RE-SPaM: Using Regular Expressions for Sequential Pattern Mining in Trajectory Databases

Leticia I. Gómez
Instituto Tecnólogico de Buenos Aires
Av. E. Madero 399
Bs.As., Argentina
lgomez@itba.edu.ar

Alejandro A. Vaisman
University of Hasselt and
Transnational University of Limburg,
Department WNI, Gebouw D, B-3590
Diepenbeek, Belgium, and
Universidad de Buenos Aires
alejandro.vaisman@uhasselt.be

Abstract

In sequential pattern mining, languages based on regular expressions (RE) were proposed to restrict frequent sequences to the ones that satisfy user-specified constraints. In these languages, REs are applied over items*. We propose a much powerful language, based on regular expressions, denoted RE-SPaM, where the basic elements are constraints over the attributes of the items. Expressions in this language may include attributes, functions over attributes, and variables. We present the data model, sketch the syntax and semantics of RE-SPaM, a set of examples, and suggest how RE-SPaM can be used in the mining process.*

1 Introduction

In many application domains, information is organized as ordered sequences. These applications can benefit from the discovery of hidden patterns in such sequences. Two main approaches had dominated the field of pattern discovery in sequences: (a) the Agrawal and Srikant [1] proposal (the one we follow in this paper), and (b) the approach of Mannila *et al.* [5]. In the former (aimed at discovering inter-transactions patterns), an *interesting* pattern is one that appears in the database at least as many times as an user-specified threshold. In the proposal, an *itemset* is an unordered, non-empty set of items and a *sequence* is an ordered list of itemsets. The support of a sequence is the fraction of the total number of transactions containing it. The authors extended their proposal [7] to support time-gap constraints, taxonomies, and time windows, resulting in the Generalized Sequential Patterns (GSP) algorithm. Although many frequent sequential patterns could be obtained using GSP, it is likely that only a few of them could be relevant to the user. Thus, Garofalakis *et. al.* [3] proposed a variation, denoted SPIRIT, where regular expressions are used to prune the information obtained. The algorithm returns only the frequent patterns that satisfy these regular expressions. We extend existing work in several ways: we propose a language based on regular expressions, called RE-SPaM, built on constraints (i.e., conditions over attributes of complex items) rather than over atomic items. These regular expressions can contain constants, attributes, and variables, substantially improving earlier proposals.

In the analysis of moving object data [4], the trajectory of an object is given by samples composed of a finite number of tuples of the form $\langle Oid, t, x, y\rangle$, such that, at a certain instant t, the object Oid was located at coordinates (x, y). Instead of *sequences of points*, we work with *semantic trajectories* [6], in the form of sequences of geometric objects, denoted *stops*, where the items to be mined are *sequences of stops* in a trajectory, along with the time spent at them by each moving object.

Mouza and Rigaux [2] proposed a language based on regular expressions for querying trajectory patterns, where each zone could be represented by its label (a constant) or by a variable (@x). Variables *can only be associated with places* (represented by labels or IDs) visited by objects. Thus, the language cannot deal with *time constraints* or categories. On the contrary, our approach allows variables associated with any attribute of an item.

2 Preliminaries and Data Model

Traditional algorithms for sequential pattern mining work over *atomic items*, i.e., literals. Each item has the time interval of the transaction associated with it. In this work we consider items as composed of attributes. The data model we now define, formalizes this. We have a set

Category	Schema
hotels	[**ID**, $categoryName, geom, star$]
restaurants	[**ID**, $categoryName, geom, typeOfFood, price$]
Eiffel Tower	[**ID**, $categoryName, geom$]
zoos	[**ID**, $categoryName, geom, price$]

Table 1. Schema of the categories

of attribute names **A**, and a set of identifier names **I**. Each attribute $attr \in \mathbf{A}$ has a domain $dom(attr)$, and each identifier $ID \in \mathbf{I}$ a domain $dom(ID)$.

Definition 1 (Category Schema) A *category schema* S is a pair (ID, A), where $ID \in \mathbf{I}$ is a distinguished attribute denoted *identifier*, and $A = \{attr | attr \in \mathbf{A}\}$. In what follows we consider the set A ordered. Thus, S has the form $(ID, attr_1, ..., attr_n)$. □

Definition 2 (Category Occurrence) *Given a category schema S, a category occurrence for S is the pair $(\langle ID, id\rangle, P)$, where ID is the ID attribute of Definition 1 above, $id \in dom(ID)$, and P is the set of pairs $\{(attr_1, v_1), ..., (attr_n, v_n)\}$, where: (a) $attr_i = A(i)$ (remember that A is considered ordered); (b) $v_i \in dom(attr_i), \forall i, i = 1..n$; (c) All the occurrences of the same category have the same set of attributes; (d) ID is unique for a category occurrence, meaning that no two occurrences of the same category can have the same value for ID. (see below)*

In what follows, for clarity reasons, we assume that $attr_0$ stands for ID. Thus, a category occurrence is the set of pairs $[(attr_0, v_0), (attr_1, v_1), ..., (attr_n, v_n)]$. □

Definition 3 (Category Instances) *A set of occurrences of the same category is denoted a* category instance. *Also, given set of category instances (see Figure 1), we extend the fourth condition in Definition 2 to hold for the whole set: ID is unique for a set of category instances, meaning that no two occurrences of categories in the set can have the same value for ID.*

The schema of each category, and a corresponding set of category instances (i.e., the stops in the trajectories), are shown in Table 1 and Figure 1, respectively. □

Adding a time interval to a category occurrence, produces an **Item**. The time interval of an item is described by its initial and final instants, and denoted [ts, tf]. A **Table of Items** (ToI) is a finite set of tuples of the form $\langle O_j, i_k\rangle$ where $i_k \in \mathcal{I}$ is an item associated with an object O_j.

Example 1 *Figure 2 shows an instance of a ToI corresponding to the category instances of Figure 1. Note that the first two items for* OID $= O_2$ *have the same ID because they correspond to the same category occurrence:* $[(categoryName, zoo), (ID, Z), (geom, pol7), (price, cheap)]$. □

Definition 4 (Valuation of an Attribute and an Item) *Let $(attr, v)$ be a pair in a category occurrence; a valuation of attr is obtained applying a function Val such that $Val(attr) = (v)$.*

Further, let I be an Item, and $\mathcal{F}$ a set of functions $\{f_1, f_2, ..., f_n\}$, such that each f_i maps the value v in a pair $(attr, v) \in I$ to a single value. In addition to v, f_i can have other constants as arguments (we denote these arguments $\mathcal{A}$). A valuation of I *with $\mathcal{F}$, denoted $\mathcal{V}(I, \mathcal{F})$ is the* item *resulting from applying $\mathcal{F}$ to I as follows: pick one f_i in $\mathcal{F}$ and apply it to the value v in a pair $(attr_j, v)$ of I, probably using some constants in $\mathcal{A}$. Repeat the process with the remaining pairs, until all pairs have been valuated.* □

Definition 5 (Transformed Subitem) *Given an item I, a set of functions $\mathcal{F}$, and a valuation of I with $\mathcal{F}$, $\mathcal{V}(I, \mathcal{F})$, any subset of $\mathcal{V}$ is called* Transformed Subitem, $TS(I)$. □

Definition 6 (Itemset) *An* itemset $(i_1, i_2, ...i_n)$ *is a non-empty set of items, where $n \geq 1$, and for all $i_k, k = 1..n$, the ts_date, ts_time, tf_date, tf_time values are the same.*

Let IS= $(i_1, i_2, ...i_n)$, *be an itemset. A* sub-itemset *of IS is a subset of $(TS(i_1), TS(i_2),... \ TS(i_n))$, where $TS(i_i)$ is any transformed subitem of i_i.*

In the moving objects setting, since each moving object can be in only one place at each moment, all itemsets belonging to the same OID will contain exactly one item. □

Definition 7 (Sequences and contiguous list) *A* sequence *is an ordered list of itemsets $\langle i_1, i_2, ..., i_m\rangle$ such that, for every pair of integers j, g, $j < g \Rightarrow Val(i_j.tf, v) < Val(i_g.ts, v)$ holds.(The* i.a *notation means that* a *is an attribute of item* i.

A sequence $\langle a_1, a_2, ..., a_n\rangle$ subsumes *another sequence $\langle b_1, b_2, ..., b_n\rangle$ if $\forall$ i ∈ 1..n, b_i is a sub-itemset of a_i.*

Given a ToI instance with tuples of the form $\langle O_j, i_k\rangle$, let us denote $Items(O_j)$ the set of items i_k associated with O_j. Also let $CL(O_j) \subseteq Items(O_j)$. We say $CL(O_j)$ is a contiguous list *for O_j, if $\forall i \in Items(O_j)$ and $i \notin CL(O_j)$, the starting time of i (denoted $v_{ts}(i)$) is less than the starting time of all the items in $CL(O_j)$, or all the starting times of the items in $CL(O_j)$ are less than $v_{ts}(i)$.* □

Definition 8 (Support) *Given a ToI instance with tuples of the form $\langle O_j, i_k\rangle$. The support of a sequence S is the fraction of the different objects O_j in the ToI, associated with a contiguous list $CL(O_j)$ which subsumes S.* □

3 RE-SPaM

We now introduce a language built over attributes of the complex items defined in Section 2. The terms in the language are constants (a literal enclosed by single quotes), attributes, variables (a literal that begins with the

Category	Instance
hotels (2 occurrences)	$[(ID, H1), (categoryName, hotel), (geom, pol1), (star, 3)]$ $[(ID, H2), (categoryName, hotel), (geom, pol2), (star, 5)]$
restaurants (3 occurrences)	$[(ID, R1), (categoryName, restaurant), (geom, pol3), (typeOfFood, French), (price, cheap)]$ $[(ID, R2), (categoryName, restaurant), (geom, pol4), (typeOfFood, French), (price, expensive)]$ $[(ID, R3), (categoryName, restaurant), (geom, pol5), (typeOfFood, Italian), (price, cheap)]$
Eiffel Tower (1 occurrence)	$[(ID, E), (categoryName, EiffelTower), (geom, pol6)]$
zoos (1 occurrence)	$[(ID, Z), (categoryName, zoo), (geom, pol7), (price, cheap)]$

Figure 1. Set of instances for the categories in Table 1

OID	Items
O_1	([(ts_date,04/08/2008), (ts_time,14:05), (tf_date,04/08/2008), (tf_time,14:33), (ID,R2),(categoryName,restaurant), (geom,pol4), (typeOfFood,French), (price,expensive)]) ([(ts_date,04/08/2008), (ts_time,15:10), (tf_date,04/08/2008), (tf_time,16:05), (ID,E), (geom,pol6)]) ([(ts_date,04/08/2008), (ts_time,17:30), (tf_date,04/08/2008), (tf_time,18:48), (ID,R3),(categoryName,restaurant), (geom,pol5), (typeOfFood,Italian), (price,cheap)]) ([(ts_date,08/08/2008), (ts_time,06:22), (tf_date,08/08/2008), (tf_time,07:05), (ID,R1), (categoryName,restaurant), (geom,pol3), (typeOfFood,French), (price,cheap)]) ([(ts_date,08/08/2008), (ts_time,10:00), (tf_date,08/08/2008), (tf_time,13:00), (ID,E), (geom,pol6)]) ([(ts_date,08/08/2008), (ts_time,17:10), (tf_date,08/08/2008), (tf_time,18:17), (ID,R1), (categoryName,restaurant), (geom,pol3), (typeOfFood,French), (price,cheap)])
O_2	([(ts_date,03/08/2008), (ts_time,11:00), (tf_date,03/08/2008), (tf_time,11:15), (ID,Z),(geom,pol7), (price,cheap)]) ([(ts_date,08/08/2008), (ts_time,18:30), (tf_date,08/08/2008), (tf_time,21:00), (ID,Z),(geom,pol7), (price,cheap)]) ([(ts_date,19/08/2008), (ts_time,09:00), (tf_date,19/08/2008), (tf_time,10:20), (ID,R1), (categoryName,restaurant), (geom,pol3), (typeOfFood,French), (price,cheap)]) ([(ts_date,19/08/2008), (ts_time,17:00), (tf_date,19/08/2008), (tf_time,18:12), (ID,R2), (categoryName,restaurant), (geom,pol4), (typeOfFood,French), (price,expensive)])

Figure 2. An instance of the ToI

R1	CONSTRAINT ← [CONDITION]
R2	CONDITION ← λ
R2	CONDITION ← EQ
R2	CONDITION ← EQ $\wedge$ CONDITION
R3	EQ ← attr = 'constant'
R3	EQ ← attr = @vble
R3	EQ ← functionName(attr, ...) = 'constant'
R3	EQ ← functionName(attr, ...) = @vble

Table 2. Grammar for constraints

'@' symbol), and functions of n arguments: an expression fn(attribute, 'ct1', 'ct2', ... , 'ct_{n-1}'), $n \geq 1$, is a function where the *first* parameter is an attribute and all the other ones are constants. □

The grammar for the constraints is given in Table 2. The regular expression language is built in the usual way, supporting the standard operators, with the usual precedence: '()','*','+','?','.','—'. The language also supports variables (strings preceded by '@').

Example 2 *The expression* $[].[price = 'cheap']$ *includes two constraints. The first one is an empty condition, satisfied by all the items in an instance of a table of items (in what follows, ToI). The second one expresses the equality condition. In our running example it is satisfied by the items identified by Z, R1 and R3.* □

We now give the intuition of RE-SPaM, and how it substantially improves other proposals. For example, existing efforts force the user to enumerate the IDs of the items to express disjunctions, like $(A|B|C|D)^*$. When the number of items becomes large, this solution would not be applicable. RE-SPaM allows writing *concise expressions* using the semantic information available.

Q1: Trajectories of tourists who visit hotel H1, then optionally stop at restaurant R3 and the Zoo, and either end at H1 or visiting the Eiffel Tower.

[ID='H1'].([ID='R3'].[ID='Z'])*.([ID='E']|[ID='H1'])

Q2: Trajectories that visit hotel H1, then, optionally visit different places, and finish at the Eiffel Tower or at H1.

[ID='H1'].[]*.([ID='E']|[ID='H1'])

Empty conditions allow avoiding the enumeration of all the items. If an expression includes an empty condition, during the mining process it is instantiated with all the IDs of the category instances.

Q3: Trajectories starting at a place such that *price* is an attribute of the item representing this kind of place, then stop either at the zoo or the Eiffel Tower, and end up going to a place that serves French food, and has the same price range as the initial stop.

[price=@x].([ID='Z'] | [ID='E']).[typeOfFood='French' ∧ price=@x]

Q3 introduces the use of variables. In our running example, 'cheap' and 'expensive' are the only possible values for prices; thus, the only valid combinations are: cheap-cheap and expensive-expensive. Sequences such as {H1 Z R1} and {Z E R2} do not satisfy the query. The first one because hotel H1 is not characterized by price, the second one because Z has cheap prices but R2 is expensive. On the other hand, the sequence {Z Z R3} does satisfy the query.

Variables can also be used to constraint items according to their structure. We call these expressions, *metadata constraints*.

Q4: The constraint [price=@x]$^+$ is verified by sequences of one or more item (not necessary the same ones), all of them with the same price. In our running example, Z, R1, R2 and R3 are the items that satisfy this constraint.

4 RE-SPaM Evaluation

For query evaluation, we work with the category instances depicted in Figure 1. Temporal information associated with item occurrences is stored in the ToI (Figure 2). Computing the support of a sequence requires computing its Transformed Subitems (Definition 5).

Example 3 *Consider the regular expression* $[price = @x]$. $[price = @x \wedge typeOfFood = \text{'French'}]$. *To obtain the Transformed Subitems we will use the function* $\mathcal{F} = \{Val\}$ *over the attributes* price *(for the first subexpression), and attributes* price *and* typeOfFood *(for the second one). Now, let us denote S the sequence of the transaction with OID=O2 composed of two sub-itemsets, the second and third lines in Figure 2, each one containing occurrences of items Z and R1 respectively. The question is: which are the sub-sequences supported by S? Since the first itemset of S is composed only by item Z, all of its sub-itemsets are obtained building subsets of the Transformed Subitem TS(Z), using* $\mathcal{F}$, *and the* price *attribute. These sub-itemsets are:* $\emptyset$ *and* $\{(price, \text{'cheap'})\}$. *Analogously, the second itemset of S is composed only by item R; thus, its sub-itemsets are obtained building subsets of TS(R1), using* $\mathcal{F}$ *and the attributes* price *and* typeOfFood. *This sub-itemsets are:* $\emptyset$, $\{(price, \text{'cheap'}), (typeOfFood, \text{'French'})\}$. *Then, the subsequences of S satisfying the regular expression are the ones whose items can be transformed to* $\{(price, \text{'cheap'})\}$, *and* $\{(price, \text{'cheap'}), (typeOfFood, \text{'French'})\}$. *In our running example, these sequences are:* $\{Z\}$, $\{R1\}$, $\{R3\}$ *for the first transformation;* $\{R1\}$ *for the second transformation; and* $\{Z \;\; R1\}$, $\{R1 \;\; R1\}$ *and* $\{R3 \;\; R1\}$ *for both of them.* □

Typically, in GSP-based algorithms, frequent sequences are computed in incremental phases. At each step *k*: (1) A temporary set C_k is built using the previous set C_{k-1}. Its elements are candidate sequences of length k. (2) Each element in C_k which contains at least one sub-sequence with support less than the minimum is discarded due to anti-monotony property (C_{k-1} is analyzed). (3) The database is accessed in order to analyze support, and each element in C_k with at least minimum support is added to the set F of frequent sequences. When an empty C_k set is obtained, F contains the frequent sequences with minimum support.

In RE-SPaM, to evaluate if a sequence satisfies a regular expression $\mathcal{R}$, we build a DFA, denoted $\mathcal{A}_\mathcal{R}$ which accepts the language generated by $\mathcal{R}$. We use an idea first proposed in the SPIRIT algorithm. There, instead of using the original constraint C, a relaxed constraint C' is used during the mining process, and the second phase above is replaced with a strategy consisting in pruning the sequences in C_k which contain at least one subsequence which *satisfies* C' and does not have minimum support. In the last phase, F is analyzed to obtain the frequent sequences that *satisfy* C. Finally, building C_k is done in three phases: (i) C_k *population*: C_k is populated using the information previously obtained. (ii) C_k *pruning by* $\mathcal{A}_\mathcal{R}$: C_k is pruned using the automaton and perhaps some extra information. If a candidate sequence does not satisfy the relaxed constraint C', it is discarded at this moment. (iii) C_k *pruning by the ToI instance*: C_k is pruned using the ToI instance, as we explain later, and added to a set F of frequent candidate sequences. Finally, F is pruned using the original constraint C. There is a *final phase*, that uses all sequences in the temporary set F and proceeds as follows. First, it uses the automaton to prune all sequences which are *not accepted*. Notice that here we are using the automaton for *acceptance verification* and not for *legal* verification.

Acknowledgments. This research has been partially funded by the Research Foundation Flanders (FWO- Vlaanderen), Research Project G.0344.05, the European Union under the FP6-IST-FET programme, Project n. FP6-14915, GeoPKDD: Geographic Privacy-Aware Knowledge Discovery and Delivery, and the Argentina Scientific Agency, project PICT 2004 11-21.350.

References

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of the Int'l Conference on Data Engineering (ICDE)*, 1995.

[2] C. du Mouza and P. Rigaux. Mobility patterns. In *Proceedings of the STDBM'04*, pages 1 – 8, Toronto, Canada, 2004.

[3] M. N. Garofalakis, R. Rastogi, and K. Shim. Mining sequential patterns with regular expression constraints. In *IEEE Transactions on Knowledge and Data Engineering*, 2002.

[4] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and quering moving objects. *ACM Trans. Database Syst.*, 25(1):1–42, 2000.

[5] H. Mannila, H. Toivonen, and I. Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 210–215, 1995.

[6] S. Spaccapietra, C. Parent, M. L. Damiani, J. A. Fernandes de Macedo, F. Porto, and C. Vangenot. A conceptual view of trajectories. In *Technical Report*, 2007.

[7] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT)*, 1996.