Discovering Complex Events in Long Sequences

Marco Botta BOTTA@DI.UNITO.IT
Dipartimento di Informatica, Universitá di Torino, C.so Svizzera 185, 10149 Torino, Italy

Attilio Giordana ATTILIO@MFN.UNIPMN.IT
Dipartimento di Informatica, Universitá Amedeo Avogadro, Spalto Marengo 33, 15100 Alessandria, Italy

Paolo Terenziani TERENZ@MFN.UNIPMN.IT
Dipartimento di Informatica, Universitá Amedeo Avogadro, Spalto Marengo 33, 15100 Alessandria, Italy

Abstract

Standard data mining techniques, taken in isolation, seem to be inadequate to discover sparce patterns in long sequences. In fact, symbolic approaches show difficulty in dealing with noise, while non-symbolic approaches, such as neural networks and statistics show difficulty in dealing with very long subsequences where relevant episodes may be interleaved with large gaps. However, dynamic programming, integrated with domain specific procedural knowledge, have been succesfully applied in difficult tasks, such as DNA decoding. Nevertheless dynamic programming is a very simple technique and it is not immediately applicable when patterns are structured and domain knowledge nedds to be taken into account. In order to circumvent this problem, we propose an incremental framework, where logics and string matching algorithms based on dynamic programming are combined together. In this framework, the knowledge extraction process follows a machine learning paradigm combining inductive and deductive learning, where deduction steps can be interleaved with induction steps aimed at augmenting a weak domain theory with knowledge extracted from the data. An extensive evaluation of the method is provided using artificial data.

1. Introduction

Discovering knowledge from long temporal or spatial sequences is a task attracting increasing attention in the literature owing to its significance in many real world applications. However, the task is not easy and the difficulty increases along with the length of the sequences and patterns to be discovered. In this paper, we propose a framework aimed at supporting the process of discovering the occurrence of complex events in very long sequences. We assume that a complex event is a partially ordered set of short chains (episodes) of elementary events (instants) interleaved with gaps where irrelevant facts may occur. Moreover, we assume the presence of noise in the instances occurring in real world sequences, which can make the events hard to recognize. Episodes are represented as strings of symbols, being a symbol a label assigned to an instant and have the same meaning as in (Cohen, 2001; Cohen & Niall, 2001). Noise may be modeled as insertion, deletion and substitution errors according to a common practice followed in Pattern Recognition.

The typical approaches used in Pattern Recognition to learn and to match patterns in sequences affected by such a kind of errors (e.g. dynamic time warping and hidden Markov Model (Rabiner, 1989)) are based on dynamic programming algorithms.

However, pure Dynamic Programming is on the one hand to simple for dealing vith structured patterms and on the other hand still has a too high computational complexity when long sequences are to be processed. Important results have been obtained in difficult tasks such as speech recognition (Rabiner, 1989) and DNA analysis (Alstchul et al., 1990; Lipman & Pearson, 1985; Gussfield, 1997), with ad hoc algorithms where dynamic programming is combined with other techniques in order to account for structural knowledge and to reduce the complexity. However, developing domain specific algorithms may be costly, and then impractical, for many data-mining applications.

In order to make easier developing task specific algorithms for extracting knowledge from the considered class of spatial/temporal sequences, we propose to integrate standard dynamic programming algorithms with a restricted first order logic language used to specify structural and positional information in the events. The structure of a complex event is represented, as a collection of episodes localized in different regions of a possibly long sequence according to a given set of constraints. Episodes are seen as logical constants and for this reason will be referred to as atomic events in the following. Dynamic programming deals with the problem of detecting episodes in sequences affected by noise, finding the most plausible instances. At the logical level, a plausibility measure accounts for the degree of matching. Logic resolution and dynamic programming strictly cooperate: the former provides information about the regions where dynamic programming matching (DP-matching) must be activated, the latter provides a quantitative plausibility estimate. The above framework recalls Michalski's two tiered concept representation and flexible matching (Bergadano et al., 1992), where DP-matching plays the role of flexible matching.

In the first part of the paper we describe the logic language and we show how DP- matching can be integrated with logic resolution implementing a plausible reasoning paradigm. Afterwards, we show how it is possible to start with incomplete event description and learn missing knowledge from examples. The learning procedure strictly follows an integrated deductive and inductive paradigm in the line of (Bergadano & Giordana, 1988). At a first step, a partial event description is deduced from imperfect domain knowledge. Then, the partial description is completed going through a two steps induction process where complex event structure is firstly discovered and then atomic event dscription is refined. The two steps can be iterated in an incremental cycle, where a human expert partecipates supplying domain knowledge.

2. Modeling Domain Knowledge

The Description Language (DL) contains a set of logical predicates to model the domain knowledge structure. We distinguish among three main types of concepts: **Events**, **Locations**, and **Constraints**. Using such concepts a domain expert can supply the structure of a complex event he/she is expecting to find in a sequence. In order to account for incomplete knowledge DL contains meta-construct to explicitly declare missing information.

Two basic tasks are defined on the sentences in DL: **find** and **learn**. Task **find** discovers plausible instances of a complex event in a sequence, whereas task **learn** aims at completing a sentence containing missing knowledge by induction from a set of sequences. In the following, the main features of DL are summarized.

2.1 The Description Language

Each (instance of) event takes place (*Happens* predicate) in a specific location. A location can be specified as a pair of natural numbers, representing the distance (the number of instants) of the starting point (function *Start*) and ending point (function *End*) of the event from a reference point (the beginning of the string where the event is found/learned).

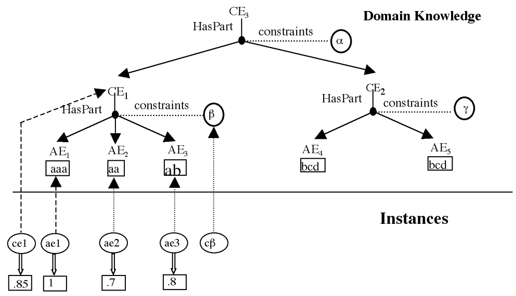

Figure 1. Example of domain knowledge describing a complex event.

Events can be either **atomic** (*AtomicEvent* predicate), or **composite** (*CompositeEvent* predicate). Atomicity is a domain-dependent property deriving from an initial abstraction step: Whenever a knowledge engineer starts to describe a domain, she/he is free to choose which events are regarded to be atomic and which are not. Atomic events are fully characterized by a string representing a chain of instants (e.g., "aab"), and are referred to by using a name. The description of a composite event consists of two main parts: (a) the definition of its components, and (b) the definition of the constraints between the components.

The first part of the description is given by means of the *HasPart* predicate (a binary predicate operating on a composite event and an event, which, in turn, may be composite or not). Currently, we focus our work on descriptions in which only conjunctions of HasPart predicates can be used; i.e., we do not consider events with alternative components. The second part constrains the relative location of the components, and is specified via the *Distance* predicate.

As an example, let us consider the domain knowledge represented in Figure 1 in a graphical form. The (composite) event CE_3 is composed by CE_1 and CE_2, which are constrained as follows (indicated by the label α in the figure and in the following):

$$\begin{aligned}\alpha : & Distance(Start(CE_2), End(CE_1), [10, 20]) \wedge \\ & Distance(End(CE_2), End(CE_1), [15, 100])\end{aligned} \quad (1)$$

In turn, CE_1 is composed by the (atomic) events AE_1, AE_2 and AE_3, and CE_2 is composed by the (atomic) events AE_4 and AE_5. β and γ are conjunctions of Distance constraints between AE_1, AE_2, AE_3 and between AE_4, AE_5 respectively. Dashed arcs represent InstanceOf predicates (e.g., the leftmost dashed arc represent InstanceOf(ce_1,CE_1)). Double arcs represent plausibilities (e.g., Plausibility(ce_1,0.85)). An equivalent representation of such a knowledge can be as in the following:

$$\begin{aligned}& CompositeEvent(CE_1) \wedge CompositeEvent(CE_2) \wedge \\ & CompositeEvent(CE_3) \wedge AtomicEvent(AE_1, "aaa") \wedge \\ & AtomicEvent(AE_2, "aa") \wedge AtomicEvent(AE_3, "aba") \wedge \\ & AtomicEvent(AE_4, "bcde") \wedge AtomicEvent(AE_5, "bcd")) \wedge \\ & HasPart(CE_3, CE_1) \wedge HasPart(CE_3, CE_2) \wedge \\ & HasPart(CE_1, AE_1) \wedge HasPart(CE_1, AE_2) \wedge \\ & HasPart(CE_1, AE_3) \wedge HasPart(CE_2, AE_4) \wedge \\ & HasPart(CE_2, AE_5) \wedge \alpha \wedge \beta \wedge \gamma\end{aligned} \quad (2)$$

A specific instance ce_1 of the composite event CE_1 can be represented as follows:

$$\begin{aligned}& InstanceOf(ce_1, CE_1) \wedge Happens(ce_1, [10, 29]) \wedge \\ & Plausibility(ce_1, 0.85) \wedge HasPart(ce_1, ae_1) \wedge \\ & HasPart(ce_1, ae_2) \wedge HasPart(ce_1, ae_3) \wedge \\ & InstanceOf(ae_1, AE_1) \wedge Happens(ae_1, [10, 12]) \wedge \\ & Plausibility(ae_1, 1.0) \wedge InstanceOf(ae_2, AE_2) \wedge \\ & Happens(ae_2, [15, 17]) \wedge Plausibility(ae_2, 0.7) \wedge \\ & InstanceOf(ae_3, AE_3) \wedge Happens(ae_3, [26, 29]) \wedge \\ & Plausibility(ae_3, 0.8)\end{aligned} \quad (3)$$

The Plausibility predicate reflects the fact that partial matching of events (see Section 3) are taken into account.

Notice that the notion of distance is very flexible: for instance, it can be either interpreted as temporal or spatial distance. Moreover, the literature about constraints in the AI field has widely shown that the distance primitive is very powerful (Davis, 1987; Dechter et al., 1991).

For instance, in the temporal area, two main frameworks based on distance constraints (also called *bounds on differences* (Davis, 1987; Dechter et al., 1991)) have been used. The STP framework only takes into account conjunctions (sets) of distances (Dechter et al., 1991), and can be used to model precise or imprecise temporal locations (dates), durations, delays between points, and different forms of qualitative temporal constraints between time points and/or time intervals (see (Meiri, 1991; Brusoni et al., 1997)). The STP framework has very nice computational properties: correct and complete propagation of the constraints (e.g., for consistency checking) can be performed in cubic time, and can give as output the *minimal network* of the constraints (i.e., the minimal constraints between each pair of entities) (Dechter et al., 1991). The minimal network can also be used for efficient query answering (Brusoni et al., 1995) and incremental insertion of constraints (Console & Terenziani, 1999). In many applications, a reference point is also taken into account (as a special entity). In such a case, considering the distance of the starting and ending points of each entity from the reference point, the minimal network induces a partial ordering among entities. In the TCSP framework, also disjunctions of distances are allowed. This allows for disjunctive temporal constraints, at the price of exponential algorithms to check consistency (Dechter et al., 1991).

2.2 Modeling Missing Knowledge

Two meta-symbols '?' and " are used to declare the ignorance about the structure of an event. More specifically, symbol '?' declares the ignorance about a specific object, such as a name or a constant, and symbol '*' indicates a sequence of objects whose length is not precisely defined. In the following, we will review the basic constructs:

1. *Undefined Event.* A composite or atomic event can be declared totally undefined by prefixing the corresponding predicate with symbol '?'. As an example, *?CompositeEvent(CE)*, means that composite event CE is totally unknown. Then, any further specification about the structure of CE is illegal. Nevertheless, constraints between CE and other external events are allowed.

2. *Undefined Constants.* Any numerical constant occurring in a predicate can be left undefined. As an example several constants in expression (1) can be left unspecified, as in the following:

$$\begin{aligned}& Distance(Start(CE_2), End(CE_1), [10, ?]) \wedge \\ & Distance(End(CE_2), End(CE_1), [?, ?])\end{aligned} \quad (4)$$

3. *Undefinded symbols.* The description of an atomic event is represented by a string of symbols. Some or all of such symbols may be symbol '?', as in the following expression:

$$\begin{aligned}& Description(AE_1, "aa??a") \wedge \\ & Description(AE_2, "???????")\end{aligned} \quad (5)$$

   Every '?' will be replaced by proper symbol during the learning phase.

4. *Undefined string.* An event description containing an arbitrary number of of symbols can be specified using meta-symbol '*' as a prfix to a meta-symbol

'?' according to the syntax that allows constructs as in the following: syntax:

$$Description(AE_1, "aa * (?)a") \wedge \\ Description(AE_2, " * (?)") \tag{6}$$

In expression (6) the event description may have whaterver number of symbols, whereas in expression (5) the size of the string is fully specified. In order to bound the number of items in a sequence, meta-symbol '*' may have numeric modifiers according to the following syntax: *{k,n}(?), where k and n are two integers delimiting the range of the number of symbols that have to be inserted.

5. *Atomic Event chain.* A chain of unknown atomic events can be specified using symbol '*' as a prefix to an undefined atomic event declaration. An example is as in the following:

$$*\{2,6\}(?AtomicEvent(AE_i) \wedge Description(AE_i, "(?)") \wedge \\ Distance(Start(AE_{i+1}), End(AE_i), [?,?])) \tag{7}$$

where a chain of atomic event containing from 2 to 6 items is declared. The index i will be automatically set to the current element of the chain in order to generate the names $AE_1, AE_2....AE_n$. In expression (7) the minimal specification for an atomic event chain has been chosen. However, other constraints may be stated, such as bounds on the size of the description and on the temporal intervals.

3. Detecting a Complex Event in Presence of Noise

Instances of a composite event are searched by operator Find(). As we will see in the following, this task plays a fundamental role also in the process of learning missing knowledge from a set of learning sequences. Plausibility is the tool to deal with noise in the data. Instead of requiring exact matching between the string bound to the instance and the nominal Description of an atomic event, partial matching is accepted. Plausibility is a numeric value in the interval [0,1], which reflects the degree of matching.

Plausible instances of a composite event are searched in a sequence by means of a two level algorithm based on Dynamic programming. At the first level (algorithm **DP-match-1()**) possible instances of atomic events (strings) are found generating a redundant lattice of atomic hypotheses. At the second level (algorithm **DP-match-2()**), global instances are selected choosing the best atomic instances that meet all the constraints.

3.1 Flexible Matching of atomic events

Algorithms for computing partial matching between strings have been widely investigated in Pattern Recognition and are extensively used for DNA analysis (Gussfield, 1997) and for automated text correction. The basic idea is that the effect of noise on a string "$x_1x_2x_3...x_n$" can be fully described by means of a set of rewriting rules corresponding to three types of errors:

1. *Insertion errors*; a sub-string "$...x_ix_{i+1}...$" has been rewritten as "$...x_iyx_{i+1}...$", being "y" a spurious element.
2. *Deletion errors*; an element "x_i" has been canceled.
3. *Substitution errors*; an element "x_i" has been replaced by another element "y".

Any string S_1 can be transformed into a string S_2 by applying a proper sequence of transformations according to rules 1, 2 or 3. However, every transformation T_i decreases the plausibility that S_2 is an instance of S_1, of an amount Δ_i As many alternative sequences may exist, a sequence requiring the minimum number of transformations, and then corresponding to the maximum plausibility, is searched using a Dynamic Programming algorithm (Gussfield, 1997).

3.2 "Find" Operator

We are now able to describe the two steps algorithm that defines the semantics of the Find() task. For the sake of space, we will describe just the basic version. However, many optimizations are possible in order to reduce the complexity both in time and in space. In the first step algorithm **DP-match-1**) scans the input sequence S left-to-right generating a lattice **G** containing all possible instances of atomic events AE_i composing a complex event CE. The algorithm behaves as follows. Every instant "x_t" (a symbol in S) is considered the possible start point of any atomic event AE_i, defined in the domain knowledge. Then, for every event AE_i, the algorithm evaluates the plausibility of being the string S_t starting in "x_t" the istance of AE_i. This is done by evaluating the similarity betweenS_t and the nominal description of AE_i. All instances having non-null plausibility are included into **G**.

In the second step algorithm **DP-match-2** searches for the most plausible global instance of CE that includes an instance for each one of the atomic events. The constraints among atomic events defines a partial order on the possible instances included in G, so that G is an acyclic oriented graph where nodes correspond to atomic event instances and edges correspond to constraints. Then, finding a maximum plausibility instance of CE can be reduced to finding a path of maximum plausibility on **G** containing an instance for every atomic event AE_i in CE. The plausibility of a path $< g_1, g_2, g_3.....g_n >$ is computed as a function $PL(< pl(g1), pl(g2), pl(g3), ...pl(gn) >)$ of the plausibility of the single atomic instances. Being **G** acyclic, if $PL(.)$ can be put into the form:

$$\begin{aligned} &PL(< pl(g_1), pl(g_2), ...pl(g_i), pl(g_{i+1}) >) = \\ &PL(PL(< pl(g_1), pl(g_2), ...pl(g_i) >), pl(g_{i+1})) \end{aligned} \tag{8}$$

where the plausibility at step i+1 only depends on the plausibility at step i and on the plausibility of node g_{i+1}, a maximum plausibility path can be found using a dynamic programming algorithm such as Dijkstra algorithm. Otherwise the computation of this step may be exponential.

In the present implementation, all constraints predicates have a boolean semantics and flexible matching is applied to atomic event descritpions only. The plausibility of an atomic event instance is computed as the ratio:

$$PL(ae_i) = \frac{sim(ae_i, desc(AE_i))}{length(desc(AE_i))} \tag{9}$$

being $sim(ae_i, desc(AE_i))$ the "local similarity" between a possible instance ae_i and the nominal description of an atomic event AE_i, evaluated according to the measure proposed in (Gussfield, 1997). Assigning weight $w = 1$ to any matched symbol, $sim(ae_i, desc(AE_i))$ corresponds to the length of the nominal description for a perfect match; in this case $PL(ae_i) = 1$, according to (9).

Finally, the plausibility of a complex event CE is computed as the weighted sum:

$$PL(CE) = \frac{\sum_{i=1}^{n} length(AE_i) pl(ae_i)}{\sum_{i=1}^{n} length(AE_i)} \tag{10}$$

In order to satisfy condition (8) only constraints between temporally (or spatially) consecutive atomic events are allowed.

4. Extracting Missing Knowledge from Data

Knowledge extraction is seen as a cooperative task where the pattern of a complex event is discovered by going through a sequence of steps in which chunks of a priori knowledge of a domain expert may be integrated with knowledge extracted from the data.

The basic induction step is carried on by operator **Learn** and is formulated as a task of *learning conjunctive characterization in presence of logical constraints.* In other words, given an incomplete description of an event CE and a set of learning sequences **D**, each one containing at least an instance of CE [1], the probelm is to discover the largest set of atomic event instances that occurr in all sequences **D** and is consistent with CE.

This problem is equivalent to a multiple alignement problem in the presence of gaps, which is exponential in the number of sequences; only approximate solutions are tractable (Gussfield, 1997). Here, an approximated multistrategy solution is proposed, which is able to account for logical constraints stated in CE' incomplete description.

The operator **learn(CE, D)** receives in input an incomplete description CE and a set **D** of learning sequences, and returns a fully specified description CE'.

The learning procedure consists of three major steps:

1. *Knowledge Extraction.* A set of overly specific inductive hypotheses **H** is generated from CE and a set of sequence pairs sampled from $D * D$.
2. *Generalization.* Few inductive hypotheses **H'**, representative for all the set, are generalized from **H**.
3. *Refinement.* Every hypothesis $h \in \mathbf{H}'$ is refined using an evolutive algorithm in order to produce a complete description CE'.

As a particular case, the refinement step can be omitted, or, alternatively, it can be activated directly on an event description without going through the full learning procedure. In the following the three steps will be described into details.

4.1 Knowledge Extraction

The knowledge extraction phase uses an integrated inductive-deductive learning strategy and generates inductive hypotheses compatible with CE, as generalization of two sequences only. Let $< S_1, S_2 >$ be a pair of sequences extracted from $D * D$, as a first step S_1 and $S_2$are aligned putting in correspondence similar subsequences occurring both in S_1 and in S_2. The algorithm used in this step was inspired to BLAST

[1] Requiring that all sequences **D** are known to be positive examples of CE may be a too strong , the condition will be lightened by accepting that some elements off **D** be irrelevant.

(Alstchul et al., 1990). However, any algorithm able to aligne two sequences in the presence of gaps can be used as well. Let $A_{1,2}$ the set of subsequences put in correspondence by the alignment algorithm. Afterwards, CE is instantiated on the set of strings $A_{1,2}$ finding the best match compatible with all defined constraints. Undefined atomic event descriptions match any string in $A_{1,2}$, whereas the plausibility for partially defined ones depends upon the symbol that have been specified. Being description CE required to satisfy condition (8) this step can be handled using algorithm **DP-match-2**. If several alternative instances exist for CE the longest is preferred. Finally, a hypothesis h is computed by completing the description CE, according to the following procedure:

Let ae_i denote the string in $A_{1,2}$ that algorithm **DP-match-2** has bound to an atomic event AE_i. If AE_i has an undefined description, it will receive string ae_i as provisional description. In an analogous way, if the description of AE_i is partially defined, it will be completed by a proper subsequence of symbols from ae_i.

Undefined constants in spatial/temporal constraints receive a tentative value induced from the instances found in S_1 and S_2.

4.2 Knowledge Generalization

The hypotheses **H** produced in the previous step tend to be too specific, i.e. the constraints may be too tight and the event description may contain irrelevant atomic event, because generated on the basis of the alignment between two sequences only. Moreover, the event description strings may be distorted if the data are noisy.

The generalization step aims at reducing these drawbacks. The underlaying assumption is that, at least some of the sequence pairs sampled from **D*D** in the previous step, has high probability of generating inductive hypotheses almost correct. As a first step, the generalization algorithm tries to discover such hypotheses, we will denote **H'** in the following. Then, the elements of **H'** undergo to a generalization process in which irrelevant atomic events are discarded and by and the range of spatial/temporal constraints are properly extended.

The algorithm works as in the following:
Let N the cardinality of set **H**. As a first step, the similarity between all $N(N-1)/2$ hypothesis pair defined in **H * H** is computed. The similarity between two hypotheses h_i, h_j is evaluated by interpreting hypothesis h_1 in the "universe" defidned by the atomic events in h_2. In other words, this means to compute the best alignment between the set atomic events in h_i and the ones in h_j. The algorithm used is a variant of **DP-match-2()** where the similarity is evaluated instead of the plausibility. The similarity measure is the same used in (9). In the case that hypotheses h_i, h_j have different number of atomic events the algorithm will implicitly select the ones which produce the best aligment, and the other will not be aligned. For every atomic event in a $AE_{k,i}$ in hypothesis h_i, it is kept a counter $f_{k,i}$, which is incremented every time it partecipates to the alignment between h_i and another hypothesis.

Afterwards, all hypotheses, whose maximum similarity is smaller than the average are eliminated from **H**, the aim is to eliminate the hypotheses, which are likely do not contain an instance of CE. Then, the generalization process begins: Every hypothesis still remaining in **H** is simplified by pruning the atomic events $AE_{k,i}$ that have been added to CE in the knowledge extraction step and that show aligment frequency $f_{k,i}$ smaller than the average. The bounds in the intervals occurring in the spatial/temporal constraints are generalized by extending them in order to be compatible with all the alignments that have been accepted.

Finally, the similarity between the hypotheses in **H** is updated in order to account for the changes due to the pruning step. Then, **H** is sorted according to the average similarity of its mambers. The k hypotheses (being k a user defined parameter) having the largest average similarity [2] are kept in a set **H'** to undergo the refinement step.

4.3 Knowledge Refinement

The refinement step aims at tuning both the atomic event descriptions and the temporal constraints in a hypothesis $h \in \mathbf{H}'$ increasing the average similarity between h and its instances in **D**, while discarding redundances and irrelevant details. In the current implementation, the refinement step is carried on by an evolutionary algorithm, of the type described in (*Bäck*, 1995), guided by a heuristic fitness function. However, this choice is due to practial reasons, because such a kind of algorithms are easy to implement and provide a good prototyping tool.

As said above, atomic event descriptions may contain irrelevant symbols as well as miss relevant ones. Then, the refinement algorithm uses a specific strategy in order to discover the most appropriate description length

[2] Notice that this corresponds to select the hypotheses, which are close to the center of the cluster induced by the similarity measure.

together with the best assignment for the symbols.

The algorithm operates as in the following:

1. All atomic event descriptions are extended with some symbols in order be sure that all descriptions are longer than necessary.

2. A phase of global optimization is executed in order to find an assignement close to the optimum for the symbols added in the previous step.

3. Atomic event descriptions are otimized one at the time by iterating the following steps, untill the proper description is found:

   (a) One symbol is dropped from the description.

   (b) The shorter description is tuned until the fitness function approch a maximum.

   The shorter description that still exhibits a fitness close to the absolute maximum found in this process is selected, and the constants in the spatio/temporal constraints are updated in order to account for the change in the length.

5. Evaluation

The algorithm is currently under evaluation using artificial data generated by a program providing explicit control on the structure of event hidden in sequences and on the degree of noise.

5.1 The artificial data

A set of discovering problems has been generated using a program, which provides explicit control on the difficulty of the task. A problem is a dataset consisting of sequences of symbols of varying lengths. Inside each sequence some words of different length have been hidden. The used words are city names taken from a predefined set of well known cities all around the world, in such a way that their position inside the sequence is easy to find by visual inspection.

City names inside a sequence are separated by a varying number of characters randomly chosen from the English alphabet. Moreover, randomly generated subsequences of length up to 15 characters have been added at the beginning and the end of each sequence.

Finally, noise in terms of insertion, deletion and substitution of characters has been added. Two set of experiments have been designed. In the first one a sequence of problems has been generated varying the number of words ($4 \leq w \leq 8$), the word length ($5 \leq L \leq 8$) and the noise level ($N \in \{0\%, 5\%, 10\%, 15\%, 30\%\}$. For every triple $< w, L, N >$, 10 different datasets has been generated for a total of 1000 discovery problems.

(a)
gzrtucmlzuwpkiliylondonnkmyszurichbtldc
berlinmsvyptnrvenicekcwbnottawasrafnbcsc

(b)
slrlqornxvondonrbmzulrzurihpsrfjberlinade
ysxfsenicwecrxbmnotstawaiglwhxgsuho

Figure 2. Typical examples of sequences. a) no noise; b) 10% noise, i.e. every character has 0.1 probability of being affected by an error.

Examples of sequence in a data set are reported in Figure 2, where cities names london, zurich, berlin, venice, ottawa can be recognized.

The second set of experiments aimed at monitoring the ability of the algorithm at discovering hidden events when relevant sequences are mixed with irrelevant ones. Starting from the datasets generated for the first experiment, new detasets are obtained by replacing a percentage (from 10% to 90%) of sequences with others not containing instances of the target event.

5.2 The results

The most important results are summarized in Table 1, where the algorithm has been activated with a very general initial description in which only the maximum length (12 character2) of an atomic event description has been stated, and both the atomic events and the spatial constraints had to be dicovered. Table 1 reportes the error rate reported as edit distance (i.e. the minimum number of corrections) between the nominal description of the target event and the one discovered by the system.

It appears that the average error rate is about 3%. Moreover, the method sems to be little sensitive with respect to the sequence length and to noise. This means that we can expect that it could scale up to much more complex tasks. Moreover, the computational cost is quite low (about 15 min, on a Pentium III 1.3Ghz). Starting from more detailed descriptions, where more information is given, the error rate decreases quickly below 1%.

The experimentation related to the second group of experiments is not yet complet. Anyhow, preliminary results show that when 50% of irrelevant sequences are present in the dataset, the performances are still very similar to the ones reported in Table 1.

Table 1. Performances obtained in the first set of experiments

		Noise Level				
w	L	0%	5%	10 %	15%	30%
4	5	0.01	0.00	0.00	0.00	0.10
4	6	0.00	0.01	0.00	0.01	0.07
4	7	0.01	0.01	0.02	0.02	0.06
4	8	0.01	0.02	0.02	0.01	0.06
5	5	0.00	0.00	0.00	0.00	0.08
5	6	0.01	0.01	0.02	0.01	0.07
5	7	0.00	0.03	0.02	0.02	0.05
5	8	0.01	0.03	0.03	0.04	0.08
6	5	0.00	0.02	0.03	0.02	0.10
6	6	0.02	0.04	0.01	0.01	0.09
6	7	0.03	0.02	0.02	0.04	0.06
6	8	0.05	0.06	0.04	0.04	0.06
7	5	0.02	0.01	0.01	0.01	0.08
7	6	0.02	0.00	0.01	0.01	0.05
7	7	0.04	0.02	0.02	0.04	0.07
7	8	0.07	0.06	0.06	0.04	0.09
8	5	0.03	0.01	0.01	0.01	0.15
8	6	0.01	0.00	0.01	0.01	0.07
8	7	0.07	0.03	0.04	0.05	0.07
8	8	0.12	0.06	0.05	0.07	0.10

6. Conclusions

A multi-strategy approach for discovering complex events in long sequences has been proposed. The most important novelty is combining a logical description language with a string matching algorithm based on dynamic programming (DP-matching) to deal with gap and noise in sequences. The robustness and the flexibility of the method have been experimentally demonstrated. In fact, the method shows a good capability of extracting event descriptions from sequences where event instances are very sparse and noisy. Preliminary experimentation shows a significant advantage with respect to pure DP-matching approaches.

References

Alstchul, S., Gish, W., Miller, W., Mayers, E., & Lipman, D. (1990). A basic local alignment search tool. *Journal of Molecular Biology*, *215*, 403–413.

Bäck, T. (1995). Generalized convergence models for tournament- and (μ, λ) -selection. *6th International Conference on Genetic Algorithms* (pp. 2–8). Pittsburg, PA.

Bergadano, F., & Giordana, A. (1988). A knowledge intensive approach to concept induction. *Proceedings of the 5th International Conference on Machine Learning* (pp. 305–317). Ann Arbor, MI: Morgan Kauffman.

Bergadano, F., Matwin, S., Michalski, R., & Zhang, J. (1992). Learning two-tiered descriptions of flexible concepts: The poseidon system. *Machine Learning*, *8(1)*, 5–44.

Brusoni, V., Console, L., Pernici, B., & Terenziani, P. (1997). Later: Managing temporal information efficiently. *IEEE Expert*, 56–64.

Brusoni, V., Console, L., & Terenziani, P. (1995). On the computational complexity of querying bounds on differences constraints. *Artificial Intelligence*, *74(2)*, 367–379.

Cohen, P. (2001). Fluent learning: Elucidating the structure of episodes. *Proceedings of 4th Symposium on Intelligent Data Analysis* (pp. 268–277).

Cohen, P., & Niall, A. (2001). An algorithm for segmenting categorical time series into meaningful episodes. *Proceedings of 4th Symposium on Intelligent Data Analysis* (pp. 198–207).

Console, L., & Terenziani, P. (1999). Efficient processing of queries and assertions about qualitative and quantitative temporal constraints. *Computational Intelligence*, *15(4)*, 442–465.

Davis, E. (1987). Constraint propagation with interval labels. *Artificial Intelligence*, *32*, 281–331.

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, *49*, 61–95.

Gussfield, D. (1997). *Algorithms on strings, trees, and sequences.* Cambridge University Press.

Lipman, D., & Pearson, W. (1985). Rapid and sensitive protein similarity searches. *Science*, *227*, 1435–1476.

Meiri, I. (1991). Combining qualitative and quantitative constraints in temporal reasoning. *Proceedings of the National Conference on Artificial Intelligence* (pp. 260–267).

Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of IEEE*, *77(2)*, 257–286.