Soft Comput (2010) 14:211–227
DOI 10.1007/s00500-008-0395-8

ORIGINAL PAPER

Automatic detection of trends in time-stamped sequences: an evolutionary approach

Lourdes Araujo · Juan Julián Merelo

Published online: 14 January 2009

Abstract This paper presents an evolutionary algorithm for modeling the arrival dates in time-stamped data sequences such as newscasts, e-mails, IRC conversations, scientific journal articles or weblog postings. These models are applied to the detection of *buzz* (i.e. terms that occur with a higher-than-*normal* frequency) in them, which has attracted a lot of interest in the online world with the increasing number of periodic content producers. That is why in this paper we have used this kind of online sequences to test our system, though it is also valid for other types of event sequences. The algorithm assigns frequencies (number of events per time unit) to time intervals so that it produces an optimal fit to the data. The optimization procedure is a trade off between accurately fitting the data and avoiding too many frequency changes, thus overcoming the noise inherent in these sequences. This process has been traditionally performed using dynamic programming algorithms, which are limited by memory and efficiency requirements. This limitation can be a problem when dealing with long sequences, and suggests the application of alternative search methods with some degree of uncertainty to achieve tractability, such as the evolutionary algorithm proposed in this paper. This algorithm is able to reach the same solution quality as those classical dynamic programming algorithms, but in a shorter time. We also test different cost functions and propose a new one that yields better fits than the one originally proposed by Kleinberg on real-world data. Finally, several distributions of states for the finite state automata are tested, with the result that an uniform distribution produces much better fits than the geometric distribution also proposed by Kleinberg. We also present a variant of the evolutionary algorithm, which achieves a fast fit of a sequence extended with new data, by taking advantage of the fit obtained for the original subsequence.

Keywords Evolutionary algorithms · Event tracking · Data time-stamped sequences · Burst detection

L. Araujo (✉)
Departamento de Lenguajes y Sistemas Informáticos,
Universidad Nacional de Educación a Distancia, Madrid, Spain
e-mail: lurdes@lsi.uned.es

J. J. Merelo
Departamento de Arquitectura y Tecnología de Computadores,
Universidad de Granada, Granada, Spain
e-mail: jj@merelo.net

1 Introduction

The analysis of information sequences has become a critical task nowadays. One of the challenges is to detect what are the emerging topics people are talking about, mainly online, what is usually called *buzz*. Creating buzz about a product (and measuring it afterwards) is critical to its success, and news agencies and governments are interested in following and measuring it in order to find out what are the most important topics in the public opinion.

However, even as buzz can be visually discovered easily, it is more difficult to detect and measure automatically, so that alerts can be set when a particular topic is *buzzing* above the rest. In order to do that, the arrival time of a significant group of terms has to be recorded, and a model designed for a set of frequencies or states applied so that state changes (increasing or decreasing buzz) can be used to detect and track buzz involving those terms.

A possible model would be as follows: let us assume a sequence of N documents arriving along a period of time T, and a set of S frequencies, ranging between 0 and 1. We have a double goal: determining the most appropriate frequency for each interval between one arrival and the next one and grouping together intervals with "similar" frequencies, i.e. with relative stability, in order to wash out the noise, i.e. random fluctuations which do not mean a real change in the frequency.

Figure 1 shows different possible fitting curves (or fits) for the frequencies of a sequence of documents which have arrived at the instants of time marked in the *X* axis. If the chosen probabilistic model does not penalize changes of state, the optimum curve would be the one which assigns to each interval the most appropriate frequency, which, in general, amounts to a change of state for each arrival. In this case, the chosen fit for the arrival marked in the *X* axis in Fig. 1 would be the lowest one. However, we want to assign the same state to consecutive intervals with similar frequency, ignoring in this way the changes produced by the randomness in the arrivals. This can be done by penalizing the change of state in different ways, i.e. by introducing in the function that measures the agreement between the fitting curve and the frequencies of document arrivals, a factor that reduce the function value with the number of changes of state.

Kleinberg (2003), whose method has inspired this work, has developed a framework which formalizes these ideas. He proposes a probabilistic automaton to model the frequency of arrival of documents in different intervals. The state of the automaton at a particular time determines the expected frequency of document occurrences, while transitions between states are probabilistically modeled (Elwalid and Mitra 1993). A burst of documents for a topic can therefore be identified by the period in which the automaton has stayed in a high frequency state. Given a sequence of documents related to a particular topic, the Viterbi dynamic programming algorithm (Forney 1973) can be applied to determine the sequence of automaton states which optimizes the measure defined by the chosen probabilistic model. This algorithm was initially designed to find the most probable path in a Markov chain which produces a given sequence of tags, and it is now extensively applied to a wide range of problems (Rabiner 1990). The Viterbi algorithm sets up a probability matrix, with one column for each date index *t* and one row for each state in the automaton. Each column has a cell for each state q_i in the automaton where the algorithm puts the maximum probability of the previous paths arriving to the node corresponding to the cell.

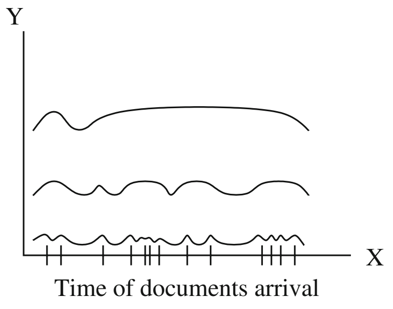

Fig. 1 Some possibles fits for a sequence of documents. The *y* axis would represent the number of documents arrived in a particular time period

The sequences of document dates to model can be obtained from different environments; in many cases they are long sequences of documents appearing along unlimited periods of time, such as chat logs, e-mails and news stories. Since a dynamic programming algorithm is limited by memory and efficiency constraints, as the length of the sequence or the number of automaton states grows, it makes sense to explore approximate or stochastic search methods in order to achieve tractability. Furthermore, what is really interesting for most applications, such as the detection of trends in sequences of documents, is the detection of significant changes of state, i.e. the detection of clear changes in the average intensity of document arrivals, not the achievement of a very precise optimum numerical result for the cost function. For these reasons, this work investigates the application of evolutionary [for which texts such as (Goldberg 1989; Michalewicz and Fogel 2004; Eiben and Smith 2003) can be good introductions] algorithms to the problem.

The aim of this work is to provide an efficient method to find a "good" fit for a sequence of dates. In our experiments those dates represent the arrivals or appearance of documents in a particular environment (such as e-mail or news feeds). Usually, the sequence presents gaps of different size between arrivals. The problem is to discern if the change in the frequency of arrivals is representative enough to be considered a change in the trend of interest on the topic in the environment, or it is just noise in the data. If we have a method to perform this discrimination, we can build a fit for the data with the frequencies which have been detected in different intervals of the arrival sequence. The obtained fit can be used for predictive applications, such as finding if interest on a topic is increasing (the last change of frequency is from lower to higher) or decreasing (otherwise), or having an estimate about when the next arrival will occur.

Moreover, there are many other applications for the obtained model. For example, it can help to detect relationships between different topics if there is a correlation in state changes for the sequences of documents collected in the same period, which would help to find emerging *concepts* composed of more than one word. The model can also be useful in identifying hierarchies between topics: the fit of one at a higher level can be obtained as the sum of the fits for those at lower level. Moreover, some correlation has been found between the buzz generated in blogs, and the Amazon online bookstore sales rank (Gruhl et al. 2005): online postings can be used to predict spikes in sales rank.

In this paper we adopt the Kleinberg model; an alternative model assuming a Poisson distribution of frequencies has been explored in another paper (Araujo et al. 2006). In addition to use an alternative algorithm for fitting frequencies, in this paper we have also studied alternative cost functions to the one proposed by Kleinberg (2003) for modeling the change of state when new documents arrive. We have built a number of artificial sequences, for which the best fit is known, in order to be able to compare the results obtained with the different cost functions. Another issue to investigate is what is the best distribution of states: uniform (all finite automata states distributed uniformly over the frequency interval) or exponential (states separated by intervals increasing exponentially in size). We have checked both distributions in real-world sequences.

We have also researched the application of our evolutionary algorithm to quickly fit a sequence where messages keep arriving, taking advantage of the previously fitted model by using it as a seed for the evolutionary algorithm.

The rest of the paper is organized as follows: next section examines the state of the art; Sect. 3 describes the model proposed by Kleinberg and the variant we are proposing here; Sect. 4 is devoted to describe an evolutionary algorithm used to find the optimal fit of frequency assignments; Sect. 5 presents and discusses the experimental results and Sect. 6 its extension to the incremental detection of changes in date sequences. Finally, Sect. 7 draws the main conclusions from this work and discusses future lines of research.

2 State of the art

Burst detection as applied to documents appearing in the web is an important part of a set of tasks usually grouped under the term *text mining* (although it can, in principle, be applied to any kind of events, as the authors did in Araujo et al. 2006). The duration of bursts, among other factors, can be used to create a *buzzness index* (Yi 2005), or to predict sales of items based on the buzz they are generating (Gruhl et al. 2005). Kumar et al. (2004) also propose using this method to analyze the evolution of links in blog communities. Different techniques for data sequences are surveyed by Muthukrishnan in (Muthukrishnan 2003), describing various models that have been proposed to describe data sequences, including Time Series Model, Cache Register Model, and Turnstile Model. The Kleinberg approach (Kleinberg 2003), as well as ours, belongs to the Time Series model category, tackled with Markov methods. There are alternatives, like the Topics Over Time (TOT) (Wang and McCallum 2006) algorithm, that uses the Latent Dirichlet allocation model, as opposed to a finite state automaton (FSA), focused on topic discovery, more than on topic tracking.

More similar to our work is the paper by Gollapudi and Sivakumar (2004), although it uses a different kind of approach (relational records and metric spaces), applied to multi-dimensional data (as opposed to the mono-dimensional data—dates used in this paper). The emphasis is also on fast and online calculation. The work by Ihler et al. (2006), has a similar aim that ours, but it uses a Poisson process to model arrivals, in the same way we did in Araujo et al. (2006).

Other papers, reviewed by Kleinberg (2006), have focused in identifying time segments in which the appearance of documents of a particular topic can be considered relatively stable. The frequency of occurrences can be very noisy, i.e. with many random changes, which hinders the identification of intervals of similar frequency. Different statistical techniques (Kleinberg 2003; Bingham et al. 2003; Girolami and Kaban 2004) have been applied to analyse temporal changes in document sequences.

Another approach is to focus on studying the rise and fall of frequencies to detect trends in sequences by identifying changes in the frequencies along given periods of time. Charikar et al. (Charikar et al. 2002) have proposed an algorithm for finding the most frequent elements in a sequence, which is also adapted to find elements whose frequencies change the most.

What we will attempt in this paper is to analyze the best way to fit a FSA to an event sequence model, optimizing at the same way algorithm speed and accuracy.

3 Problem models

One of our aims in this paper is to devise a model which generates a sequence of events and set its parameters so that the sequence under study might have been generated with it with a high probability. In order to model event occurrences in a sequence we are going to use a FSA, inspired by models for network traffic in queuing theory (Elwalid and Mitra 1993) and on Hidden Markov Models (Rabiner 1990).

According to this approach, a source emits documents at a rate which depends on the state of the FSA at a given point in time. A traffic burst begins with a transition from a state of lower rate of emission to one of higher rate. Kleinberg chooses an exponential density function $f(x) = \alpha e^{-\alpha x}, \alpha > 0$ to model the probability density of waiting times between arrivals. In the simplest case, we can distinguish between only two different states, with high and low frequency respectively. In state q_0 the automaton emits documents at a low rate, which gives rise to a probability density for the intervals $f_0(x) = \alpha_0 e^{-\alpha_0 x,}$ while state q_1 has a higher emission rate, which gives rise to a probability density for the gaps given by $f_1(x) = \alpha_1 e^{-\alpha_1 x,}$ with $\alpha_1 > \alpha_0$. Now the question is how to model the probability p with which the automaton changes its state between the emission of two consecutive documents. It is assumed that p is independent of previous emissions and state transitions. Assuming this probability distribution, we can compute the probability of a sequence $q = \{q_1, \ldots, q_n\}$ of state transitions conditioned to the sequence $x = \{x_1, \ldots, x_n\}$ of gaps observed between the $n + 1$ documents arrived in the sequence. The state sequence which maximizes this probability $P(q|x)$ is the one which minimizes the cost function $c(q|x) = -lnP(q|x)$. Applying this condition, the following formula is obtained

$$c(\mathbf{q}|\mathbf{x}) = b \ln \left(\frac{1-p}{p} \right) + \sum_{t=1}^{n} -\ln f_{i_t}(x_t) \qquad (1)$$

where b is the number of state transitions done to emit the sequence, i.e. the number of times in which $q(t) \neq q(t+1)$. In formula (1), we can observe that the fewer state transitions, the smaller the first term, while the better the state sequence fits the observed sequence of gaps x, the smaller the second term. Therefore, it is expected that the optimum sequence of states fits well the gap sequence with as few changes in the size of the gaps as possible, depending this inertia on the parameter b.

This simple two-state model is then extended to one of infinite states, providing a different one for each possible intensity of emission. Kleinberg proposes an automaton with an initial state q_0 whose corresponding density function $\alpha_0 e^{-\alpha_0 x}$ is assigned an emission rate $\alpha_0 = n/T$, where n is the number of gaps between documents emissions and T is the total length of the considered period of time; i.e., α_0 corresponds to a perfectly uniform event emission. For the remaining states $q_i, i > 0$, the assigned emission rate is $\alpha_i = \alpha_0 s^i$, where $s > 1$ is a scaling parameter, i.e. the smaller the gap, the greater the intensity. Then, by analogy with the two-state model, the cost function which the selected sequence of states must minimize, is

$$c(\mathbf{q}|\mathbf{x}) = \sum_{t=0}^{n-1} \tau(i_t, i_{t+1}) + \sum_{t=1}^{n} -\ln f_{i_t}(x_t) \qquad (2)$$

where $\tau(i, j)$ represents the cost of a state transition from the state q_i at a given time t to the state q_j at time $t + 1$. Kleinberg, who considers that the selection of $\tau(i, j)$ is very flexible, chooses $\tau(i, j)$ in such a way that the cost of changing from a lower intensity state to a higher intensity one is proportional to the number of involved states, while there is no cost for changing from higher to lower intensity states. Specifically, the cost associated to change from state q_i to q_j, where $j > i$, is defined as $(j-i)\gamma \ln n$, γ being a parameter of the model. $\mathcal{A}^*_{s,\gamma}$ denotes the automaton of infinite states with parameters s and γ. The parameter s controls the scale for the rate values of the states, while γ, which Kleinberg sets to 1 in his experiments, controls the resistance to changing state.

Computing an optimal state sequence in an automata $\mathcal{A}^*_{s,\gamma}$ with infinite states is equivalent to computing q in one of its finite restrictions $\mathcal{A}^k_{s,\gamma}$, obtained by deleting from the automaton all states but the first k of them. This result allows establishing algorithms to compute the sequence of states for the minimum cost. For this purpose, Kleinberg adopts the standard dynamic programming algorithm used for hidden Markov Models.

However, the penalty function described above is not the only possible one and we have studied others: cost functions which also penalize the change from a state with high intensity to a state of low intensity, and measures which do not depend on the number of states. These are the cost functions we will test in this paper:

- τ_a, defined as $\begin{cases} (j-i) \ln n \text{ if } j > i \\ 0, \text{ if } i \leq j \end{cases}$, the one proposed by Kleinberg, where n stands for the number of documents in the sequence.
- τ_b, defined as $|j-i| \ln n$, similar to τ_a but now there is also a cost by changing from a higher intensity state to a lower one.
- τ_c, defined as $\begin{cases} \log(j-i) \text{ if } j > i \\ 0, \text{ if } i \leq j \end{cases}$, other penalty function with zero penalty for high-to-low intensity changes.
- τ_d, defined as $\log(|j-i|)$, the counterpart of τ_c which also penalizes high-to-low intensity changes.
- τ_e, defined as $\begin{cases} \sqrt{(j-i)} \text{ if } j > i \\ 0, \text{ if } i \leq j \end{cases}$, which has zero penalty for high-to-low intensity changes.
- τ_f, defined as $\sqrt{|j-i|}$, the counterpart of τ_e.
- τ_g, defined as $\begin{cases} (j-i)/\ln E \text{ if } j > i \\ 0, \text{ if } i \leq j \end{cases}$, introduced to avoid the dependency with the number of states. E is the total number of automaton states.
- τ_h, defined as $|j-i|/\ln E$, the counterpart of τ_g which penalizes any change of state.

The results obtained with the different functions have been compared in the section devoted to the experiments (Sect. 5).

4 The search process: evolutionary algorithm

Systems based on evolutionary algorithms (Michalewicz and Fogel 2004; Hsu et al. 2002; Galvão et al. 2004) maintain a population of potential solutions to the problem and apply some selection process based on the quality or *fitness* of individuals, as natural selection does. The population is renewed by replacing some individuals with those obtained by applying "genetic" operators to selected individuals. The most usual genetic operators are *crossover* and *mutation*. Crossover obtains new individuals by mixing, often in some problem dependent way, two individuals, called parents. Mutation produces a new individual by performing some kind of random change on an individual. The production of new generations continues until resources are exhausted or until some individual in the population is fit enough. Evolutionary algorithms have proved very useful as search and optimization methods, and have previously been applied to different issues of natural language processing (Araujo 2004), such as text categorization, inference of context-free grammars, tagging and parsing.

In our case, individuals represent sequences of state transitions in the automaton. The fitness of individuals is the cost function associated to the sequence of state transitions, and depends on the chosen probabilistic model.

4.1 Individual representation

Let E be the number of states chosen for the automaton. Let us assume a sequence of $n+1$ events along a period of time T. Then, the individuals of our evolutionary algorithm could be represented as the list of automaton states corresponding to each event. Accordingly, an individual could be a list of n genes g_i, where $g_i \in \{0, \ldots, E\}$ is the state q in which the automaton is after the event i.

$q(t_1)$	$q(t_2)$	$\cdots$	$q(t_n)$

However, the occurrence of a new event does not produce a state transition in many cases. Therefore, the sequence of transitions can be represented in a more compact manner. Thus, an individual is a variable length list, in which each position, or "gene", represents the occurrence of a sub-sequence of events which do not lead to a change of state. Each gene is composed of an automaton state and of an identifier of the first and the last events in the sub-sequence. Therefore, the individuals of our evolutionary algorithm could be represented as the list of state transitions in the automaton caused by the occurrence of events.

g_1	$\cdots$	g_f
$q(t_1), (t_1, t_{k_1})$	$\cdots$	$q(t_{f_1}), (t_{f_1+1}, t_n)$

4.2 The fitness function

For the fitness function we take, quite naturally, the cost function which defines the chosen statistical model. Thus the goal of our evolutionary algorithm is to find the sequence of state transitions $\mathbf{q} = (q_{i_1}, \ldots, q_{i_n})$ which minimizes the function

$$c(\mathbf{q}|\mathbf{x}) = \sum_{t=0}^{n-1} \tau(i_t, i_{t+1}) + \sum_{t=1}^{n} -\ln \alpha_i e^{-\alpha_i x_t},$$

with the penalty cost $\tau(i, j)$ and the state parameter, α_i, of the chosen statistical model. In order to compute this function, the implicit automaton underlying the model must be completely defined, i.e. we have to assign values to each $\alpha_i, 1 \leq i < E$. We assume that the number of automaton states has previously been fixed to E. In Kleinberg's approach, a special state q_0 is assigned a document arrival rate $\alpha_0 = n/T$, which corresponds to uniform document arrivals. For the remaining states, $q_i, i > 0$, the arrival rate is $\alpha_i = \alpha_0 s^i$, where $s > 1$ is a scaling parameter, i.e. the arrival intensity increases geometrically with i. However, after performing some experiments, we have adopted a uniform distribution of state frequencies, which provides better results, specially for the sequences of real data (see Sect. 3). An estimate of the minimum frequency can be $\alpha_0 = (1/2)r^{-1}$, with r the longest interval without events. Thus the possible frequencies are $\alpha_i = \alpha_0 \times i, i = 1, \ldots, E$.

4.3 Initial population

The first step of an evolutionary algorithm is the creation of a population of individuals, or potential solutions to the problem. In our case, these individuals represent sequences of randomly generated state transitions. The simplest way of creating one such sequence is to choose a few events at random and use them to split the whole sequence into intervals, every one of which is assigned a random state. However, some preliminary experiments we have performed have shown that such a simple strategy gives rise to a search space that is too large for the algorithm to be efficient.[1] Accordingly, we choose for a state transition only those events for which the gaps with the previous event and with the following one are sufficiently different (the size of one at least 50% longer than the other). The interval before the first transition is assigned a random state, and this state is increased or decreased in successive intervals according to whether the gaps on the left and right of the partition points increase or decrease in length, respectively. The size of this change is randomly chosen.

Figure 2 shows an example of generation of an individual of the initial population. After generating a number of partition points at random, only those that correspond to a significant change of frequency in the input data, are used to define the genes of the new individual.

[1] The number of generations required to reach the optimal fit is about ten times the one required with the strategy finally adopted.

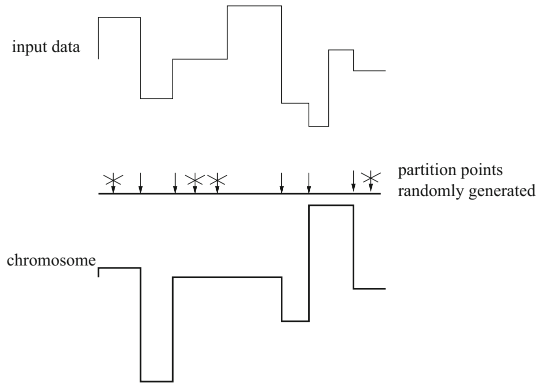

Fig. 2 Example of generation of an individual of the initial population. Only the partition points corresponding to a significant change of frequencies in the input data are accepted

4.4 Crossover operator

We have tested the classic one-point crossover and a two-point crossover operator. The first one creates two new individuals by combining two individuals in such a way that the first part of one parent up to a crossover point is combined with the second part of the other parent and vice versa. Afterwards, the best offspring substitutes the worst parent, which is a steady state strategy. The two-point crossover operator swaps the part of the parents between two crossover points chosen at random.

In both types of crossover operators, the crossover points correspond to a date in the input sequence, and the crossover gene is the one containing such date. Since, in general, the crossover gene will be different in both parents, it is necessary to design a strategy to deal with it. For the sake of simplicity, we show the adopted strategy for the one-point crossover operator. The two-point crossover operator works analogously for the two crossover genes.

The one-point crossover operator is applied following these steps:

– In order to select the crossover point, we randomly choose one of the event dates in the sequence. Then, we search in both parents the gene (crossover gene) which contains this event.
  Let us assume that the date selected as crossover point corresponds to the occurrence of event t, which belongs to the gene g_{1i} in parent 1 and to the gene g_{2j} in parent 2. The two individuals shown at the top of Fig. 3 represent that kind of parents.
– Then, the genes on the right-hand side of the crossover point (c.p.) in both parents are exchanged. The individuals at the bottom of Fig. 3 represent this type of offspring, where the gene at the crossover point is not yet specified.
– For the gene containing the crossover point (c.p.), we must decide if the subsequence of events of the crossover gene—which, in general, is different in both parents—is going to be joined or split. Experiments have shown that taking this decision at random produces bad results.[2] Accordingly, if the gaps on the left and on the right of the crossover point are comparable, the subsequence is assigned a single gene whose state is randomly selected from one of the parents. Otherwise, the subsequence is split at the crossover point in two genes, each taking the state from one parent. We consider that the gaps are similar if they differ in less than 50%. Let $d(x)$ the date of occurrence of event x. Then,

```
if (d(x) - d(x-1)) ≈ (d(x+1) - d(x)) then
    there is only a gene at the crossover position, which for offspring 1 is
```

c.p.
$q, (x - n_1, \cdots, \mathbf{x}, \cdots x + m_2)$

```
    where $q$ is randomly chosen between $q_{1i}$ and $q_{2j}$, and for offspring 2 is
```

c.p.
$q, (x - n_2, \cdots, \mathbf{x}, \cdots x + m_1)$

```
    where $q$ is randomly chosen between $q_{1i}$ and $q_{2j}$.
else
    there are two crossover genes at the crossover position, which for offspring 1 are:
```

...	c.p.		...
...	$q_{1i}, (x - n_1, \cdots, \mathbf{x})$	$q_{2j}, (x + 1, \cdots, x + m_2)$	...

```
    and for offspring 2:
```

...	c.p.		...
...	$q_{2j}, (x - n_2, \cdots, \mathbf{x})$	$q_{1i}, (x + 1, \cdots, x + m_1)$	...

Figure 4 shows an example of how one-point crossover operator creates two new chromosomes. The first one, which is a copy of the parent 1 up to the crossover point, and a copy of the parent 2 after it, will be the individual of the figure a_1 if the alternative chosen at random is not splitting the crossover gene and a_2 in the other case. Analogously, the second one will be one of the two alternative individuals, b_1 if the crossover gene is not split, or b_2 if it is.

[2] The number of generations required to reach the optimal fit is about twenty times the one required with the strategy finally adopted.

Parent 1:

g_{11}	$\cdots$	g_{1i}	$\cdots$	g_{1f_1}
$q_{11}, (t_1, \cdots)$	$\cdots$	$q_{1i}, (t - n_1, \cdots, \mathbf{t}, \cdots t + m_1)$	$\cdots$	$q_{1f_1}, (\cdots, t_n)$

Parent 2:

g_{21}	$\cdots$	g_{2j}	$\cdots$	g_{2f_2}
$q_{21}, (t_1, \cdots)$	$\cdots$	$q_{2j}, (t - n_2, \cdots, \mathbf{t}, \cdots t + m_2)$	$\cdots$	$q_{2f_2}, (\cdots, t_n)$

Offspring 1:

g_{11}	$\cdots$	g_{1i-1}	c.p.	g_{2j+1}	$\cdots$	g_{2f_2}
q_{11} $(t_1, \cdots)$	$\cdots$	q_{1i-1} $(\cdots, t - n_1 - 1)$	**?**	q_{2j+1} $(t + m_2 + 1, \cdots)$	$\cdots$	q_{2f_2} $(\cdots, t_n)$

Offspring 2:

g_{21}	$\cdots$	g_{2j-1}	c.p.	g_{1i+1}	$\cdots$	g_{1f_1}
q_{21} $(t_1, \cdots)$	$\cdots$	q_{2j-1} $(\cdots, t - n_2 - 1)$	**?**	q_{1i+1} $(t + m_1 + 1, \cdots)$	$\cdots$	q_{1f_1} $(\cdots, t_n)$

Fig. 3 Scheme of the parents and offspring in a crossover

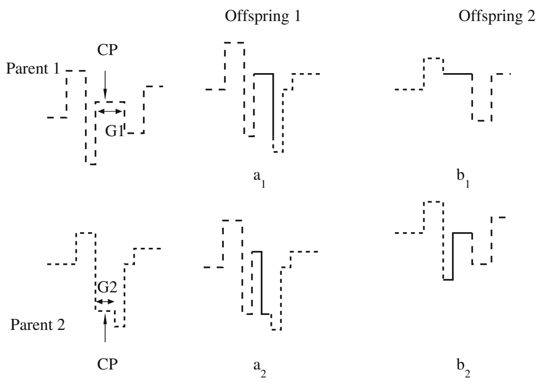

Fig. 4 Example of the one-point crossover operator. CP stands for crossover point. Each parent is identified by a different line type. The full line corresponds to the crossover gene. a_1 will be the offspring 1 if the procedure of not splitting the CP gene is randomly selected, and a_2 if it is selected to split the CP gene. Analogously, the offspring 2 will be b_1 or b_2 depending on whether the gene is split or not

4.5 Mutation operator

The mutation operator is applied to every individual of the population with a probability given by the mutation rate. Different variants of mutation have been implemented, selecting at random the one to apply in each case:

- One of these mutation operators amounts to choosing a gene at random and randomly increment or decrement its state by one unit (see Fig. 5, top).
- Other mutation operator merges two consecutive genes to produce a single one. The state of the new gene is randomly taken from one of the original genes (see Fig. 5, middle right).
- The last mutation operator splits a gene in two; each one is assigned a different state: one of them is given the state of the original gene and the other one is given the previous state plus or minus one (plus if the gap on the left of the partition point is longer than the one on the right, and minus otherwise). This operator is only applied if the gaps in both sides of the partition are different (see Fig. 5, bottom).

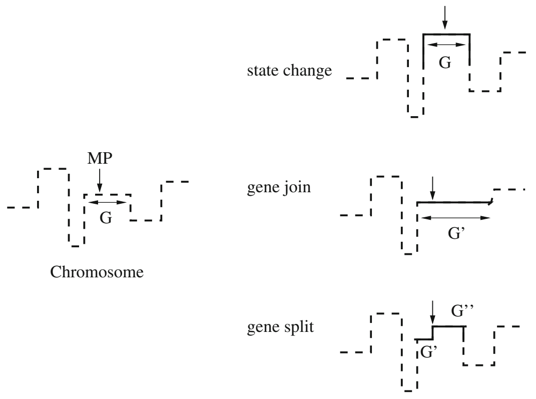

Fig. 5 Examples of the different mutation operators. *MP* stands for mutation point. The one at the top changes the state of the gene containing the mutation point; the middle one merges it with the preceding or next one, and finally the one at the bottom splits in two the gene selected for mutation

4.6 Algorithm convergence

We use a double criterion for stopping: the algorithm stops either when the specified maximum number of generations is reached or when the maximum fitness value converges, that is, when it has not changed for the last 100 generations; this number has been chosen heuristically.

5 Experimental results

Before evaluating the system on a number of real world sequences, for which the correct fit is not known in advance, we need to use artificial data, for which the fit is known, in order to test the behaviour of the adopted model and of the evolutionary algorithm. Tests for the model performed on real world sequences only can be intuitively validated if some event concerning the topic of the sequence, that causes a burst of interest, is known. For instance, in a sequence corresponding to documents on the topic *terrorism*, we can expect a burst around the date of some well known terrorist attack. However, other not so outstanding trends can not be evaluated in this way. Because of this, we have generated artificial sequences with different features, for which we know the probability distribution of the document arrivals.

Since the performance of the evolutionary algorithm is usually sensitive to settings such as the population size, the number of generations and the rates of application of the crossover and mutation operators, we have also investigated the range of values for these parameters which provide best results and compared the fits obtained by a dynamic programming algorithm with the EA; results have been published elsewhere (Araujo and Merelo 2006). Finally, we have also tested our system in some real world sequences, for which we provide other measures of the quality of the obtained fit, such as the cumulative time required for the arrival of the next document in the sequence corresponding to the obtained fit, which can be compared to the real number of messages of the fitted sequence.

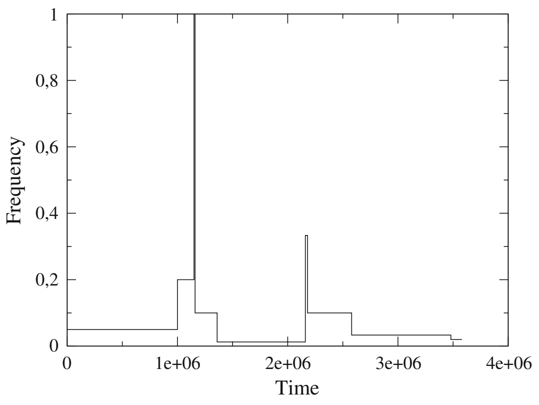

Fig. 6 Artificial sequence, *synthetic1*, generated to evaluate cost functions

5.1 Evaluating cost functions

In principle, it is not a trivial task to compare the results obtained using different cost functions. It is pointless to compare the numerical value yielded by each of them for the cost, because, in general, they are going to be different even for the same fitting curve. The best way to perform the comparison is to apply the different measures to a sequence for which we know the emission frequencies that generated it. In this case, we can compare the correct fitting curve with the curves obtained with the different cost functions. Because real world sequences present too much noise to objectively determine the best fit, we have applied this method to two kinds of artificial sequences, composed of a number of intervals spanning the whole interval. These intervals have been designed by hand to present different degrees of difficulty for the algorithm.

The program which generates sequences of the first kind reads from an input file the number of documents arriving (or event occurrences) within each interval and the gap between every two consecutive ones, which is constant along the interval. Intervals correspond to steps in the frequency-time representation. Figure 6 represents a sequence, *synthetic1*, of this type, which is composed of 220,000 dates and presents ascending and descending steps.

Figure 7 shows the fitting curves obtained by applying the Viterbi algorithm (10 states) with the different cost functions for the sequence *synthetic1*. Table 1 shows the numerical values of the frequencies appearing in the figure.

Fig. 7 Fitting curves obtained with different cost functions for the sequence *synthetic1* (Fig. 6) (clockwise from the *top-left corner*, τ_a and τ_b, τ_c and τ_d, τ_g and τ_h, τ_e and τ_f). The *arrow* marks the points in which the fit has been able to detect a narrow peak in the data

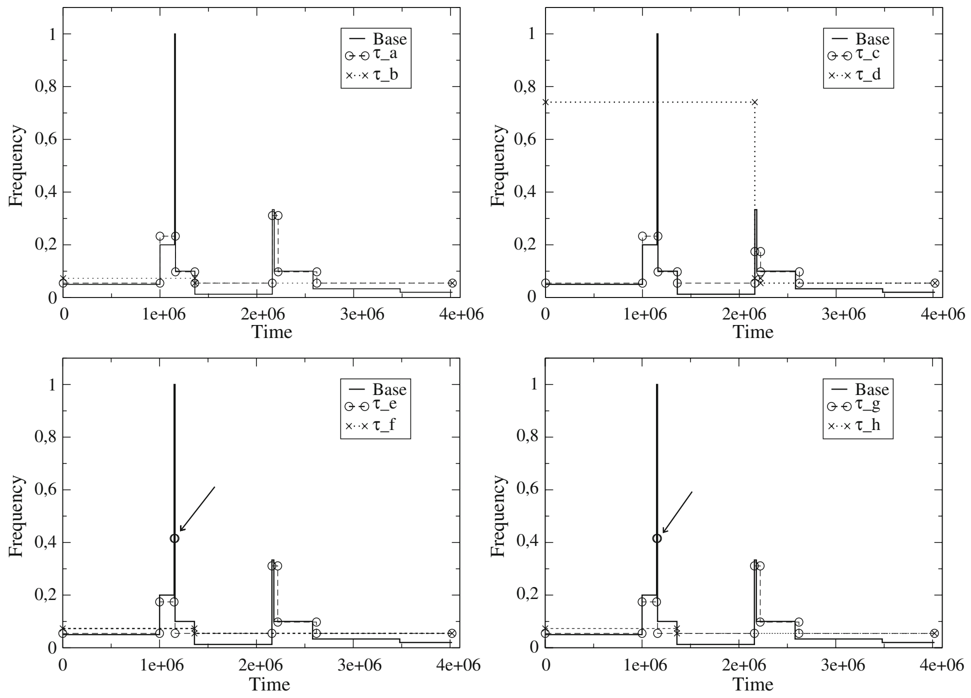

Table 1 Numerical values of the frequencies appearing in the graphics of Fig. 7 for the different cost functions

Ref. Message(State)	τ_a	τ_b	τ_c	τ_d	τ_e	τ_f	τ_g	τ_h
1,000,000 (0.5)	0.0547	0.0731	0.2327	**0.7411**	0.0547	0.0731	0.0547	0.0731
1,150,000 (0.2)	0.0547	0.0731	0.2327	0.7411	0.4153	0.0731	**0.1742**	0.0731
1,160,000 (1)	0.2327	0.0731	0.0976	0.0731	0.0547	0.0731	**0.4153**	0.0731
1,360,000 (0.1)	**0.0976**	0.0547	0.0547	0.0731	0.0547	0.0547	0.4153	0.0547
2,160,000 (0.0125)	**0.0547**	**0.0547**	0.1742	0.0731	0.3109	**0.0547**	**0.0547**	**0.0547**
2,180,000 (0.3333)	0.0547	0.0547	0.1742	0.0731	**0.3109**	0.0547	**0.3109**	0.0547
2,580,000 (0.1)	0.3109	0.0547	**0.0976**	0.0547	**0.0976**	0.0547	**0.0976**	0.0547
3,480,000 (0.0333)	**0.0547**	**0.0547**	0.0976	**0.0547**	**0.0547**	**0.0547**	**0.0547**	**0.0547**
4,020,000 (0.02)	**0.0547**	**0.0547**	**0.0547**	**0.0547**	**0.0547**	**0.0547**	**0.0547**	**0.0547**

The column at left shows the time of arrival, and the state used to generate it; values for each function are better the closer they are to the value in parenthesis. The closest value is highlighted in boldface. As it can be seen, τ_g values (next-to-last column) are consistently closer to the *real* values

We can observe that the best fits, i.e. closest to the *base* curve of frequencies used for generating the sequence, are obtained with functions τ_e and τ_g. Please note that these functions are able to detect the narrow peak in the sequence (marked with an arrow in the figure). However, we choose τ_g because it is normalized with respect to the number of states, which makes its behavior more insensitive to this parameter. Measures τ_b, τ_d, τ_f and τ_h whose changes from a higher frequency state to a lower one are also penalized, produce fitting curves that are much too flat.

The second sequence, *random1* used for evaluation has been generated via the following procedure: every time unit a document arrives with a given probability (its frequency); probabilities for this sequence are shown in Table 2. Figure 8 shows the fits obtained by applying the Viterbi algorithm to an automaton of 100 states for a sequence generated according to the probability distribution of Table 2. Table 3 shows the numerical values of the frequencies appearing in the Fig. 8. The first row of this table indicates the initial dates of each segment, the second one the last date, and the third one is the average frequency at which documents are emitted. Gaps between documents in a segment are thus random. In Fig. 8 the most accurate and smoothest fit is again obtained with the cost function τ_g. In this case τ_a produces a very poor fit.

Table 2 Probability distribution used to generate the *random1* sequence, which features random gaps

Ini	0	1001	2001	3001	4001	5001
End	1000	2000	3000	4000	5000	6000
Freq.	0.002	0.89	0.004	0.9	0.001	0.99

As a conclusion, we have used τ_g instead of the cost function proposed by Kleinberg in his paper, since it provides a better fit for known document sequences. This function avoids the dependency with the number of documents that was featured by Kleinberg's proposed one; the main difference is that while Kleinberg's function (τ_a here) uses as a scale factor the number of documents, ours (τ_g here) scales by the number of states. As can be seen in Figs. 7 and 8, the difference is bigger in the case of a randomly generated sequence (8) than in the case of a sequence with uniform gaps (7), which probably means that, in the general case, τ_g will yield better results.

Table 4 presents the results of a t test to assess the statistical significance of the differences among the different cost functions tested. The t test has been applied to the value of the area below the fitting curve resulting as

Table 3 Numerical values of the frequencies appearing in the graphics of Fig. 8 for the different cost functions

Ref. Message(State)	τ_a	τ_b	τ_c	τ_d	τ_e	τ_f	τ_g	τ_h
451 (0.002)	0.7365	0.7030	0.0179	0.0209	0.0089	0.0089	**0.0029**	0.0204
1,001 (0.89)	0.7365	0.7030	0.8832	0.9406	0.8597	0.8597	**0.8927**	0.8802
2,001 (0.004)	0.7365	0.7030	0.1477	0.0199	0.1062	0.1062	**0.0024**	0.0119
3,001 (0.9)	0.7365	0.7030	0.9316	0.9271	0.7634	0.7634	**0.8957**	0.9386
4,001 (0.001)	0.6307	0.6856	**0.0009**	0.0079	0.0044	0.0044	0.0029	0.0089
5,001 (0.99)	0.6307	0.6856	0.9925	**0.9910**	0.8063	0.8063	0.9935	0.9890

Once again, τ_g values are almost always better than the values for other functions, and always very close. The value closest to the real one is highlighted in boldface

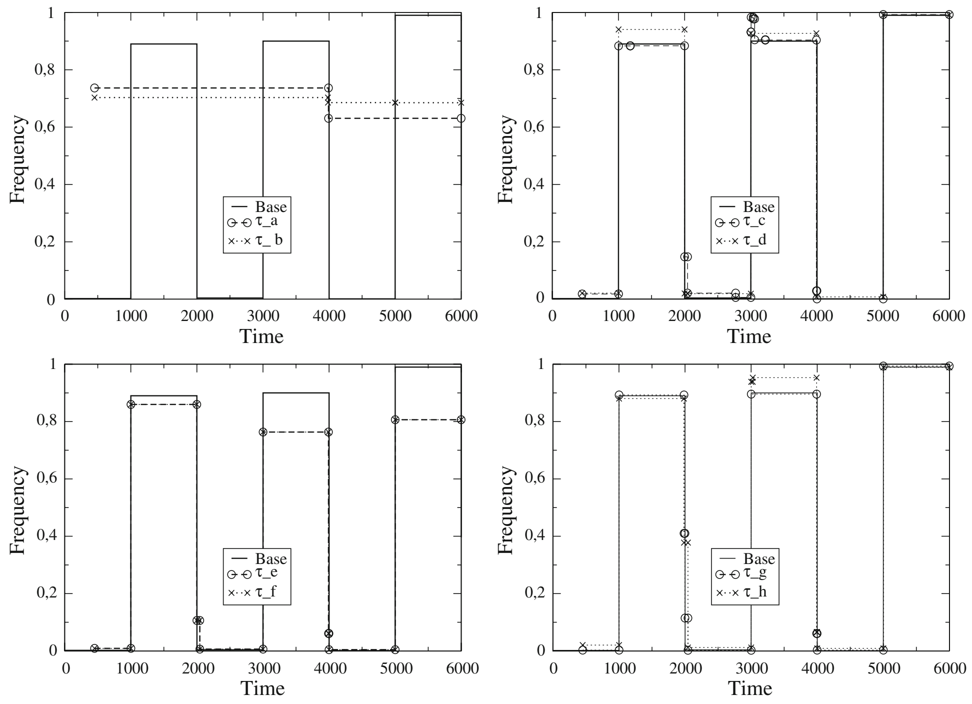

Fig. 8 Fitting curves obtained with different cost functions for the sequence generated with the probability distribution appearing in Table 2 (clockwise from the *top-left* corner, τ_a and τ_b, τ_c and τ_d, τ_g and τ_h, τ_e and τ_f). τ_g yields the best fit, as can be seen in the figure at the *bottom*

solution in each case for the data of Fig. 8. We can observe that the *t* test is 0 for the results of τ_g, and also for some other measures, thus showing its statistical significance.

5.2 Comparing the EA with a classic algorithm

Finally, in order to test the advantages of using an EA for the considered problem, we have compared its results with those obtained with the Viterbi dynamic programming algorithm. Tables 5, 6 and 7 show the values obtained for the cost function and the execution time when using a dynamic programming algorithm and the EA (best result of five runs, average and standard deviation). The cost function used has been τ_g. The EA has been run with a population size of 200 individuals, a maximum number of 200 iterations, a crossover rate of 40% and a mutation rate of 10%. The values presented for the EA are the best cost, together with its average and standard deviation and the running time (average and deviation) for five runs. Parameters for running the EA have been established heuristically; the experiments performed have been described elsewhere by the authors (Araujo and Merelo 2006). According to the obtained results, we have adopted by default the parameter setting shown in Table 8.

It can be observed that the EA always yields the shortest running time. In some cases the value obtained for the cost function with both algorithms and the number of states in the automaton is the same. There are other cases in which the value provided by Viterbi is slightly better. However, in all these cases the fitting curve obtained with both algorithms is the same and the only difference is the particular state number assigned to different steps; the relative change of state, and therefore of frequency, is maintained. For example, in the case of the *synthetic3*, Fig. 10 shows the results obtained with a 10-state automaton, with the Viterbi algorithm and the EA. We can observe that the number of steps found in each curve is the same, the size of each step and the document of the beginning and the end of each step is also the same in both algorithms, and the relative change of state is also maintained. The only difference, highlighted in the figure, is the absolute state assigned to the interval between the documents 60,000 and 75,000. Therefore, both solutions are equally useful in order to detect the burst of interest in the sequence and to estimate the future trend. Furthermore, the values obtained for the average cost and

Table 4 *T* test results for the cost functions considered. *The results are zero or near zero for* τ_a, τ_b and τ_g. The rest of the results are less than 0.5, except for τ_c and τ_h, which is just a bit greater

	τ_a	τ_b	τ_c	τ_d	τ_e	τ_f	τ_g	τ_h
τ_a	1							
τ_b	0.003	1						
τ_c	0	0	1					
τ_d	0	0	0.043	1				
τ_e	0	0	0.279	0.307	1			
τ_f	0	0	0.112	0.223	0.375	1		
τ_g	0	0	0	0	0.002	0	1	
τ_h	0	0	0.548	0.060	0.263	0.135	0.032	1

Table 5 Execution time in seconds and value of the cost function for the *synthetic2* sequence (Fig. 9) when applying the Viterbi and the Evolutionary algorithm to find the optimal sequence of automata states with different total number of states

State n.	Viterbi		Evo. Alg		
	Cost	Ex. time	Best Cost	Cost	Ex. time
5	734,721	2394.44	734,721	735242.6 ± 824.72	721.10 ± 103.12
10	704,021	4789.13	704,021	707342.2 ± 3081.98	1902.61 ± 245.23
15	699,429	7188.22	699,429	702288.4 ± 2989.61	2476.37 ± 305.70
20	696,703	9588.8	696,703	696703	2729.01 ± 294.62
25	696,210	12001.2	696,210	696210	2921.53 ± 275.33

Table 6 Execution time in seconds and value of the cost function for the sequence *synthetic3* (Fig. 9) when applying the Viterbi and the Evolutionary algorithm to find the optimal sequence of automata states with different total number of states

State n.	Viterbi		Evo. Alg		
	Cost	Ex. time	Best Cost	Cost	Ex. time
5	279,464	792.08	279,464	280897.6 ± 839.70	646.09 ± 72.12
10	277,796	1547.4	277,893	278785.4 ± 899.03	1431.92 ± 92.56
15	277,402	2319.36	277,712	279385.6 ± 980.11	1812.06 ± 115.37
20	277,306	3117.28	277,528	278980.4 ± 1114.91	2149.72 ± 136.55
25	277,260	3835.37	277,270	279472.6 ±1116.03	2161.13 ± 128.41

Table 7 Execution time in seconds and value of the cost function for the sequence *synthetic1* (Fig. 6) when applying the Viterbi and the Evolutionary algorithm to find the optimal sequence of automata states with different total number of states

State n.	Viterbi		Evo. Alg		
	Cost	Ex. time	Best Cost	Cost	Ex. time
5	795,853	2377.35	795,853	799612.8 ± 3098.99	1823.20 ± 104.62
10	794,821	4781.91	794,821	795938.2 ± 1117.64	3293.21 ± 206.75
15	794,381	5666.24	794,381	798642.0 ± 3007.618	4116.03 ± 241.21
20	794,256	9559.78	794,256	797285.0 ± 3023.408	5965.48 ± 276.92
25	794,199	11975.5	795,013	798818.8 ± 3448.08	5877.49 ± 208.23

Table 8 Parameter setting

Number of generations	200
Population size	200
Crossover type	Two-point
Crossover probability	40
Mutation probability	10
Fitness function	τ_g

standard deviation, which is below 0.5% in all cases, show that the EA is very robust. Because of the small variability observed, the fit obtained by any of the EA runs is a valid solution.

We have also studied how the number of events in the sequence to fit affect the performance on the Viterbi algorithm and the EA, for both execution time and memory. Table 9 shows the comparative results. The sequences correspond to variations in the number of documents of the artificial sequence *synthetic1*. The number of intervals and the gap between consecutive events in each interval have been maintained, but the length of the intervals has been reduced or extended to produce sequences with different event numbers. We can observe that the increase of both, the execution time and the memory requirement, with the number of document is very much larger with the Viterbi algorithm. It is more clearly shown in Fig. 11 where we can see that, though obviously the time and memory required grow with the number of documents for both algorithms, the curve for the Viterbi algorithm is much steeper than the one of the EA.

5.3 Evaluating real world document sequences

We have studied the behavior of some sequences obtained from the real world. They have been obtained from the Blogalia weblog hosting site,[3] by doing a database search of the comments table on certain words: (a) **google**, (b) **gmail**, (c) **terrorismo**[4] and (d) **atentado terrorista**.[5] Values for crossover and mutation are the same as in the previous version (40% for crossover, 10% for mutation), but the population and number of generations has been

[3] http://www.blogalia.com.

[4] terrorism.

[5] terrorist attack.

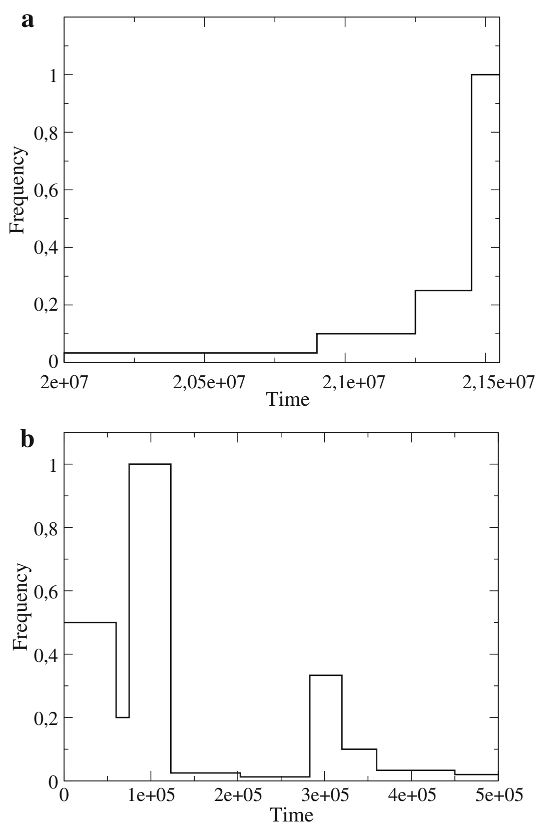

Fig. 9 Additional artificial sequences generated to evaluate the EA, which we will call *synthetic2* (**a**) and *synthetic3* (**b**)

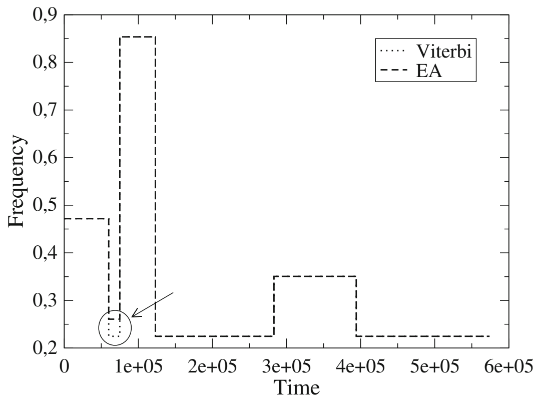

Fig. 10 Steps of the fitting curve obtained with the Viterbi algorithm and the Evolutionary algorithm for the sequence *synthetic3* (Fig. 9) with a 10 state automaton. The EA is run on a population size of 200 individuals for a maximum number of 200 generations, with a crossover rate of 40% and a mutation rate of 10%

Table 9 Comparison of the execution times and the memory requirements for sequences with different number of events

doc. number	Viterbi			Evolutionary Alg.		
	Cost	Ex. Time	Memory	Cost	Ex. Time	Memory
110,000	348,114	2624.79	36,082	348,114	2015.25	5,596
220,000	696,210	12001.2	71,041	696,210	2870.53	8,176
330,000	1,044,320	25896.11	102,842	1044,320	5964.96	11,616
440,000	1,392,430	51401.34	147,192	1,392,430	8187.25	15,052

The execution time (Ex. Time) appears in seconds, the memory appears in KB. Both algorithms have been run for an automaton of 25 states. The EA parameters have been a population size of 200 individuals, 200 generations, a crossover rate of 40% and a mutation rate of 10%

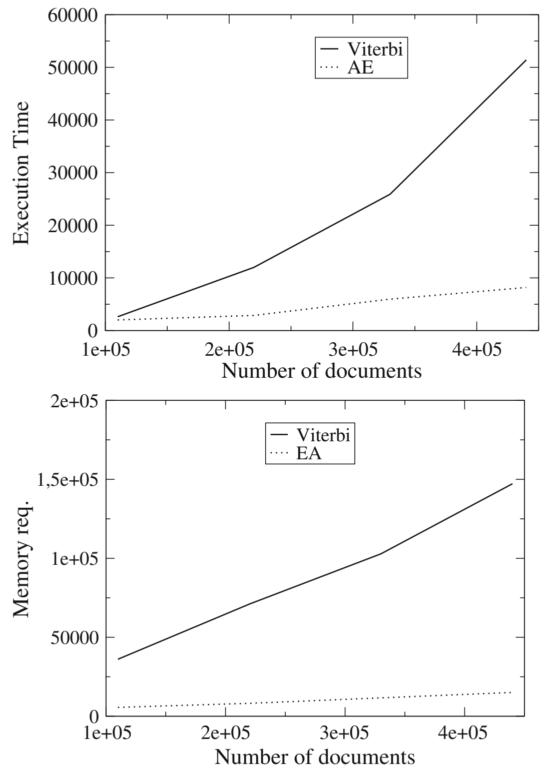

Fig. 11 Execution times and memory requirements for sequences with different numbers of documents for the Viterbi (*solid*) and the evolutionary (*dotted line*) algorithm

increased to 1,000 in the two cases, instead of 200 used in the experiments with artificial sequences. Figure 12 shows the fitting curve obtained by the EA with those parameters and using a 100 state automaton. The figure compares the results obtained using the uniform distribution of frequencies for the automata states (which we propose) with the geometric distribution proposed by Kleinberg. We can observe that the results obtained with our uniform distribution are much better than those obtained with the geometric distribution.

It becomes even clearer in the plots of Fig. 13 which show, for the input data and the two considered fits, the cumulative time required for the arrival of the next document in the sequence. In the case of the fit data, the inverse of the frequency assigned to the interval is used to calculate the required time. We can observe that the fit using the geometric distribution of frequencies severely underestimates the curve representing the input data. The goodness

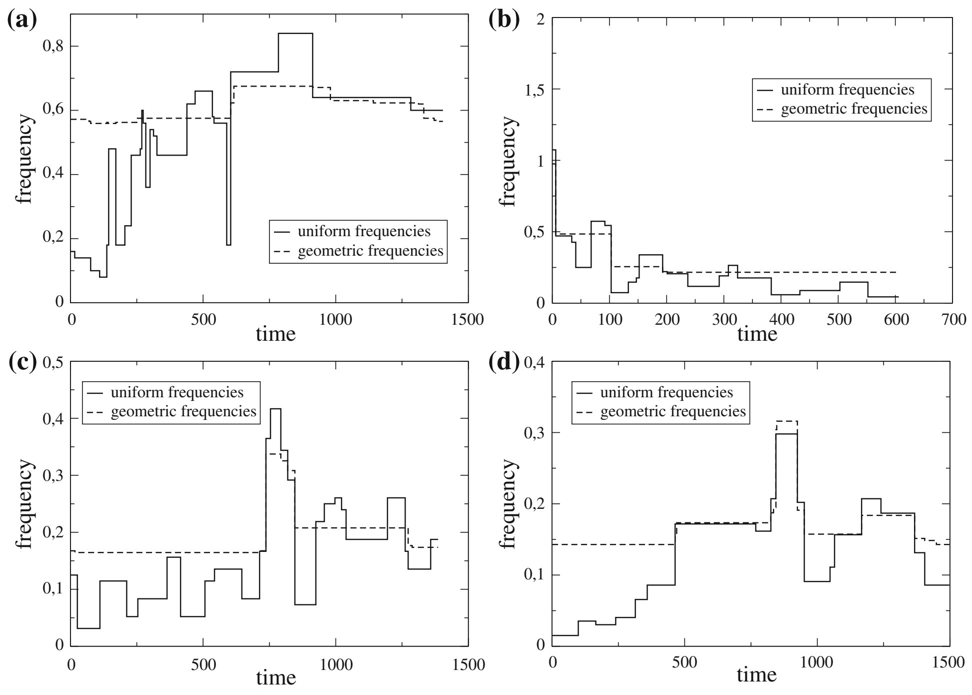

Fig. 12 Fitting curves for some real world document sequences; each sequence corresponds to blog posts from Blogalia that include the following words: **a** google; **b** gmail; **c** *atentado* and **d** *atentado terrorista*. The scale unit of x axis is days. Each graphic shows the fits obtained using our uniform distribution (*solid line*) and a geometric distribution (*dashed line*)

of the fit using our uniform distribution is patent from the plots, providing an evidence of the quality of the results. In order to provide a numerical evidence of the quality of the fits, Table 10 compares the discrepancy with the input data of the cumulative time at the date of the last document arrival, for both types of distribution of frequencies. The table shows the absolute difference as well as the percentage it represents with respect the total accumulated time. We can observe that this difference, is much smaller in the case of our uniform distribution.

We can observe that the fit for *google* (Fig. 12a) shows a great interest in the topic, along the whole period. The curve for the *gmail* sequence (Fig. 12 b) presents some bursts (approximately four), though their intensity is not very high. These bursts roughly correspond to new waves of GMail (the Google webmail system, at http://gmail.com)

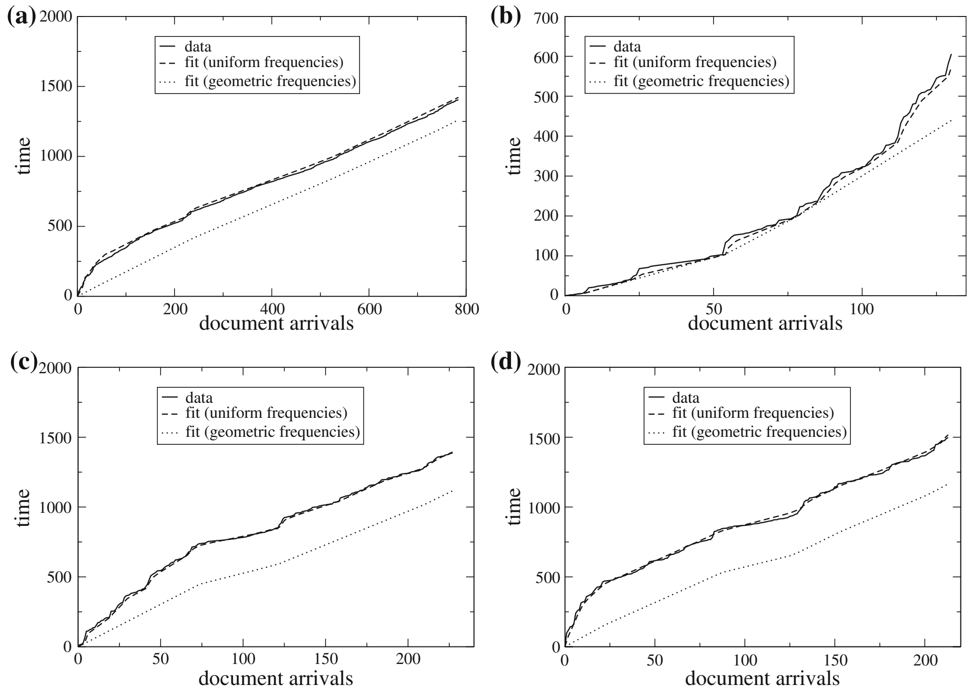

Fig. 13 Cumulative time required for the arrival of the sequence of documents in the input data and according to the frequencies generated by the fit for the sequences of Fig. 12. Each graphic shows the curves corresponding to the input data (*solid line*), and to the fits obtained using our uniform distribution (*dashed line*) and a geometric distribution (*dotted line*)

Table 10 Comparison of the discrepancy with the input data of the cumulative time of the last document arrival, using our uniform distribution of frequencies for the automata states and the geometric distribution

	Uniform freq.		Geometric freq.	
	Diff.	%	Diff.	%
google	−17.31	1.23	143.11	10.19
gmail	33.198	5.47	166.583	27.48
atentado	−7.66	0.55	269.94	19.47
at. terrorista	−18.85	1.25	332.34	22.18

invitations. The two last fits (Fig. 12 c, d) are devoted to two related topics, terrorism and terrorist attack, and thus, we can observe that although the intensity of document arrivals is greater in (d), because the topic is more general, the bursts in both sequences appear approximately at the same times, as it can be expected.

We have also checked the behaviour of the different cost functions for these real world sequences. Figure 14 shows the cumulative time required for the arrival of the sequence of documents in the input data and according to the frequencies generated by the fit obtained with each cost function. We can observe than the fit obtained using the cost function τ_g is the one which more tightly follows the curve corresponding to the input data in every sequence considered, thus confirming the results obtained with the artificial sequences. All the other cost functions show clear flaws: τ_a diverges on the `google` sequence, τ_b in gmail; in general, τ_a, τ_b and τ_d show the worst results for the sequences considered.

6 Incremental detection of changes of state in document sequences

We have also investigated the progressive detection of changes in the trends of documents sequences. Let us assume the fit for a sequence has been found, new documents arrive, and we want to detect possible changes in the trends of the corresponding topic. Clearly, the most accurate way of doing this is to apply again the algorithm for finding the best fit for the extended sequence. However, bearing in mind applications where data keeps arriving, we have investigated a way to quickly obtain a fit for an extension of a previously fitted sequence. This is done by using the previous fitting curve as a seed for the EA which searches the fit of the extended sequence. To implement this mechanism, we have modified the way in which the EA creates the initial population. A *seed individual* is created, its last gene extended with the new subsequence of document dates. Then, the initial population is created by applying the mutation operator to this seed individual, but in such a way that the last gene has a higher probability to undergo mutation. Once the initial population has been created, the EA proceeds as before.

Table 11 shows the time required to fit the sequence of Fig. 9a if a part of it, whose length appears in the first

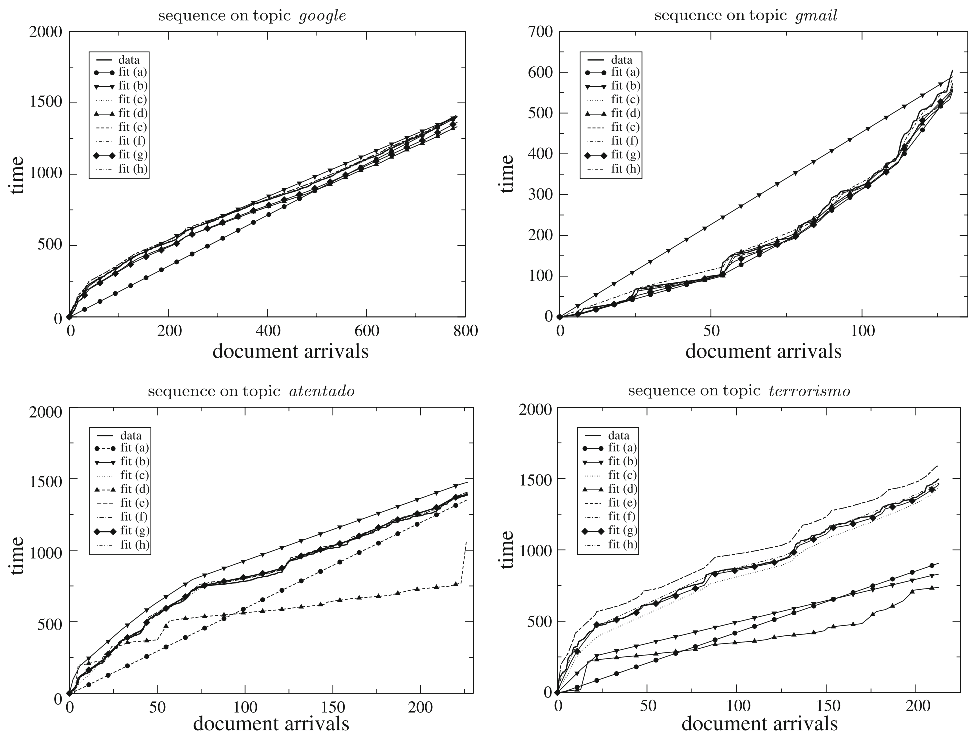

Fig. 14 Cumulative time required for the arrival of the sequence of documents in the input data and according to the frequencies generated by the fit obtained using the different cost functions for the sequences of Fig. 12. Every graphic corresponds to a different sequence, and shows the curves corresponding to the input data (*solid line*), and to the fits obtained by the different cost functions (fit (x) corresponds to the fit obtained using the cost function τ_x)

Table 11 Time (seconds) spent to fit the initial subsequence and the sequence after the arrival of new data without seed and using the previous fit as a seed for some subsequence taken from the sequence of Fig. 9a

Subst. Length	T. to fit	New Subs. length	T. w/out seed	T. w/seed	Improv. (%)
219,900	3821.31	100		141.45 (79.09)	48.64
219,000	3830.03	1,000	3895.28	144.75 (81.96)	47.70
220,000	3337.56	10,000		166.73 (79.32)	43.38

Subst. stands for subsequence, *T. to fit* for time to fit, *T. w/out* for time without, *T. w/* for time with and *Improv* for improvement

Table 12 Time (seconds) spent fitting the whole sequence using an evolutionary algorithm (*w/o seed* column) and using the previous fit as a seed for some subsequence taken from the time sequence of comments including the word 'blog' during the January 2002 to January 2006 period

Subst. length	T. to fit	New Subs. length	T. w/out seed	T. w/seed	Improv. (%)
3,032	4825.12	100	5048.49	54.6	48.31
2,632	2976.86	500		92.247	35.94
2,132	2147.50	1,000		294.97	25.74
1,132	578.79	2,000		370.41	3.7

Results correspond to the best of five runs. *Subst.* stands for subsequence, *T. to fit* for time to fit, *T. w/out* for time without, *T. w/* for time with and *Improv* for improvement

column, has already been fitted. The first column indicates the length of the previously fitted subsequence, the second one the time spent to fit it, and the third one the length of the sequence of documents which has to be added to the previous one. The fourth column presents the time needed to fit the whole sequence, while the following one presents the time spent to finish (500 generations or reach convergence) by using the previous fit as seed, with a population size of 200 individuals, a crossover rate of 40% and a mutation rate of 10%. The result for a population size of 100 individuals appears in parentheses. The last column shows the improvement achieved. We have also tested this approach on a sequence from the real world formed by the comments sent to all blogs hosted in Blogalia (http://blogalia.com) during the period January 2002 to January 2006. Table 12 shows the execution time results. In this case the algorithm works with a population size of 1,000 individual, a maximum of 10,000 iterations, crossover rate of 40%, and mutation rate of 10%. We can observe that the time required for both, artificial and the real world sequences, is very small and the use of the seed provides a large improvement in all the cases. Therefore, this mechanism can save a lot of execution time. Moreover, the quality of the results is similar to the one obtained fitting the whole sequence.

Figure 15 shows the fitting curves obtained by applying the EA to the whole sequence (Fig. 15a) and using as seed the fitting curve of the subsequence which lacks the last 1000 dates (Fig. 15b). We can see that the curves are very similar, showing the same intervals of higher interest in the topic. Furthermore, we can observe that the area corresponding to the last 1000 dates is cleaner in the fit obtained

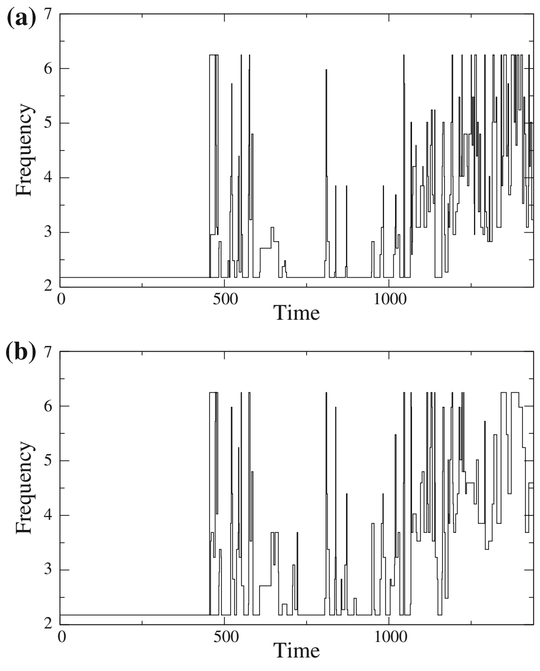

Fig. 15 Fits obtained for the sequence of blog comments including the word *blog*. In **a** (*top*) the EA has been used to fit the whole sequence, with a population size of 1,000 individuals, 10,000 generations, a crossover rate of 40% and a mutation rate of 10%. **b** shows the results when the EA is applied to a part of the sequence missing the last 1,000 dates (corresponding to the third row of Table 12), and the result of this fit is used as seed for another EA which produces the fit of the whole sequence. In this case the algorithm has been run with a population size of 1,000 individuals, 1,000 generations, a crossover rate of 40% and a mutation rate of 10%. As it can be observed, results are very similar, although the first fit is more sensitive to changes in frequency

from the seed. This is probably because in this case, due to the way in which the individuals are created, the search is centered in the area of the new dates, providing more precise results for it.

7 Conclusions and future work

In this paper, we have presented the design of a system devoted to the detection of changes on the trends of the topics of a sequence of documents, such as newscasts, e-mails, IRC (Internet Relay Chat) conversations, scientific journals or weblogs. It is based on modeling the assignment of frequencies to intervals of document arrivals (or events, in general) and obtaining an optimal fit to the data. We have designed an evolutionary algorithm to implement the model, which allows us to deal with very large sequences of documents in a reasonable time, obtaining fitting curves with a similar shape to those provided by classic dynamic algorithms. We have studied different issues of the model and its implementation, such as the criterion to change to an interval with a new frequency reflected in the cost function. We have found that penalizing the change of an interval to another with different frequency, whether it is higher or lower, provides slightly worse results than penalizing only those changes to an interval of higher frequency. We have also tested different functions for this last type of penalization, finding that some of them improve the results of the function previously used for the task of time-stamped sequence tracking. We have also observed that several functions of the tested set provide similar results. Because of this, we prefer a function which normalizes the penalization with respect of the number of frequency values considered, in order to make the results independent of this parameter. We have also found that a uniform distribution of the frequencies assigned to the automata states outperforms the geometric one proposed by Kleinberg, providing much more accurate fits of the input data. It turns out that the geometric distribution of frequencies introduces an extra penalty; with the uniform distribution the coarse-grain description of the data relies entirely on the penalizing parameters. Thus the role of tuning the penalty of the state changes is really assigned to the corresponding term of the model, which is controlled by specific parameters.

We have also designed a version of the evolutionary algorithm which dramatically reduces the time required to find the optimal fit to a sequence which is an extension of a previously fitted subsequence. This version uses the previous fit as a seed to generate the initial population, which can quickly converge if most of the sequence has been previously fitted. In this way, our system can be applied to progressively model the document sequence, and thus to detect changes on the trends of the corresponding topic. Furthermore, the fitting curves produced by the system for a sequence of documents can also be useful for other applications: the fit obtained for sequences corresponding to different topics can help to detect correlations between these topics or to study how a topic affects others.

Future lines of work will include:

1. Systematic studies of the evolutionary algorithm parameters, such as genetic operators, to improve the performance.
2. Study of correlation among document sequences, to automatically detect the occurrence of new topics composed of multi-word concepts. This can also be helped by other techniques.
3. Optimization for real-time operation, including parallelization of the algorithms.
4. More extensive checking in many different document sequences.
5. Characterization of document sequences via its model.
6. Use only the most recent part of the sequence, via a *sliding window* model that uses only the most recent information, and allows to optimize the time spent fitting the sequence to adequate it to the most recent states.

Acknowledgments This work has been supported by the Spanish MICYT projects TIN2007-68083-C02-01 and TIN2007-67581-C02-01, the Junta de Andalucia CICE project P06-TIC-02025 and the Granada University PIUGR 9/11/06 project. We are also very grateful to the anonymous reviewers, who greatly contributed to the improvement of this papers and suggested new lines of research.

References

Araujo L (2004) Symbiosis of evolutionary techniques and statistical natural language processing. IEEE Trans Evol Comput 8(1):14–27

Araujo L, Merelo JJ (2006) Automatic detection of trends in dynamical text: an evolutionary approach. http://www.citebase.org/abstract?id=oai:arXiV.org:cs/0601047

Araujo L, Cuesta JA, Merelo JJ (2006) Genetic algorithm for burst detection and activity tracking in event streams. In: Runarsson TP, Beyer HG, Burke E, Guervós JJM, Bullinaria LDWA, Rowe J, Yao X (eds) Proceedings PPSN IX, no. 4193. Lecture notes in computer science, LNCS. Springer, Berlin, pp 453–462

Bingham E, Kabán A, Girolami M (2003) Topic identification in dynamical text by complexity pursuit. Neural Process Lett 17(1):69–83

Charikar M, Chen K, Farach-Colton M (2002) Finding frequent items in data streams. In: Charikar M, Chen K, Farach-Colton M. Finding frequent items in data streams. In: Proceedings of the 29th International Colloquium on Automata, Languages, and Programming, 2002. http://citeseer.ist.psu.edu/charikar02finding.html

Eiben AE, Smith JE (2003) Introduction to evolutionary computing. Springer, Berlin

Elwalid AI, Mitra D (1993) Effective bandwidth of general Markovian traffic sources and admission control of high speed networks. IEEE/ACM Trans Netw 1(3):329–343

Forney GD (1973) The Viterbi algorithm. Proc IEEE 61(3):268–278

Galvão RK, Becerra VM, Abou-Seada M (2004) Ratio selection for classification models. Data Mining and Knowledge Discovery 8(2):151–170. doi:10.1023/B:DAMI.0000015913.38787.b3

Girolami M, Kaban A (2004) Simplicial mixtures of Markov chains: distributed modelling of dynamic user profiles. In: Thrun S, Saul L, Schölkopf B (eds) Advances in neural information processing systems 16. MIT Press, Cambridge

Goldberg DE (1989) Genetic Algorithms in search, optimization and machine learning. Addison Wesley, Reading

Gollapudi S, Sivakumar D (2004) Framework and algorithms for trend analysis in massive temporal data sets. In: CIKM'04: Proceedings of the thirteenth ACM international conference on Information and knowledge management. ACM Press, New York, pp 168–177. doi:10.1145/1031171.1031208

Gruhl D, Guha R, Kumar R, Novak J, Tomkins A (2005) The predictive power of online chatter. In: KDD'05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. ACM Press, New York, pp 78–87. doi:10.1145/1081870.1081883. http://portal.acm.org/citation.cfm?id=1081883

Hsu WH, Welge M, Redman T, Clutter D (2002) High-performance commercial data mining: a multistrategy machine learning application. Data Min Knowl Discov 6(4):361–391

Ihler A, Hutchins J, Smyth P (2006) Adaptive event detection with time-varying poisson processes. In: KDD'06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM Press, New York, pp 207–216. doi:10.1145/1150402.1150428

Kleinberg JM (2003) Bursty and hierarchical structure in streams. Data Min Knowl Discov 7(4):373–397

Kleinberg J (2006) Temporal dynamics of on-line information streams. In: Garofalakis M, Gehrke J, Rastogi R (eds) Data stream management: processing high-speed data streams. Springer, Berlin. http://www.cs.cornell.edu/home/kleinber/stream-survey04.pdf

Kumar R, Novak J, Raghavan P, Tomkins A (2004) Structure and evolution of blogspace. Commun ACM 47(12):35–39. doi:10.1145/1035134.1035162

Michalewicz Z, Fogel DB (2004) How to solve it: modern heuristics, 2nd edn. Revised and extended edn. Springer, Berlin. ISBN:3-540-22494-7

Muthukrishnan S (2003) Data streams: algorithms and applications. In: SODA'03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp 413–413. Extended version available at http://infolab.usc.edu/csci599/Fall2003/Data thms

Rabiner LR (1990) A tutorial on hidden Markov models and selected applications in speech recognition. In: Readings in speech recognition. Morgan Kaufmann Publishers Inc., Menlo Park, pp 267–296

Wang X, McCallum A (2006) Topics over time: a non-Markov continuous-time model of topical trends. In: KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM Press, New York, pp 424–433. doi:10.1145/1150402.1150450

Yi J (2005) Detecting buzz from time-sequenced document streams. In: e-Technology, e-Commerce and e-Service, 2005. EEE '05. Proceedings. The 2005 IEEE International Conference on, pp 347–352. http://ieeexplore.ieee.org/iel5/9634/30444/01402320.pdf