Relational Sequence Clustering for Aggregating Similar Agents

Grazia Bombini, Nicola Di Mauro, Stefano Ferilli, and Floriana Esposito

Università degli Studi di Bari, Dipartimento di Informatica, 70125 Bari, Italy
{bombini,ndm,ferilli,esposito@di.uniba.it}

Abstract. Many clustering methods are based on flat descriptions, while data regarding real-world domains include heterogeneous objects related to each other in multiple ways. For instance, in the field of Multi-Agent System, multiple agents interact with the environment and with other agents. In this case, in order to act effectively an agent should be able to recognise the behaviours adopted by other agents. Actions taken by an agent are sequential, and thus its behaviour can be expressed as a sequence of actions. Inferring knowledge about competing and/or companion agents by observing their actions is very beneficial to construct a behavioural model of the agent population. In this paper we propose a clustering method for relational sequences able to aggregate companion agent behaviours. The algorithm has been tested on a real world dataset proving its validity.

Key words: Sequence Clustering, Relational Sequence Similarity

1 Introduction

Many clustering methods are based on flat descriptions, in which data points are represented as a fixed-length attribute vector. However, datasets belonging to real-world domains include heterogeneous objects related to each other in multiple ways. For instance in the field of Multi-Agent System (MAS), multiple agents (artificial or human) interact with the environment and with other agents. From a behaviour analysis point of view, MASs are in many aspects similar to human society. Indeed, in order to act effectively an agent should be able to recognise the behaviours adopted by other agents. Actions taken by an agent are sequential, and thus its behaviour can be expressed as a sequence of actions. Each action can be related in different way with respect to other agents and to the environment in which agents interact. Inferring knowledge about competing and/or companion agents by observing their actions is very beneficial to construct a behavioural model of the agent population. In order to reach this goal, it is necessary to define a measure able to asses the similarity between different behaviours described by relational sequences.

In the field of unsupervised data analysis, clustering algorithms provide a useful methods to explore complex data structures. Clustering methods have been exploited in many disciplines, such as data mining, document retrieval, image segmentation and pattern classification [1–3]. A clustering method tries to aggregate data on the basis of a *similarity* (or *dissimilarity*) criteria, where groups (or *cluster*) are defined by a set of similar objects. The two key issues in clustering are the *object representation* and the design of the *similarity measure*. Sequence Clustering concerns grouping a set of sequences into clusters by using a similarity criteria capturing the difference between sequences. When the sequence are expressed in a relational language, we name the task *Relational Sequence Clustering*. Sequence data can be generated from a variety of domains, such as DNA sequencing, speech processing, customer transaction and robot sensor analysis to name a few [4, 3].

The solution that we propose is to represent each relational sequence by a set of *features* and then to exploit these features to compute a *similarity value* between sequences. This solution presents two problems: (**P1**) how to extract the features from relational sequences, and (**P2**) how to asses a similarity value between two feature-based sequence descriptions. This paper represents, to our knowledge, the first proposal for clustering relational sequences. In particular, we will use a relational learning algorithm to mine meaningful features among the relational sequences and we will use these features to construct a feature vector for each sequence. Then we adapt the Tanimoto measure [5] to compute the similarity between feature vectors, and finally the Partition Around Medoids (PAM) algorithm [6] will be used to aggregate the sequences. The proposed method has been applied to a real world problem and the results prove its validity.

2 Relational Sequences: representation and mining

In this section we present a method based on relational pattern mining, to extract meaningful features able to represent relational sequences.

2.1 Sequence Features

Before to design a clustering method, it is necessary to define an appropriate *similarity measure* between sequences. The measure will be applied to data objects represented as a set of features expressing the special properties (features) of a sequence in a specific domain. A way to represent a sequence as a feature vector is to use the patterns occurring in it as true features.

Given an alphabet of symbols $\mathcal{A}$, and let be $k \geq 1$ a positive integer, then a ***k*-gram** (k-mers), is a sequence σ of symbols over $\mathcal{A}$ of length k ($\sigma \in \mathcal{A}^k$, $|\sigma| = k$). For a given sequence $\sigma = (s_1 s_2 \dots s_t)$, the k-grams of interest are all subsequences $\sigma' = (s_i s_{i+1} \dots s_{i+k-1})$ of length k occurring in σ.

Given a sequence $\sigma = (s_1 s_2 \dots s_t)$ with $|\sigma| = t$, we define K_σ as the set of all k-grams Ω_k of σ, $1 \leq k \leq t$:

$$K_\sigma = \bigcup_{k=1}^{t} \Omega_k = \bigcup_{k=1}^{t} \{\omega_{k1}, \omega_{k2}, \dots \omega_{kn_k}\} \quad (1)$$

where $\omega_{ki} = (s_i s_{i+1} \dots s_{i+k-1})$, and $n_k = t - k + 1$.

Given a set of sequences $\mathcal{S} = \{\sigma_i\}_{i=1}^n$, $\mathcal{K}$ is the set of all k-grams on all the sequences belonging to $\mathcal{S}$: $\mathcal{K} = \bigcup_{i=1}^n K_{\sigma_i}$, where $\mathcal{K}$ represents the set of all features over $\mathcal{S}$.

2.2 Logical Background

A relational sequence is represented by a set of Datalog [7] atoms, based on a first-order *alphabet* consisting of a set of *constants*, a set of *variables*, a set of *function symbols*, and a non-empty set of *predicate symbols*. Each function symbol and each predicate symbol has an *arity*, representing the number of arguments the function/predicate has. Constants may be viewed as function symbols of arity 0. An atom $p(t_1, \dots, t_n)$ (or atomic formula) is a predicate symbol p of arity n applied to n terms t_i (i.e., a constant symbol, a variable symbols, or an n-ary function symbol f applied to n terms $t_1, t_2, \dots, t_n$). A *ground term* or *atom* is one that not contain any variables. A *clause* is a formula of the form $\forall X_1 \forall X_2 \dots \forall X_n (L_1 \vee L_2 \vee \dots \vee \overline{L}_i \vee \overline{L}_{i+1} \vee \dots \vee \overline{L}_m)$ where each L_i is a literal and $X_1, X_2, \dots X_n$ are all the variables occurring in $L_1 \vee L_2 \vee \dots \overline{L}_i \vee \dots \overline{L}_m$. Most commonly the same clause is written as an implication $L_1, L_2, \dots L_{i-1} \leftarrow L_i, L_{i+1}, \dots L_m$, where $L_1, L_2, \dots L_{i-1}$ is the *head* of the clause and $L_i, L_{i+1}, \dots L_m$ is the *body* of the clause. Clauses, literals and terms are said to be *ground* whenever they do not contain variables.

A *substitution* θ is defined as a set of bindings $\{X_1 \leftarrow a_1, \dots, X_n \leftarrow a_n\}$ where $X_i, 1 \le i \le n$ is a variable and $a_i, 1 \le i \le n$ is a term. A substitution θ is applicable to an expression e, obtaining the expression $e\theta$, by replacing all variables X_i with their corresponding terms a_i. A conjunction A is *θ-subsumed* by a conjunction B, denoted by $A \preceq_\theta B$, if there exists a substitution θ such that $A\theta \subseteq B$. A clause c_1 *θ-subsumes* a clause c_2 if and only if there exists a substitution σ such that $c_1\sigma \subseteq c_2$. c_1 is a *generalization* of c_2 (and c_2 a *specialization* of c_1) under θ-subsumption. If c_1 θ-subsumes c_2 then $c_1 \models c_2$.

Definition 1 (Relational (sub)sequence). *A* relational sequence *is an ordered list of atoms. Given a sequence $\sigma = (s_1 s_2 ... s_m)$, a sequence $\sigma' = (s'_1 s'_2 ... s'_k)$ is a* subsequence *(or* pattern*) of the sequence σ, indicated by $\sigma' \sqsubseteq \sigma$, if*

1. $1 \le k \le m$;
2. $\exists j, 1 \le j \le m - k$ *and a substitution* θ *s.t.* $\forall i, 1 \le i \le k$: $s'_i\theta = s_{j+i}$.

A subsequence occur in a sequence if exists at least a mapping from elements of σ' into the element of σ such that the previous condition are hold. In our case, that subsequence is a relational pattern*.*

The *support* of a sequence σ in a set of sequences $\mathcal{S}$ corresponds to the number of sequences in $\mathcal{S}$ containing the sequence σ: support(σ)= $|\{\sigma' | \sigma' \in \mathcal{S} \wedge \sigma \sqsubseteq \sigma'\}|$.

Now we can translate the concept of k-grams to the relational case.

Definition 2 (Relational k-gram). *Given an alphabet of atoms $\mathcal{A}$, a* relational k-gram *is a relational sequence σ of length k defined over $\mathcal{A}$.*

Given a set of relational sequences $\mathcal{S} = \{\sigma_i\}_{i=1}^n$, $\mathcal{K}$ is the set of all relational k-grams on all the sequences belonging to $\mathcal{S}$: $\mathcal{K} = \bigcup_{i=1}^n K_{\sigma_i}$, where K_{σ_i} is the set of all relational k-grams over the sequence σ_i. In particular, $\mathcal{K}$ represents the set of all relational features over $\mathcal{S}$. We define $\mathcal{K}(\alpha) \subseteq \mathcal{K}$, the set of relational k-grams having a support greater than $\alpha - 1$: $\mathcal{K}(\alpha) = \{\sigma | \sigma \in \mathcal{K} \wedge \mathit{support}(\sigma) \geq \alpha\}$.

2.3 Mining Relational Sequential Patterns

In order to select the best set of features, we use an Inductive Logic Programming (ILP) [8] algorithm , based on [9], for discovering relational patterns from sequences. It is based on a level-wise search method, known in data mining from the Apriori algorithm [10]. It takes into account the sequences, tagged with the belonging class, and the α parameter denoting the minimum support of the patterns. It is essentially composed by two steps, one for generating pattern candidates and the other for evaluating their support. The level-wise algorithm makes a breadth-first search in the lattice of patterns ordered by a specialization relation. Starting form the most general patterns, at each level of the lattice the algorithm generates candidates by using the lattice structure and then evaluates the frequencies of the candidates. Since the monotonicity of pattern frequency (if a pattern is not frequent then none of its specializations is frequent), in this phase some patterns may be discarded.

The generation of the patterns actually present in the sequences of the dataset, is based on a top-down approach. The algorithm starts with the most general patterns. These initial patterns are all of length 1 and are generated by adding an atom to the empty pattern. Then, at each step it tries to specialize all the potential patterns, discarding those that do not occur in any sequence and storing the ones whose length is equal to the user specified input parameter *maxsize*. Furthermore, for each new refined pattern, semantically equivalent patterns are detected, by using the θ-subsumption relation, and discarded. In the specialisation phase, the specialisation operator under θ-subsumption is used. Basically, the operator adds atoms to the pattern. Finally, the algorithm may use a background knowledge $\mathcal{B}$ (a set of Datalog clauses) containing constraints on how to explore the lattice.

3 Clustering Relational Sequences

Now we propose a distance function to measure the dissimilarity between two relational sequences, and the how those distances will be used in the Partition Around Medoids (PAM) algorithm to aggregate similar sequences.

3.1 Distance Function over Relational Sequences

A sequence distance function is a function d that maps a pair of sequences to a non-negative real number to measure the (dis)similarity between two sequences. A sequence distance satisfies the follow properties:

- $d(x, y) > 0$ for sequence x and y such that $x \neq y$;
- $d(x, x) = 0$ for all sequences x;
- $d(x, y) = d(y, x)$ for all sequences x and y;
- $d(x, y) \leq d(x, z) + d(z, y)$ for all sequences x, y and z.

Given a set of sequences $\mathcal{S}$, we apply the algorithm previously described in Section 2.3, and completely reported in [9], to find all the relational k-grams $\mathcal{K}(\alpha)$ over the set $\mathcal{S}$ with a support at least equal to α. $\mathcal{K}(\alpha)$ is the ordered set of *features* $\mathcal{F}$ that will be used to compute the boolean vector representation of each sequence in the following way. Given a sequence $\sigma \in S$, and $\mathcal{F} = \mathcal{K}(\alpha) = \{\omega_i\}_{i=1}^n$ the set of relational k-grams over $\mathcal{S}$, the *feature vector* of σ is a vector $V_\sigma = (f_1(\sigma), f_2(\sigma), ..., f_n(\sigma))$ where

$$f_i(\sigma) = \begin{cases} 1 \ \textit{if} \ \omega_i \sqsubseteq \sigma \\ 0 \ \textit{otherwise} \end{cases}$$

Now, the function distance $d_r(\cdot, \cdot)$ between two relational sequences σ_1 and σ_2 is computed using the classical Tanimoto measure [5]:

$$d_{r_1}(\sigma_1, \sigma_2) = \frac{n_{1\sigma_1} + n_{1\sigma_2} - 2n_{1\sigma_{12}}}{n_{1\sigma_1} + n_{1\sigma_2} - n_{1\sigma_{12}}} = \frac{2(n - n_{1\sigma_{12}})}{2n - n_{1\sigma_{12}}} \tag{2}$$

where $n_{1\sigma_i} = n = |\mathcal{F}|$ is the number of the features, and $n_{1\sigma_{12}} = |\{f_i | f_i(\sigma_1) = f_i(\sigma_2)\}|$ is the number of features with the same value in both σ_1 and σ_2. However, this basic formulation takes into account features not appearing (with value 0) in the sequences, and in case of a lot of feature this can lead to underfitting.

Equation (2) may be extended in the following way:

$$d_{r_2}(\sigma_1, \sigma_2) = \frac{n_{2\sigma_1} + n_{2\sigma_2} - 2n_{2\sigma_{12}}}{n_{2\sigma_1} + n_{2\sigma_2} - n_{2\sigma_{12}}} = \sum_{i=1}^{n} \frac{f_i(\sigma_1) + f_i(\sigma_2) - 2f_i(\sigma_1)f_i(\sigma_2)}{f_i(\sigma_1) + f_i(\sigma_2) - f_i(\sigma_1)f_i(\sigma_2)} \tag{3}$$

where $n_{2\sigma_i} = \sum_{j=1}^n f_j(\sigma_i)$ is the number of the features holding in the sequence σ_i, and $n_{\sigma_{12}} = |\{f_i | f_i(\sigma_1) = f_i(\sigma_2) = 1\}|$ is the number of features that hold both in σ_1 and σ_2.

3.2 Partition Around Medoids algorithm

Based on an appropriate objective function, a *partitional clustering* algorithm obtains a single partition of n objects into a set of k clusters. To clustering the sequences, we use the well-known k-medoids method Partition Around Medoids (PAM) [6]. Given $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^n$ a set of objects, let $\{u_h\}_{h=1}^k$ be the k cluster representatives, named *medoids*, and l_i be the cluster assignment of an object x_i, where $l_i \in \mathcal{L}$ and $\mathcal{L} = 1, ..., k$, the goal of the the k-medoids algorithm is to find the best clustering solution $\mathcal{C}$ optimizing the objective function $\mathcal{J}(C)$.

In order to find k clusters, PAM find a representative object u_i (medoid) for each cluster. This representative object is meant to be the most centrally located object for each cluster. Once the k medoids have been selected, each non-selected object is grouped with the medoid to which it is the most similar. More precisely, if $\mathbf{x}_j$ is a non-selected object, and $\mathbf{x}_i$ is a (selected) medoid, then $\mathbf{x}_j$ belongs to the cluster represented by $\mathbf{x}_i$ if $d(\mathbf{x}_j, \mathbf{x}_i) = min_{h=1,\ldots,k} d(\mathbf{x}_j, \mathbf{x}_h)$, where $d(\mathbf{x}_j, \mathbf{x}_i)$ denotes the dissimilarity, or distance, between objects $\mathbf{x}_j$ and $\mathbf{x}_i$. PAM starts with an arbitrary selection of k objects. Then in each step, a swap between a selected object $\mathbf{x}_i$ and a non-selected object $\mathbf{x}_h$ is made, as long as such a swap would result in an improvement of the quality of the clustering. In particular, to calculate the effect of such a swap between $\mathbf{x}_i$ and $\mathbf{x}_h$, PAM computes costs C_{ih} for all non-selected objects $\mathbf{x}_j$. In this work we used the function reported in Equation (3) as a similarity function in the PAM algorithm.

Tightness. Finally, the quality of the chosen medoids is measured by the average dissimilarity between a non-selected object and the medoid of its cluster, as reported in Equation (4).

$$tightness(\mathcal{C}) = \frac{1}{n} \sum_{i=1,\ldots,n} d(\mathbf{x}_i, u_i) \tag{4}$$

3.3 Evaluation of clustering solution

To evaluate the goodness of a cluster solution, it is necessary to calculate an intra-cluster similarity (how the objects within a cluster are similar) and inter-cluster similarity (how object from different clusters are dissimilar).

Entropy. Entropy indicates how much homogeneous a cluster is. For each cluster C_j in the clustering result C we compute p_{ij} , the probability that a member of the cluster C_j belongs to class i as $p_{ij} = n_j^i/n_j$ where n_j is the number of objects contained in the cluster C_j , and n_j^i is the number of data objects of the i-th class that were assigned to the cluster C_j .

The entropy of each cluster C_j may be calculated using the following formula

$$E(C_j) = -\sum_{i=1}^{c} p_{ij} log(p_{ij}) = -\sum_{i=1}^{c} \frac{n_j^i}{n_j} log \frac{n_j^i}{n_j} \tag{5}$$

where the sum is taken over all the c classes. The entropy of the entire clustering solution is then defined to be the sum of the individual cluster entropies weighted according to the cluster size:

$$E(\mathcal{C}) = \sum_{r=1}^{k} \frac{n_r}{n} E(C_j) \tag{6}$$

A perfect clustering solution should be the one that leads to clusters that contain objects from only a single class, in which case the entropy will be zero. In general, the smaller the entropy values, the better the clustering solution is.

Purity. The purity measures the extend to which each cluster contained data objects from primarily one class. The purity of the cluster C_j is defined to be

$$P(C_j) = \frac{1}{n_j} \max_i (n_j^i) \tag{7}$$

which is nothing more than the fraction of the overall cluster size that the largest class of objects assigned to that cluster represents. The overall purity of the clustering solution C is obtained as a weighted sum of the individual cluster purities and is given by

$$\mathcal{P}(\mathcal{C}) = \sum_{r=1}^{k} \frac{n_r}{n} P(C_r). \tag{8}$$

4 Experimental results

Relational dataset. In order to validate the method we performed experiments on the Greenberg data set [11]. From a behaviour analysis point of view, MASs are in many aspects similar to human's society. Indeed, in order to act effectively an agent (artificially or human) should be able to recognise the behaviours adopted by other agents. Actions taken by an agent are sequential, and thus its behavior can be expressed as a sequence of actions.

A Unix command sequence (session) can be seen as the sequence of actions taken by an agent (user) at each session. The Greenberg data set consists of 168 logs of different users of the unix chs, divided into four groups: 52 *computer scientists* (CS), 36 *experienced programmers* (EP), 55 *novice programmers* (NP) and 25 *non-programmers* (NNP). Each Greenberg's log file corresponding to a user keeps track of an entire login session. Each login session is denoted by a starting and ending time record. Each command belonging to a session, has been annotated with the current working directory, alias substitution, history use and error status. Furthermore, each command name may be followed by some options and some parameters. Each session represents a sequence and a log file is a collections of sequences.

Each shell log has been represented as a set of logical ground atoms [12] as follows: `command(e)` is the predicate used to indicate that `e` is a command. The command name has been used as a predicate symbol applied to `e`; `parameter(e,p)` indicates that `p` is the parameter of `e`. The parameter name has been used as a predicate symbol applied to `p`; `current directory(c,d)` indicates that `d` is the current directory of the command `c`; `next_c(c1,c2)` indicates that the command `c2` is the direct command successor of `c1`; `next_p(p1,p2)` indicates that the parameter `p2` is the direct parameter successor of `p1`. For instance the following shell log

```
   man mklib
   man -k mklib
```

should be translated as

```
command(c1). '$man'(c1).
   next_p(c1,c1p1). parameter(c1p1,'$mklib'). next_c(c1,c2).
command(c2). '$man'(c2).
   next_p(c2,c2p1). parameter(c2p1,'$-k').
   next_p(c2p1,c2p2). parameter(c2p2,'$mklib').
```

Results. The results obtained by the PAM algorithm on a dataset made up of 209 sequences extracted from 8 selected users (agents), two for each class, defined on 153 different command names are reported in Figure 1. In particular, we have 89 sequences for CS, 39 sequences for EP, 39 for NP and 42 for NNP.

In the first step, the set $\mathcal{K}(\alpha)$ of frequent k-grams has been mined. Here, α denotes the support of each k-gram $\sigma \in \mathcal{K}(\alpha)$ corresponding to the ratio $\mathit{support}(\sigma)/|\mathcal{S}|$, where $\mathcal{S}$ is the set of sequences to be clustered. In this experiment, α has been set to $0,02$, $0,03$, $0,04$ and $0,05$, and the algorithm extracted, respectively, 567, 153, 84 and 58 k-grams.

Even if we know the corresponding class label of each sequences we dropped it considering all the sequences as belonging to the same dummy class. In the second step, after having extracted the set of features, the sequences have been clustered adopting a values of k (the number of clusters) belonging to the set $\{4, 6, 8, 10, 12, 14, 16, 18, 20\}$. Finally, we can compute for each obtained cluster the corresponding purity value knowing the previously dropped class label for each sequence. Figure 1 shows the variation of the purity for all clustering solutions, and the corresponding plot of the entropy and tightness values. As we can see, good results have been obtained with large value of k, and choosing α equals to $0,05$ corresponding to the case of mining patterns with a large support.

5 Conclusions and Related Works

In [13] the authors investigate the problem of clustering sequence based on their structural features. In order to characterise the structural properties of a given sequence, the authors used the conditional probability distribution (CPD) of the next symbol (right after a segment of some fixed length L). The difference between the two CPDs corresponding to the two sequences is assumed to be the distance between them. The similarity between two CPDs can be measured by the variational distance or by the Kullback-Leibler divergence between the CPDs. The CPDs are represented in a concise way by probabilistic suffix tree, a variant of a suffix tree. The computation of a CPD can be expensive for large L. To define the CPD only the the frequent sequences in a cluster are used. To

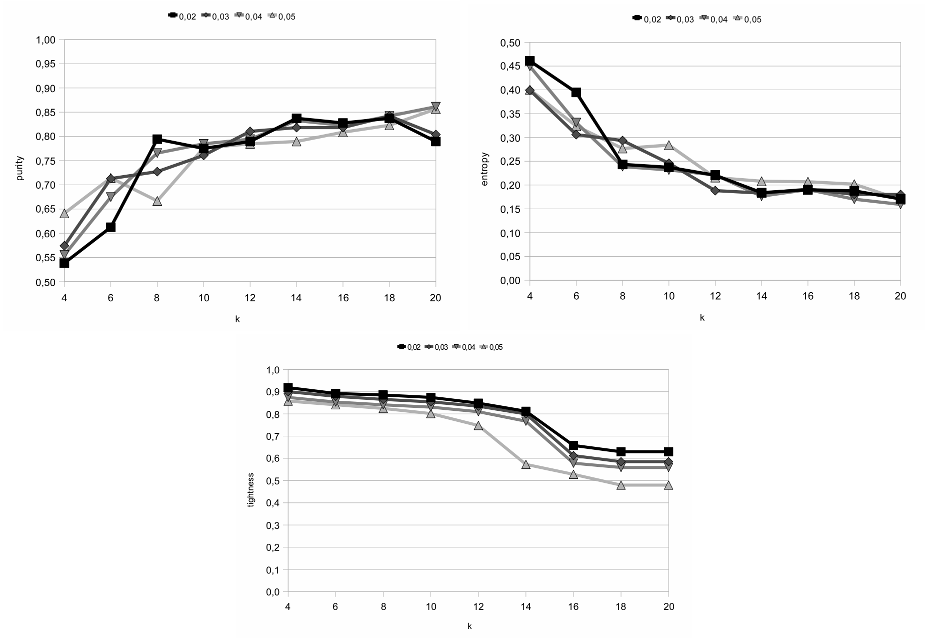

Fig. 1. Entropy (top-left), purity (top-right) and tightness (bottom) values obtained by PAM with different number of cluster

discover clusters is designed an algorithm, it compute the distance of a sequence for each cluster. However, the algorithm reported in [13] works on sequences of flat symbols.

Recent advances in artificial intelligence leading to the growth of structured data sequences and the recent interest in statistical relational learning have motivated the development of probabilistic models for relational sequences [14], such as Logical Hidden Markov Models [15] and Relational Conditional Random Fields [16]. These models have been applied to model the probabilistic nature of the sequence, but they have not been never used for clustering.

In this paper we propose a similarity measure and a clustering approach for relational sequences applied to agent behaviour aggregation. Experimental results proved the validity of the proposed approach. As a future work, we will investigate methods for extracting patterns with a high discriminative power, and we will compare different similarity functions.

Acknowledgment

This work is partially supported by the Italian MIUR-FAR project “Computational Biology Laboratory for Molecular Biodiversity” (DM19410).

References

1. Berkhin, P.: A Survey of Clustering Data Mining Techniques. In: Grouping Multidimensional Data. Springer Berlin Heidelberg (2006) 25–71
2. Filippone, M., Camastra, F., Masulli, F., Rovetta, S.: A survey of kernel and spectral methods for clustering. Pattern Recognition **41**(1) (2008) 176–190
3. Xu, R., Wunsch, D., I.: Survey of clustering algorithms. IEEE Transactions on Neural Networks **16**(3) (2005) 645–678
4. Dong, G., Pei, J.: Sequence Data Mining (Advances in Database Systems). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)
5. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification (2nd Edition). Wiley-Interscience (November 2000)
6. Kaufman, L., Rousseeuw, P.J.: Finding groups in data: an introduction to cluster analysis. John Wiley and Sons, New York (1990)
7. Ullman, J.: Principles of Database and Knowledge-Base Systems. Volume I. Computer Science Press (1988)
8. Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. Journal of Logic Programming **19/20** (1994) 629–679
9. Esposito, F., Di Mauro, N., Basile, T., Ferilli, S.: Multi-dimensional relational sequence mining. Fundamenta Informaticae **89**(1) (2008) 23–43
10. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.: Fast discovery of association rules. In: Advances in Knowledge Discovery and Data Mining. (1996)
11. Greenberg, S.: Using unix: Collected traces of 168 users. Research Report 88/333/45, Department of Computer Science, University of Calgary, Alberta, Canada (1988) Includes tar-format cartridge tape.
12. Jacobs, N., Blockeel, H., Comm, T.U.: From shell logs to shell scripts. In: Proc. 11th International Conference, ILP 2001, LNCS 2157. (2001) 80–90
13. Yang, J., Wang, W.: Cluseq: Efficient and effective sequence clustering. In: Proceedings of the 19th International Conference on Data Engineering. (2003) 101–112
14. Kersting, K., De Raedt, L., Gutmann, B., Karwath, A., Landwehr, N.: Relational sequence learning. In De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S., eds.: Probabilistic Inductive Logic Programming. Volume 4911/2008 of Lecture Notes in Computer Science. Springer (February 2008) 28–55
15. Kersting, K., Raedt, L.D., Raiko, T.: Logical hidden markov models. Journal of Artificial Intelligence Research (JAIR) **25** (2006) 425456
16. Gutmann, B., Kersting, K.: Tildecrf: Conditional random fields for logical sequences. In Fürnkranz, J., Scheffer, T., Spiliopoulou, M., eds.: Proceedings of the 15th European Conference on Machine Learning (ECML-2006). Volume 4212 of LNAI (Lecture Notes in Artificial Intelligence)., Berlin, Germany, Springer (September 2006) 174–185