On the Problem of Modeling Structured Data with the MinSOD Representative

Guido Del Vescovo, Lorenzo Livi, Fabio Massimo Frattale Mascioli, and Antonello Rizzi

***Abstract*—The possibility to face pattern recognition problems directly on structured domains (e.g., multimedia data, strings, graphs) is fundamental to the effective solution of many interesting applications. In this paper, we deal with a clustering problem defined in the string domain, focusing on the problem of cluster representation in data domains where only a dissimilarity measure can be fixed. To this aim, we adopt the MinSOD (Minimum Sum of Distances) cluster representation technique, which defines the representative as the element of the cluster minimizing the sum of dissimilarities from all the other elements in the considered set. Since the precise computation of the MinSOD have a high computational cost, we propose a suboptimal procedure consisting in computing the representative of the cluster considering only a reduced pool of samples, instead of the whole set of objects in the cluster. We have carried out some tests in order to ascertain the sensitivity of the clustering procedure with respect to the number of samples in the pool used to compute the MinSOD. Results show a good robustness of the proposed procedure. The implementations are available as part of the SPARE library, which is available as an open source project.**

***Index Terms*—Clustering strings, MinSOD representative, software library.**

I. Introduction

The recent research on Pattern Recognition and Inductive Modeling has defined effective systems able to deal with the R^n vector space. However, many interesting applications, coming for instance from computational biology, multimedia intelligent processing and computer vision, deal with structured patterns, such as, images, audio and video sequences, strings and labeled graphs [1], [2]. Usually, in order to take advantage of the existing data driven modeling systems, each pattern of a structured domain X is reduced to a set of real valued features by adopting a preprocessing function, tailored on the specific application. The design of this preprocessing procedure needs a deep expertise and it is a critical task, since useful information for the task at hand could be loss due to an excessive information compression. As a consequence, it is useful to design Pattern Recognition systems able to deal directly with structured domains. Consequently, it is fundamental the availability of effective models able to represent a set of samples belonging to X [3]. The abstraction of a cluster of objects by a representative model is useful for two main reasons. The first reason deals with computational issues. For example, comparing a particular pattern with the cluster representative instead of each single element in the cluster, of course results in a remarkable speed up. The second reason deals with the ability of a learning system to create abstract knowledge, a fundamental feature of the cerebral cortex in biological brains. The human ability to synthesize concepts describing a family of objects plays a key role in human reasoning.

Many ways are known for representing a cluster of samples in R^n considered as a metric space where, for instance, it is possible to define an Euclidean distance measure. In this case it is easy to define a way to represent a cluster, for example by the mean vector (centroid). As a consequence the sample to cluster distance can be conveniently defined as the Euclidean distance between the sample and the cluster centroid. Where necessary is it possible to adopt a more descriptive representative of a cluster by using second order statistical moments such as the covariance matrix and adopting the Mahalanobis distance for measuring sample to cluster dissimilarities. Other examples of representatives include hyperboxes, rough sets, etc. [3].

When dealing with data domains different from R^n, like structured data domain as strings and graphs, or multimedia data, using obvious and efficient ways of representing a cluster is not always possible. The element in the cluster that minimizes the Sum Of Distances (SOD) between itself and the other elements is a natural candidate for representing the cluster where more powerful abstraction techniques are not available [4]-[6]. The determination of the SOD-minimizing element can be seen as a *meta-algorithm*, since it only refers to the definition of a dissimilarity measure between samples. For this reason, the availability of a reliable and efficient implementation of this meta-algorithm is interesting, since it immediately allows the application of a number of Pattern Recognition and Machine Learning algorithms in a vast range of data domains. In fact, each inference algorithm relying on the availability of a cluster modeling method can be applied using the SOD-minimizing element as the cluster representative.

In this paper, we propose a suboptimal procedure for computing the MinSOD representative of the cluster, considering only a reduced pool of samples. We evaluate the algorithm in the framework of clustering of strings, by analyzing a suited performance measure. All implementations are available as part of the SPARE C++ library, which is briefly introduced in the next section.

II. The *SPARE* Library: A Brief Overview

SPARE – Something for PAttern REcognition – is a C++

Manuscript received July 10, 2013; revised September 15, 2013.

The authors are with Department of Information Engineering, Electronics and Telecommunications, SAPIENZA University of Rome, Via Eudossiana 18, 00184 Rome, Italy (e-mail: delvescovo, livi@diet.uniroma1.it, antonello.rizzi@uniroma1.it, mascioli@infocom.uniroma1.it).

library conceived as the core of a software framework for rapid development of Pattern Recognition and Computational Intelligence applications. *SPARE* is based on *concept-driven* template meta-programming technique, aiming to provide basic classes and procedures for rapid application development when dealing with clustering algorithms, inductive modeling systems (classification and function approximation) and optimization problems by neural networks, fuzzy logic and evolutionary computation. The set of concepts and classes have been thought in order to capture the essence of some basic algorithms, and reflects the fact that even the most complex data driven modeling system can be conceived and redesigned as an organized ensemble of small code blocks, each of them implementing specific functionalities with a standard interface. Designing a library with concepts in mind allows creating a set of classes that are not relegated to a fixed hierarchy. Concepts are non-invasive design choice. The meta-algorithms implemented in *SPARE* can be easily used to build Pattern Recognition applications dealing with customizable data spaces, without any necessity to represent objects in the real valued vector space. *SPARE* is an open source software library, released under the GNU GPL 3.0 license. The main site of the project is located at *http://libspare.org/*, where you can obtain a version of the *SPARE* library, as well as all the available documentation and tutorials.

III. The *K*-Means Clustering Procedure as a Meta-Algorithm

In the present paper, the well-known *k*-means algorithm [7] is employed in a version retaining the original clustering procedure, but not strictly referring to the R^n Euclidean space environment providing the support domain in which the *k*-means is usually presented and described. The support domain is seen as a general data space, in which some facilities like dissimilarity measures and cluster abstraction methods are defined. In this sense we talk about the *k*-means as a *meta-algorithm*.

A. Clustering Structured Data by K-Means

First, the clustering problem definition in the Euclidean space is briefly recalled. Given a finite input set $X=\{x_1, ..., x_n\}$, where $x_i \in R^n$, $i=1, ..., n$, a clustering problem consists in finding a *partition* $P=\{C_1, ..., C_k\}$ of this set in a number $k \leq n$ of clusters, with $X = \bigcup_{i=1}^{k} C_i$ and $C_i \cap C_j = \varnothing$, $\forall i, j, i \neq j$ so as to minimize the within-cluster sum of squares (WCSS)

$$\arg\min_{P} \sum_{i=1}^{k} \sum_{x_j \in C_i} \left\| x_j - \mu_i \right\|^2 \tag{1}$$

where μ_i is the *mean* vector of the *i-th* cluster C_i. This number k is called the *order* of the clustering and can be defined a priori or identified at runtime by the algorithm itself. The well known *k*-means algorithm [2] takes the number of clusters as an argument. It is a two-phase iterative *heuristic* algorithm that is based on the assignment of each input pattern x_i to the cluster with the closest mean μ_i. After the *assignment* phase follows the *update* of the means of each cluster for a maximum number of predefined steps.

In order to define clustering algorithms not necessarily dealing with the Euclidean space domain, we observe some general (*meta*) elements which play a fundamental role in the clustering process. Each cluster is modeled by a *representative*, i.e. an element not necessarily belonging to the problem data space, able to characterize the set of patterns in the cluster. The other important concept is the *dissimilarity measure* used to compute the distance between patterns and representatives. It is easy to understand that these two definitions of representative of a cluster and the sample-to-cluster dissimilarity strictly depend on the domain of the problem. For example, if $X=G$, where G is a set of graphs, the problem of deriving a representative graph is known as the *set median graph* computation [1], [5], [6].

Abstracting the input domain, it is possible to define a *generic k-means* algorithm that uses a generic representative object, configured with a generic dissimilarity measure defined between patterns and representatives. That is, it is possible to abstract the algorithm itself obtaining a *meta-algorithm* that will work on many different problems but preserving the main behavior. The same generalization could be applied virtually to any Pattern Recognition problem, such as the ones of classification and function approximation.

In Algorithm III.1 it is shown the pseudo-code of the generic *k*-means algorithm over a generic set *X*, configured with a set of representatives objects $\{\mu_1,...,\mu_k\}$ defined with a dissimilarity function $d : X \times X \to R_o^+$.

Algorithm III.1 Generic *k*-means

Input: A generic finite input set $X=\{x_1,\dots,x_n\}$, the order k of the clustering, a dissimilarity function $d : X \times X \to R_o^+$, the cluster representatives $\{\mu_1,...,\mu_k\}$, *MAX* number of allowed iterations.

Output: A partition $P=\{C_1,\dots,C_k\}$.

```
1: Initialize every representative $\mu_i$, $i=1,\dots,k$
2: $t=0$, $P = \varnothing$
3: loop
4: $t{+}{=}1$
5: Assignment Step: assign each sample $x_j$ to the cluster with the closest representative
6: $C_i^{(t)} = \{x_j : d(x_j, \mu_i) \leq d(x_j, \mu_k)$ for all $h=1,\dots,k\}$
7: Update Step: update the representatives
8: $\mu_i^{(t+1)} = R(C_i), i = 1 \to k$
9: Update the partition with the modified clusters: $P^t=\{C_1,\dots,C_k\}$
10: if $t \geq MAX \vee P^t = P^{t-1}$ then
11: return $P^t$
12: end if
13: end loop
```

The pseudo-code of the Algorithm III.1 shows that the dissimilarity evaluations between a sample x_j and a cluster C_i can be done matching the sample with the representative of this cluster, namely μ_i, using the dissimilarity function $d(\cdot;\cdot)$ (line 6), defined over the generic set *X*. The update of each representative is carried out by a generic operator $R(\cdot)$ (line 8), able to derive the representative of the modified cluster C_i. At each step *t* is induced a new partition P^t (line 9) since the stop condition is reached (line 10).

B. The Levenshtein Distance and the MinSOD Representative

Given a generic finite input set $X=\{x_1, ..., x_n\}$ of objects, the definition of the representative of this set vary considering different domains. For example, if $X \subset R^n$ the representative is simply the mean or centroid, that is, a vector $\underline{x}^* = \frac{1}{n}\sum_i \underline{x}_i$, $\underline{x}^* \in R^n$ that can be outside the set *X*. This problem can be generalized to any set of objects where it is possible to define a dissimilarity measure $d(\cdot;\cdot)$ considering this new formulation

$$x^* = \arg\min_{x_i \in X} \sum_j d\left(x_j, x_i\right) \qquad (2)$$

That is, the *set mean* in this case is taken as the element of the set *X* that minimizes the sum of the distances between the element itself and the other elements in the set. For brevity, we call this element MinSOD. To obtain the MinSOD we need to execute a quadratic number of distance calculations. That is, the total complexity depends on the complexity of $d(\cdot;\cdot)$. If we consider graphs, the distance between pairs of samples could be obtained with the well known Graph Edit Distance (GED) [8]-[10]. The computation of the (exact) GED is known to be in NP, and then the whole complexity is also in NP.

If $X=\Gamma$, where Γ is the set of strings defined on a finite alphabet Ω, the distances could be obtained with a quadratic algorithm using the *Levenshtein* edit distance [11], and the whole complexity become of the order $O(n^4)$. If *X* is a large set, the evaluation of the distance between every objects of *X* become prohibitive and some approximation mechanism must be taken into account.

In *SPARE* the MinSOD determination is achieved through a specific class, named *MinSod,* which models the *Representative* concept. The object performs an important computational speed up by only determining the MinSOD representative inside a reduced pool of samples, instead of the whole set of the samples inserted in the cluster so far. The reduced pool of samples is called the *cache* and its size is a relevant user-defined parameter. In order to work with a reduced set of samples, a *replacement policy* has to be defined to discard some samples from the pool as new samples are inserted in the set. The aim of this paper is to show that the adopted replacement policy allows a well behaved tracking of the MinSOD representative with a high automation degree, thanks to the low sensitivity to the cache size value. This statement is supported here by a test where the performance is evaluated on the basis of a quality parameter defined on a particular Pattern Recognition problem. The replacement policy of the elements adopted by the (default) *MinSod* class is simple and proved to assure uniform coverage of the input set. Each new element is inserted since the defined cache size is reached. Then, for each new sample an old one must be discarded. The old sample to discard is chosen as follows: two samples are chosen with uniform probability, and the one farthest away from the actual MinSOD representative is discarded. Preliminary tests made during the development of the class suggested that discarding deterministically the globally farthest sample from the representative can lead to poor tracking of the optimal representative due to undesired phenomena. In Algorithm III.2 is shown the pseudo-code of the behavior of the MinSod class of *SPARE*. It is worth to stress however that different MinSOD implementations are available in SPARE, providing additional and diverse functionalities.

Algorithm III.2 MinSOD Determination and Updating

Input: A generic finite input set $X=\{x_1,\ldots,x_n\}$, the size *M* of the cache *C*, a dissimilarity function $d : X \times X \rightarrow R_o^+$

Output: The MinSOD determination $\hat{x} \in X$

```
1: $C = \varnothing$
2: for all $x_i \in X$ do
3: if $|C|<M$ then
4: $C = C \cup \{x_i\}$
5: else
6: Select two elements, namely $\chi_1$ e $\chi_2$ from $C$ with uniform probability
7: if $d(\chi_1,\hat{x}) \geq d(\chi_2,\hat{x})$ then
8: $C=C \setminus \{\chi_1\}$
9: else
10: $C=C \setminus \{\chi_2\}$
11: end if
12: $C = C \cup \{x_i\}$
13: end if
14: $\hat{x} = MinSOD(C)$
15: end for
```

IV. Experiments

The aim of the presented tests is to show that the cache size parameter of the MinSOD implementation in the SPARE library is not critical, that is, its determination (under reasonable assumptions) does not affect critically the performance of the considered Pattern Recognition task. We will support this claim over a clustering problem defined over a synthetically-generated domain of strings (sequences of characters), considering two tests with different configurations and difficulties.

A. *Problem Definition and Synthetic Datasets*

Firstly, we need to describe how we generate the set of strings Γ. This set is generated using a stochastic procedure based on *Markov Chains* [12], [13]. If $|\Omega|=n$, the Markov generation process is entirely described by its *transition matrix* **T**, that is, if we are generating strings of length l, the first symbol is chosen with uniform probability on Ω, the next symbol is chosen with conditional probability $p(s_2|s_1)$, that is with a probability that depends only on the last selected symbol. A *time-homogeneous Markov chain* is characterized by the fact that the transition matrix does not change over time. The synthetic dataset used for our tests were generated using a time-homogeneous Markov chain.

Let's consider to have two distinct stationary Markov chains that we see as *classes* of strings. Our clustering problem is then defined as a two cluster problem (k=2), where each cluster is labeled with one of the two Markov chains, and the objective is to group strings that comes from the same class, that is, are generated from the same Markov chain. This is an unsupervised learning task, where the distance between the strings is obtained with the Levenshtein distance.

Once the partition is obtained, a quality index which assumes values in [0.5, 1] (since we considered k=2) is evaluated for each cluster. The quality index takes into account the *purity* of the cluster in terms of the known class labels. It is defined by the number of samples belonging to the majority class in the cluster, divided by the total cardinality of the cluster. An index value of one indicates a cluster composed by samples of the same class only. An index value of 0.5 indicates a cluster which contains half of the samples from one class, half from the other. The overall quality index is taken as the worst quality index over all clusters. We will show the results obtained over two similar tests, with the second dataset representing a more difficult clustering problem with respect to the first one. We will observe how, and if, the cache size of the MinSOD will affect the performance measure.

All tests are carried out over an *Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz* with *4 Gb* of RAM.

B. *Results*

In the first, easier, test we generate 5000 strings per class, for a total of 10000 strings. The strings are built considering an alphabet of size three, i.e., $|\Omega|=3$, and are characterized by a variable-length varying between 18 and 22 symbols. The two 3×3 transition matrices which we used to generate the two classes of strings are defined as follows:

$$\mathbf{T}_1 = \begin{bmatrix} 0.1 & 0.1 & 0.8 \\ 0.8 & 0.1 & 0.1 \\ 0.1 & 0.8 & 0.1 \end{bmatrix} \quad \mathbf{T}_2 = \begin{bmatrix} 0.1 & 0.8 & 0.1 \\ 0.1 & 0.1 & 0.8 \\ 0.8 & 0.1 & 0.1 \end{bmatrix}$$

The cache size is decremented starting from a value of 50 going down to 1, with a decrement step of 1. For each size of the cache, the same test is carried out with five different random seeds. The results for the quality index are eventually taken as the average over these runs. The worst and better case scenario is also considered. The results of this test are shown in Fig. 1. It is possible to observe a clear breakdown at a cache size of dimension 5. For higher values the results are very stable and nearly optimal. Table I reports a detailed sampling of the plot shown in Fig. 1.

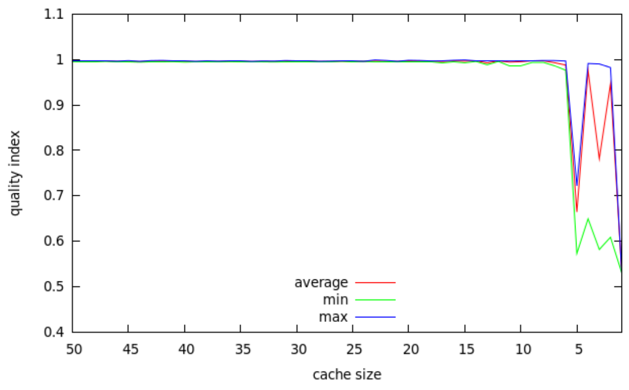

Fig. 1. Results for the first test (easy problem).

TABLE I: SAMPLING AND DETAILS OF FIG. 1.

Cache Size	Average	Min	Max	σ
5	0.6646	0.5730	0.7220	0.33190
10	0.9954	0.9866	0.9986	0.00140
15	0.9964	0.9930	0.9994	0.00077
20	0.9974	0.9962	0.9986	0.00111
25	0.9959	0.9952	0.9970	0.00102
30	0.9963	0.9954	0.9970	0.00080
35	0.9964	0.9954	0.9978	0.00077
40	0.9964	0.9946	0.9978	0.00078
45	0.9962	0.9954	0.9978	0.00083
50	0.9966	0.9956	0.9978	0.00076

The second test is carried out again considering 5000 strings per class. The string are now generated by using an alphabet of size four, $|\Omega|=4$, and considering a variable length between 28 and 32 symbols each. We have considered the two following 4×4 transition matrices, $\mathbf{T}_1$ and $\mathbf{T}_2$, defined as:

$$\mathbf{T}_1 = \begin{bmatrix} 0.1 & 0.1 & 0.7 & 0.1 \\ 0.7 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.7 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.7 \end{bmatrix} \quad \mathbf{T}_2 = \begin{bmatrix} 0.1 & 0.7 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.7 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.7 \\ 0.7 & 0.1 & 0.1 & 0.1 \end{bmatrix}$$

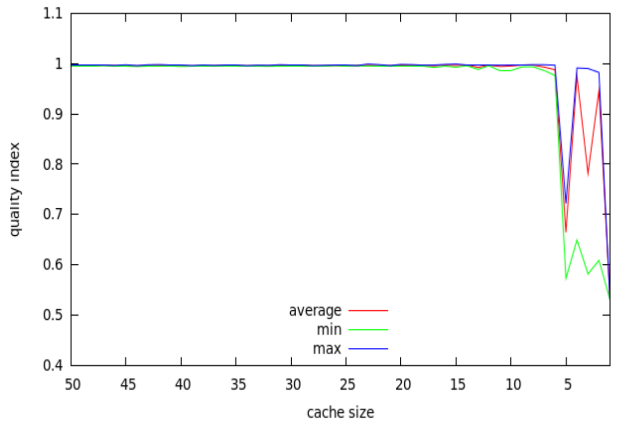

Fig. 2. Results for the second test.

The cache size of MinSOD varies as usual from 50 to 1, with a decrement step of 1; the same considerations about the random seed initialization hold also in this case. The results for this second test are shown in Fig. 2. In this case, since the problem is more difficult, it is possible to observe a non optimal behavior for a cache size less or equal to 10. In Table II we report the details of Fig. 2.

TABLE II: SAMPLING AND DETAILS OF FIG. 2.

Cache Size	Average	Min	Max	σ
5	0.7345	0.6774	0.8590	0.06708
10	0.8656	0.8144	0.9050	0.03051
15	0.8728	0.8524	0.9050	0.01795
20	0.8879	0.8744	0.9178	0.01602
25	0.8900	0.8780	0.9074	0.01126
30	0.8992	0.8780	0.9376	0.02230
35	0.9037	0.8756	0.9300	0.01917
40	0.9180	0.9030	0.9320	0.01143
45	0.9064	0.8994	0.9300	0.01185
50	0.9086	0.9000	0.9246	0.01068

The performance tests which have been carried out show a good stability of the behavior of the MinSOD algorithm with respect to the cache size variations. In fact, until a very low critical threshold for the cache size is reached, the size is quite irrelevant to on the final modeling result. A very good stability is observed with respect to the random variations in the sequence of discarded samples, which is due to the stochastic nature of the replacement algorithm. This stability is observed along the whole interval of the valid settings of the cache size, while strong variations only arise in the region where the cache size is too low to guarantee a correct tracking of the (suboptimal) representative.

V. CONCLUSIONS

In this paper, we have described a cost effective procedure for the computation of the MinSOD representative in the setting of clustering string data. Tests show that this algorithm is sufficiently robust with respect to the cache size, since for a wide range of values the established performance measure defined for the faced clustering problem is stable and of reasonable quality. It is important to remark that the MinSOD representative can be adopted in virtually every input data domain, as long as it is possible to define a dissimilarity measure on that domain. Tests have been carried out by means of the SPARE C++ library, which is available as an open source project.

ACKNOWLEDGMENT

The work was supported by the Lazio Region with the funds dedicated to the "Polo per la Mobilit à Sostenibile" (Research Center for Sustainable Mobility).

REFERENCES

[1] L. Livi and A. Rizzi, “The graph matching problem,” *Pattern Analysis and Applications*, Springer, 2012.

[2] R. Marfil, F. Escolano, and A. Bandera. “Graph-Based Representations in Pattern Recognition and Computational Intelligence,” in *Bio-Inspired Systems: Computational and Ambient Intelligence*, J. Cabestany, F. Sandoval, A. Prieto, and J. Corchado, Eds. Springer Berlin Heidelberg, 2009, vol. 5517, pp. 399-406.

[3] S. Theodoridis and K. Koutroumbas, *Pattern recognition*, Elsevier/Academic Press, 2006.

[4] X. Jiang, A. Müunger, and H. Bunke, “On median graphs: Properties, algorithms, and applications*,” IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, pp. 1144–1151, October 2001.

[5] M. Ferrer, E. Valveny, F. Serratosa, K. Riesen, and H. Bunke, “Generalized median graph computation by means of graph embedding in vector spaces,” *Pattern Recogn.*, vol. 43, pp. 1642–1655, April 2010.

[6] I. Bardaji, M. Ferrer, and A. Sanfeliu, “A comparison between two representatives of a set of graphs: median vs. barycenter graph,” in *Proc. the 2010 joint IAPR international conf. on Structural, syntactic, and statistical pattern recognition,ser. SSPR &SPR'10*, Berlin , Heidelberg: Springer-Verlag, 2010, pp. 149–158.

[7] J. B. Macqueen, “Some methods of classification and analysis of multivariate observations,” in *Proc. the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–297.

[8] H. Bunke and G. Allermann, “Inexact graph matching for structural pattern recognition,” *Pattern Recognition Letters*, vol. 1, no. 4, pp. 245–253, 1983.

[9] K. Riesen and H. Bunke, “Approximate graph edit distance computation by means of bipartite graph matching,” *Image Vision Comput.*, vol. 27, pp. 950–959, June 2009.

[10] M. Neuhaus and H. Bunke, “A quadratic programming approach to the graph edit distance problem,” in *GbRPR, ser. Lecture Notes in Computer Science*, F. Escolano and M. Vento, Eds. Springer, 2007, vol. 4538, pp. 92–102.

[11] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” *Tech. Rep*. vol. 8, 1966.

[12] L. Livi, G. Del Vescovo, and A. Rizzi, “Graph recognition by seriation and frequent substructures mining,” in *Proc. the First International Conf. on Pattern Recognition Applications and Methods*, vol. 1, pp. 186-191, Feb. 2012.

[13] L. Livi, G. D. Vescovo, and A. Rizzi. “Combining Graph Seriation and Substructures Mining for Graph Recognition,” in *Pattern Recognition - Applications and Methods, Advances in Intelligent and Soft Computing*, P. L. Carmona, J. S. Sanchez, and A. L. Fred, Eds. Springer Berlin Heidelberg, 2013, vol. 204, pp. 79-91.

Guido Del Vescovo is a post doctoral research associate at the Information Electronics and Communications Department (DIET) of the University of Rome "La Sapienza" since 2008. He received the Laurea degree in Electronics Engineering in 2004 and his Ph.D. in Information and Communication Engineering in 2008 from the University of Rome "La Sapienza". His major fields of interest include supervised and unsupervised data driven modelling techniques, neural networks, fuzzy systems, evolutionary algorithms and granular computing. He is a developer of the SPARE library (http://libspare.org), an open source C++ project for pattern recognition and machine learning.

Lorenzo Livi is a computer scientist with bachelor and master degrees got from Computer Science Department at SAPIENZA University of Rome in 2007 and 2010. Currently, he is a final year Ph.D. student of the Department of Information Engineering, Electronics, and Telecommunications (DIET) at the same University. He has also an appointment as research assistant at Ryerson University, Toronto, with Prof. Alireza Sadeghian.

During his studies, he has also worked in the ICT industry. Lorenzo's main research interests are focused on pattern recognition, soft computing, and parallel computing, mostly considering problems involving the analysis of the so-called non-geometric spaces.

Fabio Massimo Frattale Mascioli was born in Rome, Italy on June 13, 1963. He received the Laurea degree in Electronic Engineering in 1989 and the Ph.D. degree in Information and Communication Engineering in 1995 from the University "La Sapienza" of Rome. In 1996, he joined the DIET Department (ex INFOCOM) of the University "La Sapienza" of Rome as an assistant professor (researcher). Since 2000, he has been an associate professor of Circuit Theory at the same department. His research interest mainly regards neural networks and neuro-fuzzy systems and their applications to clustering, classification and function approximation problems. Currently, he is also working on circuit modeling for vibration damping, energy conversion systems, electric and hybrid vehicles, energy-mobility integrated systems. He is the author or co-author of more than eighty papers presented at international conferences or published in international scientific literature. Since 2007, he is the scientific director of the "Polo per la Mobilità Sostenibile della Regione Lazio" (Sustainable Mobility Pole of Lazio Region).

Antonello Rizzi received the Dr. Eng. degree in Electronic Engineering from the University of Rome "La Sapienza" in 1995 and the Ph.D. in Information and Communication Engineering in 2000, from the same University. In September 2000, he joined the "Information and Communication" Department (INFO-COM Dpt.) of the University of Rome "La Sapienza" as an assistant professor. Since July 2010, he joined the Department of "Information, Electronics and Telecommunications Engineering" Department of the same University.

His major fields of interest are in the area of soft computing, pattern recognition and computational intelligence, including supervised and unsupervised data driven modeling techniques, neural networks, fuzzy systems and evolutionary algorithms. His research activity concerns the design of automatic modeling systems, with particular emphasis on classification, clustering, function approximation and prediction problems. In particular, he is currently working on classification and clustering systems for structured patterns, graph matching, symbolic inductive modeling systems, Granular Computing and Knowledge Discovery systems.

Since 2008, he serves as the scientific and technical coordinator of the R&D activities in the Intelligent Systems Laboratory within the Sustainable Mobility Pole of Lazio Region. He is author of more than 90 international publications.