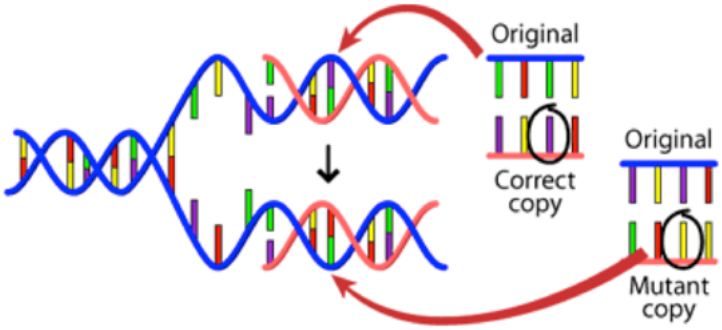


FIGURE 1. Mutation genesis. Figure source: Understanding Evolution [3].

Mutations can arise also from the interaction with external factors. If the person was exposed to certain chemicals or radiation, the chances of a mutation occurring are higher. The agents are breaking the DNA and when the cells try to repair it, some differences can happen.

Gene data is stored in lots of different ways depending on the database. Three of the existing disease-related variation databases are described in section 4. Usually, mutations are stored by gene symbol with very detailed information about the place in the DNA sequence where it happened, all kinds of locus coordinates and all kinds of classifications based on the mutation type (deletion, insertion, splicing, etc.).

At the time of this research, we are not aware of a system that stores both gene data and information about the demographics of its origin because this data is usually confidential and anonymous due to its sensitive nature. As a result of sequencing, data can be stored in standard recognized formats. Some of the most common ones are: Plain sequence format (containing one or more sequences with no extra information), FASTQ (which stores both biological sequence and quality scores and is produced by advanced sequencing instruments) and EMBL (contains an id per sequence and other relevant annotation lines before and after the sequence) [18].

2.2. **Big data and NoSQL.** *Big data* [20, 6] is a notion for storing large collections of data sets and further analyzing, visualizing and transferring them. Collecting data from all kinds of devices leads to storing a lot of data but what is truly impressive about it is not the quantity of information but what we can do with it. From a few terabytes of collected data that we had stored in 2012, we got to entire petabytes now and it is still growing.

Big data can be described as large pools of data that is captured and aggregated with advantages that lead to increasing modern economics, health care, transportation and many other industries.

NoSQL [10] means Not Only SQL, implying that when designing a software solution or product, there is more than one storage mechanism that could be used based on the needs [19]. There are a lot of NoSQL database management systems (at this moment, there are approximately 150) and they can be classified based on their data model [19]: *Key-value* (Dynamo, Riak), *Graph* (Allegro, Infinite Graph), *Multi-model* (OrientDB, FoundationDB), *Document* (MongoDB, Couchbase) and *Column* (Accumulo, Cassandra).

One main characteristic of NoSQL databases is that it is schema agnostic; this means that there is no need for an upfront schema design for allowing data storage. Another important aspect is strong consistency that can be translated in: all the clients should see the same version of data [19]. The last characteristic that NoSQL databases need to have is partition tolerance: the complete system should keep its properties even when deployed on separate servers [3].

In the context of genetic mutations, the information related to a gene and all it's possible anomalies that need to be stored in order to perform a relevant analysis is what constitutes Big Data. Each mutation entry has a series of characteristics that will be described in Section 3.3 and the number of characteristics changes constantly as the science advances. It is important to be able to save different characteristics and not be constrained by an existing rigid schema and that's why NoSQL and Genomics go hand in hand.

3. Our approach

The general idea behind our proposal is to offer a solution that provides demographics and metrics about diagnostics and mutations. It started with the idea of creating a solution that allows the medical personnel to browse through a comprehensive history of patients screened/diagnosed with certain chromosome anomalies or gene mutations.

Visualizing the number of mutations by countries needs a complete database parsing and for 1 million results, it takes a couple of minutes. One adjustment that could make this interrogation faster is splitting the work into a number of threads equal to the number of countries and each thread would count the

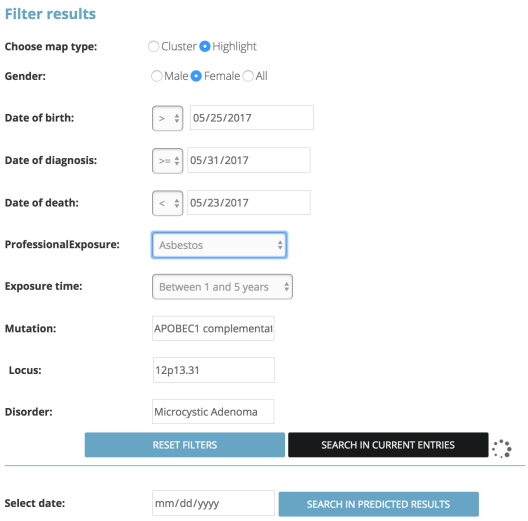


FIGURE 2. Screen shot from the application with possible filtering options.

number of mutations occurred in the given country. In the proposed solution, inserting one million entries without optimizing performance with threads, takes less than 12 minutes. Further optimization can be done and the time can be decreased by using multiple servers and batch inserting using threads. There is some bench-marking performed by Netflix that claims to have inserted 1.1 million client writes per second using Cassandra [4].

The novelty of our solution is that it stores and aggregates genetic, demographic and geographic data. The solution presented in this section is a proof of concept and the stored data is mock data based on real genes and mutations obtained by scraping existing databases.

From the visualization perspective, the originality factor is that based on the multiple filters from Figure 2, the number of people that match are displayed on the world map and the intensity of the color reflects the number of entries per country as seen in Figure 6. The solution paints a very graphic picture of the current situation on the globe, while other solutions are just listing individual mutations in tabular views. The other approach requires a lot of time, attention and work to scroll and comprehend the displayed data; However, if that approach is still needed for reports, our solution also allows exporting the data to Excel files.

3.1. **The proposed solution.** Our solution makes the following scenario possible:

"*As a doctor, I want to see the count of women that were diagnosed at the age of 25 with Breast Cancer, have the mutation KRAS, are currently under treatment and were professionally exposed to chemical agent benzyl.*"

The possible capabilities from the doctors perspective are represented in the use case diagram from Figure 3.

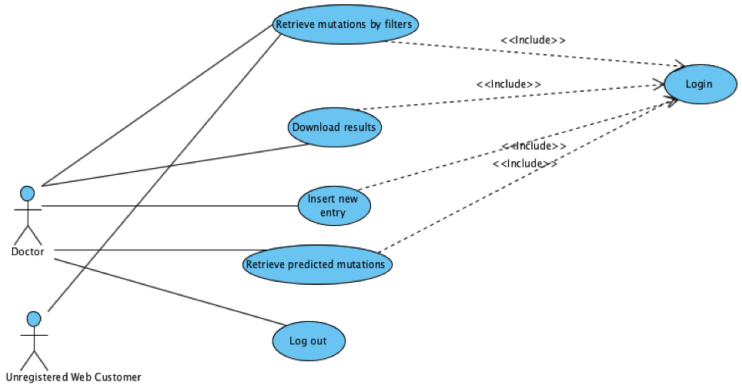


FIGURE 3. Use Case Diagram of the proposed solution.

From the technical perspective, the proposed architecture contains four main components:

(1) Creating a database that stores information about patients and their found mutations and diseases.
(2) Creating a solution for interrogating that database with regard of performance and scalability.
(3) The prediction engine that can help the doctors gain a better overview of the expansion of a mutation in a given period of time.
(4) Once the results are retrieved, they are displayed in a user friendly web interface in a way that means something for the user (position mutations geographically with the possibility of zooming in and further filtration based on gender, age, environmental conditions and so on) instead of simply listing the results in a table.

The component diagram of the proposed solution is depicted in Figure 4.

3.2. **The component for predicting future mutations.** One of the main components of the proposed solution is the one for future mutations prediction.

From a *machine learning* perspective, *predictive modelling* refers to analyzing historical information to make predictions about future [7]. *Machine*

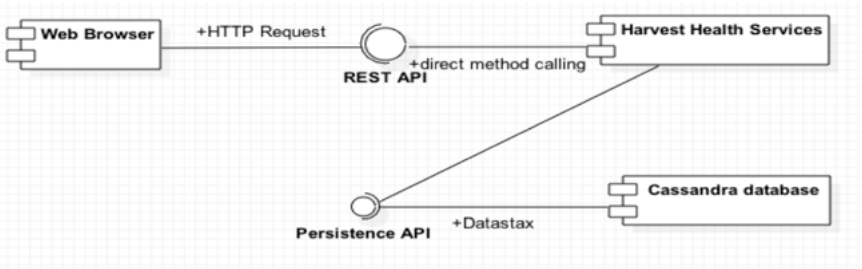


FIGURE 4. Component diagram of the proposed solution.

Learning (ML) [16] is a challenging field of *Artificial Intelligence* in which the focus is to develop adaptive computer systems, able to improve their *performance* from experience and through learning some specific domain knowledge.

Within the machine learning domain, a major emphasis is on *supervised learning*. The systems which learn from an external supervisor are connected to *predictive modelling*. The *predictive models* are able to make predictions based on some training data (i.e. historical data). In supervised learning, the learner is provided with a set of labeled examples (inputs with their known outputs) and then it will be able to generalize from the received examples and to predict the output when faced with an input instance unseen during training. The chosen software for predicting mutations is Weka [9]. The prediction engine workflow starts with the query executed on Cassandra [12] based on the users filters. The results from the database are parsed and stored into a csv file.

The *Training Model* contains 6 fields: the *country code*, the *count of mutations* (retrieved based on the country and exposure factor), the *date of diagnosis* (when the mutation was discovered and entered into the system), the *gender* of the patient, the *exposure time*(in years because it is split in intervals: less than 1 year, between 1 and 5 years, between 5 and 10, between 10 and 20, and over 20) and the most important field: the *professional exposure factor* (the chemical element to which the person was exposed).

Examples of exposure factors: Arsenic, Asbestos, Asphalt fumes, Benzene, Beryllium, 1-Bromopropane and many more.

The *Prediction Service* receives as training data the current aggregated entries from the database. The prediction flow is depicted in Figure 5.

Counts are computed for each country based on the exposure factor and other search criteria and saved as training data in an .arff file. That file is read and classified and based on it, the predicted counts are computed, then converted to JSON and finally passed on as a response to the Prediction

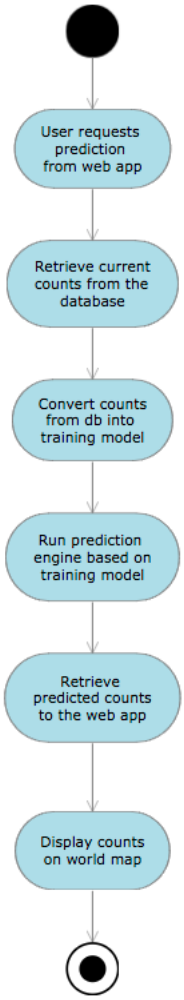


FIGURE 5. Activity diagram of the prediction flow.

Endpoint. This returned JSON will be handled by the UI similar to any other result set from Cassandra and will display the results on the World Map. As an example, Figure 6 illustrates the World map of mutation frequency by professional exposure Arsenic.

The counts from the database for professional exposure to Arsenic for less than one year are: Canada: 18, China: 1509, France: 1354, Norway: 18, Niger: 975, New Zeeland: 413 and so on.

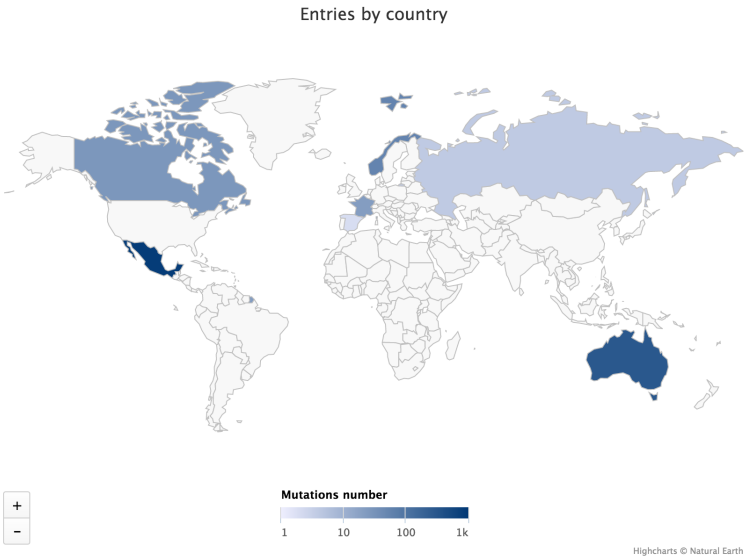


FIGURE 6. World map of mutation frequency by professional exposure Arsenic.

The results vary based on existing counts and number of years we perform the prediction. When changed to 100 years from now, the world map looks significantly different (because the current date and prediction dates are factored into the generated training data)

The predicted counts in 10 years for professional exposure to Arsenic for less than one year are depicted in Figure 7: Canada: 28, Mexico: 875, France: 22, Norway: 56, Spain: 2, Russia: 4 and Australia: 300.

3.3. **Implementation details.** *Apache Cassandra* was used for this proof of concept because of its scalability and reliability. The reading speed is more important than the insert because that is the main use case of the proposed solution: analyzing existing data.

3.3.1. *Persistence model.* The selected model for storage contains the following data: *Name*, *Identification number*, *Gender*, *Country*, *Date of birth*, *Professional exposure* (if present, with possible categories and time of exposure), *Age at diagnosis*, *Date of death*, *Submitted by* (name of the doctor), *Details*, *Disorder 1* (Mutation 1_locus22, Mutation 2_locus30 ... Mutation n_locusxy), *dots Disorder m* (Mutation 1_locus15, Mutation 2_locus13 ... Mutation p_locuskz).

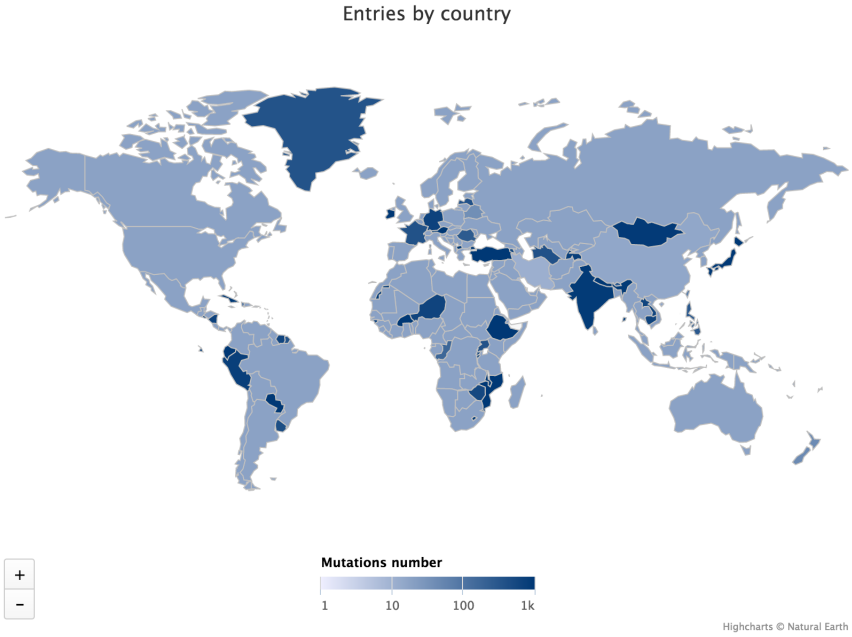


FIGURE 7. World map of mutation frequency by professional exposure Arsenic in 10 years.

It will allow extracting information about frequency of mutation in different regions with different characteristics. The number of stored rows can be dynamic for each entry and this is achieved by using a non-relational database.

Having a different number of columns for each entry is important for building a system for quickly previewing mutations. That is because a patient can have 1 disease with only one mutation while other can have 9 diseases with 1000 mutations.

The primary key contains the identification number of the person, the country identifier and the mutation entries.

A mutation entry has the following form:

$$A2M\_12p13.31\_AlzheimersDisease\_AACS_1 2q24.31\_Traheal\ Cancer$$

This means that on the gene $A2M$ (which is stored with the complete name in a separate table), on locus $12p13.31$, there is a mutation that causes the disorder: *Alzheimers disease*. But this patient has another mutation entry that is separated by comma. So the mutations column contains a string that respects the previously defined format. This way, complex information about a

patient can be stored on a single row. The same logic applies to the professional exposure field also; this is how a professional exposure entry looks like:

$$Asphaltfumes\_34000$$

Unlike mutations, this field is optional for most patients. The first part of the entry is the name of the substance to which the patient was exposed while the former represents the duration of exposure in milliseconds (the patient worked in a contaminated facility for 3 years).

The database was created with replication $=\{'class' : 'SimpleStrategy', 'replication\_factor' : 3\}$. This means that the used strategy is enough for evaluation purposes. The alternative is NetworkedTypologyStrategy which needs to be used when multiple data centers are connected. The replication factor describes the number of replicas of data on multiple nodes. It has to be specified only when using the simple strategy. The personal identification number, the mutation string and the country code compose the primary key. Those are the most important properties that define an entry and are most frequently used in searches. For searches on other criteria, indexes are created. Although indexes are not as performant as searching only for what is contained in the primary key, they were created in order to give the searches flexibility because not all the parameters are filled in for every search query.

4. Discussion and comparison to related work

The solution proposed in this paper facilitates the analysis of genetic data using a big data approach.

As a proof of concept, our proposal demonstrates the huge role that Big Data has in genetic mutations aggregation and it can be considered a starting point for similar solutions that aim to continuously innovate genetics. The presented solution allows the doctors to filter the mutations, visually inspect their frequency on the world map, predict future mutations, insert new entries and export data in various formats. It helps by aggregating all the precedent mutations correlated with a series of external factors. The doctor is able to narrow it down to a reasonable number of possibilities based on the cases that were already solved. This leads to making an informed decision of which mutations to test for. After successfully determining the current case, the specialist will introduce it to the global database, this way, helping future doctors.

We chose the NoSQL approach for implementing our prototype based on existing literature and also the following reasoning.

From a relational database perspective, this same issue could have been resolved by having a table with all genes, another with all possible mutations

by gene and a third one with all the disorders linked to multiple mutations in various genes. When trying to filter by any of the stored information, multiple joins would need to happen.

Our intuition is that performing multiple joins would take more time than the proposed solution but no concrete experiments were performed. Back to our example, the patient with 1000 mutations would be inserted into the database 1000 times having each mutation id as a foreign key or inserted once and store in column or mutation ids separated by commas. Either way, we assume performance would suffer because of the necessary joins that would need to happen.

By using a non relational database instead, the number of columns can be dynamic and it does not matter which of the stored information will be used as a filter. Using the alternative relational approach, if we want the counts of people that have certain disorders would mean joining with mutations to the the different disorder ids and then with disorders to get the names. Filtering by disorders would take more time than filtering by mutation code because the latter means a single join operation while the first means two. By using dynamic columns, it doesn't matter if the entry is searched by mutation or disorder because they are both columns on each row and it takes the same amount of time to retrieve.

We are describing in the following existing solutions for mutation analysis and prediction, comparing them with our proposal.

4.1. **Cosmic-Catalogue of somatic mutations in cancer** [15]**.** *Cosmic* [2] is a tool that allows inspecting various mutations and their frequency. It contains data from The Cancer Genome Atlas and the International Cancer Genome Consortium portals but also, data can submitted directly through their website. It is a comprehensive database that contains so far 20,981 mutations and details about each. Cosmic provides statistics about mutations but it does not contain demographics. There is no way of accessing any information about the people that have these mutations. The main capability that our solution provides and Cosmic doesn't is the visualization of incidence of mutations based on geographic location.

4.2. **The Human Gene Mutation Database** [11]**.** The *Human Gene Mutation Database* (HGMD) [17] is a database at the Institute of Medical Genetics in Cardiff, from BIOBASE that contains over 152.000 mutations. It is a comprehensive data on human inherited disease mutations to genetics and genomic research.

HGMD [11] provides all kinds of features for analyzing mutations but it does not have the capability of linking the characteristics of the person with the discovered genetic characteristics. However, it allows inspecting various

aspects of a mutation from the strictly technical point of view. It also analyzes candidate genes for finding disease linkage and predisposition.

However, there is no apparent correlation between exposure to exposure factors and the subsequent mutations, like in the solution proposed in this paper.

4.3. **Orphanet** [8]**.** *Orphanet* [8] is the solution that doctors in Romania currently use. It is a European website that has the office location in Paris and its main focus is providing content regarding rare diseases and orphan drugs [1]. Orphanet was funded by Inserm (the French National Institute of Health and Medical Research), the French Directorate General for Health and the European Commission. It contains a database that is an encyclopedia of rare diseases and it encourages the collaboration between research teams. It is basically a search engine that provides raw information that is updated annually. It also provides details about ongoing trials [13].

Although it is vastly used by Romanian specialists, Orphanet doesn't provide statistics about the place where the mutations and rare conditions occurred and in which circumstances (age, professional exposure, gender and so on). There is no way to determine the likeliness of the same mutation happening to the current patient.

Compared to the previously mentioned solutions, our approach has the already mentioned advantages: keeps the link between the occurred mutation and it's details to the person that developed it. That way, advanced correlation there can be made based on the factors that led to a mutation, a person's gender and age, diagnosis date, recovery rate and so one.

From the performance perspective, a direct comparison between the proposed solution and other gene databases is not possible because of the way other databases display the results: they are paginated and not aggregated; Users can retrieve a limited amount of data at the time (depending on the database).

Our solution's biggest contribution that none of the alternative solutions provide is the geographic clustering of mutations.

5. Conclusions and future work

We proposed in this paper a *big data* approach in mutation analysis and prediction. As a proof of concept, the presented solution demonstrates the huge role that big data has in genetic mutations aggregation and it can be considered a starting point for similar solutions that aim to continuously innovate genetics.

Future work may be done in order to enhance performance and scalability of the proposed system in order to increase the reading speed. Concrete experiments using a relational approach should be performed. The capabilities of the prediction engine could also be increased and this would lead to more accurate predictions. The basic linear prediction that is used now, can be enhanced to handle complex scenarios that take into consideration environmental factors that may lead to a mutation spreading.

Acknowledgments

The author thanks Dr. Cătană Andreea who contributed to the paper by describing the current methodologies used for genetic diagnosis and helped identifying the need for this analytical software. She also came up with the list of particularities that each entry in the database should have and provided feedback on the solutions main capabilities.

References

[1] S. Ayme and J. Schmidtke. Networking for rare diseases: a necessity for europe. *Bundesgesundheitsblatt*, 2007.

[2] S. Bamford, E. Dawson, S. Forbes, J. Clements., R. Pettett, A. Dogan, A. Flanagan, J. Teague, P.A. Futreal, M.R. Stratton, and R. Wooster. The cosmic (catalogue of somatic mutations in cancer) database and website. *Br. J. Cancer*, 91(2):355–8, July 2004.

[3] Berkeley University. Understanding Evolution - The causes of mutations. http://evolution.berkeley.edu/evolibrary/article/evo_20. Online; 2017.

[4] A. Cockcroft and D. Sheahan. The Netflix Technology Blog. https://medium.com/netflix-techblog/benchmarking-cassandra-scalability-on-aws-over-a-million-writes-per-second-39f45f066c9e. Online; 2011.

[5] B. Feldman, E.M. Martin, and T. Skotnes. Big data in healthcare hype and hope. Technical report, Dr. Bonnie 360, October 2012.

[6] L. Fernandes, M. O′Connor M., and V. Weaver. Big data, bigger outcomes. *AHIMA*, 83(10):38–43, 2012.

[7] S. Finlay. *Predictive Analytics, Data Mining and Big Data: Myths, Misconceptions and Methods*. Business in the Digital Economy. Palgrave Macmillan UK, 2014.

[8] French National Institute for Health and Medical Research. The portal for rare diseases and orphan drugs. http://www.orpha.net/consor/cgi-bin/index.php. Online; 2017.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The WEKA data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[10] R. Hecht and S. Jablonski. NoSQL evaluation: A use case oriented survey. In *2011 International Conference on Cloud and Service Computing*, pages 336–341, Dec 2011.

[11] Institute of Medical Genetics in Cardiff. The Human Gene Mutation Database. http://www.hgmd.cf.ac.uk/ac/index.php. Online; 2017.

[12] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010.

[13] S. Maiella, A. Rath, C. Angin, F. Mousson, and O. Kremp. [orphanet and its consortium: where to find expert-validated information on rare diseases]. *Revue neurologique*, 169(Suppl 1):S3–8, 2013.
[14] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A Byers-Hung. Big data: The next frontier for innovation, competition, and productivity. Technical report, McKinsey Global Institute, June 2011.
[15] Ministry for Primary Industries. COSMIC, the Catalogue Of Somatic Mutations In Cancer. http://cancer.sanger.ac.uk/cosmic. Online; v80, released 13-Feb-17.
[16] T. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.
[17] P.D. Stenson, M. Mort, E.V. Ball, K. Evans, M. Hayden, S. Heywood, M. Hussain, A.D. Phillips, and D.N. Cooper. The human gene mutation database: towards a comprehensive repository of inherited mutation data for medical research, genetic diagnosis and next-generation sequencing studies. *Human Genetics*, pages 1–13, 2017.
[18] G. Stoesser, W. Baker, and A. Broek. The embl nucleotide sequence database. *Nucleic Acids Research*, 30:21–26, 2002.
[19] T. A. M. C. Thantriwatte and C. I. Keppetiyagama. NoSQL query processing system for wireless ad-hoc and sensor networks. In *2011 International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 78–82, Sept 2011.
[20] B. Wang, L. Ruowang, and W. Perrizo. *Big Data Analytics in Bioinformatics and Healthcare*. IGI Global, Hershey, PA, USA, 1st edition, 2014.
[21] R. Wullianallur and V. Raghupathi. Big data analytics in healthcare: promise and potential. *Health Information Science and Systems*, 2(1):1–3, 2014.
[22] B. Zenger. Can big data solve healthcares big problems? *Health Byte*, 2012.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: albert.silvana@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 1, 2017
DOI: 10.24193/subbi.2017.1.07

IDENTIFYING HIDDEN DEPENDENCIES IN SOFTWARE SYSTEMS

ISTVÁN GERGELY CZIBULA, GABRIELA CZIBULA, DIANA-LUCIA MIHOLCA, AND ZSUZSANNA MARIAN

Abstract. The maintenance and evolution of software systems are highly impacted by activities such as bug fixing, adding new features or functionalities and updating existing ones. *Impact analysis* contributes to improving the maintenance activities by determining those parts from a software system which can be affected by changes to the system. There exist *hidden dependencies* in the software projects which cannot be found using common coupling measures and are due to the so called *indirect coupling*. In this paper we aim to provide a comprehensive review of existing methods for *hidden dependencies identification*, as well as to highlight the limitations of the existing state-of-the-art approaches. We also propose an *unsupervised learning* based computational model for the problem of *hidden dependencies identification* and give some incipient experimental results. The study performed in this paper supports our broader goal of developing *machine learning* methods for automatically detecting *hidden dependencies*.

1. Introduction

Maintenance activities such as bug fixes, updating existing features and adding new ones make up the majority of time and costs allocated to a software project. Each of these changes usually affects only parts of the system, and determining the affected components (classes, modules, methods etc.) is not a trivial problem. *Impact analysis* tries to identify, given a component of a software system, the other components that would be affected by changes to the former [7]. Existing methods for impact analysis usually consider only direct coupling between components, but there also exists *indirect coupling* [36], which creates *hidden dependencies*, that cannot be found using regular

Received by the editors: May 3, 2017.

2010 *Mathematics Subject Classification.* 68N30, 68T05, 62H30.

1998 *CR Categories and Descriptors.* K.6.3 [**Management of computing and information systems**]: Software Management – *Software maintenance*; I.2.6 [**Computing Methodologies**]: Artificial Intelligence – *Learning*; I.5.3 [**Computing Methodologies**]: Pattern Recognition – *Clustering*.

Key words and phrases. Impact analysis, hidden dependencies identification, machine learning, clustering.

coupling measures. Yet, not identifying them can have serious negative consequences [8].

Analyzing program dependencies has an essential role in program comprehension, change propagation, or impact analysis [22]. The software components need to be understood in the context in which they are defined and this context is expressed by the dependencies between the software components. It is fundamental for the software maintainers to discover the system's dependencies and make corresponding changes to ensure that change has been correctly spread out and the software remains stable [37]. Among the software component dependencies, *hidden dependencies* are relationships between two seemingly independent components and are produced by a data flow inside of a third software component [37].

The aim of this paper is to provide a systematic literature review on *hidden dependencies identification* (HDI), highlighting the difficulty of the problem as well as the limitations of the current state-of-the-art in this field. We are also proposing a new computational model based on *unsupervised learning* for the problem of *hidden dependencies identification*. With the broader goal of applying *machine learning* [23, 24] methods for detecting parts of a software system which are not directly coupled, we also describe the evaluation measures usually used for assessing the performance of methods for detecting hidden dependencies.

The remainder of the paper is organized as follows. The description of the HDI problem, together with an illustrative example, are given in Section 2. Section 3 presents the current state-of-the-art in *hidden dependencies identification*. Section 4 contains a discussion on the limitations of existing approaches, introduces our new *machine learning* perspective upon the problem and gives our incipient experimental results. We outline the conclusions of our paper and the directions to continue the research in Section 5.

2. Problem statement and importance

A special class of program dependencies, called *hidden dependencies* (HD) were introduced by Yu and Rajlich in [37]. The authors have also given examples of the software changes that these kinds of dependencies propagate in the code [37]. HDs are particular type of data flows [31] which have an important role in software maintenance and evolution. HDs propagate changes among the application classes of a software system and these changes are hard to detect. As shown in [31], *hidden dependencies* are found even in well designed software systems like JUnit, Drawlets, and Apache FtpServer. Thus, it is of crucial importance for software developers to detect and understand such dependencies.

A task of major importance for software developers is to understand HDs since it contributes to ease the software maintenance and evolution process. Among the software change activities that consider software dependencies we mention [31]: impact analysis [7, 27], change propagation [28], regression testing [32].

Data flows are considered to be the basis of *hidden dependencies* [31]. Since the process of analyzing the data flow in a software is not an easy task, it is very likely that software developers omit HDs rather than more explicit dependencies, introducing, in this way, bugs into the software [31].

The omission of HDs has a major impact particularly for critical computing systems. An example is a bug that was introduced during the evolution of the Minimum Safe Altitude Warning software system (MSAW) and which caused, in 1997, an aircraft crash at the Guam International Airport [9]. It has been shown that a missed HD between two software components which seemed independent has caused the bug in MSAW software: one component activated the alarm at 55 nautical miles and another component deactivated the alarm at 54 nautical miles [9].

The problem of identifying HDs is a very complex one. This is primarily because there is no exact definition for what a *hidden dependency* is.

Different methods existing in the literature were developed for detecting particular types of hidden dependencies. For example, Kagdi and Maletic considered in [17] that there is a HD between two software entities (methods or application classes) if the entities were changed at the same time in the past. Two application classes were considered by Gall et al. to be dependent [11] if they were changed by the same author and in the same time interval.

Beer et al. [5] have proposed a method for generating test data for problems involving complex linear dependencies between variables. The authors have suggested that software developers could specify restrictions on the values of variables in the source code and use them to generate the test cases. The dependencies that the authors called "complex dependencies" are able to capture semantic information that is hard to detect using traditional techniques for program analysis.

Jenkov defines in [16] a *hidden dependency* as a dependency which cannot be seen from a class's interface. Another example of a hidden dependency is the dependency on a static singleton, or static methods from within a method. One cannot observe from the interface if a class depends on static methods or static singletons [16]. These type of dependencies are hard to detect for software developers, they can be discovered only by inspecting the code [16].

2.1. **Examples of HDs.** As shown in Figure 1, in JUnit 4.4, there is a hidden dependency between the methods *getTestHeader()* of class *Failure* (Figure 2) and the method *getDescription()* of class *CompositeRunner* (Figure 3).

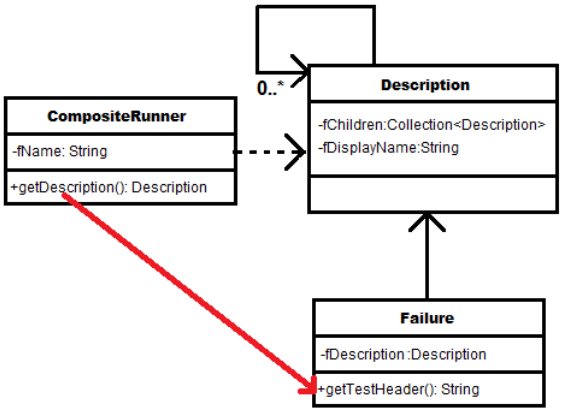


FIGURE 1. Hidden dependency in JUnit 4.4

```
package org.junit.runner.notification;

public class Failure{

    private final Description fDescription;
    ...

    /** @return a user-understandable label for the test*/
    public String getTestHeader() {
        return fDescription.getDisplayName();
    }
    ...
}
```

FIGURE 2. Failure.java

A *Description* object (see Figure 4) describes a test case or a test suite which is to be run or has been run. After execution, in case of failure, the name is printed after being decoded by the method *getTestHeader()* of the class *Failure*.

The method *getDescription()* creates a *Description* object and encodes *fName* in it. So, the method *getTestHeader()* will return it as a user-understandable label for the failed test. The methods share the *test case/suite* concept.

A justification considering pre- and postconditions, for the exemplified hidden dependency, is provided by Vanciu and Rajlich in [31].

3. LITERATURE REVIEW

In this section we present a literature review on the problem of *hidden dependencies identification* (HDI). Despite the importance of finding hidden

```
public class CompositeRunner extends Runner implements Filterable, Sortable {
    private final List<Runner> fRunners= new ArrayList<Runner>();
    private final String fName;

    public CompositeRunner(String name) {
        fName= name;
    }

    public void add(Runner runner) {
        fRunners.add(runner);
    }

    @Override
    public Description getDescription() {
        Description spec = Description.createSuiteDescription(fName);
        for (Runner runner : fRunners)
            spec.addChild(runner.getDescription());
        return spec;
    }
    ...
}
```

FIGURE 3. CompositeRunner.java

```
public class Description implements Serializable {

    private final Collection<Description> fChildren = new ConcurrentLinkedQueue<Description>();
    private final String fDisplayName;
    ...

    /** Create a <code>Description</code> named <code>name</code>.*/
    public static Description createSuiteDescription(String name, Annotation... annotations) {
        return new Description(name, annotations);
    }

    private Description(final String displayName, Annotation... annotations) {
        fDisplayName= displayName;
        fAnnotations= annotations;
    }

    /** @return a user-understandable label*/
    public String getDisplayName() {
        return fDisplayName;
    }

    /** Add <code>Description</code> as a child of the receiver.*/
    public void addChild(Description description) {
        fChildren.add(description);
    }
    ...
}
```

FIGURE 4. Description.java

dependencies, the approaches existing in the literature for this problem have moderate precision and recall values.

There are approaches which use previous versions of the software system and try to identify those classes which were changed together with respect to the same bug report [12]. Gall et al. have introduced in [12] an approach, called

CAESAR, that uses information about previous versions of a system to discover logical dependencies and change patterns among modules. The proposed method is experimentally evaluated on 20 releases of a large Telecommunications Switching System. Information such as version numbers of programs, modules and subsystems together with change reports are used for identifying common change patterns of software modules. CAESAR determines hidden dependencies which are not obvious in the source code, like modules that should be restructured. Instead of using the lines of code for the previous versions of the software, the authors use structural information about programs, modules and subsystems, together with change reports for the releases and their version numbers. The method proposed in [12] has been proved to be capable to identify bugs which were fixed in one version of the system but have appeared again, in other parts of the software, in later versions.

One of the early works is [37], where Yu and Rajlich have transformed System Dependence Graphs into Abstract System Dependence Graphs to determine which class pairs have hidden dependencies. The paper discusses how hidden dependencies impact the process of change propagation and also discusses an algorithm that indicates the possible presence of hidden dependencies. Hidden dependencies are considered to be design faults which contradict the rule "if a class A is unaware of the existence of class B, it is also unconcerned about any change to B". More exactly, a dependence between Class A and B is a hidden dependence if: (1) class A and B are not neighbors in the ASDG, i.e there is no direct dependence between A and B; and (2) there is a third class C, which is dependent on both classes, and there is data flow inside the class C that occurs between instance of class A and instance of class B. A simple algorithm for determining hidden dependencies is introduced and a JAVA example consisting of three classes collaborating to manage a session is considered.

In 2004, Hassan and Holt [14] have studied change propagation in software development. They have proposed several heuristics to predict change propagation by suggesting software entities that should be modified in accordance to the changes an entity has suffered. The heuristics have been empirically evaluated using historical data related to several open source projects. It has been experimentally shown that co-change data can be used to develop models for assisting software developers during change propagation process.

Orso et. al [25] have performed an empirical comparison of two existing dynamic impact analysis algorithms. Both algorithms use static analysis on the call-graph of the system, but they also use traces from the execution of the system to be analyzed. The first algorithm, CoverageImpact, constructs, for each execution, a vector with as many elements as methods in the system to be analyzed, and simply sets the value 1 for each executed method in this

vector. These vectors are used to determine the list of methods that were executed together with the method(s) that will be changed. This list is filtered using a static forward slice starting from each method to be changed. The second algorithm, PathImpact, constructs a so-called whole-path DAG (Directed Acyclic Graph) from the execution traces and this DAG is traversed, starting from the point that denotes the method to be changed, to determine which methods are impacted by the change. The authors have performed experiments using several versions of three Java systems to compare the precision, time and space cost of these two algorithms. The results showed that PathImpact is more precise (it returns a shorter list of methods affected by the changes to be performed in the system), but this precision comes at a significant cost of space (the whole-path DAGs need a lot more space to be stored than the binary vectors) and time. In [4], the same authors have introduced an approach that combines the precision of the PathImpact with the speed and small space overhead of the CoverageImpact method. In order to achieve this, they introduce the Execute After relation, defined for two entities, which is true if the first entity is executed after the second one. An entity can be impacted by a change to another entity only if this relation is true for them. The authors also propose a simple and fast algorithm to compute this relationship for every pair of entities and this can be done by keeping in memory only two vectors having as many elements as entities in the systems. Comparing the performance of this new algorithm to PathImpact, they conclude that it is as precise as PathImpact, but it is only slightly slower than CoverageImpact.

There are many different metrics to measure coupling between components of a software system, but most of these metrics measure direct coupling (according to [34], in a description containing almost 30 coupling metrics, only two mention indirect coupling). Indirect coupling is often considered to be simply the transitive closure of entities in direct coupling, but in many cases such transitive closures contain most of the entities from the system. Since indirect coupling can affect the maintainability of a software system as well, the authors of [34] have proposed an algorithm to detect one type of indirect coupling, which they call use-def coupling. By a simple example, they show that such use-def coupling can occur when a method returns a value (in their example this value is a String representing the type of a book), which is given as parameter to another method (in their example a method which checks whether the type of a book is suitable to the person who wants to borrow it from the library). Even if the two methods are not directly coupled (there is no direct connection between them), if the values returned by the first method are changed, errors can be introduced into the second method. They propose an algorithm to detect for each variable the point where the variable

was initialized (to see the entities to which it is coupled) and implement this algorithm in an Eclipse plug-in, called ICD (Indirect Coupling Detector).

Yang and Tempero investigate in [33] the notion of *indirect dependence* and argue that it is an important criteria for evaluating modularity. The authors claim the importance of understanding *indirect coupling* (IC) due to its "hidden" nature. They highlight the importance of determining which forms of indirect coupling may be avoided, arguing that a system with high levels of avoidable indirect coupling is "unmodular" [33]. The same authors, Yang and Tempero extend in [35] their previous study and propose metrics which express the relationship between indirect coupling and maintainability. The proposed metrics are applied to existing Java applications.

While traditional coupling measures cannot be used for finding hidden dependencies, Poshyvanyk et al. [27] have presented how a conceptual coupling measure that considers identifier names, comments and other textual elements of code can be used for impact analysis and can find hidden dependencies as well. The study reports precision and recall around 20%.

Petrenko and Raylich [26] have introduced an interactive tool called JRipples which is useful for iterative impact analysis. The proposed tool does not discover HD, the software developer having the responsibility to correctly identify the hidden dependecies during impact analysis.

In case of large software systems, computing Abstract System Dependence Graphs can be expensive, so other approaches which are based on the order in which different methods are called (call trace) have been introduced: if a method is always called after another method, there might be a dependency (hidden or not) between the classes where these methods are, as presented in [31]. Vanciu and Rajlich [31] have proposed a dynamic technique for identifying hidden dependencies. It is based on computing "execute completely after" relations which are filtered based on pre- and postconditions that are generated dynamically. For evaluation, open source software systems like JUnit, Drawlets and Apache FtpServer are used. The authors show that hidden dependencies exist even in well-designed software, like the ones considered for evaluation. For the case studies used for evaluation, the technique proposed in [31] obtained a precision between 46% and 59% for discovering hidden dependencies.

Kirbas et al. [19] have investigated the influence of the evolutionary coupling on defect proneness. A positive correlation between evolutionary coupling and defect measures, such as number of defects and defect density, have been confirmed by numerical experiments performed for a large financial legacy software system. Two evolutionary coupling measures derived from modification requests (MR) have been used in this study.

He et al. [15] have proposed Coverage and Program Structure Slicing (CPSS) as a novel solution to fault localization. CPSS is based on Reverse Data Dependence Analysis Model and integrates Coverage Based Fault Localization (CBFL) and Program Slicing by analyzing the program structure. The proposed method has been experimentally proven to be more effective than existing related methods.

Kouroshfar et al. [20] have studied the effects of architecturally dispersed co-changes on software quality. It has been experimentally shown that the changes involving multiple architectural modules are more correlated with defects than the intra-module co-changes. The study corroborates the relevance of considering architecture in predicting software defects.

Akbarinasaji et al. [3] have proposed a suite of six metrics of logical dependency among source files in a software system. The impact of these metrics on defect prediction performance has been evaluated by applying two learning models, the Logistic Regression and the Naive Bayes, on three different software projects. The metrics have been used as features of the training data, their values being derived from the timestamp information in the change history of files. The experimental results have confirmed that, if the values of logical dependency are high, they significantly improve the performance of the defect prediction models.

Bell [6] has studied the influence of hidden dependencies identification on software testing. The author has shown that increasing the efficiency and the effectiveness of testing through a good knowledge of the hidden dependencies between tests improves the software reliability. In real software systems, there are hidden dependencies between tests, which makes the testing process harder. In such situations, the tests cannot be run in parallel, since they are not independent (i.e. a test outcome is influenced by the execution of other test). It has been shown in the software engineering literature [6] that these dependencies are often difficult and hidden from the software developers. Bell has developed a software system called VMVM for detecting different types of dependencies between tests and has used detected information to significantly reduce the testing time (with around 60% in average). VMVM is a Java implementation of a technique called Unit Test Virtualization, a technique which isolates the side-effects of each unit test from other tests. It is based on a hybrid static-dynamic analysis and automatically identifies the code segments that may create side-effects. These segments are isolated in a container similar to a virtual machine.

Due to the complexity of the HDI problem, there is a continuous interest in the software engineering literature to develop more performant detectors.

4. Discussion

The evaluation measure which is usually used for estimating the performance of a process that detects hidden dependencies is the *precision* of the detection [31]. The *precision* of a HDI process is computed as the percentage of dependencies that were correctly reported as *hidden*. Since the entire set of HDs is unknown, the *recall* measure is impractical in this context.

After the in-depth analysis of the related work we presented in Section 3, we can conclude that there are a number of limitations of the approaches existing in the literature for hidden dependencies identification.

Regarding the performance of the identification process, the existing approaches have moderate *precision* values: in [26] the precision ranges from 6% to 18%, [27] reports precision around 20%, while in [4], it ranges from 30% to 40%. An improvement of the performance of HDI is achieved in [31] which reports precision between 46% and 60%.

Besides, some existing approaches rely on historical data, which is not always available (and knowledge extracted from it cannot be used for other projects), or on the creation of different graphs which can be expensive for large systems.

Even if there are a lot of approaches existing in the literature in the direction of *impact analysis* and *hidden dependencies identification*, to the best of our knowledge, the applicability of *machine learning* methods has not been investigated yet. Due to their ability to unconver hidden patterns in data, we consider that *machine learning* models would be appropriate for detecting hidden dependencies in software projects and that this direction may provide valuable results in the field.

4.1. **Our approach.** Our first objective to achieve the long-term goal of this research is to investigate how to improve *impact analysis* approaches. We are planning to reach this objective by developing new coupling measurements to improve the performance of estimating the impacts of future changes in software systems. We aim to capture in the coupling measures both the *structural* and *conceptual* aspects of coupling.

Our second research direction will be to propose *machine learning* methods for detecting *hidden dependencies* in software systems. As we deduced from reviewing the problem of *hidden dependencies identification*, none of the approaches from the literature use machine learning algorithms. Out of the existing approaches, using call trace information seems promising. We believe that *relational association rules* (RARs) [29] can be used to mine relevant patterns in the call traces. Based on our previous experience with *relational association rule mining*, we consider that RARs have the potential to improve the precision and recall values, since low values make the existing approaches

impractical to be used for real systems. Besides relational association rules, we will also investigate the applicability of unsupervised learning techniques, such as *clustering* or *self-organizing maps* [18].

In our view, the problem of HDI can be formalized as a *clustering* problem. *Clustering* [13] (also known as unsupervised classification) is able to differentiate groups of similar objects inside a given data set through detecting *hidden* patterns in data. Thus, we consider that a *clustering* approach may be useful in detecting *hidden dependencies*.

Let us consider that a software system S is represented as a set of *software entities*, $\mathcal{S} = \{e_1, e_2, \dots, e_n\}$. Depending on the granularity of the approach, a software entity $e_i \in \mathcal{S}$ can be a software component, an application class, a method or an attribute from a class, etc. The clustering approach we propose for HDI consists of three main steps and is depicted in Figure 5:

- **Data representation**. The *software entities* and the existing relationships between them (inheritance, dependency, aggregation, etc.) are extracted from the analyzed software system. Each software entity will be represented by a high-dimensional vector. The challenge will be to determine a set of *software metrics* relevant for deciding if a *hidden dependency* exists between two entities.
- **Grouping**. The set of entities extracted at the previous step are grouped in clusters using an *unsupervised learning* method (e.g. *clustering* [13] or *self-organizing map* [30]). The goal of this step is to obtain groups (clusters) which will contain software entities which depend on each other (considering both *direct* and *hidden* dependencies).
- **HD extraction**. The clusters obtained after the *Grouping* step will be filtered in order to remove the *direct* dependencies. The remaining entities from each cluster will provide a list of HDs.

Figure 5 contains a graphical representation of the solution we propose for *hidden dependencies identification*.

4.2. **Preliminary experimental results.** In this section we give some incipient experimental results which underline the effectiveness of using *unsupervised learning* for detecting software *dependencies*. We consider an experiment on an open source software framework, *Commons DbUtils* (version 1.3), a library consisting of a small set of classes which are designed to make working with JDBC easier [1]. It consists of 22 classes, placed in three packages:

- *default package* - contains the core classes and interfaces of the system.
- *handlers* - contains implementations for the *ResultSetHandler* interface from the default package.

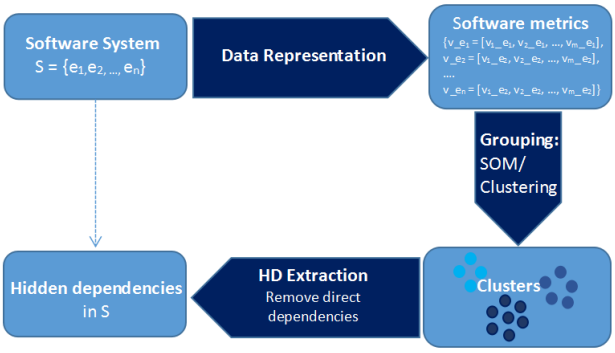


FIGURE 5. The proposed solution.

Package	Class Name
default	BasicRowProcessor(BRP), BeanProcessor (BP), DbUtils, ProxyFactory (PF), QueryLoader (QL), QueryRunner (QR), ResultSetHandler (RSH), ResultSetIterator (RSI), RowProcessor (RP)
handlers	AbstractKeyedHandler (AKH), AbstractListHandler (ALH) ArrayHandler (AH), ArrayListHandler (ALH), BeanHandler (BH), BeanListHandler (BLH), KeyedHandler (KH), ColumnListHandler (CLH), MapListHandler (MPH), MapHandler (MH), ScalarHandler (SH)
wrappers	SqlNullCheckedResultSet (SNCRS), StringTrimmedResultSet (STRS)

TABLE 1. Packages and classes in the DbUtils 1.3 system.

- *wrappers* - contains two wrappers for the *ResultSet* class from the *java.sql* package.

The exact classes from each package are presented on Table 1.

The application classes from DbUtils 1.3 are converted into a text corpus containing the elements of the implementation code (including comments, identifiers, etc.). Then, the corpus associated to the class is represented as a fixed-length feature vector of numerical values. These feature vectors are unsupervisedly learned using the implementation of *Paragraph Vector* (or `Doc2Vec`) offered by Gensim [2]. `Doc2Vec`, a model proposed by Le and Mikolov [21], is useful for expressing variable-length textual information as a fixed-length dense numeric vector (*paragraph vector*), being an alternative to common models such as bag-of-words and bag-of-n-grams. A first advantage of `Doc2Vec`

	AKH	ALH	AH	ALH	BRP	BH	BLH	BP	CLH	DU	KH	MH	MLH	PF	QL	QR	RSH	RSI	RP	SH	SNCRS	STR
AKH	1.000	0.654	0.711	0.714	0.463	0.758	0.769	0.416	0.632	0.533	0.728	0.785	0.769	0.475	0.427	0.115	0.668	0.572	0.834	0.597	0.091	0.439
ALH	0.654	1.000	0.668	0.736	0.360	0.765	0.779	0.323	0.503	0.287	0.301	0.731	0.726	0.425	0.264	0.127	0.853	0.521	0.669	0.477	0.368	0.606
AH	0.711	0.668	1.000	0.950	0.511	0.918	0.900	0.269	0.814	0.473	0.644	0.956	0.941	0.616	0.485	0.253	0.769	0.727	0.704	0.812	0.382	0.585
ALH	0.714	0.736	0.950	1.000	0.481	0.895	0.899	0.328	0.792	0.556	0.561	0.952	0.978	0.695	0.558	0.341	0.840	0.682	0.698	0.760	0.455	0.614
BRP	0.463	0.360	0.511	0.481	1.000	0.547	0.557	0.373	0.418	0.147	0.547	0.501	0.447	0.160	0.473	0.286	0.413	0.322	0.489	0.395	0.254	0.523
BH	0.758	0.765	0.918	0.895	0.547	1.000	0.979	0.384	0.825	0.360	0.628	0.915	0.897	0.589	0.414	0.116	0.771	0.547	0.760	0.822	0.342	0.592
BLH	0.769	0.779	0.900	0.899	0.557	0.979	1.000	0.380	0.787	0.408	0.579	0.899	0.902	0.610	0.398	0.042	0.774	0.548	0.769	0.781	0.330	0.582
BP	0.416	0.323	0.269	0.328	0.373	0.384	0.380	1.000	0.271	0.093	0.334	0.286	0.293	0.339	0.321	0.111	0.410	0.061	0.326	0.322	0.125	0.258
CLH	0.632	0.503	0.814	0.792	0.418	0.825	0.787	0.271	1.000	0.248	0.747	0.824	0.805	0.514	0.460	0.314	0.614	0.468	0.602	0.973	0.365	0.453
DU	0.533	0.287	0.473	0.556	0.147	0.360	0.408	0.093	0.248	1.000	0.236	0.533	0.590	0.563	0.293	0.402	0.430	0.434	0.517	0.211	0.175	0.347
KH	0.728	0.301	0.644	0.561	0.547	0.628	0.579	0.334	0.747	0.236	1.000	0.673	0.617	0.186	0.479	0.153	0.440	0.489	0.602	0.715	0.057	0.195
MH	0.785	0.731	0.956	0.952	0.501	0.915	0.899	0.286	0.824	0.533	0.673	1.000	0.969	0.592	0.488	0.299	0.808	0.718	0.734	0.802	0.366	0.580
MLH	0.769	0.726	0.941	0.978	0.447	0.897	0.902	0.293	0.805	0.590	0.617	0.969	1.000	0.682	0.516	0.337	0.804	0.658	0.713	0.767	0.372	0.553
PF	0.475	0.425	0.616	0.695	0.160	0.589	0.610	0.339	0.514	0.563	0.186	0.592	0.682	1.000	0.278	0.349	0.594	0.194	0.530	0.538	0.475	0.569
QL	0.427	0.264	0.485	0.558	0.473	0.414	0.398	0.321	0.460	0.293	0.479	0.488	0.516	0.278	1.000	0.303	0.375	0.310	0.290	0.381	0.221	0.230
QR	0.115	0.127	0.253	0.341	0.286	0.116	0.042	0.111	0.314	0.402	0.153	0.299	0.337	0.349	0.303	1.000	0.330	0.266	0.138	0.292	0.221	0.119
RSH	0.668	0.853	0.769	0.840	0.413	0.771	0.774	0.410	0.614	0.430	0.440	0.808	0.804	0.594	0.375	0.330	1.000	0.618	0.768	0.629	0.466	0.662
RSI	0.572	0.521	0.727	0.682	0.322	0.547	0.548	0.061	0.468	0.434	0.489	0.718	0.658	0.194	0.310	0.266	0.618	1.000	0.503	0.468	0.199	0.418
RP	0.834	0.669	0.704	0.698	0.489	0.760	0.769	0.326	0.602	0.517	0.602	0.734	0.713	0.530	0.290	0.138	0.768	0.503	1.000	0.592	0.145	0.483
SH	0.597	0.477	0.812	0.760	0.395	0.822	0.781	0.322	0.973	0.211	0.715	0.802	0.767	0.538	0.381	0.292	0.629	0.468	0.592	1.000	0.393	0.484
SNCRS	0.091	0.368	0.382	0.455	0.254	0.342	0.330	0.125	0.365	0.175	0.057	0.366	0.372	0.475	0.221	0.221	0.466	0.199	0.145	0.393	1.000	0.815
STRS	0.439	0.606	0.585	0.614	0.523	0.592	0.582	0.258	0.453	0.347	0.195	0.580	0.553	0.569	0.230	0.119	0.662	0.418	0.483	0.484	0.815	1.000

TABLE 2. The absolute values of cosine similarities between the classes from DBUtils 1.3.

over the traditional models is that it considers the semantics of the words or, more formally, the distance between the words [21]. Therefore, *private* will be closer to *protected* than to *boolean*. An additional advantage over bag-of-words is that it also takes into consideration the words order, at least in a small context.

In our experiment with DBUtils 1.3, we computed feature vectors consisting of 300 numerical features. We give in Table 2 the absolute values of the cosine similarities between all pairs of feature vectors.

Our focus is to test if an *unsupervised learning* model is able to capture the *coupling* relationship between the application classes thus avoiding to limit the definition of *coupling* to a predefined similarity function (like cosine similarity). A *self-organizing map* will be used in our experiment as an *unsupervised learning* model. SOMs [30] are a type of artificial neural network which are trained to provide a low-dimensional representation of the input space, called a *map* [10]. The main characteristic of a SOM is that it preserves the topological ordering of the input data, more exactly the input instances which are close to each other in the input space will also be close to each other on the output map.

The 22 application classes from DbUtils 1.3 are mapped on a SOM having a *torus* topology. For visualizing the SOM, the U-Matrix method [18] is used. Figure 6 illustrates the U-Matrix visualization of the SOM trained on the application classes from DbUtils 1.3. The darker regions on the U-Matrix represent data that are similar while the data falling in the lighter regions are dissimilar. Visualizing the U-Matrix for the resulting map, we observe three regions corresponding to the three packages presented in Table 1. The classes from the *default* package are displayed in *red*, those from the *handlers* package in *green*, while the third package *wrappers* is marked with *blue*.

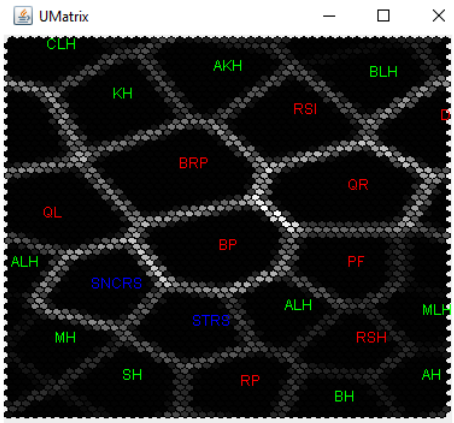


FIGURE 6. U-Matrix visualization.

Analyzing the U-matrix from Figure 6 we observe two application classes (*RowProcessor* and *ResultSetHandler*) which seem to be misplaced in the *handlers* package. But these two misplacements are explainable, considering the conceptual coupling measurement we used in our experiment. The *ResultSetHandler* class is conceptually coupled to the classes from the *handlers* package and this type of coupling is deduced from its source code. The *RowProcessor* class is close on the map to the *BeanHandler* class. Analyzing the source code of the *BeanHandler* class we found that it contains an attribute of type *RowProcessor*, which justifies their closeness on the map. Moreover, inspecting the source code of *RowProcessor* class, we observe that it operates with (Java) *beans* and this is expressed in its code (identifiers, comments, etc). Thus, the representation of the classes using `Doc2Vec` captured the conceptual relationship between the classes. We can conclude that the map depicted in Figure 6 empirically confirms our hypothesis that unsupervised *machine learning* models (the *self-organizing map*, in our case) are able to express *dependencies* (*conceptual*, in our case) between software entities. For capturing the direct coupling between software entities, we should consider not only the *conceptual* coupling, but also the *structural* one.

In our experiment we have focused only on *direct* dependencies, but we are confident that using an appropriate data representation (i.e. vectorial representation of the *application classes*), a SOM will be effective for depicting more complex dependencies (like *hidden dependencies*) from a software system. Further work will investigate different vectorial representations for the

software entities which are appropriate for capturing more complex software dependencies.

5. Conclusions and future work

This paper presented in detail the problem of identifying hidden dependencies in software systems, a problem of major importance during the maintenance and evolution of software systems. We discussed about evaluating the performance of the detection process and we identified the main limitations of the existing state-of-the-art approaches.

We proposed a new computational model based on clustering for the problem of *hidden dependencies identification*. Such a *machine learning* perspective has not been proposed in the literature so far. As further work we will investigate *software metrics* useful in the *Data representation* step from our approach, as well as different clustering algorithms useful in the *Grouping* step. Regarding the *impact analysis*, we target to develop coupling measurements which capture both the structural and the conceptual aspects of coupling.

Acknowledgments

This work was supported by a grant of the Romanian National Authority for Scientific Research, CNCS−UEFISCDI, project number PN-II-RU-TE-2014-4-0082.

References

[1] Commons DbUtils. http://commons.apache.org/proper/commons-dbutils/index.html.

[2] RaRe TECHNOLOGIES. https://github.com/RaRe-Technologies/gensim.

[3] Shirin Akbarinasaji, Behjat Soltanifar, Bora Çağlayan, Ayse Basar Bener, Andriy Miranskyy, Asli Filiz, Bryan M. Kramer, and Ayse Tosun. A metric suite proposal for logical dependency. In *Proceedings of the 7th International Workshop on Emerging Trends in Software Metrics*, WETSoM '16, pages 57–63, New York, NY, USA, 2016. ACM.

[4] Taweesup Apiwattanapong, Alessandro Orso, and Mary Jean Harrold. Efficient and precise dynamic impact analysis using execute-after sequences. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 432–441, New York, NY, USA, 2005. ACM.

[5] A. Beer and S. Mohacsi. Efficient test data generation for variables with complex dependencies. In *2008 1st International Conference on Software Testing, Verification, and Validation*, pages 3–11, April 2008.

[6] Jonathan Bell. *Making Software More Reliable by Uncovering Hidden Dependencies*. PhD thesis, Graduate School of Art and Sciences, Columbia University, 2016.

[7] Lionel C. Briand, Juergen Wuest, and Hakim Lounis. Using coupling measurement for impact analysis in object-oriented systems. In *Proceedings of the IEEE International Conference on Software Maintenance*, ICSM '99, pages 475–482, Washington, DC, USA, 1999. IEEE Computer Society.

[8] Daniel Conte de Leon and Jim Alves-Foss. Hidden implementation dependencies in high assurance and critical computing systems. *IEEE Trans. Softw. Eng.*, 32(10):790–811, October 2006.

[9] D. Conte de Leon and J. Alves-Foss. Hidden implementation dependencies in high assurance and critical computing systems. *IEEE Transactions on Software Engineering*, 32(10):790–811, Oct 2006.

[10] N. Elfelly, J.-Y. Dieulot, and P. Borne. A neural approach of multimodel representation of complex processes. *International Journal of Computers, Communications & Control*, III(2):149–160, 2008.

[11] Harald Gall, Karin Hajek, and Mehdi Jazayeri. Detection of logical coupling based on product release history. In *Proceedings of the International Conference on Software Maintenance*, ICSM '98, pages 190–, Washington, DC, USA, 1998. IEEE Computer Society.

[12] Harald Gall, Karin Hajek, and Mehdi Jazayeri. Detection of logical coupling based on product release history. In *Proceedings of the International Conference on Software Maintenance*, ICSM '98, pages 190–198, Washington, DC, USA, 1998. IEEE Computer Society.

[13] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[14] Ahmed E. Hassan and Richard C. Holt. Predicting change propagation in software systems. In *Proceedings of the 20th IEEE International Conference on Software Maintenance*, ICSM '04, pages 284–293, Washington, DC, USA, 2004. IEEE Computer Society.

[15] Hui He, Dongyan Zhang, Min Liu, Weizhe Zhang, and Dongmin Gao. A coverage and slicing dependencies analysis for seeking software security defects. *The Scientific World Journal*, 2014:1–10, 2014.

[16] Jakob Jenkov. Understanding Dependencies. Technical report, Tech and Media Labs, 2014.

[17] Huzefa Kagdi, Michael L. Collard, and Jonathan I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *J. Softw. Maint. Evol.*, 19(2):77–131, March 2007.

[18] S. Kaski and T. Kohonen. Exploratory data analysis by the self-organizing map: Structures of welfare and poverty in the world. In *Neural Networks in Financial Engineering. Proceedings of the Third International Conference on Neural Networks in the Capital Markets*, pages 498–507. World Scientific, 1996.

[19] Serkan Kirbas, Alper Sen, Bora Caglayan, Ayse Bener, and Rasim Mahmutogullari. The effect of evolutionary coupling on software defects: An industrial case study on a legacy system. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, pages 6:1–6:7, New York, NY, USA, 2014. ACM.

[20] Ehsan Kouroshfar, Mehdi Mirakhorli, Hamid Bagheri, Lu Xiao, Sam Malek, and Yuanfang Cai. A study on the role of software architecture in the evolution and quality of software. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, pages 246–257, Piscataway, NJ, USA, 2015. IEEE Press.

[21] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[22] Steffen Lehnert. A taxonomy for software change impact analysis. In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, IWPSE-EVOL '11, pages 41–50, New York, NY, USA, 2011. ACM.

[23] Thomas M. Mitchell. *Machine learning*. McGraw-Hill, Inc. New York, USA, 1997.
[24] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.
[25] Alessandro Orso, Taweesup Apiwattanapong, James Law, Gregg Rothermel, and Mary Jean Harrold. An empirical comparison of dynamic impact analysis algorithms. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 491–500, Washington, DC, USA, 2004. IEEE Computer Society.
[26] Maksym Petrenko and Vclav Rajlich. Variable granularity for improving precision of impact analysis. In *ICPC*, pages 10–19. IEEE Computer Society, 2009.
[27] Denys Poshyvanyk, Andrian Marcus, Rudolf Ferenc, and Tibor Gyimóthy. Using information retrieval based coupling measures for impact analysis. *Empirical Softw. Engg.*, 14(1):5–32, February 2009.
[28] Vaclav Rajlich. A model for change propagation based on graph rewriting. In *Proceedings of the International Conference on Software Maintenance*, ICSM '97, pages 84–91, Washington, DC, USA, 1997. IEEE Computer Society.
[29] Gabriela Serban, Alina Câmpan, and Istvan Gergely Czibula. A programming interface for finding relational association rules. *International Journal of Computers, Communications & Control*, I(S.):439–444, June 2006.
[30] Panu Somervuo and Teuvo Kohonen. Self-organizing maps and learning vector quantization for feature sequences. *Neural Processing Letters*, 10:151–159, 1999.
[31] R. Vanciu and V. Rajlich. Hidden dependencies in software systems. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10, Sept 2010.
[32] Lee White, Khaled Jaber, Brian Robinson, and Václav Rajlich. Extended firewall for regression testing: An experience report. *J. Softw. Maint. Evol.*, 20(6):419–433, November 2008.
[33] H. Y. Yang and E. Tempero. Indirect coupling as a criteria for modularity. In *Assessment of Contemporary Modularization Techniques, 2007. ICSE Workshops ACoM '07. First International Workshop on*, pages 10–10, May 2007.
[34] Hong Yul Yang, E. Tempero, and R. Berrigan. Detecting indirect coupling. In *2005 Australian Software Engineering Conference*, pages 212–221, March 2005.
[35] Hong Yul Yang and Ewan Tempero. Measuring the strength of indirect coupling. In *Proceedings of the 2007 Australian Software Engineering Conference*, ASWEC '07, pages 319–328, Washington, DC, USA, 2007. IEEE Computer Society.
[36] Hong Yul Yang, Ewan Tempero, and Rebecca Berrigan. Detecting indirect coupling. In *Proceedings of the 2005 Australian Conference on Software Engineering*, ASWEC '05, pages 212–221, Washington, DC, USA, 2005. IEEE Computer Society.
[37] Zhifeng Yu and V. Rajlich. Hidden dependencies in program comprehension and change propagation. In *Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on*, pages 293–299, 2001.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: {istvanc,gabis,marianzsu}@cs.ubbcluj.ro,mdir1308@scs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 1, 2017
DOI: 10.24193/subbi.2017.1.08

USING COMPUTATIONAL INTELLIGENCE MODELS FOR ADDITIONAL INSIGHT INTO PROTEIN STRUCTURE

MARIA-IULIANA BOCICOR[1], ALESSANDRO PANDINI[2], GABRIELA CZIBULA[1], SILVANA ALBERT[1], AND MIHAI TELETIN[1]

Abstract. Proteins are large, complex molecules with crucial roles in the functioning of living organisms. Understanding the underlying mechanisms by which proteins achieve their structures and substructures, as well as those involved in the conformational transitions may contribute to a deeper comprehension of the involved biological processes. This paper investigates a new machine learning perspective upon analyzing protein conformational transitions and introduces a new formalization for the problem, with the more general goal of uncovering interesting patterns in protein conformational transitions. This study represents the starting point of a research which is being conducted in order to obtain a better comprehension of proteins' structures and, implicitly, functions, by investigating *computational intelligence* methods for analyzing and deducing proteins conformational transitions.

1. Introduction

Proteins are large, complex molecules with crucial roles in the functioning of living organisms: they can be building blocks in the body (structural proteins), they catalyze biochemical reactions in metabolism (enzymes) or they may execute key tasks in maintaining the cellular environment. Moments after a protein is synthesized it folds, forming a stable three-dimensional (3D) structure, which is known to define the protein's function and which is entirely dictated by the linear sequence of amino acids composing the protein [26]. According to various external factors from the protein's environment (e.g. temperature, interaction with other molecules), modifications in the protein structures occur during their biological functions. Thus, a protein

Received by the editors: May 14, 2017.

2010 *Mathematics Subject Classification.* 68T05, 62H30.

1998 *CR Categories and Descriptors.* I.2.6 [**Computing Methodologies**]: Artificial Intelligence – *Learning*; I.5.3 [**Computing Methodologies**]: Pattern Recognition – *Clustering*.

Key words and phrases. Protein conformations, Computational Intelligence, Machine learning, Self-organizing maps.

will acquire a limited number of alternative conformations (belonging to the same fold), having the ability to transition between them [25]. Understanding protein conformational transitions and protein dynamics is essential for the comprehension of biomolecular interactions. This is of paramount importance in the process of developing new drugs that can inhibit proteins' uncontrolled behaviour, which can arise in pathological cases (such as protein incorrect folding or mutations) [15].

The contribution of the paper is summarized as follows. Our first goal is to explore a new *machine learning* perspective upon studying protein conformational transitions. Starting from the current state-of-the-art which refers to the analysis of conformational changes in proteins, we propose a new computational model for the problem of predicting protein conformational transitions. Secondly, we aim to provide an intuition upon the applicability of *machine learning* techniques for uncovering interesting patterns in the structure of proteins. The study performed in this paper represents the starting point of a research which is being conducted in order to obtain a better comprehension of proteins' structures and, implicitly, functions, by investigating *computational intelligence* methods for analysing and deducing proteins conformational transitions. The long-term goal of our research is to contribute to a better understanding and to offer additional insight into the construction and functioning of proteins.

The rest of the paper is organized as follows. Section 2 presents the motivation of our approach, highlighting the importance and relevance of understanding *protein conformational transitions*, but the difficulty of the problem as well. The biological background related to our approach is given in Section 3. The current state-of-the-art, as well as the limitations of existing approaches related to the analysis of protein structure are presented in Section 4. Section 5 introduces our *machine learning* perspective on the problem, together with an incipient computational model. A case study which highlights the applicability of *machine learning* methods for analyzing protein conformational transitions is described in Section 6. The conclusions of the paper, as well as directions for continuing our research are pointed out in Section 7.

2. Motivation

Although the stable 3D structure of a protein is defined by a unique topology (i.e. fold), this structure is not static and it is now widely accepted that proteins are dynamic objects [25]. According to various external factors from the protein's environment (e.g. temperature, interaction with other molecules), modifications in proteins' structures occur during their biological functions. A protein will thus acquire a limited number of conformations and will have

the ability to transition between alternative conformations. Understanding protein dynamics and how these conformational transitions occur is essential for the comprehension of biomolecular interactions, which is of paramount importance in the process of developing new drugs that can inhibit proteins' uncontrolled behaviour [15].

When investigating the role of conformational transitions in biological function from a computational perspective, the first stage is devising a formalisation of the problem in question, which involves specific domain knowledge and thus a collaboration between biologists, chemists, physicists and computer scientists. Various formal abstractions of problems related to protein structure, or their equivalent transformations have been proven to be NP-hard or NP-complete [7, 8], which means that there are no algorithms which can solve these problems in realistic time. The complexity of the *protein conformational transitions* problem is further increased by the high dimensionality of the space to be explored. For such classes of problems, heuristic techniques inspired from artificial intelligence and mathematical optimisation are certainly suitable candidates. In addition to the difficulties mentioned above, obtaining sufficient relevant experimental biological data for thorough analyses and understanding is time-consuming and financially expensive.

Both the importance and the complexity of the problem motivate us to investigate the usefulness of *machine learning* models and methods for the analyzing and detecting the conformational changes in proteins. Our perspective on the problem is new, to the best of our knowledge it has not been investigated in the literature, yet. We are confident that *machine learning* based solutions are applicable and may lead to interesting and valuable information, due to these models' ability to discover hidden patterns in data.

3. Background

Proteins are large molecules, having significant roles in the structure, development and functioning of living organisms. They are composed of basic building blocks - *amino acids* - small molecules which chain together in order to create proteins. The amino acids sequence forms the primary structure of the protein, which can be represented as a string of symbols representing the 20 amino acids (they are encoded by the letters of the alphabet). Although the sequence of amino acids is linear, the protein does not have an extended conformation, as intramolecular forces between the amino acids lead to a folding of the protein. As soon as it is synthesized as a linear sequence of amino acids, a protein folds in a matter of seconds to a stable three dimensional structure called the protein's native state. This structure of the protein is very important, as it defines the protein's function. However, proteins are

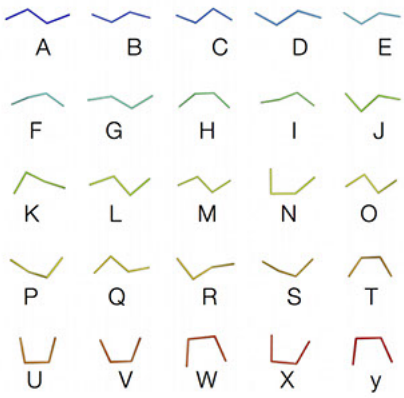


FIGURE 1. Structural elements and their associated symbols of the SA. Figure source: Alessandro Pandini [19].

dynamic molecules and undergo slight changes in their structures, according to the function they are fulfilling and depending on environmental conditions. Understanding and tracing these conformational changes (or transitions) could help us gain new insight into the way proteins function.

When studying protein conformations, it can be noticed that there are several frequently occurring conformations for small fragments. These so-called *states* have been determined with various methods and encoded in Structural Alphabets (SA) [18], which contain codes for the re-occurring short conformations. There are several types of SA, derived using various methods [16]. These are particularly useful in computational applications, as they allow representing a three dimensional structure via a one dimensional array (a sequence of characters of the alphabet), thus facilitating analysis of protein structure.

In our study we employ the structural alphabet derived by Pandini et al. in [18]. This is composed of 25 codes, represented by 25 letters of the (conventional) alphabet, each letter representing the short structural (three dimensional) element composed of four amino acids in the linear sequence of the protein. The structural elements and their associated letters of the SA are depicted in Figure 1. The structural element is characterized by the two angles between consecutive amino acids (more specifically, between the alpha carbon atoms of these amino acids) and by the torsion angle formed by all four atoms [18].

Let us consider a protein Pr, whose primary sequence is composed of n amino acids: $Pr = p_1p_2 \cdots p_n$. Then, a structural conformation of protein Pr can be represented as a sequence of letters of length $n-3$, where each letter encodes the structure formed by four amino acids in the primary sequence:

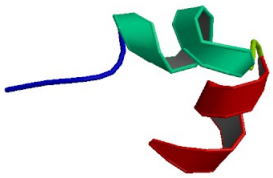


FIGURE 2. 3D view of protein 1HP9. Image from the RCSB PDB (`www.rcsb.org`) [2] of PDB ID 1HP9 [24] [2].

$\mathcal{C} = s_1 s_2 \cdots s_{n-3}$. Slight changes in a protein's conformation lead to different conformations (thus different representations). One could imagine a sliding window of length four, passing over the protein's structure and each group of four amino acids is represented by a symbol of the SA. An as example, we present protein $1HP9$ [1] (a toxin from scorpion venom), which has a short primary sequence (22 amino acids): GHACYRNCWREGNDEETCKERC. A three dimensional view of this protein is illustrated in Figure 2. Five possible SA representations of this protein from the analysis of conformations available in a database of molecular simulations [13] are shown below (the symbols in these representations are symbols of the structural alphabet [18]):

- `QSUWNSVVVPRIJUUVVUV`
- `QSUWNSVVVPRIJUUVUUV`
- `RSUWNSVVVPRIKUUVVUV`
- `QSUWNSVVVPRGKUUVVUV`
- `QSUWNSVVVPRGKUUVVUV`

It can be noticed that all these conformations have the same length of 19 symbols ($= 22 - 3$) and there are very slight differences among them. The amount of changes is consistent with the timescale of the original simulation: conformations were recorded at intervals of 1 picosecond.

4. LITERATURE REVIEW

Several theoretical models have been proposed for modelling conformational transitions, among which we mention those introduced by Miyashita et al. [14], Whtford et al. in [27], Skjaerven et al. [22]. These were employed by physics-based computational methods, such as molecular dynamics [17] or Monte Carlo [12] to simulate the movement of atoms. However, although having the potential to offer valuable information about protein structure, these simulations

[1] http://www.rcsb.org/pdb/explore/explore.do?structureId=1hp9

[2] This image is used according to RCSB PDB Policies & References: `http://www.rcsb.org/pdb/static.do?p=general_information/about_pdb/policies_references.html`.

are extremely computationally expensive and thus their time intervals are considerably shorter than those of real biological conformational changes. Normal Mode Analysis [22] and simplifications of it have been used in several cases for modelling protein conformational transitions: Schuyler et al. present in [21] a tool which is able to generate a transition pathway from a source to a destination conformation and Al-Bluwi et al. use in [1] robotics inspired methods (motion planning algorithms) to model conformational transitions.

Another technique presented by Haspel et al. in [9], who propose to trace conformational changes from a start to a goal conformational state by mapping the protein to a reduced representation, capturing low-energy conformations with the help of a coarse-grained physics based energy function and applying a sampling-based motion planning algorithm (again, inspired from robotics). The limitations of these later solutions are that they either use relatively simple energy functions (which thus only consider a small number of energy parameters), or they provide approximations of the paths, which require further refinement.

Raveh et al. introduce in [20] an approach called *PathRover* that, based on initial external constraints can generate motion pathways. The motion planning algorithm takes into account any available prior information and incorporates it into the algorithm of rapidly exploring random trees (RTT). This solution's main advantage is that by using initial constraints, it narrows down the search in high-dimensional spaces thus being significantly faster. They managed to do that by integrating their solution into *Rosetta* - modelling framework that aggregates algorithms for computational modeling and analysis of protein data. In order to successfully integrate it, they had to provide energy functions, optimising protocols and techniques for sampling.

The generated pathways are the result of partial data assimilation in sampling-based motion planning of molecules. As a result, each pathway has to form a sequence that satisfies all the initial restrictions while consisting of clash-free low-energy conformations. The challenge still remains in extracting physical features from simulated motion and being able to bridge experimental and computational observations. Significantly less options are explored in [20] because of the use of partial input but there is no learning involved based on existing findings.

Cortés et al. propose in [4] a computational approach based on path planning. The technique is intended to predict the motions of the molecules of the proteins. It is mentioned that motion planning techniques have lots of applications in computational biology and that they can be successfully applied on protein study. The proposed approach is split in two main stages, a *geometric filtering phase* and an *energy based computation* applied only on the

solutions extracted from the first stage. One of the advantages of this split is the increase in computational speed. The approach analysis shown that the filtering stage is very effective and that it is capable to present very important knowledge to biologists. However, the second stage still has some limitations, since it cannot exploit all the provided knowledge.

The study we conducted on the current state-of-the-art on the problem of identifying proteins conformational transitions revealed that a *machine learning* based computational model has not been investigated in the literature, yet.

5. Theoretical model. Our proposal

As opposed to other approaches in the literature (Section 4), we tackle the problem of determining conformational transitions in proteins from a different angle and we derive a different formalization for it, starting from a data set of more than 300 proteins and their associated conformations. As described in Section 3, a protein Pr or length n can be viewed as a word over the alphabet of 20 letters representing amino acids $\mathcal{A} = \{G, P, A, V, L, I, M, C, F, Y, W, H, K, R, Q, N, E, D, S, T\}$: $Pr = p_1 p_2 \dots p_n$, where $p_i \in \mathcal{A}, \forall i \in \{1, 2, \dots, n\}$.

For each protein we are given thousands of different conformations, obtained by molecular dynamics simulations. Each conformation is converted into its SA representation. The structural alphabet is composed of the 25 letters shown in Figure 1: $\mathcal{SA} = \{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y\}$. It is important to remark that although same symbols are being used both for amino acids and for structural elements, these are actually completely different concepts (this is important to be remembered when processing and experimenting on the data).

For each protein $Pr = p_1 p_2 \dots p_n$ in the data set, we are given a large number m of experimentally determined conformations (for the data set we use, $m = 10000$). Therefore, for each protein we have a set $\mathcal{S} = \{c_j \mid c_j = c_j^1 c_j^2 \dots c_j^{n-3},\ j \in \{1, 2, \dots, m\},\ c_j^k \in \mathcal{SA},\ k \in \{1, 2, \dots n-3\}\}$ of conformations. Considering all these conformations, a distribution matrix is computed for each protein, which holds information about the SA elements' distribution, for each position k, $\forall k \in \{1, 2, \dots, n-3\}$. This frequency matrix can be interpreted as a "profile" of the protein dynamics where for each fragment position we have a probabilistic measure of the occurrence of each letter in the alphabet. An example of such a matrix, for the 5 conformations of the protein $1HP9$ presented in Section 3, is given in Table 1. For each position in the SA representation we compute the probability of occurrence of each symbol of

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
G	0	0	0	0	0	0	0	0	0	0	0	0.4	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0.6	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0	0	0.4	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0.6	0	0	0	0	0	0
N	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Q	0.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0.2	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
S	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
U	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0
V	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	0	1
W	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TABLE 1. Distributions of SA symbols for the example presented in Section 3.

the SA on that specific position. For simplicity, in Table 1 we only show the distributions of the symbols occurring in the 5 conformations, but on a real world example, all symbols of the SA are considered.

Furthermore, other useful biological information about the protein and the composing amino acids can be considered (structure related information). For instance, a property an amino acid is characterized by is the relative solvent accessibility (RSA), which measures the solvent exposure of the amino acid. This property is numerical and it can have different values for the same amino acid, belonging to different structural environments. Another example would be the amino acid's hydrophobicity, a physical property which measures how much the amino acid is repelled by water. This is important, as hydrophobic forces are decisive factors in the protein folding process.

Considering the input information described above, we formulate the problem of determining protein conformational transitions as follows:

- *Given:*
  - A protein, as a string of amino acids.
  - Other structurally significant, biologic characteristics of amino acids (e.g. RSA values, hydrophobicity).
  - A *small* number of conformations (e.g. 10 conformations, determined using molecular dynamic methods).
- *The requirement is to solve any or both problems below:*

- Generate a matrix of probability distributions similar to the one presented in Table 1, corresponding to all m possible conformations (even though these are not known).
- Generate all m possible conformations for the protein.

Our aim is to further formalize the problem, considering various combinations of possible input data in order to be able to approach it from a machine learning perspective. Nonetheless, both requirements are difficult and conventional machine learning techniques are very probably not sufficient for satisfying results, therefore a more thorough investigation, as well as new or hybrid techniques are demanded in order to solve any of the two formulations of the problem.

6. Experiments

In this section we aim to give an empirical confirmation of our hypothesis that *machine learning* methods are applicable for analyzing proteins conformational transitions. More specifically, our focus is to highlight that *unsupervised learning* methods are able to capture patterns among the conformations of the same protein, as well as relationships between related proteins, relations which are confirmed from a biological perspective.

We considered an experiment consisting of *seven* proteins (codes: 1ASH, 1DLW, 1ECA, 1C52, 1CCR, 1APQ, 1COU in the Protein Data Bank [2]), taken from three different superfamilies (1.10.490.10, 1.10.760.10, 2.10.25.10). The superfamilies for the proteins were determined using **CATH Protein Structure Classification** database [3] which is a publicly available online resource that provides information on the evolutionary relationships of protein domains [5]. In this database, two proteins are considered in the same superfamily if there is a similarity between their three-dimensional structure [11].

Table 2 illustrates the superfamilies for the seven proteins considered in our experiment, as well as the similarity index between the proteins belonging to the same superfamily, as provided by the FATCAT algorithm (Flexible structure AlignmenT by Chaining Aligned fragment pairs allowing Twists) [28].

From Table 2 we observe that the proteins from the first two families have a similarity index about 20%, while the proteins from the third family have the lowest similarity index of about only 5%.

In order to test our hypothesis that *unsupervised learning* models are able to capture the biological relationships between proteins data, we performed the following experiment.

#	Superfamily	Proteins	Similarity index
1	**1.10.490.10**	{1ASH, 1DLW, 1ECA}	1ASH - 1DLW: 20.57% 1ASH - 1ECA: 25.85% 1ECA - 1DLW: 19.08%
2	**1.10.760.10**	{1C52, 1CCR}	1C52 - 1CCR: 27.10%
3	**2.10.25.10**	{1APQ, 1COU}	1APQ - 1COU: 4.92%

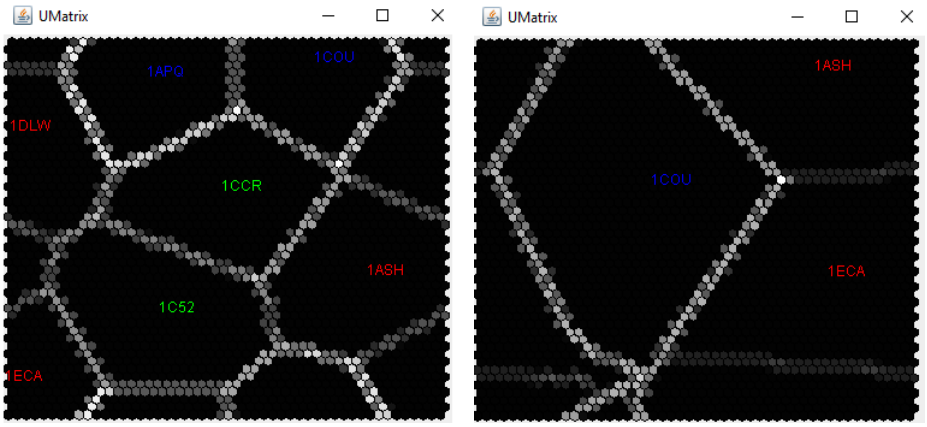
TABLE 2. Sample proteins

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
0	0	0	0	0	0	0.021	0	0.031	0.021	0.031	0	0	0.052	0	0.052	0.042	0.063	0.105	0	0.221	0.305	0.052	0	0

TABLE 3. Probabilities of occurrence of SA symbols for the example presented in Section 3.

We are further considering the theoretical model introduced in Section 5. For each protein, 10000 conformational transitions are known. The specific data we use is retrieved from MoDEL, a database which includes representatives from different protein families and fold arrangements [13]. Our current experiment's goal is to investigate whether biologically relevant correlations could be found within the given numerical data and to mine this given data in order to discover significant signals than can later be used by machine learning strategies to solve the problem described and defined in Sections 3 and 5. For this purpose, we use a further simplified representation of a protein: instead of the frequency matrix, we use a frequency vector, constructed as follows. For each of the 25 letters l_i ($1 \leq i \leq 25$) from the structural alphabet and each protein Pr, we compute the probability $p_{l_i}^{Pr}$ of occurrence of each letter l_i in the conformational transitions of protein Pr. Thus, a protein Pr may be visualized as a 25-dimensional vector containing the probabilities of occurrence of the symbols from the structural alphabet in the given protein, $Pr = (p_{l_1}^{Pr}, p_{l_2}^{Pr}, \ldots, p_{l_{25}}^{Pr})$. For the protein example presented in Section 3 (1HP9), including the 5 presented conformations, the frequency vector is presented in Table 3.

Considering the above modelling, each of the seven proteins considered in our case study is represented as a multi-dimensional vector. Our focus is to test if the conformational transitions of the proteins provide useful information regarding their three-dimensional structure and if an *unsupervised learning* model is able to capture this type of biological relationships between the proteins.



(A) Proteins 1ASH, 1DLW, 1ECA, 1C52, 1CCR, 1APQ and 1COU.

(B) Proteins 1ASH, 1ECA and 1COU.

FIGURE 3. U-Matrix visualization.

We will use a *self-organizing map* (SOM) as an *unsupervised learning* model. SOMs are known to be powerful *data mining* tools for visualizing high-dimensional data. A *self-organizing map* [23] is a type of artificial neural network that is trained using *unsupervised learning* to provide a low-dimensional representation of the high-dimensional input space, called a *map* [6]. The *topological mapping* is the main characteristic of the unsupervised mapping provided by a SOM, more exactly the input samples which are close to each other in the input space will be mapped into neighboring neurons on the output map.

6.1. **Results and discussion.** We mapped the *seven* proteins described above (considering their 25-dimensional representations) on a SOM having a *torus* topology. For the SOM visualization, we use the U-Matrix method [10] with the following interpretation: the lighter regions express data that are dissimilar while darker regions contain data that are similar.

Figure 3a depicts the U-Matrix visualization of the SOM trained on the *seven* proteins. Visualizing the U-Matrix for the resulting map, we clearly observe three regions corresponding to the three protein families described in Table 2.

Figure 3b illustrates the U-Matrix visualization of the SOM trained on only *three* proteins: 1ASH, 1ECA and 1COU. From these, only the first two belong to the same superfamily. This can be visualized on the U-Matrix, since there is

a clear separating boundary between the protein 1COU and the class formed by the other two proteins.

The results previously described and depicted in Figures 3a and 3b indicate the potential of unsupervised *machine learning* models (the *self-organizing map*, in our case) to uncover patterns encoded in the conformational transitions of proteins.

7. Conclusions and future work

In this paper we have investigated the problem of analyzing the conformational transitions of proteins, with the more general goal of contributing to a comprehensive understanding of the problem. We presented the current state-of-the-art approaches and we proposed a new computational perspective on the problem, based on *machine learning*. Our proposal represents the starting point of a research initiated on the topic approached in this paper, our long-term goal being to offer additional insight into the construction and functioning of proteins.

We also highlighted, through a *data mining* experiment, that the information obtained through analyzing proteins conformational transitions capture the relationships between related proteins, relations which are confirmed from a biological perspective.

Starting from the computational model proposed in Section 5, future work will be done in order to apply concrete supervised *machine learning* methods (e.g. *artificial neural networks*, *support vector machines*) for predicting the conformational transitions of proteins, as well as the matrix of probability distributions associated to protein conformations.

References

[1] Ibrahim Al-Bluwi, Marc Vaisset, Thierry Siméon, and Juan Cortés. Modeling protein conformational transitions by a combination of coarse-grained normal mode analysis and robotics-inspired methods. *BMC Structural Biology*, 13(1):S2, 2013.

[2] H.M Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28:235–242.

[3] CATH: Protein Structure Classification Database at UCL. CATH - Gene3D. `http://www.cathdb.info`.

[4] J. Cortés, T. Siméon, V. Ruiz De Angulo, D. Guieysse, M. Remaud-Siméon, and V. Tran. A path planning approach for computing large-amplitude motions of flexible molecules. *Bioinformatics*, 21(1):116–125, January 2005.

[5] Natalie L. Dawson, Tony E. Lewis, Sayoni Das, Jonathan G. Lees, David Lee, Paul Ashford, Christine A. Orengo, and Ian Sillitoe. Cath: an expanded resource to predict protein function through structure and sequence. *Nucleic Acids Research*, 45(D1):D289, 2017.

[6] N. Elfelly, J.-Y. Dieulot, and P. Borne. A neural approach of multimodel representation of complex processes. *International Journal of Computers, Communications & Control*, III(2):149–160, 2008.

[7] Aviezri S. Fraenkel. Complexity of protein folding. *Bulletin of Mathematical Biology*, 55(6):1199 – 1210, 1993.

[8] Christophe Guyeux, Nathalie M.-L. Cote, Jacques M. Bahi, and Wojciech Bienia. Is protein folding problem really a NP-complete one? First investigations. *Journal of Bioinformatics and Computational Biology*, 12(01):1350017–1350041, 2014.

[9] N. Haspel, M. Moll, M. L. Baker, W. Chiu, and L. E. Kavraki. Tracing conformational changes in proteins. In *2009 IEEE International Conference on Bioinformatics and Biomedicine Workshop*, pages 120–127, Nov 2009.

[10] S. Kaski and T. Kohonen. Exploratory data analysis by the self-organizing map: Structures of welfare and poverty in the world. In *Neural Networks in Financial Engineering. Proceedings of the Third International Conference on Neural Networks in the Capital Markets*, pages 498–507. World Scientific, 1996.

[11] Michael Knudsen and Carsten Wiuf. The CATH database. *Human Genomics*, 4:207–212, 2010.

[12] I. Lotan, F. Schwarzer, and J.C. Latombe. Efficient energy computation for monte carlo simulation of proteins. *Lecture Notes in Computer Science*, 2812:354–373,, 2003.

[13] Tim Meyer, Marco D'Abramo, Adam Hospital, Manuel Rueda, Carles Ferrer-Costa, Alberto Prez, Oliver Carrillo, Jordi Camps, Carles Fenollosa, Dmitry Repchevsky, Josep Lluis Gelp, and Modesto Orozco. MoDEL (molecular dynamics extended library): A database of atomistic molecular dynamics trajectories. *Structure*, 18(11):1399 – 1409, 2010.

[14] Osamu Miyashita, Peter G. Wolynes, and Jos N. Onuchic. Simple energy landscape model for the kinetics of functional transitions in proteins. *The Journal of Physical Chemistry B*, 109(5):1959–1969, 2005.

[15] G. Morra, M. Meli, and G. Colombo. Molecular dynamics simulations of proteins and peptides: from folding to drug design. *Current Protein and Peptide Science*, 9:2181–2196, 2008.

[16] B. Offmann, M. Tyagi, and A.G. de Brevern. Local protein structures. *Current Bioinformatics*, 2(3):165–202, 2007.

[17] Kei-ichi Okazaki, Nobuyasu Koga, Shoji Takada, Jose N. Onuchic, and Peter G. Wolynes. Multiple-basin energy landscapes for large-amplitude conformational motions of proteins: Structure-based molecular dynamics simulations. *Proceedings of the National Academy of Sciences*, 103(32):11844–11849, 2006.

[18] A. Pandini, A. Fornili, and J. Kleinjung. Structural alphabets derived from attractors in conformational space. *BMC Bioinformatics*, 11(97):1–18, 2010.

[19] Alessandro Pandini. Structural alphabet tools for molecular simulations. `http://people.brunel.ac.uk/~csstaap2/software.html`. [Online; accessed 12-May-2017].

[20] B. Raveh, A. Enosh, O. Schueler-Furman, and D. Halperin. Rapid sampling of molecular motion with prior information constraints. *PLoS Computational Biology*, 5(2), February 2009.

[21] Adam D. Schuyler, Robert L. Jernigan, Pradman K. Qasba, Boopathy Ramakrishnan, and Gregory S. Chirikjian. Iterative cluster-nma: A tool for generating conformational transitions in proteins. *Proteins: Structure, Function, and Bioinformatics*, 74(3):760–776, 2009.

[22] Lars Skjaerven, Siv M. Hollup, and Nathalie Reuter. Normal mode analysis for proteins. *Journal of Molecular Structure: {THEOCHEM}*, 898(13):42 – 48, 2009.

[23] Panu Somervuo and Teuvo Kohonen. Self-organizing maps and learning vector quantization for feature sequences. *Neural Processing Letters*, 10:151–159, 1999.

[24] K.N. Srinivasan, V. Sivaraja, I. Huys, T. Sasaki, B. Cheng, T.K. Kumar, K. Sato, J. Tytgat, C. Yu, B.C. San, S. Ranganathan, H.J. Bowie, R.M. Kini, and P. Gopalakrishnakone. kappa-hefutoxin1, a novel toxin from the scorpion heterometrus fulvipes with unique structure and function. importance of the functional diad in potassium channel selectivity. *J.Biol.Chem*, 277:30040–30047, 2002. PDB ID: 1HP9.

[25] Nobuhiko Tokuriki and Dan S. Tawfik. Protein dynamism and evolvability. *Science*, 324(9524):203–207, 2009.

[26] D. Voet and J. Voet. *Biochemistry*. Wiley, 4 edition, 2011.

[27] Paul C. Whitford, Osamu Miyashita, Yaakov Levy, and Jos N. Onuchic. Conformational transitions of adenylate kinase: Switching by cracking. *Journal of Molecular Biology*, 366(5):1661 – 1671, 2007.

[28] Yuzhen Ye and Adam Godzik. FATCAT: a web server for flexible structure comparison and structure similarity searching. *Nucleic Acids Research*, 32:582–585, 2004.

[1] Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: {iuliana, gabis, albert.silvana}@cs.ubbcluj.ro, tmic1334@scs.ubbcluj.ro

[2] Department of Computer Science, Brunel University, London, England

E-mail address: alessandro.pandini@brunel.ac.uk