CERIAS Tech Report 2005-82

RELIABLE IDENTIFICATION OF SIGNIFICANT SETS OF EPISODES IN EVENT SEQUENCES

by Robert Gwadera

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

RELIABLE IDENTIFICATION OF SIGNIFICANT SETS OF EPISODES IN EVENT SEQUENCES

A Thesis

Submitted to the Faculty

of

Purdue University

by

Robert Gwadera

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2005

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Gwadera Robert. Ph.D., Purdue University, December, 2005. Reliable Identification of Significant Sets of Episodes in Event Sequences. Major Professors: Wojciech Szpankowski and Mikhail Atallah.

In this thesis we present a solution to the problem of identification of significant sets of episodes in event sequences. In order to determine the significance of an episode in a monitored event sequence, we compare its observed frequency to its frequency in a reference sequence. The reference sequence in our work is represented by a variable-length Markov model of generating symbols in the reference sequence. An episode is significant if the probability that it would have a given frequency by chance, in the reference sequence, is very small. In order to identify significant episodes we first show how to select the sliding window size to ensure that a discovered episode is meaningful and then we show how to compute a lower threshold for under-represented and an upper threshold for overrepresented significant episodes. The frequency of occurrence alone is not enough to determine significance, i.e., an infrequent episode can be more significant than a frequent one, and the significance depends on the structure of the episode and on probabilistic characteristics of the reference and monitored event streams. As an extension, we propose a novel method for providing approximate answers, with probabilistic guarantees, to a class of ad hoc sliding window queries referencing past data in data streams. The queries in that class compute the frequency of past windows that satisfy given join conditions among tuples in a window comprising multiple streams. To represent the join conditions consisting of intra-stream and inter-stream constraints between tuples in the window we introduce a concept of a 2D-episode.

1 INTRODUCTION

Stream data mining has been of great interest in many applications, including intrusion detection, alarm correlation systems in telecommunication networks, web usage analysis and computational biology. Systems designed to mine stream data usually involve a sliding "window of observation" within which the analysis is confined. We are interested in patterns of activity in an event stream, which is a chronologically ordered sequence of events (e.g., activities in a computer system, transactions in a database, etc). The patterns of activity are specified as episodes, where an episode is a partially ordered collection of events occurring as a subsequence within a window of a given size. The notion of an occurrence is as a subsequence rather than as a substring (that is, contiguity is not required), a requirement dictated by practical considerations because (for example) an "interesting" (e.g., suspicious) sequence of events does not have to be contiguous in the event stream. Furthermore we are interested in episodes that are significant with respect to a reference sequence. There are two special (basic) types of episodes that were defined: (1) a *serial episode* is a sequence of events that occur in a specified order; (2) a *parallel episode* is an unordered collection of events; and (3) an arbitrarily complex *composite episodes* can be build from an event and/or an episode by a serial and/or a parallel composition. Episode mining was introduced by H. Manila, H. Toivonen and I. Verkamo in their work "Discovery of frequent episodes in event sequences" [1]. In that paper, a sliding window was shifted a certain number of consecutive events in an event stream and the number of windows containing at least one occurrence of an episode was counted. Given a window size and a user defined threshold, an episode was considered to be frequent if the fraction of windows in which it occurred, exceeded the threshold. Then the task of mining was to discover all frequent episodes from a given class of episodes.

This thesis solves and validates through experimental results the following fundamental problems: (1) selecting the sliding window size to guarantee that an occurrence of a set of subsequence patterns is meaningful in the sense that the set does not occur almost surely in every window in an event sequence; and (2) identification of significant sets of episodes with respect to a reference sequence.

The difficulty of the problem of selecting the sliding window size stems from the fact that for an appropriately large window size any subsequence pattern will almost surely occur in every window in an event stream because the probability of its existence (at least one occurrence) as a subsequence in a window of a given size will be close to one.

The difficulty of the problem of identification of significant sets of episodes stems from the fact that the frequency of occurrence alone is not enough to determine significance, i.e., an infrequent episode can be more significance than a frequent one and the significance depends on the following factors: (1) the structure of the episode; and (2) the probabilistic characteristics of the reference and the monitored event streams. In order to decide whether a discovered episode in the monitored event sequence is significant with respect to a reference sequence, we compare its observed frequency to its frequency in the reference sequence. In our work the reference sequence is represented by a a variable-length probabilistic Markov model.

There are three main challenges faced and resolved in this research. The first is theoretical to prove that the frequency of an episode is normally distributed in Markov sources, in order to derive a formula for significance thresholds. The second is algorithmic to design efficient algorithms for computing the thresholds for any combination of episode type/model of the reference sequence. Finally, the third is experimental to find an appropriate Markov model structure and a corresponding method of parameter estimation to ensure optimality in the sense of space/time efficiency and prediction accuracy.

The reliability of this method for detecting significant sets of episodes stems from the fact that it provides: (1) an analytic formula for selecting the sliding window size; and (2) guarantees on accuracy of the threshold mechanism.

2 MINING STREAM DATA

Data mining means extracting (mining) knowledge from large amounts of data. The term data mining is actually a misnomer since the process of gold mining from sand is termed gold mining rater than sand mining. Therefore a more appropriate term would be *knowledge mining from data*. Unfortunately this term has not been widely accepted because of its length. Abstractly, the term mining can be associated with a process of discovering small set of treasures (precious information) from a great amount of raw material. In any case, the misnomer *data mining* become a popular term that depending on a context, is synonymous to the more specific terms like: knowledge mining from databases, knowledge discovery in databases, knowledge extraction, data/pattern analysis. More specifically data mining is about discovering *interesting knowledge* from large amount of data. Thus, the research in data mining involves designing intelligent methods for extracting interesting data patterns given a large amount of data from a given domain and a notion (measure) of interestingness. There are several *objective measures of interestingness* that are based on the structure and statistics of discovered patterns. In general each objective interestingness measure is associated with a threshold, which may be specified by the user. There are also *subjective measures of interestingness* that are based on user beliefs and find interesting patterns if they are unexpected or contradict user's beliefs. Measures of interestingness can be used in two different ways: (1) during the data mining step to guide the discovery process improving the search efficiency by pruning away subsets of the pattern space that do not satisfy the given interestingness constraints; or (2) after the data mining step to rank the discovered patterns with respect to their interestingness and to filter out uninteresting patterns. Thus, methods to assess pattern interestingness, and their use to improve data mining efficiency are at the heart of data mining research.

In this dissertation we are interested in mining patterns occurring as subsequences in an event stream. The notion of an occurrence as a subsequence rather than as substring (that is, contiguity is not required), is dictated by practical considerations because (for example) an interesting (anomalous) sequence of events need not be contiguous in the event stream. Furthermore systems designed to mine stream data usually involve a sliding window of observation within which the analysis is confined. This is done for two reasons: (1) the event stream is usually too long, and without a limited window approach it would involve having to save too much state; and (2) the event stream can be so long (e.g., in a continuously monitoring system) that *any* subsequence (bad or good) would likely occur within it. As an example of the need to confine the analysis to such a limited sliding window, note that three failed login attempts (with failure due to wrong password) are significant if they occur in rapid succession, but quite innocuous if they occur within a one-month interval. In this study we do not use the notion of real calendar time such as a one month interval, instead we use the number of events as a proxy for time. This is why our interval length defined by the window size is not the difference between two time stamps, but rather the size of a (contiguous) substring of the event stream. More specifically Definition 2.0.1 summarizes the most important, considered in stream data mining, types of occurrences of a pattern in an event sequence and Definition 2.0.2 summarizes corresponding problems in pattern matching.

Definition 2.0.1 *Given a stream of symbols* $S = s_1 s_2 \ldots$, *and a pattern* $e = e_1 e_2 \ldots e_m$ *of length* m, *both over an alphabet* $\mathcal{A} = \{a_1, a_2, \ldots, a_{|\mathcal{A}|}\}$,

- e *is a* **substring** *of* S *if there exists an integer* j *such that* $s_{j+i} = e_i$ *for* $1 \leq i \leq m$
- e *is a* **subsequence** *of* S *if there exist integers* $1 \leq i_1 < i_2 < \ldots < i_m$ *such that* $s_{i_1} = e_1, s_{i_2} = e_2, \ldots, s_{i_m} = e_m$
- e *is a* w**-windowed subsequence** *of* S *if* e *is a subsequence of* S *and* $i_m - i_1 < w$

- *e is a* **minimal w-windowed subsequence** *of S if e is a w-windowed subsequence of S and there does not exist any sub-window of w where e occurs as a subsequence.*

Definition 2.0.2 *Given a stream of symbols $S = s_1 s_2 \ldots$, and a pattern $e = e_1 e_2 \ldots e_m$ of length m, both over an alphabet $\mathcal{A} = \{a_1, a_2, \ldots, a_{|\mathcal{A}|}\}$,*

- *the* **pattern matching problem** *is to find whether e is a substring of S*
- *the* **plain subsequence matching problem** *is to find whether e is a subsequence of S*
- *given moreover a window of size w*
  - *the* **window-existence subsequence matching problem (WESP)** *is find whether e is a w-windowed subsequence of S*
  - *the* **window-accumulated subsequence matching problem (WASP)** *is to find the number of w-windows of S within which e is a w-windowed subsequence of S.*

In [2] the WESP and the WASP were defined. In [3] a probabilistic analysis of the plain subsequence matching problem was presented. This thesis presents the first probabilistic analysis of the WESP and the WASP.

3 NOTATION

This chapter introduces a notation that is used throughout the thesis.

We consider an infinite event sequence $S = S[1]S[2]\ldots$ over an alphabet $\mathcal{A} = \{a_1, a_2, \ldots, a_{|\mathcal{A}|}\}$ and an episode α over $\mathcal{A}$ in one of the following forms:

1. Single subsequence pattern (serial episode) $e = [e[1], e[2], \ldots, e[m]]$

2. Set of subsequence patterns $\mathcal{E} = \{e_1, e_2, \ldots, e_{|\mathcal{E}|}\}$, where $e_i = [e_i[1], e_i[2], \ldots, e_i[m_i]]$ is a serial episode for $1 \leq i \leq |\mathcal{E}|$ and by an occurrence of the set $\mathcal{E}$ we mean at least one occurrence of at least one member of $\mathcal{E}$

3. Set of all permutations of a set of symbols $e = \{e[1], e[2], \ldots, e[m]\}$ (parallel episode).

We use the superscript $\exists$ to mean "at least one occurrence as a subsequence".

Given an event sequence S and an episode α we analyze the following quantities:

- $\Omega^{\exists}(n, w)$: the observed frequency of the episode defined as the number of windows of size w, that contain an occurrence of the episode in the event sequence out of n shifts of the sliding window

- $\overline{\Omega^{\exists}}(n, w) = \frac{\Omega^{\exists}(n,w)}{n}$: the observed relative frequency of the episode defined as a fraction of windows of size w that contain an occurrence of the episode in the event sequence out of n shifts of the sliding window

- $\mathcal{W}^{\exists}(w)$: the set of all distinct windows of size w containing an occurrence of the episode

- $C^{\exists}(w)$: the number of windows of length w containing an occurrence of the episode (cardinality of $\mathcal{W}^{\exists}(w)$)

- $P^{\exists}(w)$: the probability that the episode occurs in the window of size w in the event sequence. We call this probability the *probability of existence* of the episode. Note that $\overline{\Omega^{\exists}}(n, w)$ is an estimator of $P^{\exists}(w)$.

- $\tau_{\ell}(w)$: the lower significance threshold for under-represented episodes

- $\tau_{u}(w)$: the upper significance threshold for over-represented episodes

Whenever the episode or the event sequence are not implied we specify them explicitly in the formulas and use $\overline{\Omega^{\exists}}(n, w, \alpha, S)$, $P^{\exists}(w, \alpha, S)$, etc.

Also we use $\mathcal{W}^{\exists}(w, m_1, m_2)$, $P^{\exists}(w, m_1, m_2)$ instead of $\mathcal{W}^{\exists}(w, \mathcal{E})$ and $P^{\exists}(w, \mathcal{E})$ in recursive formulas, where $\mathcal{E} = \{e_1, e_2\}$ and $m_i = |e_i|$. We also occasionally use index $m_i - k$ to mean "dropping the last k symbols of e_i", e.g., $P^{\exists}(w, m_1 - k, m_2)$ implies a pattern that is the prefix of e_1 of length $m_1 - k$ and that the second pattern is all of e_2.

4 MINING FREQUENT EPISODES

4.1 Introduction

Episode mining was formally introduced by Manila, Toivonen and Verkamo as the problem of discovering frequent episodes in event sequences [1], where an episode was defined as a partially ordered collection of events occurring as a subsequence in an event stream. In terms of pattern matching, in episode mining we are interested in finding w-windowed subsequences or minimal w-windowed subsequence. The problem of episode mining was motivated by the fact that in many applications it is of interest to discover recurrent (frequent) sets of events occurring close to each other in an event stream. That paper considered occurrences of episodes as w-windowed subsequences and minimal w-windowed subsequences in an event stream. The process of mining episodes corresponds to the Window-Accumulated Subsequence Matching Problem (WASP).

In frequent episode mining the interestingness of mined episodes is measured using the frequency of an episode defined as a fraction of windows in which the episode occurs and given a user-defined minimum frequency threshold the episode is frequent if its frequency exceeds the threshold.

More formally, the problem of discovering frequent episodes can be expressed as follows.

Given:

- $\mathcal{A} = \{a_1, a_2, \ldots, a_{|\mathcal{A}|}\}$: an alphabet
- $S = S[1]S[2]\ldots$: an infinite event sequence
- α: an episode

- w: a user defined sliding window size w
- τ_{min}: a user defined minimum frequency threshold τ_{min}, where episode α is frequent if $\overline{\Omega^{\exists}}(n, w, \alpha, S) > \tau_{min}$
- $\mathcal{E}$: a class of episodes defined as an arbitrary set of subsequence patterns (e.g. all subsequence patterns of length m over $\mathcal{A}$)

discover all frequent episodes from the given class $\mathcal{E}$ of episodes.

Once the frequent episodes are discovered, they can be used to obtain rules (association rules) that describe relationships between events in the given event sequence.

Paper [1] defined three types of episodes: a *serial episode*, a *parallel episode* and an arbitrarily complex *composite episode*.

4.2 Episodes

An episode is defined as a partially ordered collection of events occurring as a subsequence in an event stream. Episodes can be abstractly represented as directed acyclic graphs (DAG). Given an event stream S generating events from an alphabet $\mathcal{A}$ we consider the following types of episodes:

1. *Serial episode* $e = [e[1], e[2], \ldots, e[m]]$ is a sequence of events $e[1], e[2], \ldots, e[m]$. An occurrence of e means an occurrence of pattern e as a subsequence. In the graph representation a serial episode corresponds to a single path from the first event of the episode to the last one.

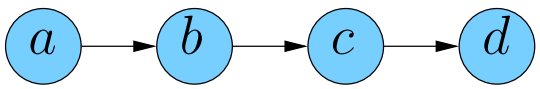

Figure 4.1.: A serial episode $e = [a, b, c, d]$ in the graph representation

2. *Parallel episode* $e = \{e[1], e[2], \ldots, e[m]\}$ is an unordered collection of events $e[1], e[2], \ldots, e[m]$. Formally, a parallel episode corresponds to the set of all

permutations of events of the episode. An occurrence of a parallel episode corresponds to a logical OR of occurrences of permutations of e as a subsequence. Alternatively we can view an occurrence of a parallel episode e as a logical AND of events $e[1], e[2], \ldots, e[m]$. In the graph representation a parallel episode corresponds to a single node containing all events of the episode. The parallel episode case captures situations where the ordering of the events within the window of observation does not matter, e.g., the events correspond to market basket items scanned by cashier.

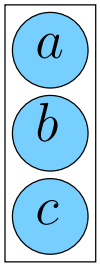

Figure 4.2.: A parallel episode $e = \{a, b, c\}$ in the graph representation

3. *Composite episode* corresponds to an arbitrary DAG (AND-graph) built from events and episodes by a serial and/or a parallel composition. A partial ordering implied by a composite episode corresponds to a set of serial episodes $\mathcal{E} = \{e_1, e_2, \ldots, e_{|\mathcal{E}|}\}$ where e_i has length m_i, for $1 \leq i \leq |\mathcal{E}|$. An occurrence of the set $\mathcal{E}$ corresponds to at least one occurrence of at least one member of the set (a logical OR of occurrences of members of $\mathcal{E}$).

4.3 Algorithm for discovering frequent episodes

Algorithm 1 from [1] computes a collection of frequent episodes $\mathcal{F}_m(S, w, \mathcal{E}, \tau_{min})$ given an event sequence S, a window size w, a class of episodes $\mathcal{E}$ (e.g. all parallel episodes o length 5) and a minimum frequency threshold τ_{min}. The idea of the algorithm is based on the idea of the *Apriori algorithm* [4] developed for finding

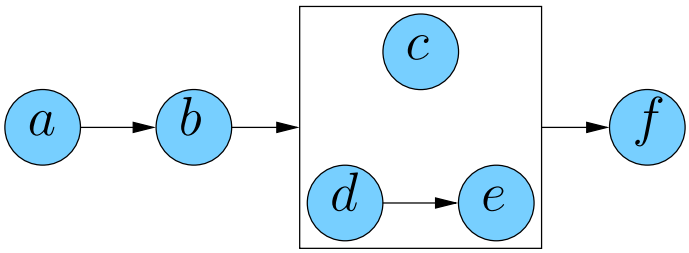

Figure 4.3.: A set of episodes $\mathcal{E} = \{[a, b, c, d, e, f], [a, b, d, c, e, f], [a, b, d, e, c, f]\}$

frequent itemsets in transaction databases. The name of the algorithm stems from the fact that it uses *prior knowledge* of frequent itemset properties. The Apriori algorithm employs an iterative approach called *level-wise* search, where it uses k-itemsets to construct $k + 1$-itemsets. First it finds 1-itemsets called L_1 and then it recursively uses L_{k-1} to find L_k, requiring one full scan of the database, until no more frequent k-itemsets can be found. The algorithm cuts the search space by using the *Apriori property* (monotone property) saying that if a set is not frequent then all its supersets cannot be frequent.

Thus, Algorithm 1 is an adaptation of the Apriori algorithm to discovering frequent episodes. Similarly to frequent itemset, frequent episodes are monotone (Apriori property) meaning that, *if an episode is frequent in an event sequence then all its subepisodes are frequent*. Let $\mathcal{C}_k$ be the set of candidate episodes of length k and let $\mathcal{F}_k$ be the set of frequent episodes of length k. The two-step level-wise search that uses the Apriori property to cut the search space is presented below.

1. *Join*: construct a set of candidate episodes C_k by joining the set of frequent episodes $\mathcal{F}_{k-1}$ with itself. Let $\mathcal{F}_{k-1}[i]$ and $\mathcal{F}_{k-1}[j]$ be two members of the set $\mathcal{F}_{k-1}$ such that their events are sorted lexicographically. Then we join two elements $\mathcal{F}_{k-1}[i] \bowtie \mathcal{F}_{k-1}[j]$ in order to create a valid candidate frequent episode of length k if they share a prefix of length $k - 2$, i.e, if $\mathcal{F}_{k-1}[i][1 : k - 2] = \mathcal{F}_{k-1}[j][1 : k - 2]$. Also in order to avoid creating duplicates we require that $\mathcal{F}_{k-1}[i][k - 1] < \mathcal{F}_{k-1}[j][k - 1]$.

2. *Prune*:

   (a) *Apriori property check*: if any $k-1$-subepisode β of a k-episode α in C_k is not in L_{k-1} then remove α from C_k.

   (b) *Database pass*: perform a scan of the event sequence S to determine frequencies of candidate episodes in C_k and remove those for which the frequency is less than the minimum threshold τ_{min}.

Algorithm 1 and its functions present a pseudocode of the algorithm for discovering frequent episodes.

Algorithm 1: Discovering frequent episode

Input: $S[1:n]$, w, $\mathcal{E}$, τ_{min}
Output: $\mathcal{F}_m(S, w, \mathcal{E}, \tau_{min})$
begin
  $\mathcal{F}_0 = \emptyset$
  for $k = 1$ ***to*** m **do**
    $\mathcal{C}_k = candidateEpisodeGenerate(\mathcal{F}_{k-1}, \tau_{min})$
    for $i = 1$ ***to*** n **do**
      foreach $c \in \mathcal{C}_k$ **do**
        if $I_i^{\exists}(w, c)$ **then**
          $\Omega^{\exists}(w, n, c) = \Omega^{\exists}(w, n, c) + 1$;
    $\mathcal{F}_k(S, w, \mathcal{E}, \tau_{min}) = \{c \in \mathcal{C}_k | \frac{\Omega^{\exists}(w,n,c)}{n} > \tau_{min}\}$
end

Function `candidateEpisodeGenerate`($\mathcal{F}_{k-1}$, τ_{min})

Input: $\mathcal{F}_{k-1}$, τ_{min}
Output: $\mathcal{C}_k$
begin
  for $i = 1$ ***to*** $|\mathcal{F}_{k-1}|$ **do**
    for $j = 1$ ***to*** $|\mathcal{F}_{k-1}|$ **do**
      if $\mathcal{F}_{k-1}[i][1 : k-2] = \mathcal{F}_{k-1}[j][1 : k-2]$ ***and***
      $\mathcal{F}_{k-1}[i][k-1] < \mathcal{F}_{k-1}[j][k-1]$ **then**
        $c = \mathcal{F}_{k-1}[i] \bowtie \mathcal{F}_{k-1}[j]$;
        if ***not*** *hasInfrequentSubset(c, $\mathcal{F}_{k-1}$)* **then**
          $\mathcal{C}_k = \mathcal{C}_k \cup c$;
  return $\mathcal{C}_k$;
end

Function `hasInfrequentSubset`(c, $\mathcal{F}_{k-1}$)

Input: c, $\mathcal{F}_{k-1}$
Output: $TRUE$ or $FALSE$
begin
  foreach $e \in (k-1)$-*subset* ***of*** c **do**
    if $e \notin \mathcal{F}_{k-1}$ **then**
      return $TRUE$;
  return $FALSE$;
end

4.4 Episode rules

The knowledge of frequent episodes can be used to find associations between events in an event stream called episode rules. Formally, *episode rules* are implications of the form

$$\beta \;\Rightarrow\; \alpha \quad [c, s, w]$$

where β is a subepisode of α, $c = confidence(\beta \Rightarrow \alpha)$ and $s = support(\beta \Rightarrow \alpha)$ defined as follows

$$\begin{aligned} support(\beta \Rightarrow \alpha) &= P(\beta \cap \alpha) \\ confidence(\beta \Rightarrow \alpha) &= P(\alpha|\beta) \\ &= \frac{P(\beta \cap \alpha)}{P(\beta)} \end{aligned}$$

Depending on the type of frequent episode e the implications are of the form

1. $e = e[1]e[2]\ldots e[k]\ldots e[m]$ is serial

$$e[1]e[2]\ldots e[k] \;\Rightarrow\; e[1]e[2]\ldots e[k]\ldots e[m]$$

2. $e = \{e[1], e[2], \ldots, e[m]\}$ is parallel

$$\beta \;\Rightarrow\; \alpha$$

where $\alpha, \beta \in e$ and $\alpha \cap \beta = \emptyset$

As an example rule, consider a frequent serial episode rule $home, people \Rightarrow faculty \quad [0.2, 00.5, 30]$ from the log file of the web server of Department of Computer Science at Purdue University. The rule says that if the *home* page is accessed followed by the *people* page then in 20% the $faculty$ page is accessed within a window of 30 seconds and this activity establishes 5% of the windows of length 30 in the log file.

Algorithm 2 presents a pseudocode of the algorithm for computing episode rules.

Algorithm 2: Computation of episode rules

Input: c, $\mathcal{F}_k(S, w, \tau_{min})$, $minimumConfidence$

Output: $\mathcal{R}$ set of episode rules

begin

  foreach $\alpha \in \mathcal{F}_k(S, w, \tau_{min})$ **do**

    foreach $\beta \in$ *subepisodes of* α **do**

      if $\frac{freq(\alpha)}{freq(\beta)} > minimumConfidence$ **then**

        $\mathcal{R} = \mathcal{R} \cup \beta \Rightarrow \alpha \quad \left[\frac{freq(\alpha)}{freq(\beta)}, frequency(\alpha), w\right]$

end

5 FINDING OCCURRENCES OF EPISODES

This chapter reviews algorithms for computing $\Omega^{\exists}(n, w)$ given an event sequence S, a window size w and an episode.

5.1 Serial episode

This section presents the standard dynamic programming algorithm for finding $\Omega^{\exists}(n, w)$ for a single subsequence pattern $e = [e[1], e[2], \ldots e[m]]$ [2]. The idea of the algorithm is based on the definition of an occurrence of a serial episode as a subsequence within a w-window as presented in Definition 2.0.1. Thus, first the algorithm recognizes an occurrence of e as a subsequence in S and then is checks whether the occurrence falls withing the w-window. The algorithm maintains an array $Q[1 : m]$ such that $Q[j]$ is the starting position of the most recent occurrence of the substring $e[1 : j]$ in S. This means that if $Q[j] = i, i \neq 0$ then there is an occurrence of $e[1 : j]$ starting at position i in S. Thus, if $i - Q[m] < m$ then the occurrence falls within the w-window. The time complexity of this algorithm is $O(mn)$ since at every position i in S, in the worst case, it needs to update all entries in Q. As an improvement of this algorithm one could use a search structure (a tree or a hash table) for finding the proper elements of Q given an input symbol $S[i]$. Also, since i is of length $\log(n)$ one could store in $Q[j]$ the distance between the current position and the starting position of the most recent occurrence of $e[1]$ giving $\log(w)$ instead of $\log(n)$. Algorithm 3 presents a pseudocode of the described algorithm.

Algorithm 3: Computation of $\Omega^{\exists}(n, w)$ for a serial episode $e[1:m]$ in an event sequence $S[1:n]$

```
Input: S[1:n], e[1:m], w
Output: Ω^∃(n, w)
begin
    for j = 1 to m do
        Q[j] = 0;
    Ω^∃(n, w) = 0;
    for i = 1 to n do
        if S[i] == e[1] then
            Q[1] = i;
        for j = 2 to m do
            if S[i] == e[j] then
                Q[j] = Q[j − 1];
        if i − Q[m] < w then
            Ω^∃(n, w) = Ω^∃(n, w) + 1;
end
```

5.2 Parallel episode

This section presents our algorithm for finding $\Omega^{\exists}(n, w)$ for a parallel episode $e = \{e[1], e[2] \dots e[m]\}$ [5]. The main advantage of the algorithm presented in this section with comparison to the algorithm in [1] is that our algorithm: (1) can compute occurrences for all window sizes in one pass; (2) computes the minimal occurrences; (3) computes occurrences when instead of a window of size w we associate a time to live *ttl* with each symbol of the pattern e; and (4) the space required by the data structure is independent of the window length w.

Let $e' = \{e'[1], e'[2], \dots e'[m']\}$ be a set of cardinality m' obtained from e by eliminating duplicates and then sorting it. Let c_i for $i = 1, 2 \dots m'$ be the number of

times an alphabet symbol $e'[i]$ occurs in e. We build a binary tree over symbols in e' as leaves. Figure 5.1 shows the tree. Each node in the tree contains the usual tree

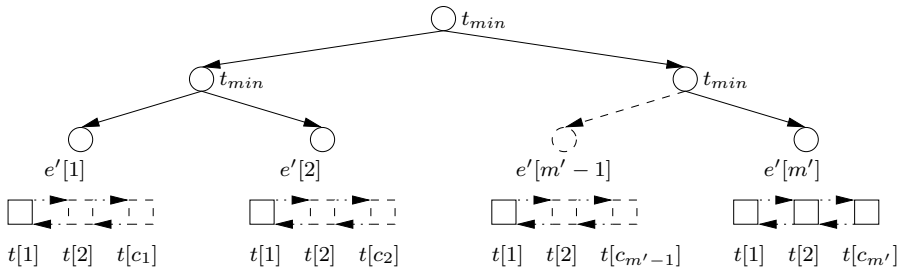

Figure 5.1.: Data structure for finding occurrences of a parallel episode e

pointers: *parent*, *lchild*, *rchild* and a search key interval $[s_{min}, s_{max}]$ where s_{min} is the smallest key in the subtree and s_{max} is the largest key in the subtree. In addition, depending on its type a node keeps the following specific information:

- root: t an event counter, counts the number of elapsed (scanned) events, where by "time" we mean the position in the event sequence
- internal node:
  - t_{min} the minimum time of the arrival for the subtree rooted at this node
- leaf node:
  - search key value s_i
  - *dlist*: doubly linked list containing one element for each symbol $e'[i]$ in the pattern e. The purpose of the list is to keep track of the most recent occurrences of the symbol $e'[i]$ sorted by arrival times. Let $t[i][1], t[i][2] \ldots t[i][c_i]$ for $i = 1, 2 \ldots m'$ be the times of the occurrence of symbol $e'[i]$ from the left to the right in the list then they must satisfy the condition $t[i][j] < t[i][j+1]$ for $j = 1, \ldots c_i - 1$. So the leftmost element of the list contains the oldest occurrence of $e'[i]$ and the rightmost element contains the most recent occurrence of $e'[i]$.

The tree supports the following operations:

- **update(s)**: when a new symbol s arrives the time t is incremented by one and the search tree structure is used to find the proper leaf. If the search finds leaf $e'[i]$ then the leftmost element of the doubly linked list with time $t[i][1]$ is removed and a new element with the current time t is attached to the right end of the list. Once the new element is attached to the list the time of the leftmost element, as the oldest one, is propagated up the tree as long as it is smaller then t_{min} of the internal node on the path to the root. This operation takes $\log(m)$ time (the height of the tree).

- **exists**: if $t - t_{min} + 1 \leq w$ at the root node then at least one permutation of e occurs as a subsequence within the window. This operation takes $O(1)$.

Algorithm 4 presents a pseudocode for finding $\Omega^{\exists}(n, w)$ using the presented tree data structure.

Algorithm 4: Tree based algorithm for finding occurrences of a parallel episode

```
input  : S[1 : n], e[1 : m], w
output: Ω∃(n, w)
begin
    Ω∃(n, w) = 0;
    tree.build(S);
    for i = 1 to n do
        tree.update(T[i]);
        if tree.exists then
            Ω∃(n, w) = Ω∃(n, w) + 1;
end
```

The time complexity for finding $\Omega^{\exists}(n, w)$ is $O(n \log m)$ because we perform n calls to **update(s)**, each requiring $O(\log m)$. The space is determined by the number of nodes in the tree, which is $O(m)$. The presented tree structure can also handle a

problem when instead of a window of size w we associate a time to live ttl with each symbol of the pattern e. In such a case each leaf stores the expiration time $expt = t+ttl$ and the each internal node stores the minimum expiration time $expt_{min}$ in its subtree. Operation **update(s)** remains the same and **exists** returns $true$ if $expt_{min} > t$ at the root node. The presented data structure can be extended to handle multiple parallel episodes (details are omitted).

5.3 Set of serial episode

This section presents an algorithm for finding $\Omega^{\exists}(n, w)$ for a set of serial episodes $\mathcal{E} = \{e_1, e_2, \ldots, e_{|\mathcal{E}|}\}$ [6]. This algorithm is a generalization of Algorithm 3 to an arbitrary set of serial episodes. We could use $|\mathcal{E}|$ instances of the Algorithm 3 independently but it would be inefficient since some of the members of $\mathcal{E}$ could potentially share the same prefix. Therefore we build a trie T from members of $\mathcal{E}$ and combine the computation of the nodes that depend on the common prefixes. The result is a significant saving in computational cost if there are many common prefixes. Algorithm 5 presents a pseudocode of the presented algorithm, where

- $T.root$ is the root of the trie
- for each $node \in T$ we have
  - $node.parent$ is the parent
  - $node.child$ is the list of children
  - $node.v$ is the starting position in S of the most recent occurrence of the substring ending at $node$
  - $node.s$ is the symbol associated with the node
- $waits(c)$ is a list of nodes waiting for symbol c
- $leaf$ is the list of leaves in lexicographic order.

It is easy to see that Algorithm 5 reduces to Algorithm 3 if the trie T consists of a single path.

Algorithm 5: Computation of $\Omega^{\exists}(n, w)$ for a set of serial episodes represented as a trie $T = (V, E)$ in an event sequence $S[1 : n]$

```
Input: S[1 : n], T = (V, E), w
Output: Ω^∃(n, w)
begin
    foreach node ∈ T do
        node.v = 0;
    Ω^∃(n, w) = 0;
    for i = 1 to n do
        for j = 1 to |T.root.child| do
            if T.root.child[j].s = S[i] then
                T.root.child[j].v = i;
        for j = 1 to |waits(S[i])| do
            waits(S[i])[j].v = waits(S[i])[j].parent.v;
        for j = 1 to |T.leaf[j]| do
            if i − T.leaf[j].v < w then
                Ω^∃(n, w) = Ω^∃(n, w) + 1;
                break;
end
```

6 MINING SIGNIFICANT EPISODES

Given a class of episodes, a window of size w, an event sequence S and a minimum frequency threshold, the problem of discovering frequent episodes was defined as finding all episodes from the class, whose frequency is above the minimum frequency threshold [1]. A class of episodes may be specified as an arbitrary set of subsequence patterns. In many applications it is of interest to define a class of episodes as a set of disjoint sets of subsequence patterns, and the task is then to find all sets of subsequence patterns that exceed the minimum frequency threshold. As an example of such an application consider approximate subsequence matching where an approximate occurrence of a subsequence pattern means an occurrence of a member of a set of subsequence patterns that are similar in an approximate sense to the subsequence pattern. Then the problem is to find frequent subsequence patterns in terms of approximate occurrences.

Episode mining was used in many applications including: intrusion detection [7–10], alarm correlation systems in telecommunication networks [11], web usage analysis and computational biology [12]. However, there were reports that episode mining tends to produce a large number of meaningless or redundant patterns (see, e.g., [8]).

Therefore, in this thesis we address the fundamental problem of identification of significant sets of episodes. The solution to this problem involves a solution to the following two problems: (1) selecting the sliding window size to guarantee that an occurrence of a set of subsequence patterns is meaningful in the sense that the set does not occur almost surely in every window in an event sequence; and (2) identification of significant sets of episodes with respect to a reference sequence. Thus, in order to identify a significant episode one has to make sure it is meaningful. Clearly every significant episode is meaningful, but the reverse is not true. The

reason that episode mining may produce a large number of redundant patterns is the fact that depending on the probabilistic characteristic of the event stream a serial episode and many of its permutations may have the same probability of existence.

The problem of selecting the sliding window size to guarantee that an occurrence of a set of episodes is meaningful goes beyond episode mining, and concerns all aspects of information extraction using the sliding window approach that is extensively used in stream query processing and security applications. The problem stems from the fact that for an appropriately large window size any subsequence pattern will almost surely occur in every window in an event stream because the probability of its existence (at least one occurrence) as a subsequence in a window of a given size will be close to one. Consequences of too large a window size may include a needlessly high computational cost of executing a continuous multi stream window join query that satisfies a join condition for every shift of the window [13] and the generation of many false alarms in a monitoring system. We solve the problem of selecting the sliding window size by providing an analytical formula for the probability of existence for an arbitrary set of subsequence patterns that uses a variable-length probabilistic Markov model of the event stream.

Recall that a k-order variable-length Markov model is a Markov model where the contexts (memory) are allowed to be of variable length. Such reduced models are also called *tree models* since they can be represented by a context tree that can range from a full tree for an ordinary k-order Markov model to an empty tree for 0-order Markov model (memoryless model). The attractiveness of tree models stems from the fact that they provide a sparse representation of a sequence by reducing the number of parameters to be estimated since for real-life data the actual memory length varies. Furthermore they can be efficiently learned while monitoring the sequence. The variable-length Markov models are in details discussed in Chapter 9.

The problem of identification of significant sets of episodes is motivated by applications where there is a notion of a normal (reference) event stream, in which case it is of interest to discover significant deviations from the normal behavior. Accord-

ing to this notion an episode is significant if the probability that it would occur by chance a specific number of times, in the reference event stream, is very small. Note that the frequency of occurrence alone is not enough to determine significance, i.e., an infrequent episode can be more significance than a frequent one and the significance depends on the following factors: (1) the structure of the episode; and (2) the probabilistic characteristics of the reference and the monitored event streams. In general, while monitoring an event stream there are three phenomena that could make an anomaly with respect to the reference sequence:

1. *Foreign symbol*: a symbol not found in the reference event stream
2. *Under-represented episode*: an episode that is too infrequent in the monitored event stream
3. *Over-represented episode*: an episode that is too frequent in the monitored event stream

Detection of an anomaly caused by an occurrence of a foreign symbol is straightforward. More difficult is the problem of discovering over-represented and under-represented episodes, which is the topic of our work. We solve the problem of identification of significant sets of episodes by providing analytical formulas for significance thresholds for under-represented and over-represented episodes, using a variable-length probabilistic Markov model as a model of the reference stream. The advantage of a probabilistic Markov model of the reference sequence, over profiles of normal behavior in a form of frequent episodes [7], is that the Markov model allows answering queries referring to arbitrary episodes in the reference sequence, and also provides a sparse summarization of the reference sequence. The ability to compute the frequency of an arbitrary episode in the reference sequence is particularly needed in episode mining where we discover a collection of frequent episodes and can compute their significance "on the fly". Furthermore such a probabilistic model can be conveniently built "by hand" using expert knowledge. Thus, given a collection of discovered frequent episodes, from a given class and using a given window size, we

identify significant episodes in a two step process where we first filter out meaningless episodes by computing their probability of existence, and then we identify significant episodes using the thresholds.

This process efficiently removes all irrelevant subsequence patterns that may be discovered in the standard frequent episode mining algorithm as reported in [8]. Applications of the significance thresholds include intrusion detection and computational biology. An over-represented episode in a particular segment of DNA with respect to a model may suggest a biological function and an under-represented episode may suggest a vulnerability to certain diseases.

The significance thresholds for episodes can be viewed as an analogy to the χ^2 significance test for market basket item sets [14], where given an itemset and a confidence level α, the test computes the χ^2 statistic to assess a deviation between actual frequencies of subitemsets of the itemset and their expected values assuming independence of the items. Thus, the reference model was a memoryless model, and if $\chi^2 \geq \chi^2_\alpha$ the itemset was significant, meaning the items were dependent.

7 IDENTIFICATION OF SIGNIFICANT SETS OF EPISODES

This chapter presents the definitions and solutions without derivations that are given in Chapters 8 and 10.

7.1 Problem definition

The problem of identification of significant sets of episodes in an even sequence S can be stated as follows. Given:

- a variable-length Markov model M with parameter set $\Theta(M)$ of a reference event sequence
- $\overline{\Omega^{\exists}}(n, w)$: observed frequency of a set of episodes
- $\beta(b)$: a target (user defined) significance level (e.g., $\beta(b) = 10^{-5}$),

is the set of episodes significant with respect to the reference sequences. In other words the question is what is the upper threshold $\tau_u(w)$ and the lower threshold $\tau_\ell(w)$ such that if $\overline{\Omega^{\exists}}(n, w) \notin [\tau_\ell(w), \tau_u(w)]$ then the episode is significant at the level $\beta(b)$, i.e., $P(\overline{\Omega^{\exists}}(n, w) > \tau_u(w)) \leq \beta(b)$ and $P(\overline{\Omega^{\exists}}(n, w) < \tau_\ell(w)) \leq \beta(b)$?

7.2 Solution

In chapter 8 we prove that if an event sequence is generated by a Markov chain X that satisfies the following conditions:

1. X has finite state space; and
2. X is irreducible and aperiodic;

then $\overline{\Omega^{\exists}}(n, w)$ satisfies the central limit theorem in that sequence leading to the following formulas for the thresholds:

$$\begin{cases} \tau_u(w) &= P^{\exists}(w) + b\sqrt{P^{\exists}(w)(1 - P^{\exists}(w))} \\ \tau_\ell(w) &= P^{\exists}(w) - b\sqrt{P^{\exists}(w)(1 - P^{\exists}(w))} \\ \beta(b) &= \frac{1}{\sqrt{2\pi}} \int_b^{\infty} e^{\frac{-t^2}{2}} dt \end{cases} \tag{7.1}$$

The graphical representation of the thresholds is presented in Figure 7.1.

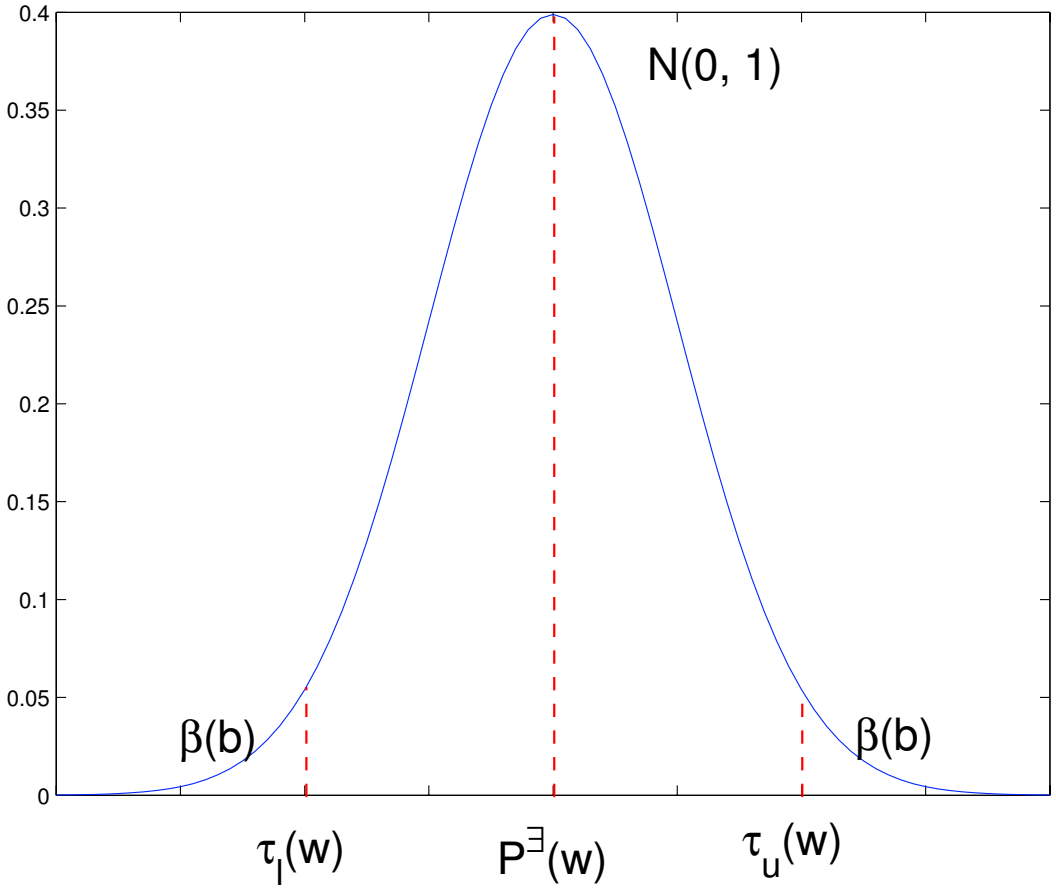

Figure 7.1.: The upper and lower threshold

The difficulty of efficiently computing $P^{\exists}(w)$ stems from the fact that the structure of the tree model (variable-length Markov model) of the stream can range from a full tree, through a sparse tree, to the empty tree (see Chapter 9 for a review on the variable-length Markov models).

7.3 Classification of episodes

Given an episode with observed frequency $\overline{\Omega^{\exists}}(n, w)$ and a significance level $\beta(b)$, the episode can be classified using the upper threshold $\tau_u(w)$ and the lower threshold $\tau_\ell(w)$ as follows:

- *meaningless*: if $\overline{\Omega^{\exists}}(n, w) \approx 1$ and $P^{\exists}(w) \approx 1$, i.e., the window size w is too large
- *significant*:
  - if $\overline{\Omega^{\exists}}(n, w) > \tau_u(w)$ for over-represented episodes, i.e, $P(\overline{\Omega^{\exists}}(n, w) > \tau_u(w)) \leq \beta(b)$
  - if $\overline{\Omega^{\exists}}(n, w) < \tau_\ell(w)$ for under-represented episodes, i.e, $P(\overline{\Omega^{\exists}}(n, w) < \tau_\ell(w)) \leq \beta(b)$
- *normal*: if $\overline{\Omega^{\exists}}(n, w) \in [\tau_\ell(w), \tau_u(w)]$

7.4 Episode ranking with respect to significance

Given a collection of episodes $\mathcal{C}$ (e.g., frequent episodes) discovered in S, we rank the episodes with respect to significance by finding a maximum $b(e)$ (called $b(e)_{max}$) for every episode $e \in \mathcal{C}$ such that for that episode:

- $\overline{\Omega^{\exists}}(n, w) \geq \tau_u(w)$, where $\beta(b(e)_{max}) = P(\overline{\Omega^{\exists}}(n, w) \geq \tau_u(w))$ for over-represented episodes
- $\overline{\Omega^{\exists}}(n, w) \leq \tau_\ell(w)$, where $\beta(b(e)_{max}) = P(\overline{\Omega^{\exists}}(n, w) \leq \tau_u(w))$ for under-represented episodes

and then we sort the episodes according to b_{max}.

7.5 Building a model of the reference sequence

Depending on whether a model M and parameters $\Theta(M)$ are given prior to monitoring the event sequence S, we can distinguish the following cases according to their consequences for our method:

1. M and $\Theta(M)$ are given in which case the thresholds detect a significant deviation from both M and $\Theta(M)$

2. M is given but $\Theta(M)$ is estimated while monitoring S in which case the thresholds detect a significant deviation from M

3. M is not given and we can build M and $\Theta(M)$ while monitoring S in which case the thresholds detect outliers with respect to M and $\Theta(M)$.

7.6 Selecting the window size to guarantee that an occurrence of an episode is meaningful

Given an event sequence S that is generated by a Markov chain that has finite state and is irreducible and aperiodic, an occurrence of an episode in a window of size w in that sequence is meaningful if $P^{\exists}(w) < 1$. Furthermore for every episode in that sequence there exists such a window size w_{sat} for which $P^{\exists}(w_{sat}) \approx 1$ and $P^{\exists}(w_{sat}) \approx 1$ for every $w > w_{sat}$, where w_{sat} depends on the probabilistic characteristic of S and on the structure of the episode. Figure 7.2 shows the shape of a graph of $P^{\exists}(w)$ for a single subsequence pattern, and the corresponding standard deviation $\sqrt{P^{\exists}(w)(1 - P^{\exists}(w))}$. The shape of the graphs for $P^{\exists}(w)$ and $\sqrt{P^{\exists}(w)(1 - P^{\exists}(w))}$ for an arbitrary episode is similar because, in general, $P^{\exists}(w) \approx 1 - \Theta(\rho^w)$ for $0 < \rho < 1$ [15] and the difference is only in the slope of $P^{\exists}(w)$.

Thus, given a variable-length Markov model M with parameters $\Theta(M)$ of an event sequence S, we can find an appropriate w using a formula for $P^{\exists}(w)$. Depending on whether the model and an episode are given prior to monitoring S we can distinguish the following cases

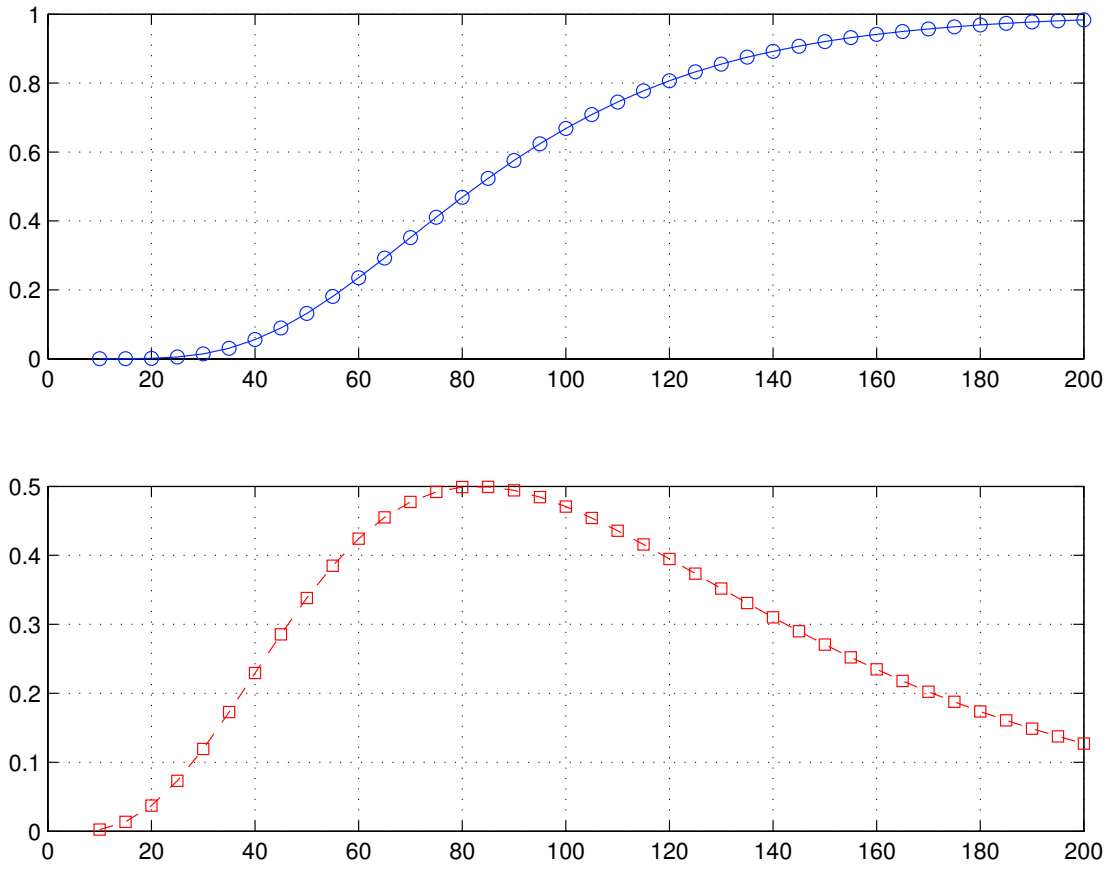

Figure 7.2.: $P^{\exists}(w)$ and $\sqrt{P^{\exists}(w)(1-P^{\exists}(w))}$ for a single subsequence pattern

- *Designing the sliding window size*: given a priori knowledge of episodes of interest and a model M with parameters $\Theta(M)$ of S, we can select an appropriate w prior to monitoring S
- *Validation of the sliding window size*: given a proposed w, an episode (e.g., a frequent episode) discovered in S and a model M with parameters $\Theta(M)$ learned while monitoring S, we can validate w for the discovered episode

A proper approach to designing the sliding window size that is dedicated for the upper or the lower threshold, depends on the type of threshold. Given a significance

level $\beta(b)$, the window size w should be such that $\tau_u(w) < 1$ for over-represented episodes and $\tau_\ell(w) > 0$ for under-represented episodes.

8 ANALYSIS OF THE SIGNIFICANCE THRESHOLDS

In this chapter we show that $\overline{\Omega^{\exists}}(n, w)$ satisfies the central limit theorem (CLT) under certain conditions in Markov sources.

We consider an event sequence $S = S[1], S[2] \ldots$, that is generated by a k-order Markov source over an alphabet $\mathcal{A}$.

Definition 8.0.1 *A k-order Markov source is a sequence of random variables $Z_1, Z_2, \ldots$ over a finite alphabet $\mathcal{A}$ of cardinality $|\mathcal{A}|$ if there exists a minimum integer k such that*

$$P(Z_1 = z_1, \ldots Z_n = z_n) =$$
$$P(Z_1 = z_1, \ldots Z_k = z_k) \prod_{i=k+1}^{n} P(Z_i = z_i | Z_{i-1} = z_{i-1}, \ldots, Z_{i-k} = z_{i-k})$$

for all $n \geq k$ and every sequence $z_1, \ldots, z_n \in \mathcal{A}$.

A k-order Markov source can be equivalently represented by a k-order finite state Markov chain.

Definition 8.0.2 *A finite state k-order stationary Markov chain is a sequence of random variables $X_1, X_2, \ldots$, over a state alphabet $\mathcal{Q}$ of cardinality $\mathcal{A}^k$, where*

$$P(X_{n+1} = j | X_1 = i_1, \ldots, X_n = i_n) = P(X_{n+1} = j | X_n = i_n) = p_{i_n, j}$$

for every n and every sequence $i_1, \ldots i_n \in \mathcal{Q}$. The $p_{i_n,j}$ is a transition probability from state i_n to state j, where $i_n, j \in \mathcal{Q}$.

Note, that the definition states that: (1) the probability of a state depends on only the previous state; (2) the transition probabilities do not vary with n.

Now we review some fundamental properties of Markov chains that are essential in the analysis of $\overline{\Omega^{\exists}}(n, w)$. Let

$$p_{i,j}^{n} = P(X_{m+n} = j | X_m = i) = \sum_{k_1 \ldots k_{n-1}} p_{ik_1} p_{k_1 k_2}, \ldots, p_{k_{n-1} j}$$

be the *n-the order transition probability*. A Markov chain is *irreducible* if for all i, j $p^n_{i,j} > 0$ for some n [16]. The period of a state i is the greatest common divisor of the set of natural numbers n for which $p^n_{i,i} > 0$. A Markov chain is *aperiodic* if the period is equal to 1 for all states [16].

Throughout the rest of this chapter we assume that the event sequence S is represented with a k-order Markov chain.

Observe that $\Omega^{\exists}(n, w) = \sum_{i=1}^{n} I_i^{\exists}(w)$ where

$$I_i^{\exists}(w) = \begin{cases} 1 & \text{the episode occurs in the } i\text{-th window} \\ 0 & \text{otherwise} \end{cases}$$

The independence of the elements of the sequence of $I_1^{\exists}(w), I_2^{\exists}(w), \ldots$ is violated in two different ways:

1. observation windows overlap within $w-1$ events meaning
   $|P(I_{i+k}^{\exists}(w) = 1 | I_i^{\exists}(w) = 1) - P(I_{i+k}^{\exists}(w) = 1)| \neq 0$ for $1 \leq k \leq w-1$ (see Figure 8.1)

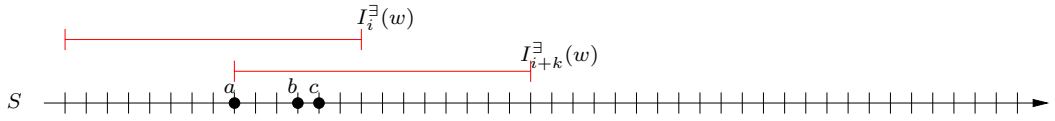

Figure 8.1.: Dependency due to window overlap

2. the event sequence S is not memoryless meaning
   $|P(I_{i+k}^{\exists}(w) = 1 | I_i^{\exists}(w) = 1) - P(I_{i+k}^{\exists}(w) = 1)| \neq 0$ for $k > w-1$ (see Figure 8.2).

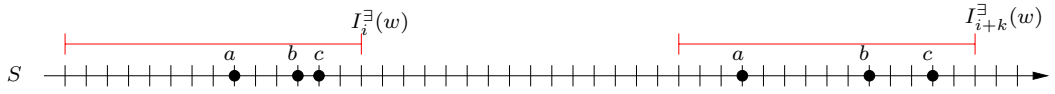

Figure 8.2.: Dependency due to memory of the event sequence

Because of the dependence of the elements of the sequence $I_1^{\exists}(w), I_2^{\exists}(w), \ldots$, the standard version of the CLT theorem does not apply to $\overline{\Omega^{\exists}}(n, w)$. However, the CLT also holds for certain weakly dependent sequences (mixing sequences), where a dependence between the current event and a future event in the sequence decreases monotonically and at a certain rate with the distance between the events. One of the examples of such a sequence is a $\phi(n)$-mixing sequence of random variables, where the dependence between two random variables A and B in the sequence is measured in terms of $|P(B|A) - P(B)|$. Other measures of dependence are discussed in [17].

Definition 8.0.3 *Let $\phi(1), \phi(2), \ldots$ be a sequence of numbers such that as $n \rightarrow \infty$, $\phi(n) \rightarrow 0$. A stationary sequence of random variables $Y_1, Y_2, \ldots$ is ϕ-mixing if $|P(E_2|E_1) - P(E_2)| \leq \phi(n)$ for every j, k such that $E_1 = f(Y_1, \ldots, Y_j)$ and $E_2 = f(Y_{j+n}, \ldots, Y_{j+n+k})$.*

In this definition, E_1 is an event that depends only on $Y_1, \ldots, Y_j$, and E_2 is an event that depends only on $Y_{j+n}, \ldots, Y_{j+n+k}$. The condition requires that E_1 and E_2 are almost independent in the sense that $|P(E_2|E_1) - P(E_2)|$ is small for large n. Thus, the ϕ-mixing sequence captures the dependence in Figure 8.1 and Figure 8.2. Notice that, according to the definition of a ϕ-mixing sequence, the $w-1$-dependent sequence $I_1^{\exists}(w), I_2^{\exists}(w), I_3^{\exists}(w), \ldots$ is ϕ-mixing with $\phi_n = 0$ for $|i - j| > w - 1$.

The CLT for a ϕ-mixing sequence is as follows.

Theorem 8.0.1 *(Ibragimov and Linnik, 1971) [18] Let $Y_1, Y_2, \ldots$ be a strictly stationary ϕ-mixing sequence with $\mathbf{E}[Y_i^2] < \infty$ and let $\overline{Y}(n) = \frac{1}{n}\sum_{i=1}^{n} Y_i$. If*

$$\sum_n \sqrt{\phi(n)} \;<\; \infty \tag{8.1}$$

then

$$\mathbf{Var}[Y_i] \;=\; \mathbf{E}[Y_1^2] - \mathbf{E}[Y_1]^2 + 2\sum_{k=1}^{\infty} \mathbf{E}[Y_1 Y_{1+k}] \tag{8.2}$$

converges absolutely and if $\mathbf{Var}[Y_i] > 0$ then as $n \rightarrow \infty$

$$\frac{\sqrt{n}(\overline{Y}(n) - \mathbf{E}[Y_i])}{\sqrt{\mathbf{Var}[Y_i]}} \xrightarrow{d} N(0, 1) \tag{8.3}$$

Now we will prove that $I_1^{\exists}(w), I_2^{\exists}(w), \ldots$ satisfies the conditions of Theorem 8.0.1. To accomplish it we will show the following: (1) the necessary condition for a Markov chain to be ϕ-mixing; and (2) that $I_1^{\exists}(w), I_2^{\exists}(w), \ldots$, as a function of a ϕ-mixing Markov chain, is ϕ-mixing and also satisfies Theorem 8.0.1.

The following theorem quantifies the dependence of elements in a k-order Markov chain and presents conditions under which the chain is ϕ-mixing.

Theorem 8.0.2 *[16] Let $X_1, X_2, \ldots$ be a Markov chain and let $p_{ij}^n = P(X_{m+n} = j | X_m = i)$ be the n-th order transition probability. If the state space is finite and the chain is irreducible and aperiodic then there is a stationary distribution π, and*

$$|p_{ij}^n - \pi_j| \quad \leq \quad A\rho^n, \tag{8.4}$$

where $A \geq 0$ and $0 \leq \rho < 1$.

Thus, theorem 8.0.2 implies that an irreducible, aperiodic and finite Markov chain is ϕ-mixing with the exponential rate of convergence ($\phi(n) = A\rho^n$) that is even stronger than necessary for Theorem 8.0.1 to hold. Markov chains satisfying Theorem 8.0.2 are called *uniformly ergodic*.

Clearly, the sequence $I_1^{\exists}(w), I_2^{\exists}(w), \ldots$ is a function of the underlying k -order Markov chain $X_1, X_2, \ldots$ representing the stream S, where $I_i^{\exists}(w) = f(X_i)$. Note however that a function of a Markov chain is not necessarily Markovian.

The following corollary follows form Theorems 8.0.1 and 8.0.2 since for stationary, irreducible, aperiodic and finite state Markov chains $X_1, X_2, \ldots$ the $\phi(n)$ goes to zero at an exponential rate. Also the corollary describes necessary condition on a function of a Markov chain $Y_i = f(X_i)$ under which the CLT holds for $\overline{Y}(n)$.

Corrolary 1 *(Ibragimov and Linnik, 1971) Let $X_1, X_2, \ldots$ be an irreducible, aperiodic and finite Markov chain with stationary distribution π and let $Y_i = f(X_i)$ and let $\overline{Y}(n) = \frac{1}{n}\sum_{i=1}^n f(X_i)$ where $f : X \to R$ is a real-valued function. If $\mathbf{E}[Y_i^2] < \infty$ then for any initial distribution, as $n \to \infty$*

$$\frac{\sqrt{n}(\overline{Y}(n) - \mathbf{E}[Y_i])}{\sqrt{\mathbf{Var}[Y_i]}} \xrightarrow{d} N(0,1) \tag{8.5}$$

From the above we obtain the following theorem.

Theorem 8.0.3 *Let S be an event sequence that is generated by a Markov chain that has a finite state space and is irreducible and aperiodic then*

$$\lim_{n\to\infty} P\left(a \leq \frac{\sqrt{n}\left(\overline{\Omega^{\exists}}(n,w) - \mathbf{E}\left[I_i^{\exists}(w)\right]\right)}{\sqrt{\mathbf{Var}[I_i^{\exists}(w)]}} \leq b\right) = \frac{1}{\sqrt{2\pi}}\int_a^b e^{\frac{-t^2}{2}}dt$$

where $\mathbf{E}[I_i^{\exists}(w)] = P^{\exists}(w)$ *and* $\mathbf{Var}[I_i^{\exists}(w)] = \mathbf{Var}[I_1^{\exists}(w)] + 2\sum_{k=1}^{\infty}\mathbf{Cov}[I_1^{\exists}(w), I_{1+k}^{\exists}(w)]$.

Proof. Easily follows from Corollary 1.

Theorem 8.0.3 leads to (7.1) for the thresholds.

Now we will derive a computational formula for $\mathbf{Var}[I_i^{\exists}(w)]$. Let $P(I_1^{\exists}(w) \cap I_{1+k}^{\exists}(w))$ be the probability that $I_1^{\exists}(w) = 1$ and $I_{1+k}^{\exists}(w) = 1$ and let $P(I_1^{\exists}(w))$ be the probability that $I_1^{\exists}(w) = 1$, where clearly $P(I_1^{\exists}(w)) = P^{\exists}(w)$ because of the stationarity of the chain. Then $\mathbf{Var}[I_i^{\exists}(w)]$ can be expressed as follows

$$\begin{aligned}
\mathbf{Var}[I_i^{\exists}(w)] &= \mathbf{Var}[I_1^{\exists}(w)] + 2\sum_{k=1}^{\infty}\mathbf{Cov}[I_1^{\exists}(w), I_{1+k}^{\exists}(w)] \\
&= \mathbf{E}[I_1^{\exists}(w)^2] - (\mathbf{E}[I_1^{\exists}(w)])^2 + 2\sum_{k=1}^{\infty}(\mathbf{E}[I_1^{\exists}(w)I_{1+k}^{\exists}(w)] - (\mathbf{E}[I_1^{\exists}(w)])^2) \\
&= P^{\exists}(w) - (P^{\exists}(w))^2 + 2\sum_{k=1}^{\infty}\left(P(I_1^{\exists}(w) \cap I_{1+k}^{\exists}(w)) - (P^{\exists}(w))^2\right) \\
&= P^{\exists}(w)(1 - P^{\exists}(w)) + 2\sum_{k=1}^{\infty}\left(P(I_1^{\exists}(w))P(I_{1+k}^{\exists}(w)|I_1^{\exists}(w)) - (P^{\exists}(w))^2\right) \\
&= P^{\exists}(w)(1 - P^{\exists}(w)) + 2P^{\exists}(w)\sum_{k=1}^{\infty}\left(P(I_{1+k}^{\exists}(w)|I_1^{\exists}(w)) - P^{\exists}(w)\right)
\end{aligned}$$

From Theorem 8.0.2 it follows that $\lim_{k\to\infty}\left(P(I_{1+k}^{\exists}(w)|I_1^{\exists}(w)) - P^{\exists}(w)\right) = 0$ leading to the following approximation of $\mathbf{Var}[I_i^{\exists}(w)]$

$$\mathbf{Var}[I_i^{\exists}(w)] \approx P^{\exists}(w)(1 - P^{\exists}(w)). \tag{8.6}$$

The following example illustrates an application of the concepts presented in this section.

Example. Consider an event sequence S over an alphabet $\mathcal{A} = \{0, 1\}$. Assume that S is represented with an irreducible and aperiodic 2-order Markov chain $X_1, X_2, X_3, \ldots$ having a state space $\mathcal{Q} = \{00, 01, 10, 11\}$ and a stationary distribution vector $\pi = [\pi_1, \pi_2, \pi_3, \pi_4]$. Also consider an episode $e = [0, 1]$. Then clearly $I_i^{\exists}(w) = f(X_i)$, where $I_i^{\exists}(w) = 1$ if $X_i = 01$ and $I_i^{\exists}(w) = 0$ otherwise. Therefore, $\overline{\Omega^{\exists}}(n, w)$ satisfies Theorem 8.0.3 with $\mathbf{E}[I_i^{\exists}(w)] = \pi_2$ and $\mathbf{Var}[I_i^{\exists}(w)] = \pi_2(1 - \pi_2) + 2\pi_2 \sum_{k=1}^{\infty}(p_{2,2}^{(k)} - \pi_2)$. Since from Theorem 8.0.2 we know that $(p_{2,2}^{(k)} - \pi_2)$ goes to zero exponentially fast we obtain $\mathbf{Var}[I_i^{\exists}(w)] \approx \pi_2(1 - \pi_2)$.

9 VARIABLE-LENGTH MARKOV MODEL

In this chapter we review variable-length Markov models.

Given an alphabet $\mathcal{A} = \{a_1, a_2, \ldots, a_{|\mathcal{A}|}\}$ a k-order variable-length Markov model is characterized by a set of conditional probabilities of generating a symbol $x \in \mathcal{A}$ for every context $c \in \mathcal{C}$, where $\mathcal{C} \subset \bigcup_{k=1} \mathcal{A}^k$ is a set of variable-length contexts. The set of contexts $\mathcal{C}$ is called the *model* and the corresponding set of conditional probabilities $\Theta(\mathcal{C}) = \{\theta_{1,1}, \theta_{1,2}, \ldots \theta_{|\mathcal{A}|,|\mathcal{C}|}\}$ is called the *set of parameters*, where $\theta_{i,j} = P(a_i|c_j)$. A conditional probability $P(a_i|c)$ can be easily estimated using the maximum likelihood (ML) estimate $\hat{P}(a_i|c) = \frac{N(ca_i)}{\sum_{i=1}^{|\mathcal{A}|} N(ca_i)}$, where $N(s)$ is the observed number of occurrences of a string s in the stream of length n. An unconditional probability distribution in a memoryless model is estimated using $\hat{P}(a_i) = \frac{N(a_i)}{n}$.

In an ordinary k-order full Markov model the there are $|\mathcal{C}| = |A|^k$ contexts and $|\Theta(\mathcal{C})| = |A|^{k+1}$ parameters to be estimated. However, such a full Markov model can be inefficient since for real-life data the actual memory length varies. Thus, the number of parameters can be significantly reduced by allowing contexts (memory) to be of variable length. Such reduced models, first considered in [19] were termed the *variable-length Markov chains/models* or *tree models* [20] since they can be conveniently represented with a context tree structure. Clearly, a memoryless model corresponds to the empty tree and a full model to a full tree. Thus, the advantage of variable-length Markov models over full models is that they efficiently capture the redundancies that are typical for real-life data.. In terms of Markov chains, $\mathcal{C}$ is the set of states and $\Theta(\mathcal{C})$ is the set of transition probabilities in a variable-length Markov chain.

9.1 Context algorithm

In [19] an algorithm *context* was presented for estimating a minimal context tree of a variable-length Markov model in time $O(n\log(k))$, where k is a given depth of the tree. Given a sequence of length n the algorithm builds an optimal context tree in two stages. In the first stage the algorithm grows a large context tree based on observed contexts and recorded occurrences of each symbol in every context. In the second stage the algorithm prunes the tree, by examining every terminal node and pruning it to its parent node if the probability distribution for the two nodes does not differ significantly according to a pruning criterion. As a pruning criterion, in [19,21] the minimization of the stochastic complexity and in [22–24] the *Kullback-Leibler distance* was proposed respectively. Let wu be a, time reversed, terminal node in a context tree, where u is the oldest symbol of the context. Then the Kullback-Leibler distance can be expressed as follows $\Delta_{wu} = \sum_{a\in\mathcal{A}} \hat{P}(a|wu)\log\left(\frac{\hat{P}(a|wu)}{\hat{P}(a|w)}\right) N(wu)$ and if $\Delta < K(n)$, then we prune wu to w, where $K(n) \sim C\log(n)$ and $C > 2|\mathcal{A}| + 3$. In [25] the *context tree weighting* algorithm was proposed for computing probability $P(S)$ of a Markov source S using an average over all possible models having orders less than a given order k.

Figures 9.1 and 9.2 show examples of a 2-order full Markov model and a 2-order variable-length Markov model of DNA respectively, where a filled (black) node means that the node is an internal node as well as a leaf node reflecting a variable-length context. Thus, variable-length Markov models efficiently capture redundancies that are typical for real-life data and therefore they are particularly well suited for our method for summarizing event sequences where space efficiency is the key issue.

9.2 Interpolated Markov model

In this section we turn our attention to a class of variable-length Markov models called interpolated Markov model (IMM) that does not optimize the state space but builds a variable-length Markov model implicitly as a result of parameter estimation

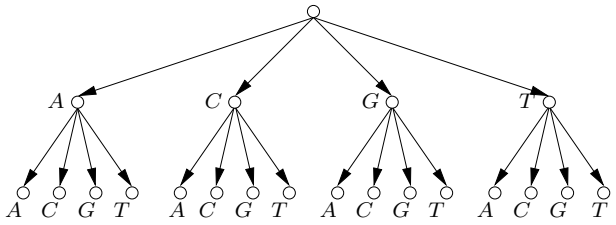

Figure 9.1.: 2-order full Markov model of DNA

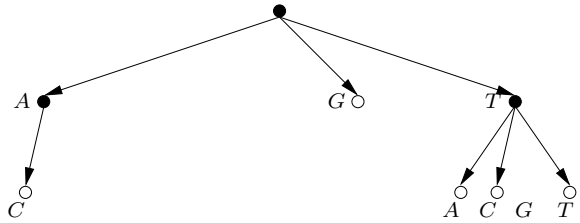

Figure 9.2.: 2-order variable-length Markov model of DNA

from sparse data. Consider a k-order full Markov model. Then a higher-order model should always do at least as well as, and frequently better than a lower-order model. However, in practice, when using a k-order model, if the training sequence is of length N then there are only $N - k$ strings of size $k + 1$ available to estimate $|\mathcal{A}|^{k+1}$ conditional probabilities and frequencies of some of the $|\mathcal{A}|^k$ contexts become too small or even zero. Deriving a model of too high order form such sparse data will lead to over-fitting.

The problem of parameter estimation of Markov models from sparse data is known as *smoothing* and has been widely discussed in the literature on language modeling [26]. The smoothing is a technique for adjusting the maximum likelihood estimates of probabilities to produce more accurate probabilities. The name smoothing comes from the fact that these methods tend to make the probabilities more uniform, by adjusting low probabilities upward and higher probabilities downward. Not only do smoothing methods generally prevent zero probabilities, but they

also improve the accuracy of the model as the whole. Whenever a probability is estimated from a fewer counts, smoothing has the potential to significantly improve estimation.

Techniques as *back-off* [27] and *interpolation* [28] have been implemented to deal with sparse data. The back-off model backs off to lower order models depending on counts of respective contexts. The interpolated model is a Markov chain with a new structure, where a conditional probability of order k is a combination of equal and lower order probabilities weighted by interpolation parameters, giving hight weight to probability estimates corresponding to high frequency contexts and lower weight to estimates corresponding to low frequency contexts. We use the notation $c_j[1:n]$ for $n = k, k-1, \ldots 1$ to denote a suffix of length n of context c_j of length k and we omit the notation for $n = k$, i.e., we write c_j instead of $c_j[1:k]$ in cases where k is implied. We are interested in Markov models that define conditional probabilities $P(a_i|c_j)$ as a linear combination of conditional probabilities corresponding to suffixes of c_j. The following recursion defines a value of the interpolated conditional probability in IMM:

$$P_{IMM}(a_i|c_j) \quad = \quad \lambda(c_j) \cdot P(a_i|c_j) + (1 - \lambda(c_j)) \cdot P_{IMM}(a_i|c_j[1:k-1]), \quad (9.1)$$

where $0 \leq \lambda(c_j) \leq 1$ and $P(a_i|c_j)$ is the probability estimate using the maximum likelihood (ML) estimate from the training data. For contexts c_j not observed in the training data, i.e., if $n(c_j a_i) = 0$ then we set $P(a_i|c_j) = P(a_i|c_j[1:n])$ for $n = max_{1 \leq n \leq k}\{n|n(c_j[1:n], a_i) > 0\}$ and this is exactly the place in the computation of parameters of an IMM where a variable-length Model is being implicitly built. The value of the parameter $\lambda(c_j)$ can be interpreted in many ways depending on the following interpretations of the IMM [29]:

- Context model interpretation: the parameters combine the predictions from contexts of varying length. Since longer contexts support stronger predictions and shorter contexts have more accurate statistics the interpolation of the predictions of different context lengths results in more accurate prediction than from a fixed context.

- State model interpretation: the parameters are hidden transitions from a higher order Markov model to a lower Markov model where the interpolation parameters model our beliefs about how much of the past is necessary to predict a state transition in an underlying Markov source of unknown order.

- Nonuniform model interpretation: the parameters are beliefs about conditional independence with probability $(1 - \lambda(c_j))$ that the future symbol does not depend on c_j.

In general if the frequency of context c_j is sufficiently high, the value of $\lambda(c_j)$ is close to 1. In the opposite case $\lambda(c_j)$ is close to zero and the interpolation probability $P_{IMM}(a_i|c_j)$ gains more from $P_{IMM}(a_i|c_j[1:k-1])$. However the problem of finding interpolation parameters is still more of an art than an exact science. In our experiments we assume a given order of IMM and use a modification of the method based on χ^2-test introduced in [30].

9.2.1 Computing interpolation parameters

In this section we present the χ^2-confidence based interpolation method introduced in the GLIMMER gene finding algorithm [30] for computing the $\lambda(c_i)$ in (9.1). Algorithm 6 presents a pseudocode of the algorithm.

Algorithm 6: k-order IMM parameter estimation

input : $n(c_j), n(c_j, a_i), N, k$
output: k-order $P_{IMM}(a_i|c_j)$
begin
  for $j = 1$ ***to*** $|\mathcal{A}|^k$ **do**
    $th = (N - k + 1)P(c_j)$;
    if $n(c_j) \geq th$ **then**
      $\lambda(c_j) = 1$
    else
      $\chi^2 = 0$;
      for $i = 1$ ***to*** $|\mathcal{A}|$ **do**
        $\chi^2 += \frac{(n(c_j,a_i) - n(c_j) * P_{IMM}(a_i|c_j[1:k-1]))^2}{n(c_j) * P_{IMM}(a_i|c_j[1:k-1])}$;
      $q = qchisquare(\chi^2, \mathcal{A} - 1)$;
      if $q \geq 0.5$ **then**
        $\lambda(c_j) = \frac{q * n(c_j)}{th}$;
        for $i = 1$ ***to*** $|\mathcal{A}|$ **do**
          $P_{IMM}(a_i|c_j) = \lambda(c_j)\frac{n(c_j,a_i)}{n(c_j)} + (1 - \lambda(c_j)) \cdot P_{IMM}(a_i|c_j[1 : k-1])$;
      else
        $\lambda(c_j) = 0$;
end

The algorithm takes as its input the following parameters: $n(c_j)$ the frequency of context c_j, $n(c_j, a_i)$ the frequency of string $c_j a_i$, k the order of the IMM and N the length of the training set. The algorithm computes the goodness-of-fit test to asses the discrepancy between $n(c_j, a_i)$ and $n(c_j) \cdot P_{IMM}(a_i|c_j[1 : k-1])$ with H_0 being the hypothesis that $P_{IMM}(a_i|c_j[1 : k])$ fits $P_{IMM}(a_i|c_j[1 : k-1])$ for $i = 1, 2, \ldots |\mathcal{A}|$, where $q = 1 - p_{value}$ for $\chi^2_{\mathcal{A}-1}$. Thus, if $q \geq 0.5$ then the algorithm rejects hypothesis H_0 and sets $\lambda(c_j) = \frac{q * n(c_j)}{th}$. Otherwise the algorithm backs up

to $P_{IMM}(a_i|c_j[1:k-1])$ by setting $\lambda(c_j) = 0$. The GLIMMER system used a fixed value for $th = 400$. We interpreted the threshold as the expected number of occurrences of a context c_i of length i as a string in the training set for 0-order Markov source. Thus, we set $th = \mathbf{E}[n(c_j)] = (N-k+1)P(c_j)$, where $P(c_j)$ is the probability of the context in the 0-order Markov model. Alternatively we could use $th = (N-k+1)P(c_j) - \sqrt{\mathbf{Var}[n(c_j)]}$.

9.3 Change detection and a sequence of models

While representing the entire history of a stream with one model with parameters the prediction accuracy of the model may deteriorate if there were changes of the underlying (true) model along the history of the stream. Therefore, if such changes happen one should detect them and represent each homogeneous interval in the history with a different model. As a method for detecting changes in model we propose the Jensen-Shanon distance [31]. The method detects a change by considering consecutive blocks of the same size in the sequence. Given three consecutive blocks b_{i-1}, b_i, b_{i+1} each of size n the method computes the distance as $M(b_i) = H(b_{i-1}, b_{i+1}) - 0.5(H(b_{i-1}) + H(b_{i+1}))$, where $H(x) = -\sum_{a\in\mathcal{A}} \frac{N_x(a)}{n} \log\left(\frac{N_x(a)}{n}\right)$, $H(x,y) = -\sum_{a\in\mathcal{A}} \frac{N_x(a)+N_y(a)}{2n} \log\left(\frac{N_x(a)+N_y(a)}{2n}\right)$ and $N_x(a)$ is the number of occurrences of symbol a in block x. If $M(b_i) > M_{th}$ then with high probability there is a change in block b_i, where $M_{th} = -\frac{\log(\alpha)}{2n}$ and α (e.g. $\alpha = 10^4$) is the probability of false detection [32]. The algorithm may be applied recursively top-down if an exact point of change is needed.

Since we have a limited amount of main memory resources we can only store a limited (fixed) number of models for a given stream. This implies that we have a *sliding window of models*, where the leftmost (oldest) model is removed from consideration when it is appropriately old and a newly learned model takes the rightmost position. Thus, besides the models themselves our method must maintain the points in time where each model started.

10 ANALYSIS OF THE PROBABILITY OF EXISTENCE OF AN EPISODE

In this chapter we analyze $P^{\exists}(w)$ for the following types of episodes: (1) a single subsequence pattern (a serial episodes); and (2) a set of subsequence patterns (a set of serial episodes) including the parallel episode case. The reason that we consider those two cases of $P^{\exists}(w)$ separately is that $P^{\exists}(w)$ for an arbitrary set of subsequence patterns is not a superposition of the results for a serial episode and poses different challenges.

Thus, given:

- a variable-length Markov model M with parameter set $\Theta(M)$ of an event sequence S
- a set of episodes $\mathcal{E}$

the goal is to find a formula for $P^{\exists}(w)$ for an arbitrary set of serial episodes $\mathcal{E}$.

In general $P^{\exists}(w)$ can be expressed as follows

$$P^{\exists}(w) \quad = \quad \sum_{x \in \mathcal{W}^{\exists}(w)} P(x),$$

where $\mathcal{W}^{\exists}(w)$ is the set of all distinct windows of length w that contain the set of episodes as a subsequence and $P(x)$ is the probability of x as a string in a given model.

10.1 Single subsequence pattern

In this section we derive a computational formula for $P(w)$ for a single subsequence pattern $e = [e[1], e[2], \ldots e[m]]$. While deriving the formula we derived the following: (1) an expression for $\mathcal{W}^{\exists}(w, e)$; (2) a general algorithm for computing $P(w, e)$; and (3) a formula for $C^{\exists}(w, e)$.

Assuming a 0-order Markov model of S we derived the following: (1) an exact formula for $P(w, e)$; (2) an asymptotic approximation of $P(w, e)$, which is of the form $P^{\exists}(w) = 1 - \Theta(\rho^w)$ for large w and $0 < \rho < 1$; and (3) a fast dynamic programming algorithm for computing $P(w, e)$.

10.1.1 Analysis of $\mathcal{W}^{\exists}(w, e)$

Let $\mathcal{W}^{\exists}(w, e)$ be the set of all distinct windows of length w containing e as a subsequence. Then the probability of existence of e can be expressed as follows

$$P^{\exists}(w, e) = \sum_{x \in \mathcal{W}^{\exists}(w,e)} P(x), \tag{10.1}$$

where $P(x)$ is the probability of string x in a given Markov model.

We will now show that a recursive formula for enumerating the elements of $\mathcal{W}^{\exists}(w, m)$ has the form below. Recall that the notation $\mathcal{W}^{\exists}(a, b)$ when $b < m$, means the set of windows of size a that contain the b-prefix of e (= the string consisting of the first b symbols of e).

$$\begin{cases} \mathcal{W}^{\exists}(w, m) = (\mathcal{A} - \{e[m]\}) \times \mathcal{W}^{\exists}(w-1, m) \cup \{e[m]\} \times \mathcal{W}^{\exists}(w-1, m-1) & w, m > 0, \\ \mathcal{W}^{\exists}(w, 0) = \mathcal{A}^w & w > 0, \\ \mathcal{W}^{\exists}(0, m) = 0 & m > 0, \\ \mathcal{W}^{\exists}(0, 0) = 1. \end{cases}$$

That the elements generated at each level of the recursion are distinct can be seen by observing that we divide $\mathcal{W}^{\exists}(w, m)$ into two subsets: strings that have $e[m]$ as their last symbol, and strings that have other symbols than $e[m]$ as their last symbol. We now turn our attention to showing that we do generate all strings of $\mathcal{W}^{\exists}(w, m)$. Consider all $\binom{w}{m}$ positions of a window where e may occur as a subsequence. We claim that the recursion considers all positions where the m respective symbols of e can occur, and that it considers these m-tuples of positions in a decreasing lexicographic order, that is, tuple $(i_1, i_2, \ldots, i_m)$ is considered before tuple $(i_1', i_2', \ldots, i_m')$ if the former is lexicographically larger than the latter.

Simply observe that $\mathcal{W}^{\exists}(w, m)$ can be split into two disjoint subsets:

- Windows having $e[m]$ at their last position. Because the last window symbol is fixed as $e[m]$ for all of them, their enumeration effectively becomes that of the windows o size $w-1$ that contain the $(m-1)$-prefix of e (= the string consisting the the first $m-1$ symbols of e). This latter enumeration is what we mean by the notation $\mathcal{W}^{\exists}(w-1, m-1)$.

- Windows not having $e[m]$ at their last position. Because the last symbol cannot be considered part of an occurrence of e, their enumeration effectively becomes that of the windows of size $w-1$ that contain e. This latter enumeration is what we mean by the notation $\mathcal{W}^{\exists}(w-1, m)$.

From the above, it is straightforward to obtain the following theorem.

Theorem 10.1.1 *The set of all distinct windows of length w, that contain a string e of length m as a subsequence, can be enumerated as follows.*

$$\mathcal{W}^{\exists}(w,m) = \bigcup_{\sum_{k=1}^{m+1} n_k = w-m} (\mathcal{A}-e[1])^{n_1} \times \{e[1]\} \times \ldots \times (\mathcal{A}-e[m])^{n_m} \times \{e[m]\} \times \mathcal{A}^{n_{m+1}}$$

The expression for $\mathcal{W}^{\exists}(w,m)$ can be represented as a recursion graph given in Figure 10.1. The graph has one start vertex (0) and one end vertex (m). The vertex set V of the graph, excluding vertex (0), consists of numbers $1, 2 \ldots m$ corresponding to indexes of symbols in e. The edge set E consists of edges from vertex (i) to (i+1) and self-loops from vertex (i) to itself for $0 \leq i \leq m$. The label for vertex from (i) to (i+1) is equal to $e[i+1]$. The label for self-loops of vertex (i) is equal to $\mathcal{A} - e[i+1]$ if $0 \leq i < m$ and $\mathcal{A}$ if $i = m$. The powers n_i for $i = 1, 2, \ldots m+1$ symbolize the number of times the i-th self-loop is used on the path from vertex (0) to (m). Thus, $\mathcal{W}^{\exists}(w,m)$ is equal to all paths from (0) to (m) of length w

Based on Theorem 10.1.1 we can divide the elements of $\mathcal{W}^{\exists}(w,m)$ into equivalence classes $\mathcal{V}^{\exists}$ with respect to the ordered sequences of $(n_1, n_2, \ldots, n_{m+1})$ for which $\sum_{i=1}^{m+1} n_i = w-m$. Clearly, the number of such ordered partitions is $\binom{w-m+m+1-1}{w-m} = \binom{w}{m}$.

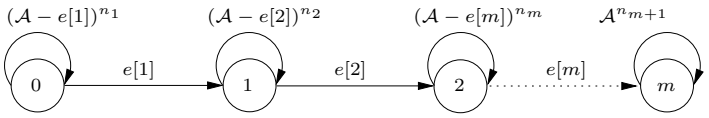

Figure 10.1.: Graphical interpretation of the solution to $\mathcal{W}^{\exists}(w, e)$

Table 10.1: Enumeration of $\mathcal{W}^{\exists}(3, 2)$ for $\mathcal{A} = \{a, b\}$ and $e = [b, a]$ using Theorem 10.1.1

i	$W(3,2)[i]$	n_1	n_2	n_3
2	$\underline{ba}a$	0	0	1
3	$\underline{ba}b$	0	0	1
4	$\underline{b}b\underline{a}$	0	1	0
1	$a\underline{ba}$	1	0	0

Example. Let $\mathcal{A} = \{a, b\}$ $e = [b, a]$ and $w = 3$. We generate $\mathcal{W}^{\exists}(3, 2)$ in Table 10.1 and compute $P^{\exists}(w, e)$. From (10.1) we obtain $P^{\exists}(3, 2) = P(e)(P(a) + P(b) + P(b) + P(a)) = 2P(e)$.

10.1.2 Algorithm for computing $P^{\exists}(w, e)$

This section presents a general algorithm for computing $P^{\exists}(w)$ for Markov sources [33].

Let x_i be the i-th symbol of a window $x \in \mathcal{W}^{\exists}(w, e)$ then the probability of the window in a k-order Markov source can be computed as follows

$$P(x) \;=\; P(x_0)P(x_1|x_0)\dots P(x_{k-1}|x_{k-2}\dots x_0)\cdot\prod_{i=k}^{w-1} P(x_i|x_{i-1}\dots x_{i-k})$$

where k is the order of the model of the source. However, using (10.1) directly is computationally very expensive. Furthermore computing $P(x)$ for every window x independently would be inefficient because many windows share the same prefix.

Therefore we propose a computational method where we enumerate the windows according to the depth-first traversal of a trie build from the members of $\mathcal{W}^{\exists}(w)$ without the trailing $\mathcal{A}^{n_{m+1}}$ that contributes a factor of 1 to the computation of the probability. The idea of this method is that the probability of each distinct prefix of $\mathcal{W}^{\exists}(w, e)$ is computed once. An example of such a trie for $e = [a, b, c]$ and $w = 4$ is shown in Figure 10.2.

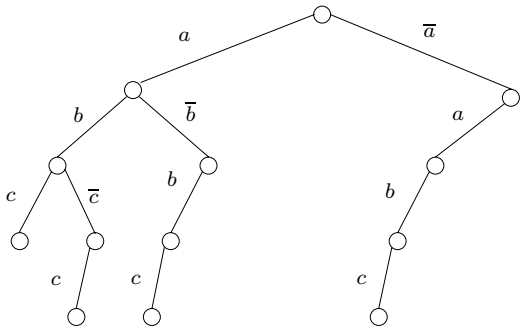

Figure 10.2.: A trie for the set of windows of length $w = 4$ containing $e = [a, b, c]$ as a subsequence

10.1.3 Analysis of $C^{\exists}(w, m)$

Recall that $C^{\exists}(w, m)$ denotes the cardinality of $\mathcal{W}^{\exists}(w, m)$.

The recurrence for $C^{\exists}(w, m)$ follows directly form the one for $\mathcal{W}^{\exists}(w, m)$. Namely,

$$\begin{cases} C^{\exists}(w,m) &= (|\mathcal{A}|-1)C^{\exists}(w-1,m) + C^{\exists}(w-1,m-1) & w,m>0, \\ C^{\exists}(w,0) &= |\mathcal{A}|^w & w>0, \\ C^{\exists}(0,m) &= 0 & m>0, \\ C^{\exists}(0,0) &= 1. \end{cases}$$

We use the method of generating functions to find the solution for $C^{\exists}(w, m)$. For an in-depth discussion of generating functions see, for example, [34]. We obtain the following theorem.

Theorem 10.1.2 *The number of all windows of length w over an alphabet $\mathcal{A}$ which contains at least one occurrence of a pattern of length m is equal to:*

$$C^{\exists}(w,m) = \sum_{k=0}^{w-m} \binom{k+m-1}{k} (|\mathcal{A}|-1)^k |\mathcal{A}|^{w-m-k}.$$

Proof. We leave m as a free variable and define the following family of generating functions

$$W_m(x) = \sum_{w=0} C^{\exists}(w,m) x^w$$

where x is a complex number. From the above recurrence we obtain

$$\begin{cases} W_m(x) = (|\mathcal{A}|-1)\sum_{w=1} C^{\exists}(w-1,m)x^w + \sum_{w=1} C^{\exists}(w-1,m-1)x^w & m>0, \\ W_0(x) = \sum_{w=0} C^{\exists}(w,0)x^w & m=0. \end{cases}$$

We now work with $W_m(x)$ for $m > 0$.

$$\begin{aligned} W_m(x) &= (|\mathcal{A}|-1)\sum_{w=1} C^{\exists}(w-1,m)x^w + \sum_{w=1} C^{\exists}(w-1,m-1)x^w \\ &= (|\mathcal{A}|-1)x\sum_{w=1} C^{\exists}(w-1,m)x^{w-1} + x\sum_{w=1} C^{\exists}(w-1,m-1)x^{w-1} \\ &= (|\mathcal{A}|-1)x\sum_{w=0} C^{\exists}(w,m)x^w + x\sum_{w=0} C^{\exists}(w,m-1)x^w \\ &= (|\mathcal{A}|-1)xW_m(x) + xW_{m-1}(x). \end{aligned}$$

We represent $W_m(x)$ in the form of a first order recurrence with respect to m.

$$\begin{aligned} W_m(x)(1-(|\mathcal{A}|-1)x) &= xW_{m-1}(x) \\ W_m(x) &= \frac{x}{(1-(|\mathcal{A}|-1)x)} W_{m-1}(x) \\ &= x^m \frac{1}{(1-(|\mathcal{A}|-1)x)^m} W_0(x). \end{aligned}$$

Using the fact that

$$W_0(x) = \sum_{w=0} |\mathcal{A}|^w x^w = \frac{1}{(1-|\mathcal{A}|x)}$$

we obtain

$$W_m(x) = x^m \frac{1}{(1-(|\mathcal{A}|-1)x)^m} \frac{1}{(1-|\mathcal{A}|x)}.$$

Denoting by $[x^w]f(x)$ the coefficient at x^w of $f(x)$, we find

$$C^{\exists}(w,m) \;=\; [x^w]W_m(x).$$

Since

$$[x^w]\frac{1}{(1-(|\mathcal{A}|-1)x)^m} \;=\; \binom{w+m-1}{w}(|\mathcal{A}|-1)^w,$$

and

$$[x^w]\frac{1}{(1-(|\mathcal{A}|-1)x)^m}\frac{1}{(1-|\mathcal{A}|x)} \;=\; \sum_{k=0}^{w}\binom{k+m-1}{k}(|\mathcal{A}|-1)^k|\mathcal{A}|^{w-k}$$

we finally obtain

$$[x^w]W_m(x) \;=\; \sum_{k=0}^{w-m}\binom{k+m-1}{k}(|\mathcal{A}|-1)^k|\mathcal{A}|^{w-m-k}.$$

10.1.4 Analysis of $P^{\exists}(w,e)$ for 0-order Markov models

The probability of existence $P^{\exists}(w,e)$ for a serial episode $e = [e[1], e[2], \ldots e[m]]$ in a 0-order Markov source satisfies the following recurrence

$$\begin{cases} P^{\exists}(w,m) &=& (1-P(e[m]))P^{\exists}(w-1,m) + P(e[m])P^{\exists}(w-1,m-1) & w,m>0, \\ P^{\exists}(w,0) &=& 1 & w>0, \\ P^{\exists}(0,m) &=& 0 & m>0, \\ P^{\exists}(0,0) &=& 1 & . \end{cases}$$

The recurrence for $P^{\exists}(w,m)$ follows directly from the one for $\mathcal{W}^{\exists}(w,m)$. Indeed, consider a window of size w. Observe that either the last symbol of the pattern, $e[m]$, does not occur at the w-th position of the window or it does occur. In the former situation e must occur within the window of size $w-1$ leading to the term $(1-P(e[m]))P^{\exists}(w-1,m)$ of the above recurrence. The latter situation provides the second term of the recurrence.

In Section 10.1.5 we solve the above recurrence using generating functions. In Section 10.1.6 we apply Cauchy's residue theorem to obtain an asymptotic expansion of $P^{\exists}(w,m)$ for fixed m and large w. In Section 10.1.7 we present a fast dynamic programing algorithm for $P^{\exists}(w,m)$. We summarize our results in the next theorem.

Theorem 10.1.3 *Consider a 0-order Markov source with p_i being the probability of generating the i-th symbol of e and $q_i = 1 - p_i$. Let also*

$$P(e) = \prod_{i=1}^{m} p_i$$

- *Then for all m and $w \geq m$ we have*

$$P^{\exists}(w,m) = P(e) \sum_{i=0}^{w-m} \sum_{\sum_{k=1}^{m} n_k = i} \prod_{k=1}^{m} q_k^{n_k}. \tag{10.2}$$

- *Let now m be fixed and assume $i \neq j$ implies $p_i \neq p_j$. Then as $w \to \infty$*

$$P^{\exists}(w,m) \approx 1 - P(e) \sum_{i=1}^{m} \frac{(1-p_i)^w}{p_i} \prod_{j \neq i}^{m} \frac{1}{p_j - p_i} \tag{10.3}$$

Proof. We prove (10.2) in Section 10.1.5 and (10.3) in Section 10.1.6.

Notice that the asymptotic approximation reveals the anticipated fact about the behavior of $P^{\exists}(w,m)$, i.e., that $P^{\exists}(w,m) = 1$ as $w \to \infty$, and the rate of convergence is exponential.

10.1.5 Exact solution

As before we use the method of generating functions to find the solution to $P^{\exists}(w,m)$. Let

$$W_m(x) \;=\; \sum_{w=0} P^{\exists}(w,m) x^w.$$

From the above recurrence we find

$$\begin{cases} W_m(x) &= q_m \sum_{w=1} P^{\exists}(w-1,m) x^w + p_m \sum_{w=1} P^{\exists}(w-1,m-1) x^w & m > 0, \\ W_0(x) &= \sum_{w=0} P^{\exists}(w,0) x^w & m = 0. \end{cases}$$

where $q_m = 1 - p_m$. We now work with $W_m(x)$ for $m > 0$

$$\begin{aligned} W_m(x) &= q_m \sum_{w=1} P^{\exists}(w-1,m)x^w + p_m \sum_{w=1} P^{\exists}(w-1,m-1)x^w \\ &= q_m x \sum_{w=1} P^{\exists}(w-1,m)x^{w-1} + p_m x \sum_{w=1} P^{\exists}(w-1,m-1)x^{w-1} \\ &= q_m x \sum_{w=0} P^{\exists}(w,m)x^w + p_m x \sum_{w=0} P^{\exists}(w,m-1)x^w \\ &= q_m x W_m(x) + p_m x W_{m-1}(x). \end{aligned}$$

We represent $W_m(x)$ in the form of the first order recurrence with respect to m

$$\begin{aligned} W_m(x)(1 - q_m x) &= p_m x W_{m-1}(x) \\ W_m(x) &= \frac{p_m x}{(1 - q_m x)} W_{m-1}(x) \\ &= \prod_{i=1}^{m} p_i x^m \prod_{i=1}^{m} \frac{1}{(1 - q_i x)} W_0(x) \end{aligned}$$

Using the fact that

$$\sum_{w=0} x^w = \frac{1}{(1-x)},$$

we obtain

$$\begin{aligned} W_m(x) &= \prod_{i=1}^{m} p_i x^m \prod_{i=1}^{m} \frac{1}{(1 - q_i x)} \frac{1}{(1-x)} \\ &= P(e) x^m \prod_{i=1}^{m} \frac{1}{(1 - q_i x)} \frac{1}{(1-x)}. \end{aligned}$$

But $P^{\exists}(w,m) = [x^w]W_m(x)$, and since

$$\prod_{i=1}^{m} \frac{1}{(1 - q_i x)} = \prod_{i=1}^{m} \sum_{w=0} q_i^w x^w,$$

and

$$\begin{aligned} [x^w] \prod_{i=1}^{m} \sum_{w=0} q_i^w x^w &= \sum_{\sum_{k=1}^{m} n_k = w} q_1^{n_1} q_2^{n_2} \ldots q_m^{n_m} \\ &= \sum_{\sum_{k=1}^{m} n_k = w} \prod_{k=1}^{m} q_k^{n_k} \end{aligned}$$

we use the partial sum property to derive the following

$$[x^w]\prod_{i=1}^{m}\sum_{w=0} q_i^w x^w \frac{1}{(1-x)} \;=\; \sum_{i=0}^{w}\sum_{\sum_{k=1}^{m} n_k=i}\prod_{k=1}^{m} q_k^{n_k},$$

we finally obtain

$$\begin{aligned}[x^w]W_m(x) &= P(e)[x^{w-m}]\prod_{i=1}^{m}\sum_{w=0} q_i^w x^w \frac{1}{(1-x)} \\ &= P(e)\sum_{i=0}^{w-m}\sum_{\sum_{k=1}^{m} n_k=i}\prod_{k=1}^{m} q_k^{n_k}.\end{aligned}$$

This proves (10.2) of Theorem 10.1.3.

10.1.6 Asymptotic approximation

Now we estimate $P^{\exists}(w,m)$ asymptotically as $w \to \infty$ and m fixed, that is, we prove (10.3) of Theorem 10.1.3. In our previous derivations we obtained

$$W_m(z) = P(e)z^m \prod_{i=1}^{m} \frac{1}{(1-q_i z)}\frac{1}{(1-z)}.$$

Observe that the exact value of $P^{\exists}(w,m)$ is equal to the coefficient of $W_m(x)$ at x^w which – we recall – we denote as $[x^w]W_m(x)$. By the Cauchy coefficient theorem (cf. [34]) we know that

$$P^{\exists}(w,m) = [z^w]W_m(z) = \frac{1}{2\pi i}\oint W_m(z)z^{-w-1}dz$$

where z is a complex variable and the integration is over a small circle around $z = 0$. To evaluate this integral we use another Cauchy result known as the Cauchy residue theorem [34]. For this we enlarge the circle around $z = 0$ so that it contains all singularities $W_m(z)$. In our case, the radius r of such a circle must satisfy $r > (1-p_{\max})^{-1}$. Then

$$P^{\exists}(w,m) = -\sum_{p} Res[W_m(z)z^{-w-1}, z=p] + O(r^{-w})$$

where $Res[f(z), z = a]$ is the residue of $f(z)$ at $z = a$. We recall that if $f(z) = \frac{\phi(z)}{\varphi(z)}$, where $\phi(z)$ and $\varphi(z)$ are analytic functions in $z = a$ subject to $\varphi(z) = 0$, $\varphi^{'}(z) \neq 0$ and $\phi(z) \neq 0$, then a is a pole of $f(z)$ and

$$Res\left[\frac{\phi(z)}{\varphi(z)}, z = a\right] = \frac{\phi(a)}{\varphi^{'}(a)}.$$

Therefore,

$$Res[W_m(z)z^{-w-1}, z = 1] = -P(e)1^{m-w-1}\prod_{i=1}^{m}\frac{1}{(1-q_i)} = -1.$$

Similarly for $z = \frac{1}{q_i}$ we have

$$\begin{aligned} Res\left[W_m(z)z^{-w-1}, z = \frac{1}{q_i}\right] &= (-1)\frac{1}{q_i}P(e)\left(\frac{1}{q_i}\right)^{m-w-1}\prod_{j\neq i}^{m}\frac{1}{\left(1-\frac{q_j}{q_i}\right)}\frac{1}{1-\frac{1}{q_i}} \\ &= P(e)\frac{(1-p_i)^w}{p_i}\prod_{j\neq i}^{m}\frac{1}{p_j-p_i}. \end{aligned}$$

Putting everything together we obtain

$$\begin{aligned} P^{\exists}(w,m) &= -Res[W_m(z)z^{-w-1}, z = 1] - \sum_{i=0}^{m} Res\left[W_m(z)z^{-w-1}, z = \frac{1}{q_i}\right] \\ &= 1 - P(e)\sum_{i=1}^{m}\frac{(1-p_i)^w}{p_i}\prod_{j\neq i}^{m}\frac{1}{p_j-p_i} + O(r^{-w}). \end{aligned}$$

10.1.7 Fast algorithm

Finally, we propose a dynamic programming algorithm for computing $P^{\exists}(w,m)$. Let $Q[i,j]$ denote the product $\prod_{k=1}^{j} q_k^{n_k}$ such that $\sum_{k=1}^{j} n_k = i$ then

$$\begin{cases} Q[i,j] = \sum_{k=0}^{i} Q[i-k, j-1]\cdot q_j^k & 1 < j \leq m, \quad 1 < i \leq w-m \\ Q[i,1] = q_1^i & \\ Q[0,j] = 1 & 1 \leq j \leq m \\ P^{\exists}(w,m) = P(e)\sum_{i=0}^{w-m} Q[i,m] & \end{cases}$$

The time complexity of the algorithm is $O((w-m)^2 \cdot m)$ and is equal to the space required to build the table $Q[w-m, m]$. Let $v[a:b]$ denote the substring of a string

v between indexes a and b such that $a < b$ and $1 \leq a, b \leq w$. Let $p[1:m]$ be an array with probabilities $p_1, p_2, \ldots, p_m$ of the symbols in e.

Algorithm 7: Computation of $P^{\exists}(w, m)$

```
input  : w, m, p[1 : m]
output: P^∃(w, m)
begin
    for j = 1 to m do
        Q[0, j] = 1;
    P^∃(w, m) = 1;
    for i = 1 to w − m do
        Q[i, 1] = (1 − p[1])^i;
        for j = 2 to m do
            Q[i, j] = 0;
            for k = 0 to i do
                Q[i, j] = Q[i, j] + Q[i − k, j − 1] ∗ (1 − p[j])^k;
        P^∃(w, m) = P^∃(w, m) + Q[i, m];
    P^∃(w, m) = P(e) · P^∃(w, m);
end
```

10.2 Set of subsequence patterns

This section extends the results presented in Section 10.1 to the case of a set of subsequence patterns, monitored simultaneously for an occurrence, including the important special case of the set of all permutations of a set of symbols (parallel episode). The parallel episode case captures situations where the ordering of the events within the window of observation does not matter, e.g., market basket items scanned by a cashier. The reason we distinguish the parallel episode case is because we will take advantage of the structure of permutations of a single pattern to design

an efficient algorithm instead of representing the parallel episode as a set of serial episodes.

Thus, the goal of the current section is to quantify $P^{\exists}(w, \mathcal{E})$ for a set $\mathcal{E} = \{e_1, e_2, \ldots, e_{|\mathcal{E}|}\}$. In order to compute $P^{\exists}(w, \mathcal{E})$, we need a formula for $\mathcal{W}^{\exists}(w, \mathcal{E})$. However, a considerable source of difficulty is the fact that $\mathcal{W}^{\exists}(w, \mathcal{E})$ for $|\mathcal{E}| > 1$ is not equal to the enumeration of sets $\mathcal{W}^{\exists}(w, e_i)$ because in general $\mathcal{W}^{\exists}(w, e_i) \cap \mathcal{W}^{\exists}(w, e_j) \neq \emptyset$ for $i \neq j$ where $i, j \leq |\mathcal{E}|$ and therefore considering $\mathcal{W}^{\exists}(w, e_i)$ and $\mathcal{W}^{\exists}(w, e_j)$ independently would lead to a failure of the probabilistic analysis of $P^{\exists}(w, \mathcal{E})$ due to double-counting of respective probabilistic events. Thus, we would need to apply an inclusion-exclusion principle. To appreciate the difficulty of this extension, consider the much-simplified case when there are only two subsequence patterns ($\mathcal{E} = \{e_1, e_2\}$) and no symbol is common to e_1 and e_2. Even in this case $\mathcal{W}^{\exists}(w, e_1) \cap \mathcal{W}^{\exists}(w, e_2) \neq \emptyset$ for appropriately large w. Add to this the fact that the different patterns typically do have common symbols or common subsequences or possibly common prefixes, that they may have different lengths, and the problem becomes nontrivial with a solution that strongly depends on the structure of the patterns in the set. This fact excludes any analytical solution to the case $|\mathcal{E}| > 1$.

The goal of the current section is a computational formula for $P^{\exists}(w, \mathcal{E})$. We provide a recurrence system for constructing $\mathcal{W}^{\exists}(w, \mathcal{E})$. Because the recurrence contains conditional statements, representing interactions of symbols of members of $\mathcal{E}$, we cannot find an analytical solution to the recurrence. Therefore we propose an efficient algorithmic method for enumerating $\mathcal{W}^{\exists}(w, \mathcal{E})$ using recursion graphs, which leads to a formula for $P^{\exists}(w, \mathcal{E})$. We applied our theoretical results by running an extensive series of experiments on real data. We used a part of *Wal-Mart* sales data.

10.2.1 Set of serial episodes

For the sake of the presentation we focus throughout this section on the case where either $\mathcal{E} = \{e_1, e_2\}$, or $\mathcal{E}$ is the set corresponding to a parallel episode but our derivations will easily be seen to generalize to an arbitrary set of episodes $\mathcal{E}$.

Let $\mathcal{W}^{\exists}(w, \mathcal{E})$ be the *set* of all distinct windows of length w containing at least one occurrence of at least one member of $\mathcal{E} = \{e_1, e_2\}$ as a subsequence. Then the probability of existence of $\mathcal{E}$ can be expressed as follows

$$P^{\exists}(w, \mathcal{E}) \;\; = \sum_{x \in \mathcal{W}^{\exists}(w,\mathcal{E})} P(x), \tag{10.4}$$

where $P(x)$ is the probability of string x in a given Markov model.

We could use Theorem 10.1.1 and express $\mathcal{W}^{\exists}(w, \mathcal{E})$ in terms of $\mathcal{W}^{\exists}(w, e_1)$ and $\mathcal{W}^{\exists}(w, e_2)$ using the inclusion-exclusion principle as $\mathcal{W}^{\exists}(w, e_1, e_2) = \mathcal{W}^{\exists}(w, e_1) \cup \mathcal{W}^{\exists}(w, e_2) - \mathcal{W}^{\exists}(w, e_1) \cap \mathcal{W}^{\exists}(w, e_2)$. However, it would be inefficient for the following reasons: (1) $\mathcal{W}^{\exists}(w, e_1) \cap \mathcal{W}^{\exists}(w, e_2) \neq \emptyset$ for appropriate large w and then the cardinality $|\mathcal{W}^{\exists}(w, e_1) \cap \mathcal{W}^{\exists}(w, e_2)|$ is exponential in w; and (2) the inclusion-exclusion "blows up" for $|\mathcal{E}| > 2$.

Below we present some examples of two sets $\mathcal{W}^{\exists}(w, e_1)$, $\mathcal{W}^{\exists}(w, e_2)$ and their intersections.

Example: Consider $\mathcal{A} = \{a, b, c, d\}$ and the following cases:

- $\mathcal{E} = \{[a, b], [c, d]\}$ and $w = 3$. Then $\mathcal{W}^{\exists}(w, e_1) \cap \mathcal{W}^{\exists}(w, e_2) = \emptyset$
- $\mathcal{E} = \{[a, b], [a, c]\}$ and $w = 3$. Then $\mathcal{W}^{\exists}(w, e_1) \cap \mathcal{W}^{\exists}(w, e_2) = \{[a, b, c], [a, c, b]\}$
- $\mathcal{E} = \{[a, b], [a]\}$ and $w = 3$. Then $\mathcal{W}^{\exists}(w, e_1) \cap \mathcal{W}^{\exists}(w, e_2) = \mathcal{W}^{\exists}(w, e_1)$, i.e. this case is equivalent to $\mathcal{E} = \{a\}$ because an occurrence of ab implies an occurrence of a in the window

Clearly, the example shows that the cardinality of $\mathcal{W}^{\exists}(w, \mathcal{E})$ depends on symbols in corresponding positions in e_1 and e_2 that determine the size of the intersection. This

implies that, unlike the solution to $P^{\exists}(w, e)$, we cannot find an analytical solution to $P^{\exists}(w, \mathcal{E})$. Therefore, we adopt a computational approach to finding $P^{\exists}(w, \mathcal{E})$ with the goal of providing an efficient algorithm that uses a graph to represent the dependences (interactions) of symbols in e_1 and e_2. We start by presenting a recurrence defining $\mathcal{W}^{\exists}(w, \mathcal{E})$.

In the 0-order Markov model the probability of a pattern is equal to a product of individual probabilities of symbols. Thus, to any recursive formula for $\mathcal{W}^{\exists}(w, m_1, m_2)$ corresponds a similar formula for $P^{\exists}(w, m_1, m_2)$ (and vice-versa). We now show that $P^{\exists}(w, m_1, m_2)$ satisfies the following recurrence, where for the purpose of readabil-

ity of the formula we assume that e_1 and e_2 are reversed, i.e., $e_1 = [e_1[m], e_1[m-1], \ldots e_1[1]]$ and $e_2 = [e_2[m], e_2[m-1], \ldots e_2[1]]$.

$$
\begin{cases}
\textbf{if} \quad e_1[M_1] \neq e_2[M_2] \quad \textbf{then} \\
P^{\exists}(W, M_1, M_2) = P(e_1[M_1])P^{\exists}(W-1, M_1-1, M_2) + \\
\qquad\qquad P(e_2[M_2])P^{\exists}(W-1, M_1, M_2-1) + \\
\qquad\qquad (1 - P(e_1[M_1]) - P(e_2[M_2]))P^{\exists}(W-1, M_1, M_2) \\
\qquad\qquad \textbf{for} \quad w > 0, M_1, M_2 > 0 \\
\textbf{if} \quad e_1[M_1] = e_2[M_2] \quad \textbf{then} \\
P^{\exists}(W, M_1, M_2) = P(e_1[M_1])P^{\exists}(W-1, M_1-1, M_2-1) + \\
\qquad\qquad (1 - P(e_1[M_1]))P^{\exists}(W-1, M_1, M_2) \\
\qquad\qquad \textbf{for} \quad W > 0, M_1, M_2 > 0 \\
P^{\exists}(W, 0, 0) = 1 \qquad \textbf{for} \quad W \geq 0 \\
P^{\exists}(0, M_1, M_2) = 0 \qquad \textbf{for} \quad M_1, M_2 > 0 \\
P^{\exists}(1, M_1, M_2) = 1 \qquad \textbf{for} \quad min\{M_1, M_2\} = 0
\end{cases}
$$

Indeed, consider all $\binom{w}{m_1}$ and $\binom{w}{m_2}$ positions in a window in $\mathcal{W}^{\exists}(w, m_1, m_2)$ where e_1 or e_2 can occur as a subsequence. We can think of those positions as m_1-tuples and m_2-tuples respectively. Now let consider all windows in $\mathcal{W}^{\exists}(w, m_1, m_2)$ sorted according to increasing lexicographic order of those tuples. Then depending on whether the first symbols of e_1 and e_2 are equal or not there are two cases. If $e_1[m_1] \neq e_2[m_1]$ then there are three cases: either $e_1[m_1]$ is the first symbol in the window giving the term $P(e_1[m_1])P^{\exists}(w-1, m_1-1, m_2)$, or $e_2[m_2]$ is the first

symbol in the window giving the term $P(e_2[m_2])P^{\exists}(w-1, m_1, m_2-1)$, or none of the above which leads to the term $(1-P(e_1[m_1])-P(e_2[m_2]))P^{\exists}(w-1, m_1, m_2)$. If $e_1[m_1] = e_2[m_2]$ then there are two cases depending on whether the first symbol of the window is equal to $e_1[m_1]$ or not. From the above discussion it is clear that the shape of the recursion tree is determined by interactions between symbols in e_1 and e_2, i.e., whether their symbols at pairs of positions are equal or not. Therefore, in order to find a solution to $P^{\exists}(w, m_1, m_2)$ we have to enumerate all pairs of indices (M_1, M_2) such that $P^{\exists}(W, M_1, M_2)$ appears in the recursion tree (not all such pairs of indices qualify). This recursion tree in a form of a graph is now described more formally (as stated earlier, in addition to depicting the recurrence, the graph also describes all elements of $\mathcal{W}^{\exists}(w, m_1, m_2)$. Below we show a recursive description of the graph.

Let $G(\mathcal{E}) = (V, E)$ be an edge-labeled directed graph defined as follows. The vertex set V is a subset of all the pairs (i, j), $0 \leq i \leq m_1$, $0 \leq j \leq m_2$. Figure 10.3 contains an inductive definition of $G(\mathcal{E}) = (V, E)$.

Let $Edges(path)$ and $Vertices(path)$ denote the sequence of consecutive edges and (respectively) vertices in any $path$, except that $Vertices(path)$ does not include the last vertex on $path$ (why this is so will become apparent below). Let $\mathcal{R}$ be the set of all distinct simple paths (i.e., without self-loops) from the start-vertex to any end-vertex. Let $\mathcal{L}_w$ be the set of all distinct paths of length w, *including self-loops*, from the start-vertex to all end-vertices (that is, self-loops do count towards path length). Then the set of all windows containing $\mathcal{E} = \{e_1, e_2\}$ as a subsequence can be enumerated as follows

$$\mathcal{W}^{\exists}(w, \mathcal{E}) \quad = \quad \{Edges(path) : path \in \mathcal{L}_w\}. \tag{10.5}$$

Examples of $G(\mathcal{E})$ are shown in Figure 10.4 and in Figure 10.5. For every vertex (i, j) we annotated its self-loop with $n_{i,j}$ denoting the number of times a path from the start-vertex to the end-vertex uses the loop.

- $(0,0)$ is in V.
- If (i,j) is in V, $i < m_1$, and $e_1[i+1] \neq e_2[j+1]$ then $(i+1,j)$ is also in V, and an edge from (i,j) to $(i+1,j)$ labeled $e_1[i+1]$ exists in E.
- If (i,j) is in V, $j < m_2$, and $e_1[i+1] \neq e_2[j+1]$ then $(i,j+1)$ is also in V, and an edge from (i,j) to $(i,j+1)$ labeled $e_2[j+1]$ exists in E.
- If (i,j) is in V, $i < m_1$ and $j < m_2$, and $e_1[i+1] = e_2[j+1]$ then $(i+1,j+1)$ is also in V, and an edge from (i,j) to $(i+1,j+1)$ labeled $e_1[i+1]$ $(= e_2[j+1])$ exists in E.
- A self-loop from vertex (i,j) to itself exists and has label equal to (i) $\mathcal{A}$ if $i = m_1$ or $j = m_2$, (ii) $\mathcal{A} - \{e_1[i+1]\} - \{e_2[j+1]\}$ if $i < m_1$ and $j < m_2$.

The following observations, in which we do not count self-loops towards the in-degree and out-degree of a vertex, follow from the above definition of $G(\mathcal{E})$.

- The in-degree of vertex $(0,0)$ (start vertex) equals zero, the out-degree of vertices $(m_1,j),(i,m_2)$ for $i \leq m_1, j \leq m_2$ (end-vertices) equals zero.
- The in-degree and out-degree of every vertex (i,j) is at most $|\mathcal{E}|$.
- $|V| = O(m_1 m_2)$ and $|E| = O(|\mathcal{E}| m_1 m_2)$.

Figure 10.3.: Inductive definition of $G(\mathcal{E})$

Example: Consider $\mathcal{A} = \{a,b,c,d\}$, $\mathcal{E} = \{[a,b],[a,c]\}$ and $w = 3$. We use $G(\mathcal{E})$ from Figure 10.5 and applying (10.5) we enumerate elements of $\mathcal{W}^{\exists}(3,\mathcal{E})$ which are shown in Table 10.2. Clearly, using the presented method we generate all members of $\mathcal{W}^{\exists}(3,\mathcal{E})$ without duplicates. If we compare Figure 10.1 with Figures 10.4 and 10.5 it is clear that $G(\mathcal{E})$ for a set containing only one episode reduces to the graph

Table 10.2: Enumeration of $\mathcal{W}^{\exists}(3, \mathcal{E})$ for $\mathcal{A} = \{a, b, c, d\}$ and $\mathcal{E} = \{[a, b], [a, c]\}$ using $G(\mathcal{E})$ from Figure 10.5

End vertex	$\mathcal{W}^{\exists}(3, \mathcal{E})$	$n_{0,0}$	$n_{1,1}$	$n_{2,1}$	$n_{1,2}$
(2, 1)	$ab \times A$	0	0	1	0
(2, 1)	aab	0	1	0	0
(2, 1)	adb	0	1	0	0
(2, 1)	bab	1	0	0	0
(2, 1)	cab	1	0	0	0
(2, 1)	dab	1	0	0	0
(1, 2)	$ac \times A$	0	0	1	0
(1, 2)	aac	0	1	0	0
(1, 2)	adc	0	1	0	0
(1, 2)	bac	1	0	0	0
(1, 2)	cac	1	0	0	0
(1, 2)	dac	1	0	0	0

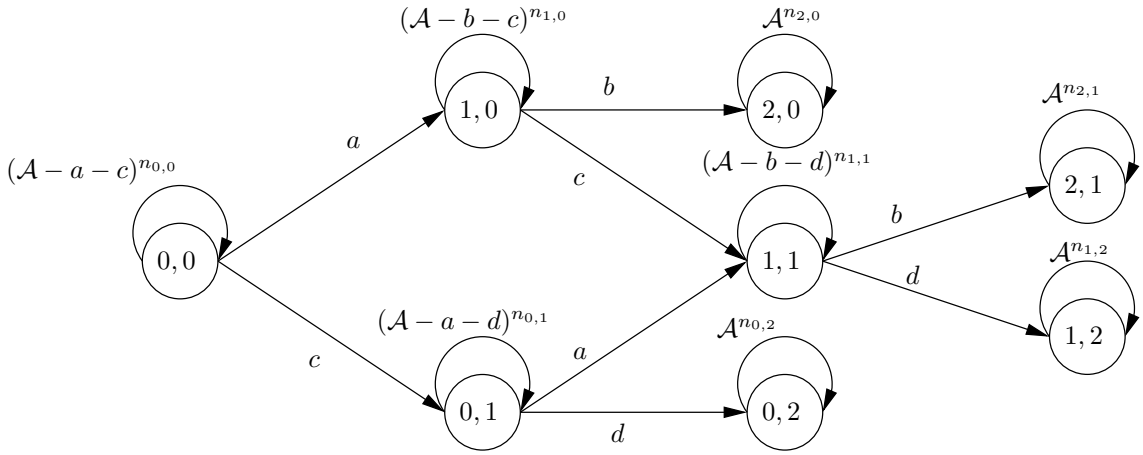

Figure 10.4.: $G(\mathcal{E})$ for $\mathcal{E} = \{[a, b], [c, d]\}$

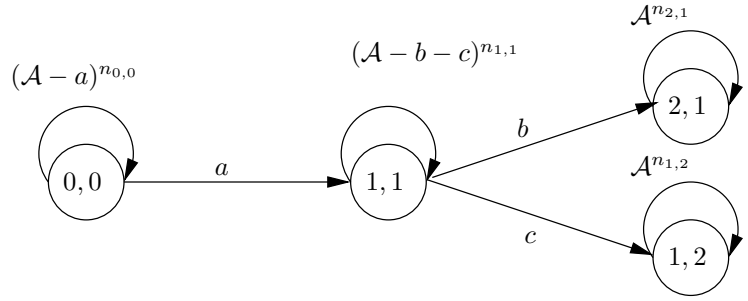

Figure 10.5.: $G(\mathcal{E})$ for $\mathcal{E} = \{[a, b], [a, c]\}$

representing $\mathcal{W}^{\exists}(w, e)$ for a serial episode. The formulas (10.4), (10.5) and (10.2) lead to the following theorem.

Theorem 10.2.1 *Consider a* 0*-order Markov event sequence with* $P(e_i[k])$ *being the probability of generating the* k*-th symbol of* $e_i \in \mathcal{E} = \{e_1, e_2\}$*. Let also*

$$P(Edges(path)) = \prod_{edge \in Edges(path)} P(label(edge)).$$

Then for m_1, m_2 *and* $w \geq m_i$ *we have*

$$P^{\exists}(w, m_1, m_2) = \sum_{path \in \mathcal{R}} P(Edges(path)) \sum_{g=0}^{w-|Edges(path)|} \sum_{(i,j) \in Vertices(path), \sum n_{i,j}=g} \prod_{(i,j)} [1 - P(\{e_1[i+1]\} \cup \{e_2[j+1]\})]^{n_{i,j}},$$

where $0 \leq n_{i,j} \leq w - |Edges(path)|$.

Proof. Easily follows from (10.4), (10.5) and (10.2).

Thus, the reason that $Vertices(path)$ does not include the last vertex is because in the memoryless model $P(\mathcal{A}^{n_{i,j}}) = 1$ for $0 \leq n_{i,j}$.

For the purpose of our experiments, we implemented an efficient dynamic programming algorithm based on Theorem 10.2.1. The time complexity of the algorithm is $O(|R|w^2)$, where $m = \min\{m_1, m_2\}$ since we consider $|R|$ paths and each of them requires $O(w^2)$ [15]. In section 11.4.2 we present evidence that Theorem 10.2.1 works on real data by showing that $\overline{\Omega^{\exists}}(n, w)$ closely approximates $P^{\exists}(w, \mathcal{E})$.

10.2.2 Parallel episode

We also solved $P^{\exists}(w, \mathcal{E})$ for an important special case when $\mathcal{E}$ consist of all permutations of a single pattern, which is the case of a parallel episode. Using Theorem 10.2.1 directly for a parallel episode as a set of patterns would be inefficient because we would then need to consider a graph having a disastrous $|V| = O(m^{m!})$. In order to simplify the graph $G(\mathcal{E})$ that would result from all permutations, and bring its number of vertices down to a manageable size (quantified below), we exploit the structure of the set of all permutations to design a different graph. Notice that, for a parallel episode, every path in $\mathcal{R}$ is a permutation of symbols in e. In addition, the out-degree of a vertex is at most m if all symbols of e are different. Furthermore a transition from from $P^{\exists}(k, i, j, \ldots)$ to $P^{\exists}(k+1, i', j', \ldots)$ takes place for any symbol of e not seen so far since the order of symbols does not matter. This observations allow us to introduce a variant of $G(\mathcal{E})$ called $G_{\|}(e)$.

Let $G_{\|}(e) = (V, E)$ be a directed edge-labeled graph defined as follows. The vertex set V consists of submultisets of size $i = 0, 1, \ldots, m$ of the multiset of symbols in $e = \{e[1], e[2], \ldots, e[m]\}$. Let $e' = \{e'[1], e'[2], \ldots e'[m']\}$ be the set of cardinality m' obtained from e by eliminating duplicates and then sorting it. Let c_i for $i = 0, 1 \ldots m'$ be the number of times symbol $e'[i]$ occurs in e. We represent the submultisets as integer vectors of the form $(i_1, i_2, \ldots, i_{m'})$, where $0 \leq i_j \leq c_j$ if the vertex contains i_j symbols $e'[j]$ in its submultiset. Figure 10.6 contains an inductive definition of $G_{\|} = (V, E)$, where $\cup$ is multiset union (e.g.$\{c\} \cup \{c\} = \{c, c\} = \cup^2 c$).

- $(0, 0, \ldots, 0)$ and $(c_1, c_2, \ldots, c_{m'})$ are in V.
- If $(i_1, i_2, \ldots, i_{m'})$ is in V and $\sum_{j=1}^{m'} i_j < m$ then $(k_1, k_2, \ldots, k_{m'})$ is in V such that $\sum_{j=1}^{m'} k_j = \sum_{j=1}^{m'} i_j + 1$ and an edge with label equal to $\{\cup^{k_1} e'[1], \cup^{k_2} e'[2], \ldots, \cup^{k_m} e'[m]\} - \{\cup^{i_1} e'[1], \cup^{i_2} e'[2], \ldots, \cup^{i_m} e'[m]\}$ exists in E.
- A self-loop from vertex $(i_1, i_2, \ldots, i_{m'})$ to itself exists and has label equal to (i) $\mathcal{A}$ for $(c_1, c_2, \ldots, c_{m'})$, (ii) $\mathcal{A} - \bigcup_{i_j=1} \{e'[i_j]\}$ otherwise.

The following observations, in which we do not count self-loops towards the in-degree and out-degree of a vertex, follow from the above definition of $G_{\|}(e)$.

- The in-degree of the start-vertex $(0, 0, \ldots, 0)$ equals zero, the out-degree of the end-vertex $(c_1, c_2, \ldots, c_{m'})$ equals zero. The in-degree and out-degree of every vertex is at most m.
- $|V| = O(2^m)$ and $|E| = O(|V|m)$.

Figure 10.6.: Inductive definition of $G_{\|}(e)$

Examples of $G_{\|}(e)$ are shown in Figure 10.7 and in Figure 10.8. As in the case of $G(\mathcal{E})$ for every vertex $(i_1, i_2, \ldots, i_{m'})$ we annotated its self-loop with $n_{i,j}$ denoting

the number of times a path from the start-vertex to the end-vertex uses the loop. The next theorem is an adoption of Theorem 10.2.1 to the case of a parallel episode.

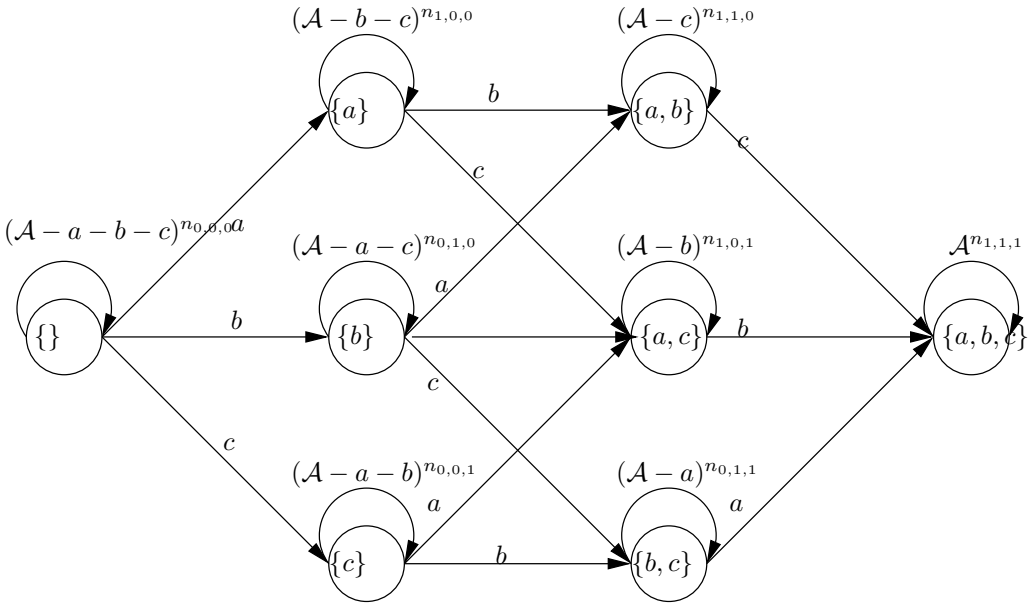

Figure 10.7.: $G_{\|}(e)$ for $e = \{a, b, c\}$ and $\mathcal{A} = \{a, b, c, d\}$

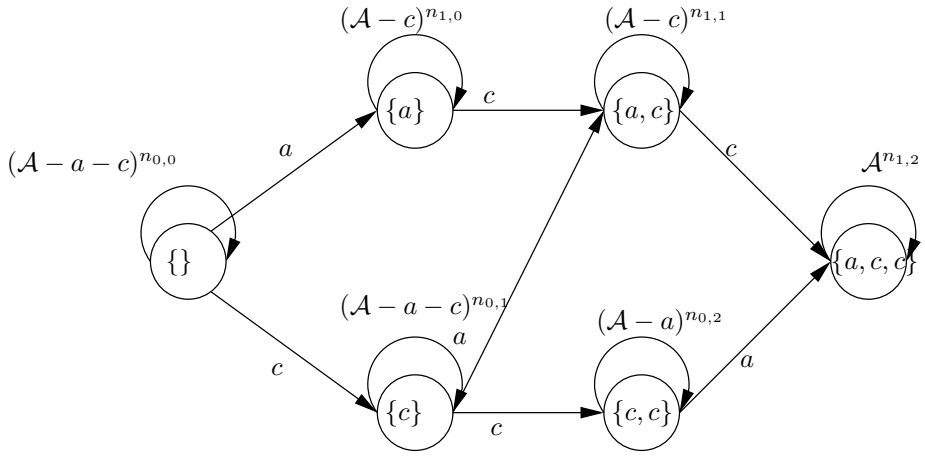

Figure 10.8.: $G_{\|}(e)$ for $e = \{a, c, c\}$ and $\mathcal{A} = \{a, b, c, d\}$

Theorem 10.2.2 *Consider a 0-order Markov event sequence with $P(e[k])$ being the probability of generating the k-th symbol of e where e is a parallel episode with*

$$P(e) = \prod_{i=1}^{m} P(e[i])$$

then for all m and $w \geq m$ we have

$$P^{\exists}(w) = P(e) \sum_{path \in \mathcal{R}} \sum_{g=0}^{w-m} \sum_{(i_1,\ldots,i_{m'}) \in Vertices(path), \sum n_{i_1,\ldots,i_{m'}} = g} \prod \left(1 - P\left(\bigcup_{i_j=0} \{e[i_j]\}\right)\right)^{n_{i_1,\ldots,i_{m'}}},$$

where $0 \leq n_{i_1,\ldots,i_{m'}} \leq w - m$.

Proof. Easily follows from (10.5) and (10.2).

For the purpose of our experiments we implemented an efficient dynamic programming algorithm based on Theorem 10.2.2. The time complexity of the algorithm is $O(m!w^2)$, since we consider $|\mathcal{R}| = O(m!)$ paths and each of them requires $O(w^2)$ [15]. In section 11.4.1 we present evidence that Theorem 10.2.2 works on real data by showing that $\overline{\Omega^{\exists}}(n, w, \mathcal{E})$ closely approximates $P^{\exists}(w, \mathcal{E})$.

11 EXPERIMENTAL RESULTS

The main purpose of experiments, presented in this chapter, was to test the derived analytic formulas for $P^{\exists}(w)$ for real data since the accuracy of the threshold mechanism in detecting significant episodes is determined by the accuracy of the respective formulas for $P^{\exists}(w)$ in predicting frequencies of episodes in the reference sequence. To accomplish this we compared $\overline{\Omega^{\exists}}(n, w)$ with $P^{\exists}(w)$ for different values of w using the following error metric d

$$d = \left[\frac{1}{r}\sum_{i=1}^{r}\frac{|\overline{\Omega^{\exists}}(n, w_i) - P^{\exists}(w_i)|}{\overline{\Omega^{\exists}}(n, w_i)}\right] 100\% \tag{11.1}$$

where $w_1 < w_2 < \ldots w_r$ are the tested window sizes.

We used the following sources in experiments in this chapter: English text in Section 11.1, web accesses in Section 11.2, DNA in sections 11.3 and *Wal-Mart* transactions in Section 11.4.

11.1 English text source

In this section we consider an English text represented by an on-line version of *War and Peace* by Leo Tolstoy. The purpose of the experiments in this section was to test how the formula for $P^{\exists}(w, e)$ for 0-order model works for a source that apparently is not 0-order. The attractiveness of 0-order models of a reference sequence stems from the fact that we can use the presented fast algorithms for computing $P^{\exists}(w)$ in such models. An English text is of course not memoryless. As an example consider string "th" which occurs more frequently than "tz" or "ts". So in this example the letter "h" will occur more likely if the previous letter was "t". Thus, in order to test the formulas we assumed a 0-order model of the text and estimated the model parameters from frequencies of observed symbols.

11.1.1 Serial episode

In this experiment we compare $\overline{\Omega^{\exists}}(n, w, e)$, discovered using Algorithm 3, with $P^{\exists}(w, e)$, computed using Algorithm 7. We set an example episode $e = [g, w, a, d, e, r, a]$ and compared $\Omega^{\exists}(n, w)$ with $P^{\exists}(w)$ for selected values of w. Figure 11.1 illustrates the results that show two main facts: $P^{\exists}(w)$ approaches 1 as w goes to infinity and $P^{\exists}(w)$ very closely approximates $\overline{\Omega^{\exists}}(n, w)$ with $d = 12\%$.

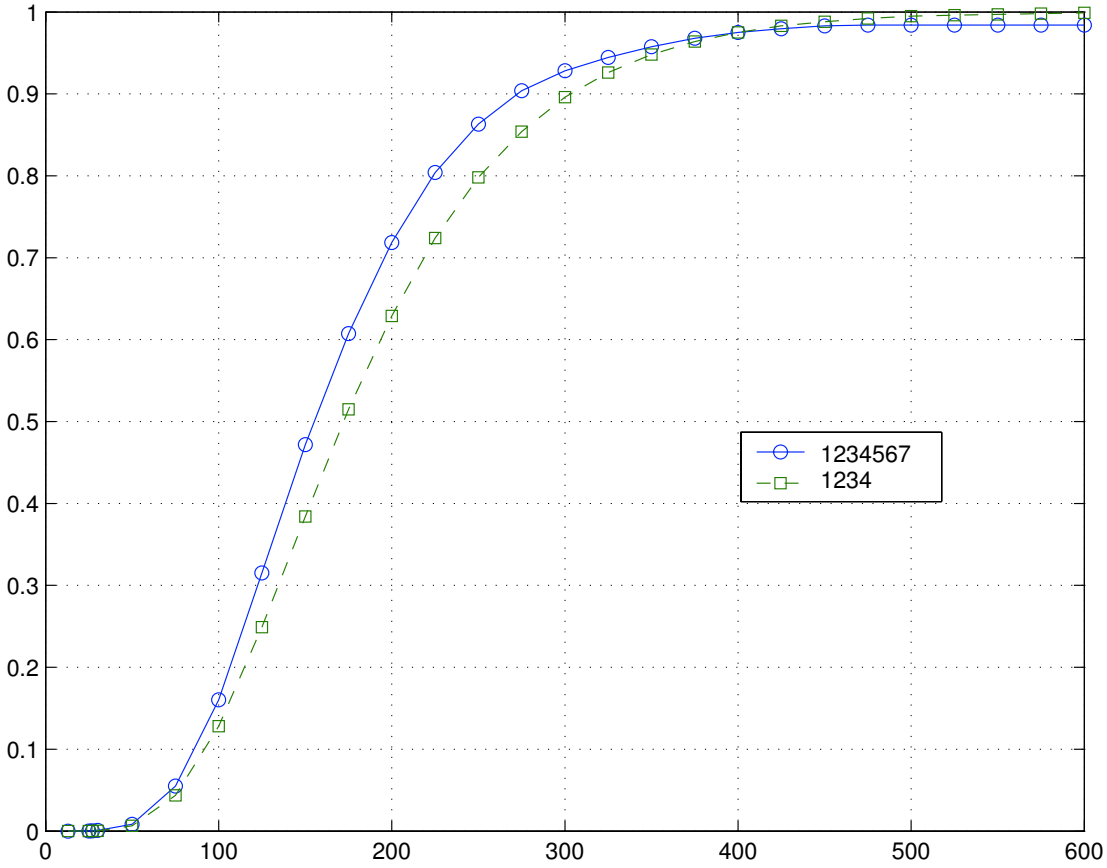

Figure 11.1.: $\overline{\Omega^{\exists}}(n, w)$ and $P^{\exists}(w)$ for a serial episode

11.1.2 Upper threshold

In this experiment we demonstrate the application of the upper threshold $\tau_u(w)$. We assumed that the reference sequence is represented by a 0-order model and computed the parameters from the monitored sequence. We set $\beta(b) = 10^{-8}$, $w = 100$ and kept injected an example episode $e = [w, o, j, c, i, e, c, h]$ as a w-windowed subsequence into different places of the source. After each insertion we discovered $\overline{\Omega^{\exists}}(n, w)$ and checked whether it exceeded $\tau_u(w)$. To make it more interesting we considered two values of gaps between inserted symbols of e: $gap = 0$ and $gap = 11$. In other words we injected e as $s_1 g^{gap} s_2 g^{gap} \dots g^{gap} s_m$, where $g \in \mathcal{A}^+$. The results are shown in Figure 11.2.

The horizontal dash-dot line shows $P^{\exists}(w, e)$ and the solid line shows $\tau_u(w)$. We can see that if $gap = 0$, then we only need two episodes to exceed $\tau_u(w)$ versus three if $gap = 11$. This makes a sense if we notice that if the episode is stretched to the window boundaries ($gap = 11$) then it is more noise-like compared to the case when $gap = 0$, which suggests an intentional action (attack) and should be detected early. Clearly, $\tau_u(w)$ provides a sharp detection of significant episodes.

11.2 Web accesses

In this section we consider web accesses represented by logs of user accesses to the music machines web site (currently at `http://machines.hyperreal.org`), which recorded accesses from 1/01/99 through 4/30/99 . We focused on the `http://machines.hyperreal.org/manufacturers/` web page containing links to 81 pages of manufacturers of music instruments and created an event sequence by considering only unique accesses made by the same originating host to one of the manufacturers, i.e, if a given host made many accesses to the same manufacturer per session then we treated it as one access and consider the first access only. The purpose of the experiment in this section was to test how the formula for $P^{\exists}(w, e)$ for 0-order model works for a source that apparently is not 0-order. The web accesses are not

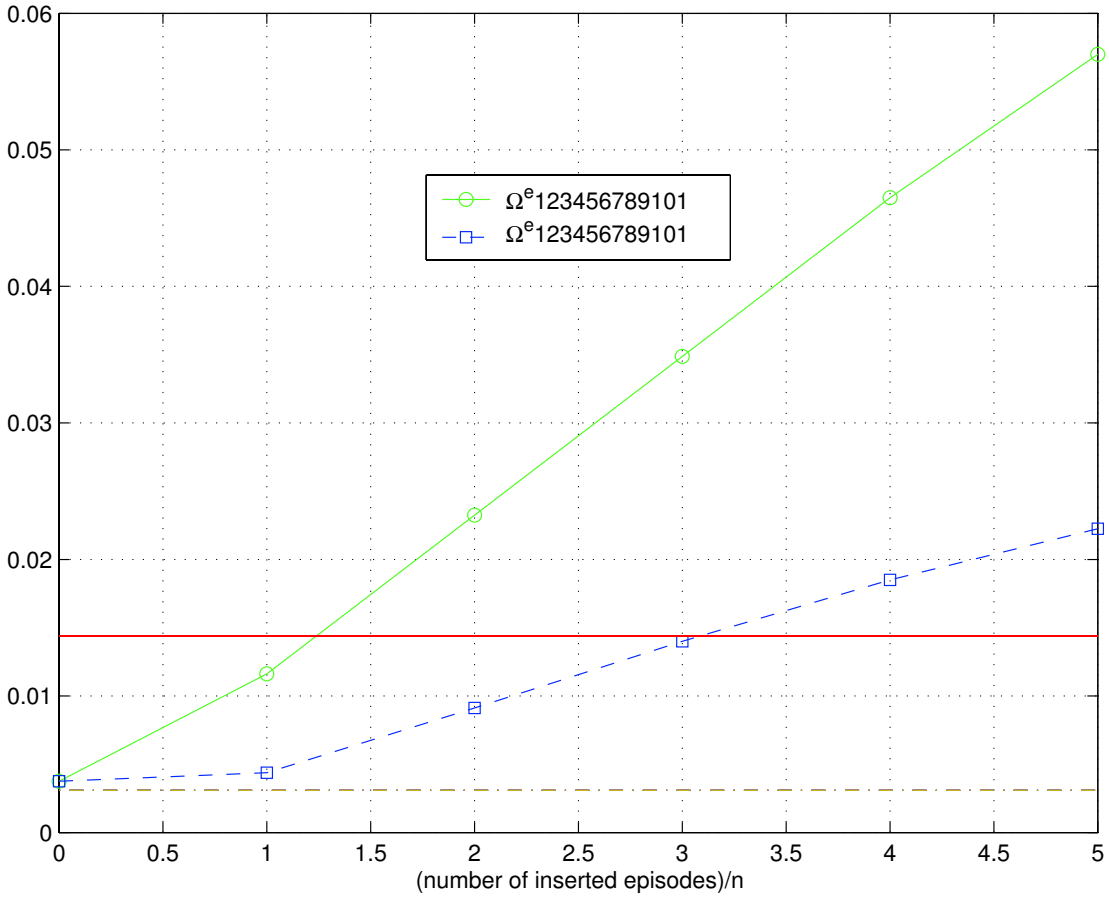

Figure 11.2.: Detection of injected serial episode

memoryless not only because of hierarchical structure but also because of correlations between links. For example a person looking for a product in an on-line store will most likely visit all manufacturers of the search product category.

11.2.1 Serial episode

In this experiment we compare $\overline{\Omega^{\exists}}(n, w, e)$ discovered using Algorithm 3 with $P^{\exists}(w, e)$, computed using Algorithm 7. We set an example episode $e = \{Akai, ARP, Korg, Moog, Yamaha, Casio, Sequential\}$ and compared $\Omega^{\exists}(n, w)$ with $P^{\exists}(w)$ for

selected values of w. Figure 11.3 illustrates the results that show that $P^{\exists}(w, e)$ still provides a good approximation of $\overline{\Omega^{\exists}}(n, w)$ with $d = 14\%$.

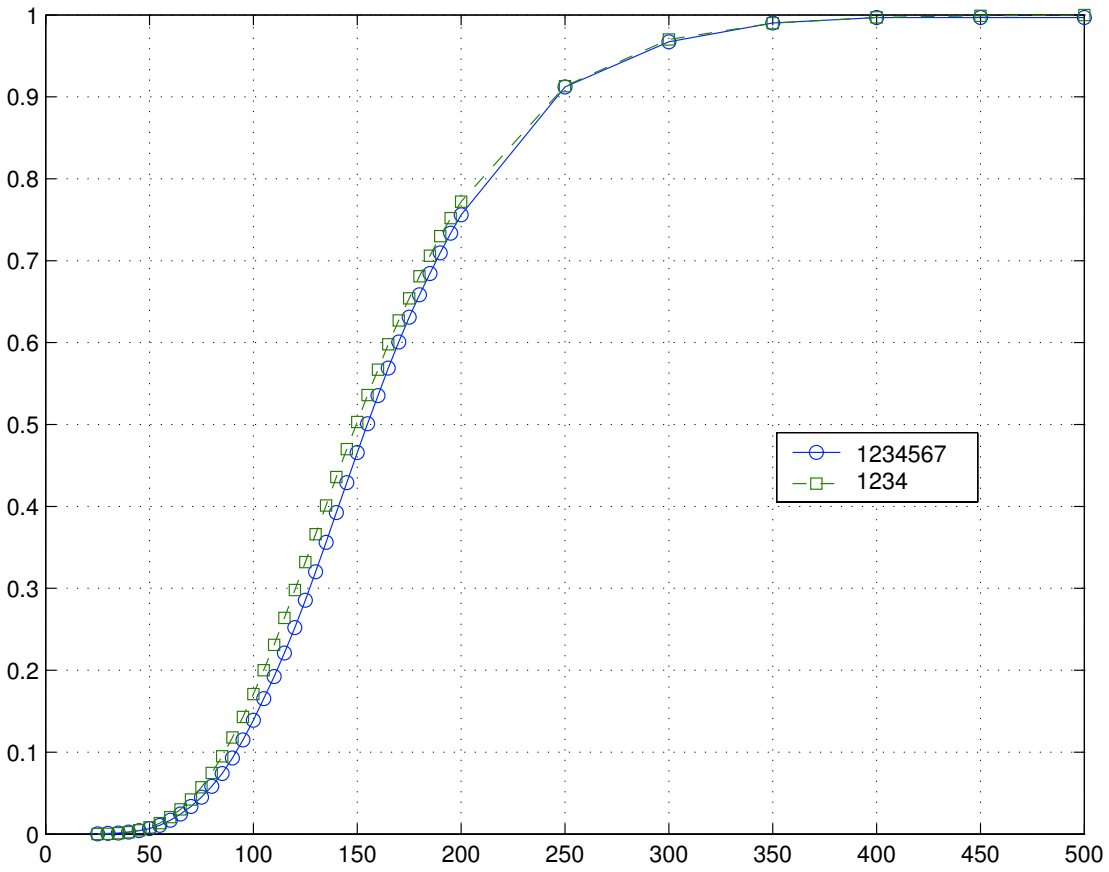

Figure 11.3.: $\overline{\Omega^{\exists}}(n, w)$ and $P^{\exists}(w)$ for the web accesses

11.3 DNA

In this section we consider genomic data represented as strings of nucleotide symbols over the alphabet $\mathcal{A} = \{A, C, G, T\}$. The main purpose of experiments in this section was to test the prediction accuracy of computational formula for $P^{\exists}(w)$ for variable-length Markov models learned using the IMM algorithm (Section 9.2) and the context algorithm (Section 9.1).

Markov models of DNA sequences have frequently been used in gene finding algorithms [30], where the interest was in finding strings of symbols instead of subsequences. Furthermore, we do not score the testing sequence using a trained Markov model, as in the work on gene discovery. Adapting the method of scoring the sequence for episode discovery would mean training a separate Markov model for every combination of window size and episode type. Note that in the episode framework we consider $\overline{\Omega^{\exists}}(n, w)$, which is a function of a Markov chain rather than a well defined structure (coding/non-coding region) of the sequence as in the gene discovery methods. Therefore, in our method for the reliable detection of significant episodes we use a Markov model only to compute the thresholds. Because of the sequential nature of Markov sources we considered only serial episodes. We do not test the thresholds directly in experiments by computing its value and simulating occurrences of significant episode as in Section 11.1.2 because we already showed that the accuracy of the thresholds is determined by prediction accuracy of the formula for $P^{\exists}(w)$. Therefore, we test the formulas for thresholds indirectly by focusing on the predictive performance of the formula for $P^{\exists}(w)$ for Markov models. Perhaps the most intriguing question is whether we can improve our detection method on DNA sequences in terms of accuracy by employing a k-order Markov model for $k > 0$ rather than the 0-order Markov model. As we will see in experiments the answer to this question is affirmative. To evaluate the performance of a model we used (11.1).

11.3.1 Full versus IMM for the same training and testing sequence

In this experiment we experimentally confirm the correctness of our theoretical results including Theorem 8.0.3 and the algorithm for $P^{\exists}(w)$ (Section 10.1.2). We expected to achieve a better prediction accuracy using the 5-order (full and IMM) comparing to the 0-order. To exclude a possibility of model over-fitting we used the the same sequence of *Haemophilius influenzea* as a training and testing source. We selected an example serial episode $e = [C, C, G, T]$ and for each k-order full

model for $k = 0, 1, 2, 3, 4, 5$ and 5-order IMM we compared $\overline{\Omega^{\exists}}(n, w)$ with $P^{\exists}(w)$ for $w = [5, 20]$ by computing the prediction error (11.1). The computed prediction errors are represented with a bar graph in Figure 11.4 that shows that: (1) the prediction error decreases monotonically starting from 1-order full model up; and (2) 5-order (full and IMM) gives the best prediction significantly outperforming the 0-order. This validates our theoretical and algorithmic results. 5-order IMM performs closely to 5-order full model since the training source was sufficiently large and the IMM did not use the lower order models.

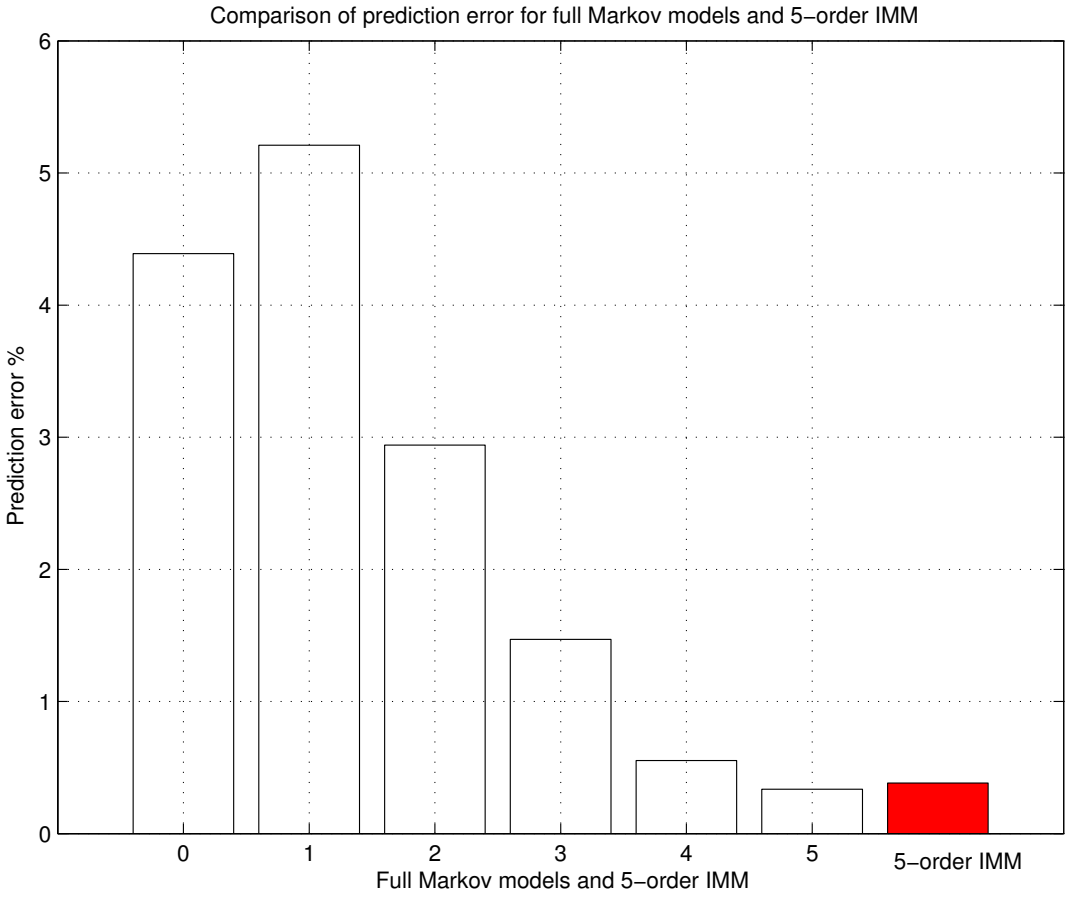

Figure 11.4.: Prediction error d between $\overline{\Omega^{\exists}}(n, w)$ and $P^{\exists}(w)$ for a serial episode using a k-order full Markov models for $k = 0, 1, 2, 3, 4, 5$ and 5-order IMM

11.3.2 Full versus IMM for training sequence different form testing sequence

In this experiment we compared the full 5-order with 5-order IMM. We used *Haemophilius influenzea* for computing the conditional probabilities and we tested the performance of both models on *Helicobacter pylori*. We expected the IMM to perform better than the full model because of its smoothing properties while we expected the full model to suffer from over-fitting. Also we did not expect a significant improvement in accuracy of IMM because the training set of size (1,830,025) was sufficiently large to find all context strings. We selected an example serial episode $e = [C, C, G, T]$ and for each k-order full model for $k = 0, 1, 2, 3, 4, 5$ and 5-order IMM we compared $\overline{\Omega^{\exists}}(n, w)$ with $P^{\exists}(w)$ for $w = [5, 20]$ by computing the prediction error (11.1). The results, shown as a bar graph in Figure 11.5, confirm our expectations and the IMM performed slightly better than 5-order full model. Also 1-order full turned out to be the winner probably because there is a difference in the structure of DNA of *Helicobacter pylori* and *Haemophilius influenzea* and the 1-order captured the necessary structure without over-fitting.

11.3.3 Frequent episodes

In this section we experiment on a DNA sequence of *Escherichia coli* from `ftp://ftp.ncbi.nlm.nih.gov/genomes/Bacteria/Escherichia_coli_K12/`. The purpose of this experiment was to demonstrate the applicability of the upper threshold and frequent episode mining for recognizing coding regions of the DNA. The idea of using a Markov model of coding and non-coding regions of DNA for gene identification is not new [35] but the novelty of our approach is to use frequent episodes mining instead of pattern matching and sequence scoring. A DNA sequence can be segmented into three functional parts: (1) protein-coding (gene) in the direct strand; (2) gene shadow (protein-coding in the complementary strand); and (3) non-coding [35]. Because the coding and non-coding regions exhibit different probabilistic characteristics, we considered these two regions independently. For this purpose we created two

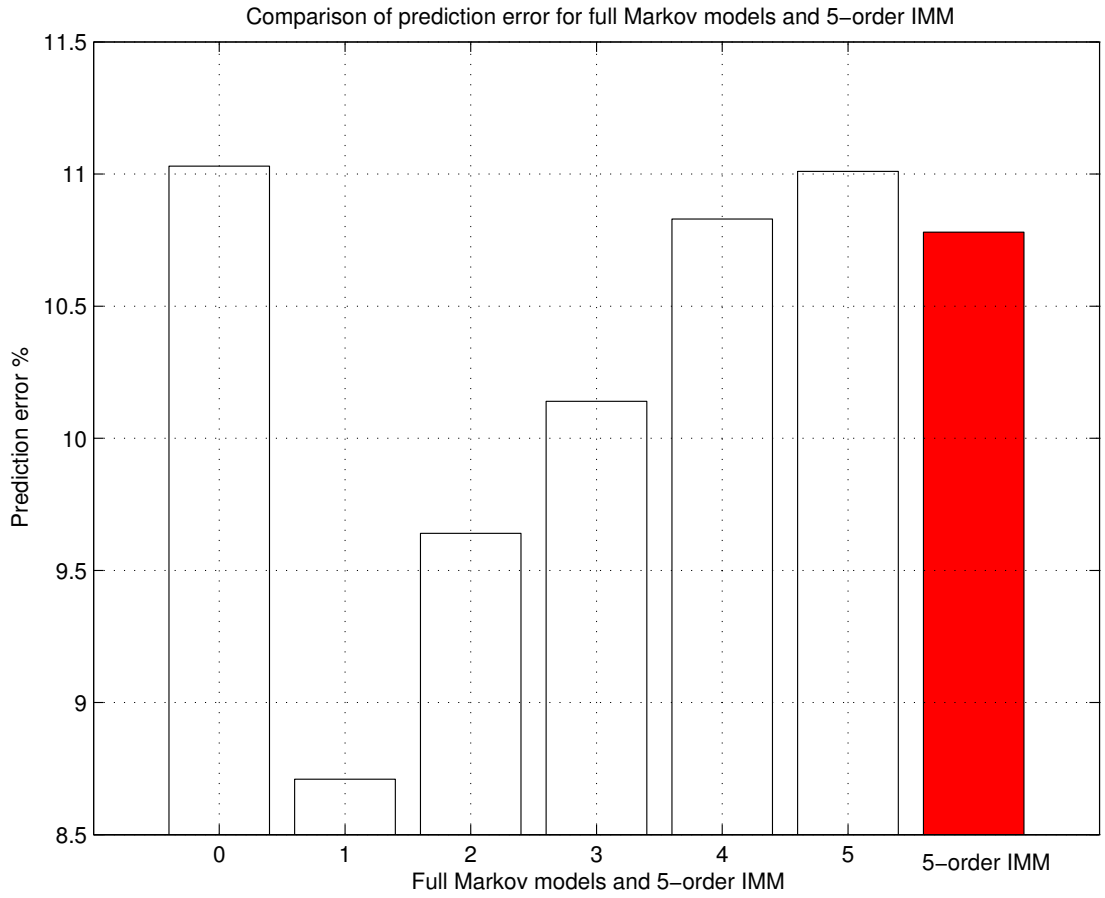

Figure 11.5.: Prediction error d between $\overline{\Omega^{\exists}}(n, w)$ and $P^{\exists}(w)$ for a serial episode using a k-order full Markov models for $k = 0, 1, 2, 3, 4, 5$ and 5-order IMM

sequences: (1) a coding sequence by merging protein coding and shadow regions; and (2) non-coding sequence by merging respective regions. We treated the non-coding sequence as a reference sequence and built a variable-length Markov model in order to recognize the coding sequence in terms of frequent significant episodes. To accomplish this we selected a class of serial episodes of length 6, the window size $w = 15$, and we discovered the most frequent episodes from this class in the coding sequence. Then for each of the 50 most frequent episodes we computed the upper threshold for the significance level $\beta(b) = 0.01$. As it turns out, all the 50 most frequent episodes,

discovered in the coding sequence, were significant with respect to the non-coding sequence. This clearly confirms that there is a significant difference in the structure between the coding sequence and non-coding sequence. We also ranked the 50 most frequent significant episodes with respect to significance. The results, presented in Figure 11.6, show that frequency of occurrence does not always monotonically determine the significance rank. The most significant of the most frequent 50 episodes in the coding sequence was episode $[G, C, G, C, G, C]$.

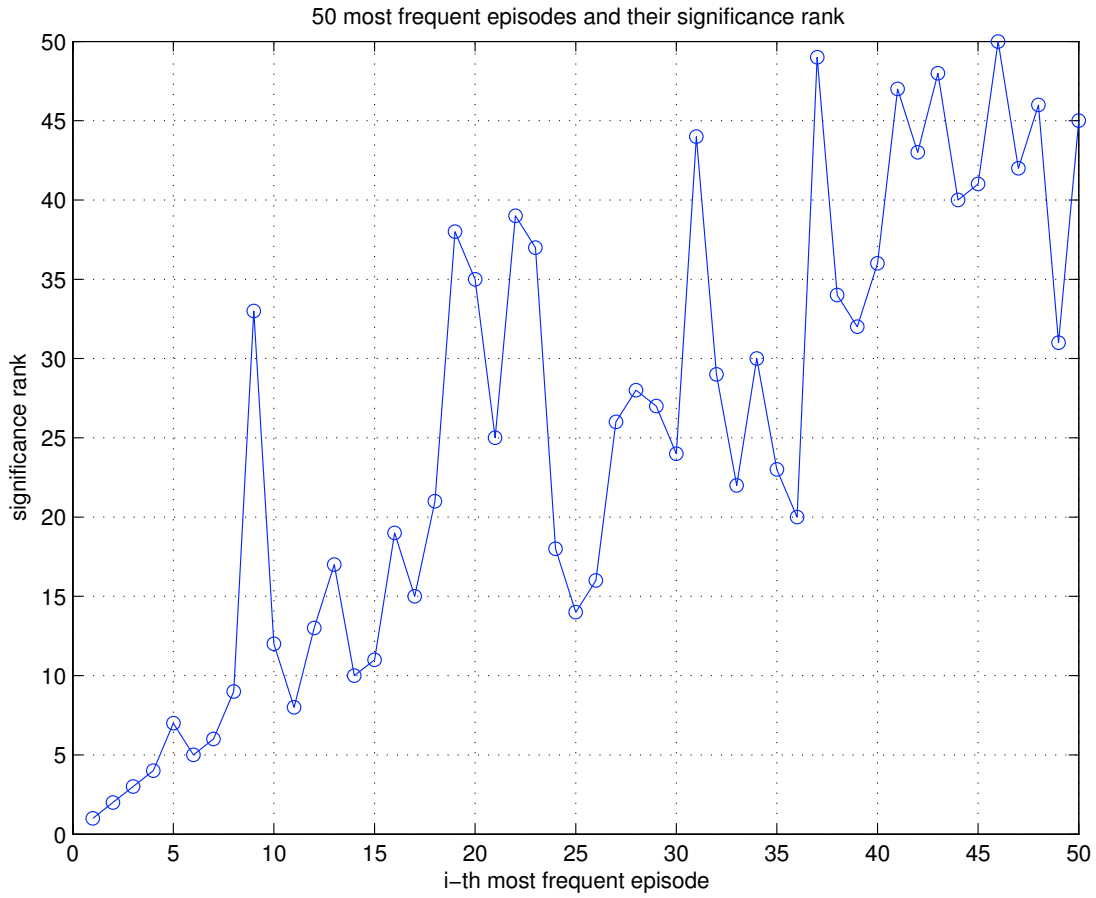

Figure 11.6.: 50 most frequent significant episodes and their significance rank

11.4 *Wal-Mart* transactions

In this section we use *Wal-Mart* data available to the Department of Computer Sciences, Purdue University. The database contains part of *Wal-Mart* sales data for the years 1999 and 2000 in 135 stores. We selected one of the stores, one category of items of cardinality 35 ($|\mathcal{A}| = 35$) and extracted records from table $Item\_Scan$, sorted by scan time. The choice of *Wal-Mart* data in experiments was motivated by the fact that the transaction data is an ideal real source for testing the formula for $P^{\exists}(w)$ for a parallel episode. Given the transaction data we also tested the formulas for $P^{\exists}(w)$ for sets of serial episodes and compared the results to $P^{\exists}(w)$ for a parallel episode. In Sections 11.4.1-11.4.3 we compare $\overline{\Omega^{\exists}}(n, w)$ with $P^{\exists}(w)$ for different values of w and using the 0-th order Markov model of the event sequence. Finally, in section 11.4.4 we give an example of a less frequent episode that is more significant than a more frequent episode. Whenever we referenced an alphabet symbol a_i we used the index i instead of its real name.

11.4.1 Parallel episode

We selected an example parallel episode $e = \{0, 4, 5, 6, 9, 10, 17\}$ and for $w \in [10, 180]$ we compare $\overline{\Omega^{\exists}}(n, w)$, discovered using Algorithm 4, with $P^{\exists}(w)$ computed using Theorem 10.2.2. The results are shown in Figure 11.7, that indicate an exceptionally close fit between $\overline{\Omega^{\exists}}(n, w)$ and $P^{\exists}(w)$ with $d = 2\%$. The results confirm our expectations that the 0-th order Markov model and a parallel episode models well sources as the *Wal-Mart* item scans where the source seems to generate events independently.

11.4.2 Set of two serial episodes

We selected a serial episode $e_1 = [0, 4, 5, 6, 9, 10, 17]$ and its permuted version $e_2 = [0, 6, 5, 4, 10, 9, 17]$. This case reflects a situation when a pattern of interest is

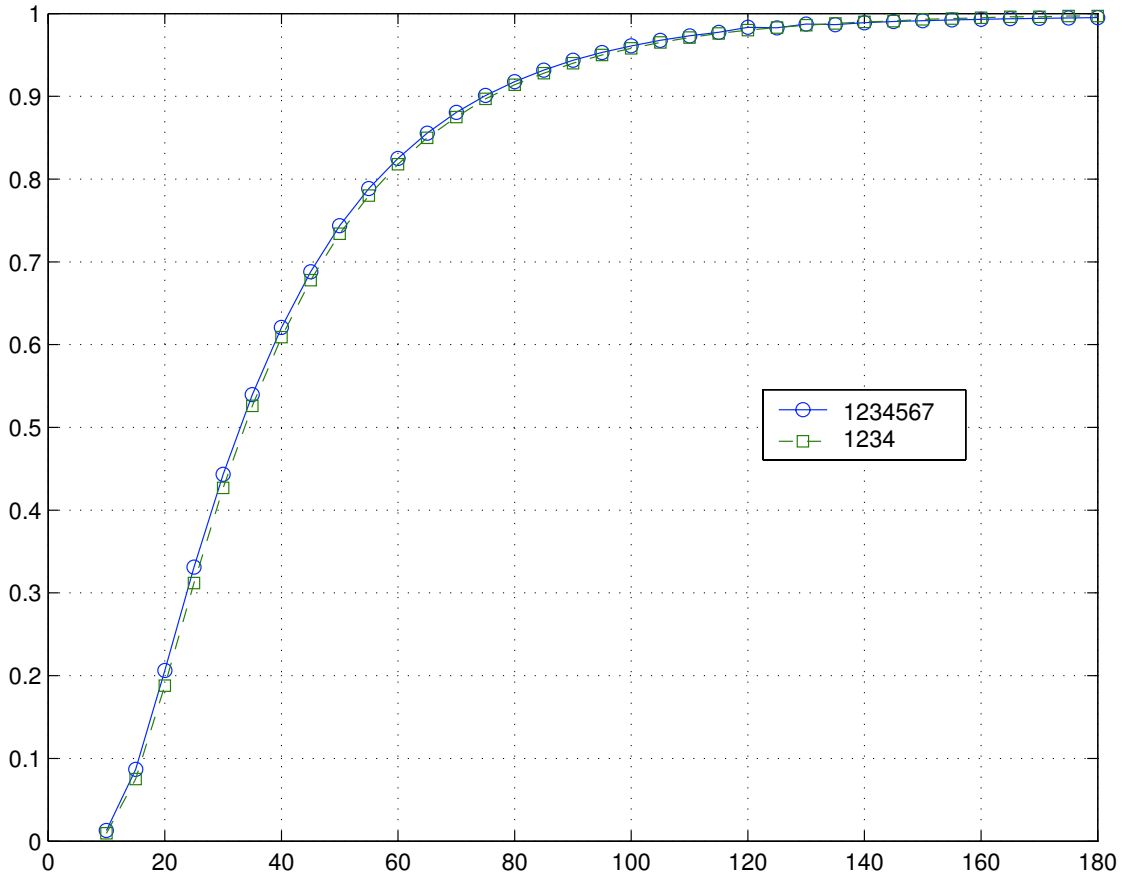

Figure 11.7.: $\overline{\Omega^{\exists}}(n, w)$ and $P^{\exists}(w)$ for a parallel episode, using *Wal-Mart* data

only partially restricted and the serial case is too restrictive but the parallel case too relaxed. For $w \in [10, 180]$ we compare $\overline{\Omega^{\exists}}(n, w)$, discovered using Algorithm 5, with $P^{\exists}(w)$ computed using Theorem 10.2.1. The results are shown in Figure 11.8, which indicate a very close fit between $\overline{\Omega^{\exists}}(n, w)$ and $P^{\exists}(w)$ with $d = 13\%$. Notice that since the reference sequence is modeled by a 0-th order Markov source any set of two episodes where e_2 is a permutation of e_1 will have the same $P^{\exists}(w)$. This property clearly shows why episode mining may produce redundant patterns. If an event sequence is almost memoryless then the algorithm for discovering the most frequent serial episodes from a given class of serial episodes will return the

most frequent serial episode and all its permutations. Clearly, this problem does not concern discovering frequent parallel episodes.

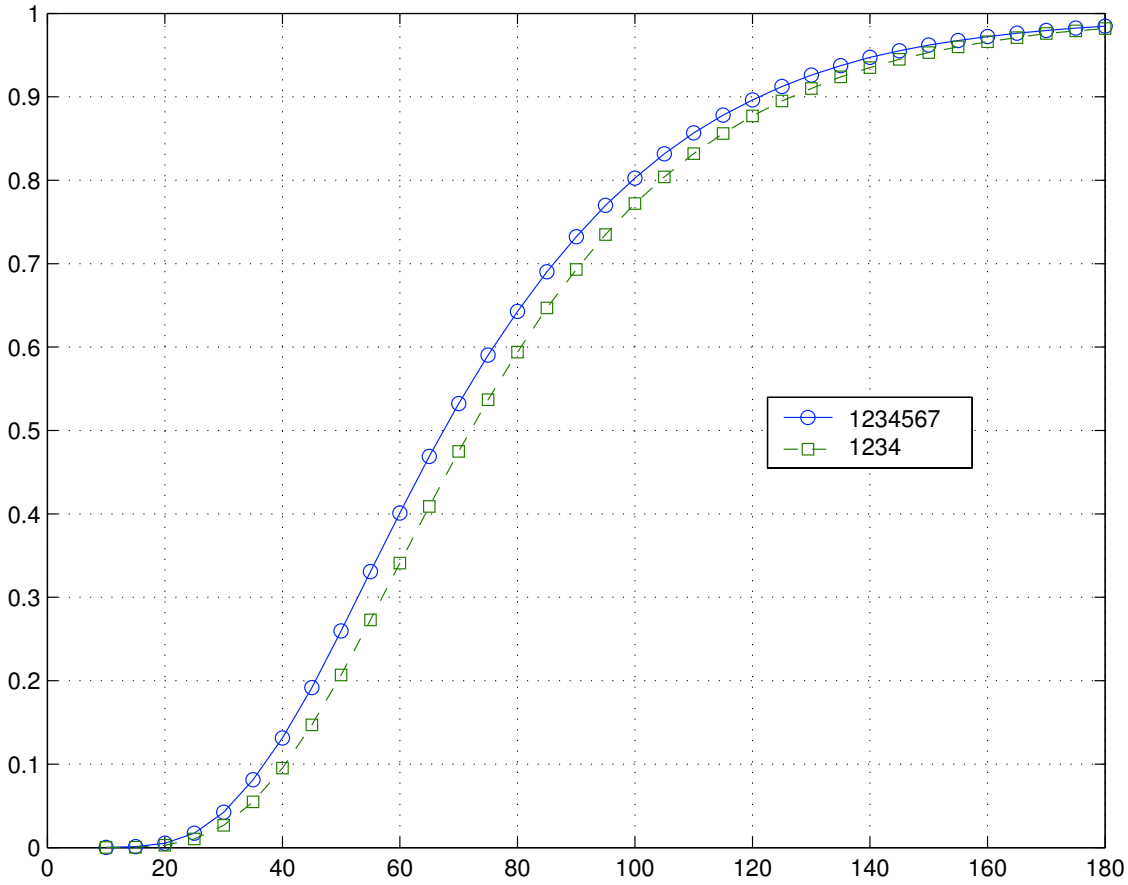

Figure 11.8.: $\overline{\Omega^{\exists}}(n, w)$ and $P^{\exists}(w)$ for a set $\mathcal{E} = \{e_1, e_2\}$ of serial episodes, using *Wal-Mart* data

11.4.3 Comparison of the three cases: parallel, two serial and one serial

In this experiment we investigate the relationship between $P^{\exists}(w)$ and $\overline{\Omega^{\exists}}(n, w)$ for an episode in the following three forms: parallel e_1, set of two serial $\{e_1, e_2\}$ (e_2 is a permutation of e_1) and serial e_1. For the first two cases we use the results

obtained in the previous experiments. To obtain the results for the serial episode for $w \in [10, 180]$ we ran the respective algorithms.

The resulting graphs for $P^{\exists}(w)$ are shown in Figures 11.9 and the corresponding graphs for $\overline{\Omega^{\exists}}(w, n)$ are shown in Figure 11.10. The figures confirms our expectations that the serial and parallel cases of an episode e establish the lower and upper bound on the probability of existence of e in window of size w.

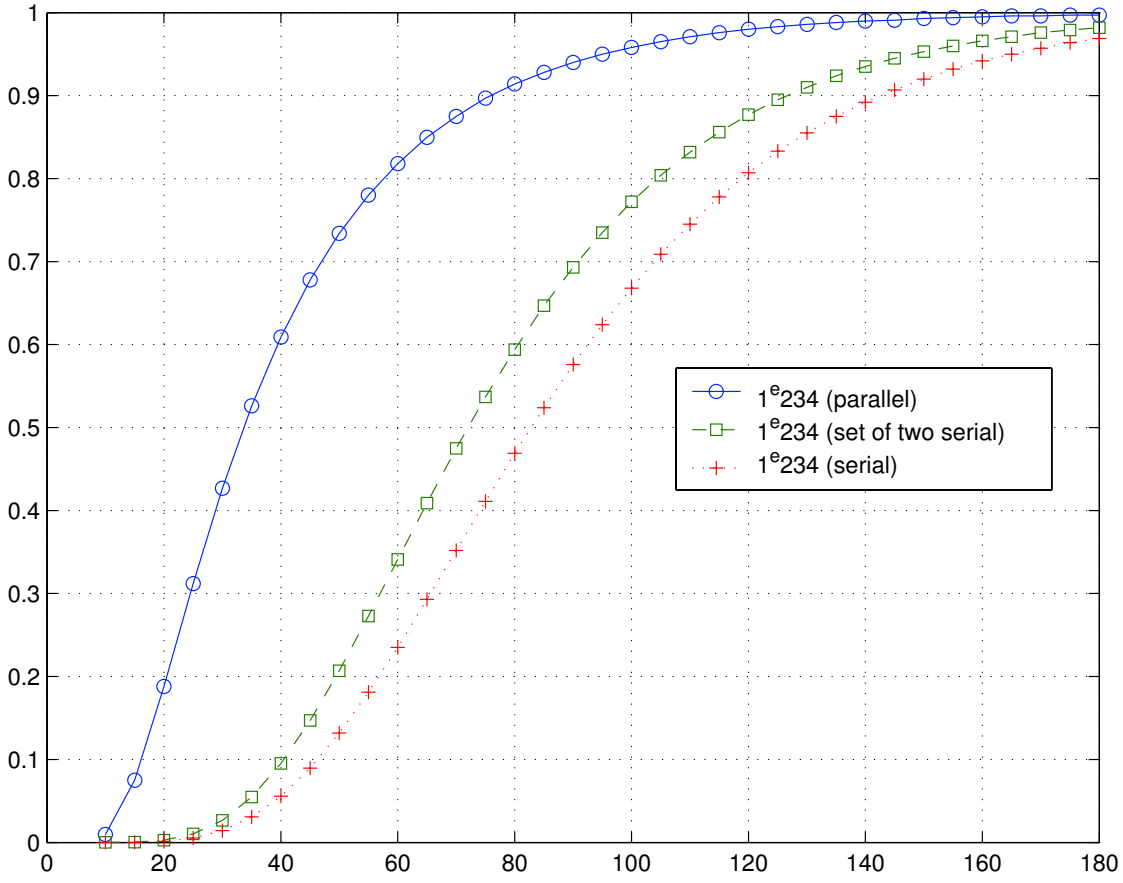

Figure 11.9.: $P^{\exists}(w)$ for three cases: parallel, set of two serial and serial, using *Wal-Mart* data

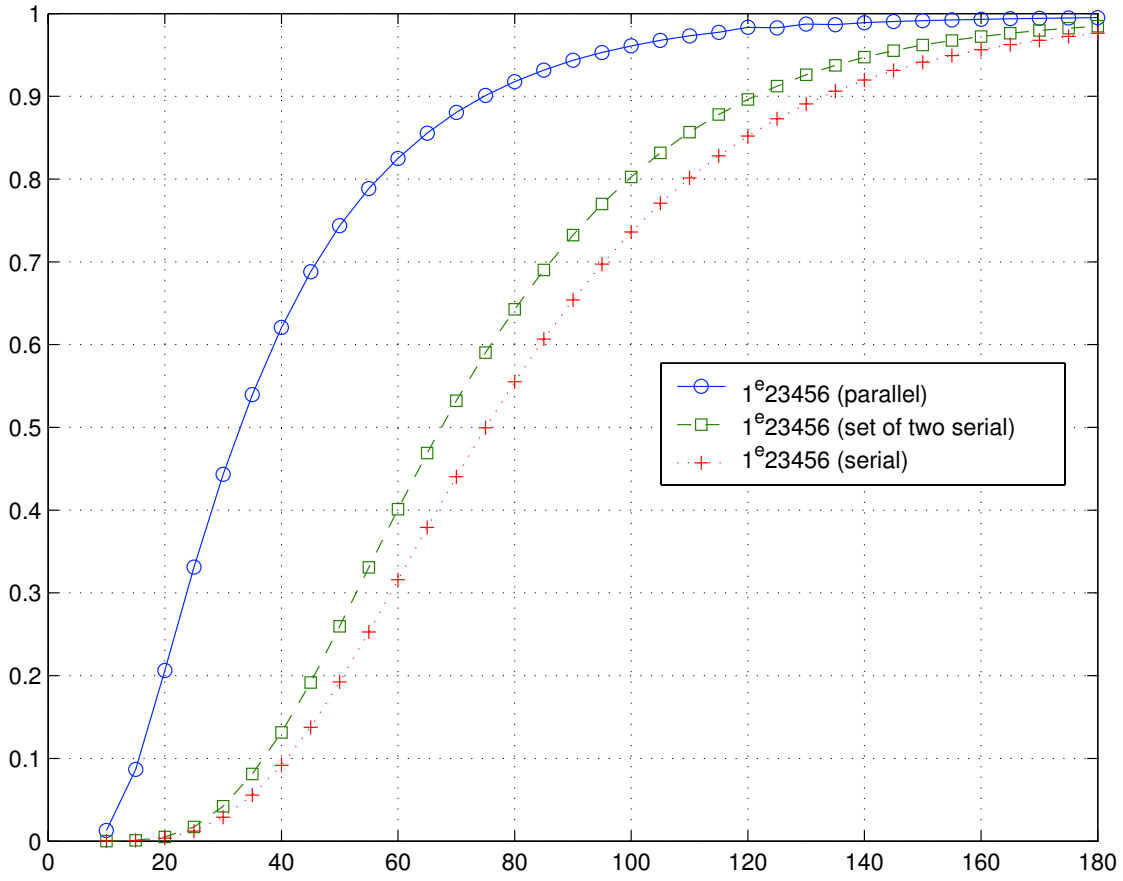

Figure 11.10.: $\overline{\Omega^{\exists}}(n, w)$ for three cases: parallel, set of two serial and serial, using *Wal-Mart* data

11.4.4 Frequency of an episode does not determine significance

In this experiment we show that, for a given w and episode length m, a lower-frequency episode can be more significant than a higher-frequency episode. We selected three example parallel episodes e_1, e_2, e_3 of length $m = 7$ each and for each of them we computed the pair $\tau(w, e_i)$, $P^{\exists}(w, e_i)$ for $w = 50$, $\beta(b) = 10^{-4}$ and the reference model was built for the whole sequence. The results, presented in Table 11.1, show that $\tau(w, e_1) > P^{\exists}(w, e_1) > \tau(w, e_2) > P^{\exists}(w, e_2) > \tau(w, e_3) > P^{\exists}(w, e_3)$,

Table 11.1: An example of a lower-frequency episode (e_3) that is more significant than higher-frequency episodes (e_2 and e_1)

Episode	w	$\overline{\Omega^{\exists}}(n,w)$	$P^{\exists}(w)$	$\tau_u(w)$
$e_1 = \{0,4,5,6,9,10,19\}$	50	4.64e-01	5.07e-02	5.19e-01
$e_2 = \{7,14,15,17,20,26,29\}$	50	3.49e-02	5.40e-02	5.93e-02
$e_3 = \{8,12,14,18,29,31,32\}$	50	2.26e-03	1.28e-03	2.12e-03

$\overline{\Omega^{\exists}}(n,w,e_3) < \overline{\Omega^{\exists}}(n,w,e_2) < \overline{\Omega^{\exists}}(n,w,e_1)$ and only e_3 is significant even though it is the least frequent among the episodes.

12 A SLIDING WINDOW AD HOC QUERY ANSWERING WITH PROBABILISTIC GUARANTEES

As a variation on the solved problem of identification of significant sets of episodes, we can also use the consequences of Theorem 8.0.3 to provide approximate answerers to sliding window episode queries referencing past data in data streams where we use $P^{\exists}(w)$ as an approximation of $\overline{\Omega^{\exists}}(w, n)$ and the probabilistic guaranties follow form the fact that $\overline{\Omega^{\exists}}(w, n)$ is normally distributed.

Thus, in this chapter we present a novel method for providing approximate answers, with probabilistic guarantees, to a class of ad hoc sliding window queries referencing past data in data streams. The class of queries comprises aggregate sliding window stream join queries computing the frequency of past windows that satisfy given join conditions between multiple streams. To represent the join conditions we introduce a concept of a two-dimensional episode (2D-episode) that defines a set of tuples satisfying given intra-stream and inter-stream constrains in the window. Using our method we can also compute iceberg queries computing the most frequent 2D-episodes from a class of 2D-episodes without the use of any external memory, i.e., "on-the-fly", whereas an exact answer to such a query requires a polynomial number of passes over the history of the data streams. As a summarization method, for each stream independently, we propose to use a space efficient variable-length Markov model learned while observing the data streams. Experiments, conducted on *Wal-Mart* transactions, demonstrate remarkable accuracy of our method, thereby confirming our theoretical analysis.

Systems designed to monitor continuous streams of data are becoming increasingly important in many applications including network monitoring, monitoring sensors installed in complex systems (traffic control, passenger aircrafts, power-plants), monitoring stock data reported from various exchanges. The purpose of such a multi-

stream monitoring application is to discover an unusual ("interesting") behavior in the environment being monitored in order to raise an alarm when it is appropriate. Given the streams of data from multiple sensors the monitoring systems employ a sliding window approach since the streams are unbounded in size and without a sliding window approach the systems would have to save to much state. Furthermore the window cannot be so long such that any subsequence would likely occur within it. Given such an observation window, comprising multiple streams, an unusual activity in the environment being monitored is manifested by a configuration (sequence) of events satisfying certain intra-stream and inter-stream constraints between the events in the configuration. In many applications given a certain configuration of events in the current window it is of interest to know whether some known or some unusually frequent/rare configurations of events occurred in the past, where the frequency of occurrence is defined as the number of windows containing the configuration out of a given number of shifts of the sliding window. However, since the streams are unbounded in size and events arrive at high rate on-line, the amount of storage required to compute an exact answer to a query referencing past events is also unbounded. Therefore, high-quality approximate answers to such queries, using appropriate main memory data reduction and summarization methods of past events, are of great importance. Furthermore, the monitoring system must be able to answer queries that are not known in advance. This type of queries is called *ad hoc queries* referencing past data in event streams. As an example of such a query, consider a situation where a router has been compromised and one needs to find out whether there were some signatures, that occurred within a window of a given size, on multiple network interfaces of the router.

Here, we propose a novel method for providing approximate answers with probabilistic guarantees, without the use of any external memory, to sliding window stream join queries computing the frequency of past windows that satisfy given join conditions between multiple streams. To represent the join conditions we introduce a concept of a two-dimensional episode (2D-episode) that defines a set of tuples (a

configuration of events) satisfying given intra-stream and inter-stream constraints in the window. Using our method we can also compute iceberg queries computing the most frequent 2D-episodes from a class of 2D-episodes without the use of any external memory, i.e., "on-the-fly", where an exact answer to such a query requires a polynomial number of passes over the history of the data streams. As a summarization method, for each stream independently, we propose to use a space efficient variable-length Markov model learned while observing the data streams. We provide approximate answers to the queries by computing the expected value of the frequency of windows containing an occurrence of a 2D-episode using the variable-length Markov models of the streams. The probabilistic guarantees follow from the fact that the frequency of windows containing a 2D-episode is normally distributed and one can set a narrow confidence interval on the difference between an observed frequency and the expected value of the frequency.

To the best of our knowledge, this is the first work on providing approximate answers to ad hoc sliding window queries using variable-length Markov model of stream data as a summarization method. Also this is the first work defining 2D-episode queries.

12.1 2D-episodes

To represent the join conditions between tuples in a window comprising multiple data streams we introduce a concept of a two-dimensional episode (2D-episode) as an extension of the concept of an episode, defined in [1], to multiple streams of data monitored simultaneously. Formally, a 2D-episode is a set of tuples satisfying the following types of time constraints:

1. *intra-stream constraints*
2. *inter-stream constraints*

where the constraints may concern given symbols and/or symbol variables. We use small letters to denote symbols (e.g., $S_1.a$) and capital letters to denote symbols

variables (e.g., $S_2.X$), where we use a notation $S_1.a$ and $S_2.X$ to denote a symbol a in stream S_1 and symbol variable X in stream S_2 respectively.

A 2D-episode can be represented as a graph where nodes represent tuples (symbols) and directed edges between nodes represent the time constraints. We distinguish the following types of nodes

1. *symbol nodes*: drawn as filled circles
2. *symbol variable nodes*: drawn as empty circles. They are used to represent equijoins between symbols in different streams

We also distinguish two types of edges: (1) directed edges: corresponding to intra-stream and inter-stream constraints and drawn as solid lines (2) undirected edges: corresponding to an equijoin and drawn as dashed lines.

In general, a 2D-episode can be viewed as an AND/OR graph, where we have the AND semantics unless otherwise stated. The OR semantics can be used to specify sets of 2D-episodes where by a join condition we mean a logical OR of occurrences of the members of the set.

12.1.1 Intra-stream, inter-stream constraints and an equijoin

This example illustrates a query computing the number of windows satisfying a join between three streams, where the join condition consists of an occurrence of a 2D-episode with intra-stream, inter-stream constraints and an equijoin. A graph representing the 2D-episode and a window containing an occurrence of the 2D-episode is shown in Figure 12.1.b and 12.1.a respectively.

The 2D-episode in Figure 12.1 has the following constraints: (1) equijoin: $S_1.X = S_2.X = S_3.X$ (2) intra-stream constraints: $S_1.X \rightarrow S_1.a \rightarrow S_1.b$, $S_2.X \rightarrow S_2.c$ (3) inter-stream constraints: $S_2.c \rightarrow S_3.d$, $S_2.c \rightarrow S_3.e$, where $S_1.a, S_1.b \in \mathcal{A}_1$, $S_2.c \in \mathcal{A}_2$, $S_3.d, S_3.e \in \mathcal{A}_3$ and $X \in \mathcal{A}_1 \cap \mathcal{A}_2 \cap \mathcal{A}_3$.

A SQL query corresponding to the 2D-episode in Figure 12.1 is shown in Figure 12.2. To count windows containing at least one occurrence of a specified 2D-episode

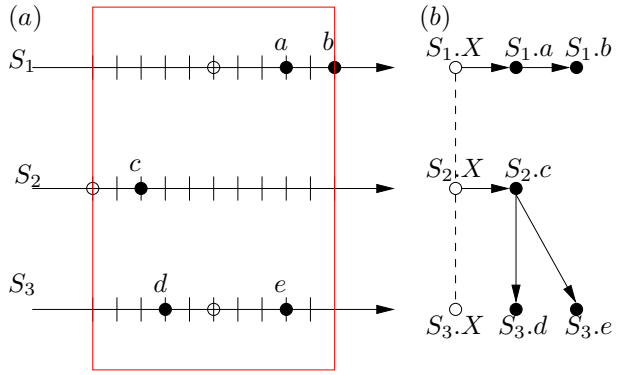

Figure 12.1.: A window join (a) corresponding to an occurrence of a 2D-episode (b)

we use an SQL aggregate operator WCOUNT(*) that can be converted to a standard SQL. To specify 2D-episodes we introduce WINDOW() construct that is similar to the WINDOW(A, B) construct introduced in [36]. WINDOW() operator can be used as follows:

1. WINDOW(S)=w: where S is a stream and w is a positive integer. Defines a window of size w in stream S

2. WINDOW=w: defines a window of size w over all streams

3. WINDOW(S)[i] = 'a' AND WINDOW(S)[j] = 'b' AND $i < j$: where i, j are integer variables. In this construct i and j are used to specify the intra-stream constraint $S.a \rightarrow S.b$. Similarly we can use the subscript variables to specify inter-stream constraints.

12.2 Approximate query answering

In order to provide approximate answers to the queries we use yet another aspect of Theorem 8.0.3. While in the reliable identification of significant episodes we used Theorem 8.0.3 to derive the formulas for the thresholds here we use $P^{\exists}(w)$ as the the approximation of $\overline{\Omega^{\exists}}(w, n)$ given a variable-length Markov model of the past

```
SELECT WCOUNT(*)
FROM S[1], S[2], S[3]
WHERE
WINDOW(S[1])[c1]=WINDOW(S[2])[c2]
AND
WINDOW(S[2])[c2]=WINDOW(S[3])[c3]
AND
WINDOW(S[1])[i1]='a' AND
WINDOW(S[1])[i2]='b' AND
WINDOW(S[2])[j1]='c' AND
WINDOW(S[3])[k1]='d' AND
WINDOW(S[3])[k2]='e' AND
c1 < i1 AND
c2 < j1 AND
i1 < i2 AND
j1 < k1 AND
j1 < k2
WINDOW=11
```

Figure 12.2.: Query corresponding to Figure 12.1

events, where the probabilistic guarantees follow directly from Theorem 8.0.3. Thus, building on the research on reliable identification of significant episodes we present a method for providing approximate answers to ad hoc queries counting the number of windows, that join multiple streams on the presence of a 2D-episode containing intra-stream and inter-stream constraints.

Consider two event streams S_1 and S_2 over alphabets $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively, monitored using a window of size w_1 over S_1 and a window of size w_2 over S_2. In our framework there is no requirement that the windows must be of the same size as long as they end at the same position in all streams and are shifted simultaneously

one event at a time. Thus, the case $w_1 = w_2$ is equivalent to a "single window" over both streams and throughout this chapter we assume the single window case unless otherwise specified. Assume our summarization method has built a pair consisting of a model with parameters (M_1, Θ_1) and (M_2, Θ_2) corresponding to each stream. Let $\Omega^{\exists}_{\bowtie}(n, w)$ denote the random variable corresponding to an observed number of past windows containing (joining on) a 2D-episode and let $\overline{\Omega^{\exists}_{\bowtie}}(n, w) = \frac{\Omega^{\exists}_{\bowtie}(n,w)}{n}$ be the relative frequency of the number of windows. Clearly $\overline{\Omega^{\exists}_{\bowtie}}(n, w)$ satisfies the CLT with $\mathbf{E}[\overline{\Omega^{\exists}_{\bowtie}}(n, w)] = P^{\exists}_{\bowtie}(w)$ and $\mathbf{Var}[\overline{\Omega^{\exists}_{\bowtie}}(n, w)] \approx P^{\exists}_{\bowtie}(w)(1 - P^{\exists}_{\bowtie}(w))$, where $P^{\exists}_{\bowtie}(w)$ is the probability that a window ending at a given position in the streams contains (joins on) a 2D-episode. We call this probability *probability of existence of a join* or shortly *probability of a join*. Then we approximate $\Omega^{\exists}_{\bowtie}(n, w)$ using its expected value $P^{\exists}_{\bowtie}(w)$, where the probabilistic guarantees can be easily written as follows

$$P\left(P^{\exists}_{\bowtie}(w) - b \cdot \sigma(w, n) < \overline{\Omega^{\exists}_{\bowtie}}(n, w) < P^{\exists}_{\bowtie}(w) + b \cdot \sigma(w, n)\right) = \Phi(b, b),$$

where $\sigma(w, n) = \sqrt{P^{\exists}_{\bowtie}(w)(1 - P^{\exists}_{\bowtie}(w))}$ and $\Phi(x_1, x_2)$ is the probability that a normal random variable assumes a value in the range $[x_1, x_2]$. As an example consider a confidence level of $\Phi(b, b) = 0.99$ for which $b = 2.58$.

The difficulty of efficiently computing $P^{\exists}_{\bowtie}(w)$ stems from the following facts: (1) a structure of the tree models (variable-length Markov models) of the streams can range from a full tree to the empty tree; (2) presence of dependencies between streams (e.g. streams generate symbols dependently if an occurrence of a given symbol in one stream depends on occurrences of symbols in other stream). To capture the dependencies one would need to maintain a Markov model of such a dependence while monitoring the streams; and (3) presence of inter-stream and/or equijoin constraints in the 2D-episode.

In the following sections, we present preliminary results by considering the case where the streams are modeled as 0-order Markov sources and generate symbols independently of each other. Given such a model of the streams we present efficient algorithms for computing the probability of a join $P^{\exists}_{\bowtie}(w)$ for queries containing inter-stream and intra-stream constraints.

12.3 Problem definition

The problem of computing an answer to an ad hoc sliding window query computing the frequency of a 2D-episode can be formally defined as follows.

Given:

- event streams $S_1, S_2, \ldots, S_k$ over finite alphabets $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ where the stream are synchronized meaning that the symbols are generated at the same rate in all streams
- we assume that each stream S_i for $1 \leq i \leq k$ is represented by a sequence of (variable-length Markov model, parameters) $(M_{i,1}, \Theta_{i,1}), \ldots (M_{i,n_i}, \Theta_{i,n_i})$ summarizing the history of the stream, where each pair (model, parameters) correspond to a change of probability distribution in the event stream
- sliding windows sizes $w_1, w_2, \ldots, w_{|k|}$ in each stream
- n the number of past tuples
- a 2D-episode
- a sliding window query computing the frequency of past windows satisfying a join condition given by the 2D-episode,

compute the frequency of a join $\overline{\Omega^{\exists}_{\bowtie}}(n, w)$ using a formula for $P^{\exists}_{\bowtie}(w)$ efficiently (in time polynomial in the window sizes and in time independent on the number of past tuples), with probabilistic guarantees and using only the main memory.

12.4 Solutions

In this section we provide solutions to: (1) queries computing the frequency of past windows containing at least one occurrence of a 2D-episode consisting of intra-stream (Section 12.4.1) and inter-stream (Section 12.4.2) constraints (2) iceberg queries computing the most frequent 2D-episodes from a class of 2D-episodes (Section

12.4.3). For each type of query we present an example SQL query using the syntax introduced in Section 12.1.1.

For clarity of the presentation we provide the solutions to the case of two streams S_1 and S_2 although the results easily generalize to a larger number of streams. Furthermore, we assume that the streams generate symbols independently of each other and they are modeled as 0-order Markov sources. Also, we consider a 2D-episode e, where by e_1 and e_2 we denote subepisodes of e corresponding to streams S_1 and S_2 respectively.

12.4.1 Intra-stream constraints case of $P^{\exists}_{\bowtie}(w)$

Since the streams generate symbols independently and there are no inter-stream constraints occurrences of windows containing subepisodes e_1 in S_1 and e_2 in S_2 of episode E are independent of each other. Therefore,

$$P^{\exists}_{\bowtie}(w) = P^{\exists}(w, e_1) \cdot P^{\exists}(w, e_2), \tag{12.1}$$

where $P^{\exists}(w, e_1)$ and $P^{\exists}(w, e_2)$ is the probability of existence of episode e_1 and e_2 respectively.

Computing $P^{\exists}_{\bowtie}(w)$ takes $O(w^2)$ time and $O(w \cdot \max(|e_1|, |e_2|))$ space using Algorithm 7.

Query: intra-stream constraints

This example illustrates a query computing the number of windows satisfying a join between three streams, where the join condition consists of an occurrence of a 2D-episode with intra-stream constraints. This query is an example of a query where the window sizes are different. A graph representing the 2D-episode and a window containing an occurrence of the 2D-episode is shown in Figure 12.3.b and 12.3.a respectively.

The 2D-episode in Figure 12.3.a has the following intra-stream constraints: $S_1.a \rightarrow S_1.b \rightarrow S_1.c$ corresponding to a serial episode, $S_2.d \rightarrow S_2.e$ corresponding to a serial episode and unconstrained symbols $S_3.g$, $S_3.h$, $S_3.f$ corresponding to a parallel episode.

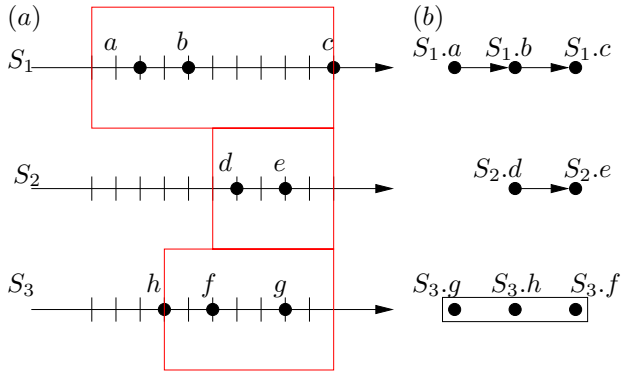

Figure 12.3.: A window join (a) corresponding to an occurrence of a 2D-episode containing only intra-stream constraints (b)

12.4.2 Inter-stream constraints case of $P^{\exists}_{\bowtie}(w)$

In the presence of inter-stream constraints only a subset of occurrences of e_1 and a subset of occurrences of e_2 satisfy the inter-stream constraints. Therefore, we cannot compute $P^{\exists}_{\bowtie}(w)$ for e as a product of corresponding probabilities of existence of e_1 and e_2.

Enumerative approach

Let $C^{\exists}_{inter}$ be a set of inter-stream constraints and let $\mathcal{W}^{\exists}{}_1(w)$ and $\mathcal{W}^{\exists}{}_2(w)$ be the sets of windows of sizes w containing episode e_1 and e_2 as a subsequence respectively. Therefore, we can express a computational formula for $P^{\exists}_{\bowtie}(w)$ as follows

$$P^{\exists}_{\bowtie}(w) = \sum_{C^{\exists}_{inter}} P(\mathcal{W}^{\exists}{}_1(w, e_1)[i]) \cdot P(\mathcal{W}^{\exists}{}_2(w, e_2)[j]) \tag{12.2}$$

where the $\mathcal{W}^{\exists}{}_1(w, e_1)[i]$, $\mathcal{W}^{\exists}{}_2(w, e_2)[j]$ are the i-the and j-th element in the respective set and $i \leq |(\mathcal{W}^{\exists}{}_1(w)|, j \leq |(\mathcal{W}^{\exists}{}_2(w)|$. Thus, we could enumerate elements from both sets that satisfy the inter-stream conditions $C^{\exists}_{inter}$, using Theorem 10.1.1. However, such a method would require $O\left(\binom{w}{|e_1|} \cdot \binom{w}{|e_2|}\right)$.

Merging streams

An alternative method is based on the observation that we can reduce the inter-stream constraints to intra-stream constraints by merging streams S_1, S_2 over alphabets $\mathcal{A}_1$, $\mathcal{A}_2$, and by creating one stream S over alphabet $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$. Probabilities of symbols in the new alphabet are computed as $P(S_1.i \times S_2.j) = P(S_1.i) \cdot P(S_2.j)$ where $S_1.i$ and $S_2.j$ is the i-th and j-th symbol in $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively.

We illustrate this idea in the following example.

Example 1.. Let $\mathcal{A}_1 = \{a, b\}$, $\mathcal{A}_2 = \{c, d\}$ and we have the following intra-stream and inter-stream constraints $C^{\exists}_{intra} = \{S_1.a \rightarrow S_1.b, S_2.a \rightarrow S_2.b\}$ and $C^{\exists}_{inter} = \{S_1.a \rightarrow S_2.c, S_2.d \rightarrow S_1.b\}$. Thus, we merge the alphabets and obtain alphabet $\mathcal{A} = \{A = ac, B = ad, C = bc, D = bd\}$. After reducing the inter-stream constraints to corresponding intra-stream constraints over alphabet $\mathcal{A}$ we obtain the following set of serial episodes $\mathcal{E} = \{A \rightarrow A \rightarrow B \rightarrow C, A \rightarrow A \rightarrow B \rightarrow D, A \rightarrow A \rightarrow D \rightarrow C, A \rightarrow A \rightarrow D \rightarrow D, A \rightarrow C \rightarrow B \rightarrow C, A \rightarrow C \rightarrow B \rightarrow D, A \rightarrow C \rightarrow D \rightarrow C, A \rightarrow C \rightarrow D \rightarrow D\}$. Figure 12.4 shows a 2D-episode with intra-stream and inter-stream constraints and the corresponding set of episodes $\mathcal{E}$ represented as a trie.

The advantage of this method versus the enumerative approach is that, given $\mathcal{E}$ we can compute $P^{\exists}_{\bowtie}(w)$ efficiently using Theorem 10.2.1 for sets of serial episodes. Thus, the probability of the join can be computed as follows

$$P^{\exists}_{\bowtie}(w) = P^{\exists}(w, \mathcal{E}). \tag{12.3}$$

Computing $P^{\exists}_{\bowtie}(w)$ takes $O(|\mathcal{E}| \cdot w^2)$ time and $O(w \cdot \max(|E_1|, |E_2|))$ space. However, this method does not scale well as the number of streams grows.

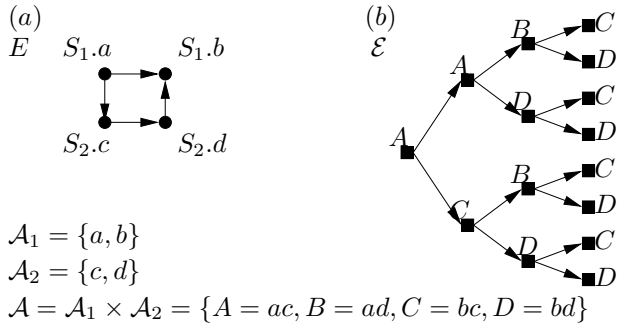

Figure 12.4.: Conversion from a 2D-episode E (a) to a set of serial episodes $\mathcal{E}$ represented as a trie (b)

Query: inter-stream and intra-stream constraints

This example illustrates a query computing the number of windows satisfying a join between three streams, where the join condition consists of an occurrence of a 2D-episode with intra-stream and inter-stream constraints. A graph representing the 2D-episode and a window containing an occurrence of the 2D-episode is shown in Figure 12.5.b and 12.5.a respectively.

The 2D-episode in Figure 12.5.b has the following constraints (1) intra-stream constraints: $S_1.a \rightarrow S_1.b$, $S_2.d \rightarrow S_2.e$ and $S_3.g \rightarrow S_3.h$ (2) inter-stream constraints: $S_3.f \rightarrow S_1.a$, $S_1.b \rightarrow S_2.d$ and $S_1.c \rightarrow S_3.g$.

12.4.3 Iceberg queries

We can compute iceberg queries computing the most frequent 2D-episodes from a class of 2D-episodes without the use of external memory. The idea of the standard Apriori-like algorithm for computing frequent episodes [1] is based on a level-wise search, where at each level the algorithm generates candidate frequent subepisodes of size k using frequent episodes of size $k-1$ and then performs a full database scan to compute frequencies of the candidates in order to select frequent episodes, i.e.

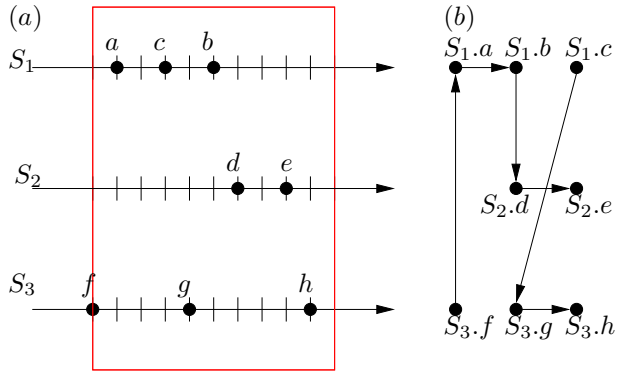

Figure 12.5.: A window join (a) corresponding to a 2D-episode containing intra-stream and inter-stream constraints (b)

episodes whose frequency is greater than a user defined minimum frequency, of size k. The search starts from subepisodes of size $k = 1$ and finishes if no more frequent episodes can be found.

The idea of our method is to compute frequencies of candidate frequent 2D-subepisodes using formulas for $P^{\exists}_{\bowtie}(w)$ instead of performing multiple database scans over the history of data streams. Thus, given a set of candidate frequent k-2D-episodes C_k we compute their frequencies, in order to obtain a set of frequent k-2D-episodes L_k. Let $\mathcal{F}_m$ be the set of frequent 2D-episodes in multiple streams over an alphabet $\mathcal{A}$, within a window of size w, satisfying a minimum frequency *min_support* and from a class of 2D-episodes $\mathcal{E}$ consisting of m events. We use symbol variables to specify classes. As an example consider a class $\mathcal{E} = \{S_1.X \rightarrow S_1.Y, S_2.X \rightarrow S_2.Y, S_1.X = S_2.X, S_1.Y = S_2.Y\}$ that describes 2D-episodes of size $m = 4$ such that the same sequence of two symbols (a serial episode) occurs in both streams simultaneously. We call such a special class the class of serial 2D-episodes of size 2.

Algorithm 8 computes a collection of the most frequent 2D-episodes, from a class of 2D-episodes, without the use of external memory in stream environment.

Algorithm 8: Algorithm for computing frequent 2D-episodes, from a class of 2D-episodes, without the use of external memory in stream environment

Input: m, w, $min\_support$, $\mathcal{E}$, $\mathcal{A}$
Output: $\mathcal{F}_m$
begin
  for $i = 1$; $i \leq |\mathcal{A}|$; $i = i + 1$ **do**
    if $P^{\exists}_{\bowtie}(w, a_i) > min\_support$ **then**
      $\mathcal{F}_1 = \mathcal{F}_1 \cup a_i$
  for $k = 2$; $k \leq m$ ***and*** $\mathcal{F}_{k-1} \neq \emptyset$; $k = k + 1$ **do**
    $\mathcal{C}_k = candidateGenerate(\mathcal{F}_{k-1}, min\_support)$
    foreach $c \in \mathcal{C}_k$ **do**
      if $P^{\exists}_{\bowtie}(w, c) > min\_support$ **then**
        $\mathcal{F}_k = \mathcal{F}_k \cup c$
end

12.5 Experiments

The purpose of our experiments was to test applicability of the analytical formulas for $P^{\exists}_{\bowtie}(w)$ for real sources. Therefore, we chose *Wal-Mart* transaction available to the Department of Computer Sciences, Purdue University. The database contains a part of *Wal-Mart* sales data for the years 1999 and 2000 in 135 stores. We selected two stores, one category of items of cardinality 35 ($|\mathcal{A}| = 35$) and for each store we extracted one stream of item scans sorted by the scan time corresponding to streams S_1 and S_2 respectively. Thus, the streams are independent of each other and there is no dependency between symbols in a given stream.

To evaluate the quality of an answer to a query we used (11.1), where in place $\overline{\Omega^{\exists}}(w, n)$ and $P^{\exists}(w)$ in that formula we used $\overline{\Omega^{\exists}_{\bowtie}}(n, w)$ and $P^{\exists}_{\bowtie}(w)$ respectively.

12.5.1 Intra-stream constraints

In this experiment we test the quality of an answer to a query computing the frequency of a join on a 2D-episode containing only intra-stream constraints. The 2D-episode is show in Figure 12.6.

To compute the error between observed frequencies and answers to the query we choose 20 values of the window size w and for each of them we computed (11.1) for $P^{\exists}_{\bowtie}(w)$, computed using (12.1), and $\overline{\Omega^{\exists}_{\bowtie}}(n, w)$. The results, presented in Figure 12.7, show that in the worst case an answer differs by 33% from the observed value. The average error is equal to 16%. Furthermore, the envelope of the bar graph in Figure 12.7 resembles the shape of the standard deviation curve in Figure 7.2. This validates our theoretical results and proves that our method provides good approximation of observed frequencies. In particular it validates (12.1).

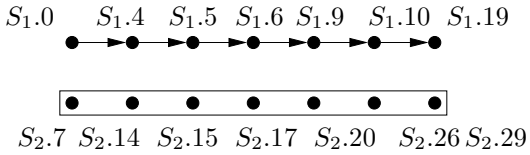

Figure 12.6.: 2D-episode containing intra-stream constraints

12.5.2 Intra-stream and Inter-stream constraints

In this experiment we test the quality of an answer to a query computing the frequency of a join on a 2D-episode containing both the intra-stream and inter-stream constraints. The episode is show in Figure 12.8.

To compute the error between observed frequencies and answers to the query we choose 20 values of the window size w and for each of them we computed (11.1) for $P^{\exists}_{\bowtie}(w)$, computed using (12.2), and $\overline{\Omega^{\exists}_{\bowtie}}(n, w)$. The results, presented in Figure 12.9, show that in the worst case an answer differs by 30% from the observed value. The

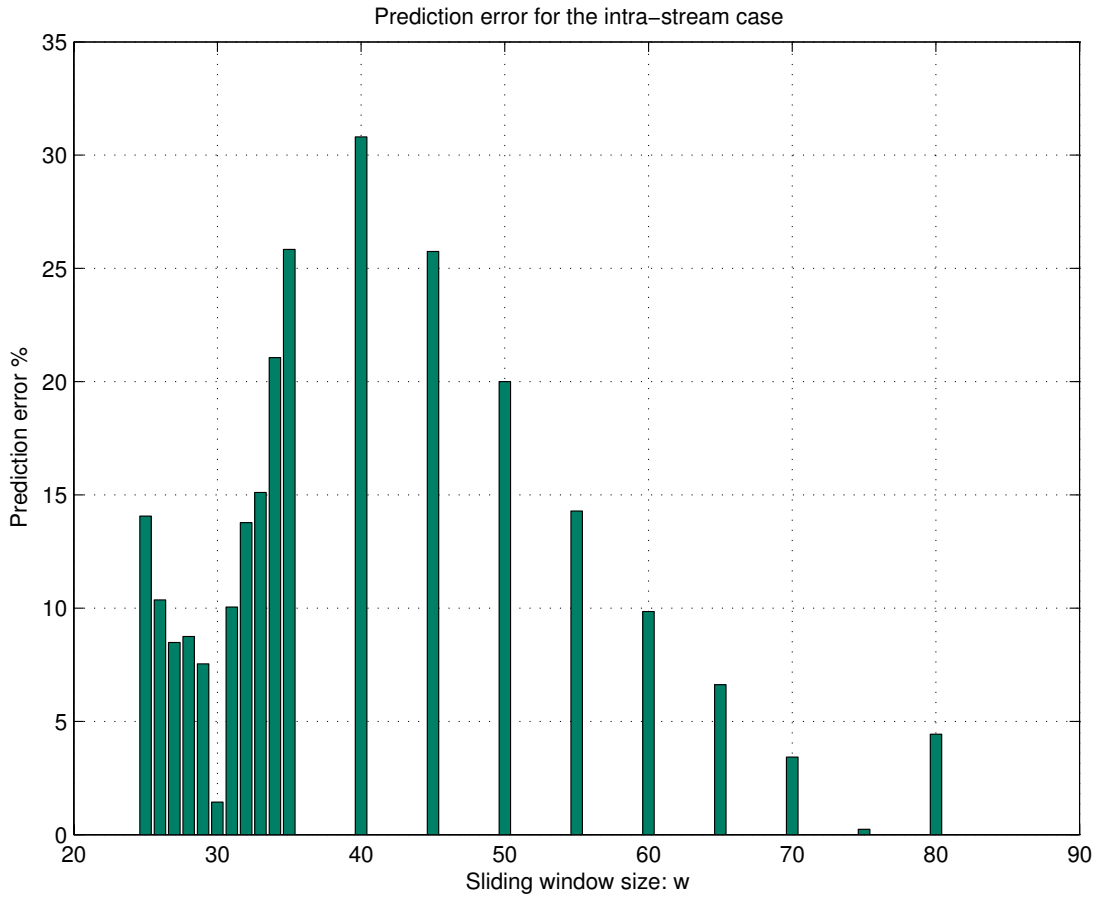

Figure 12.7.: Comparison between analytic frequencies and observed frequencies for 2D-episode from Figure 12.6

average error is equal to 13%. This validates our theoretical results and proves that our method provides quality answers. In particular it validates (12.2).

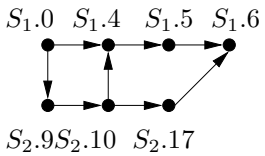

Figure 12.8.: 2D-episode containing intra-stream and inter-stream constraints

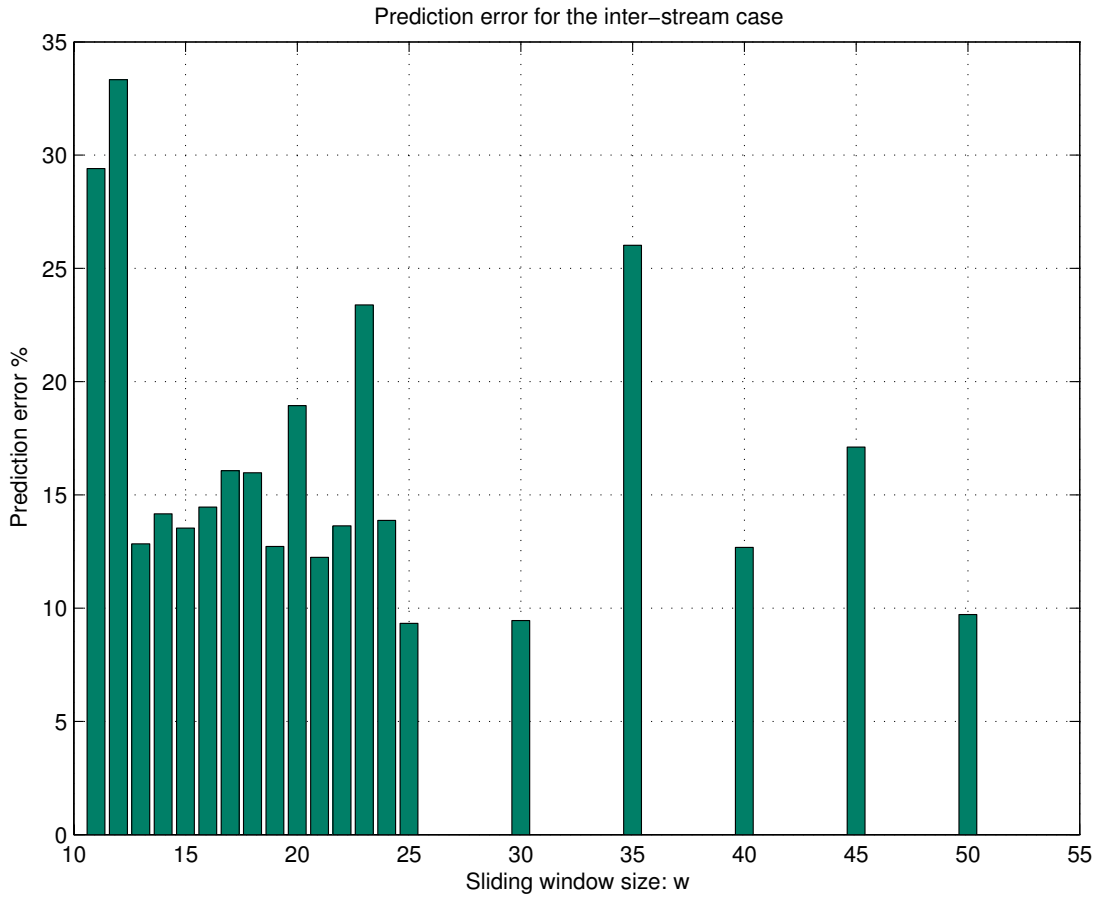

Figure 12.9.: Comparison between analytic frequencies and observed frequencies for 2D-episode from Figure 12.8

12.5.3 Iceberg queries

In this experiment we use our query mechanism to compute "on-the-fly" approximate answers to iceberg queries computing the most frequent 2D-episodes from a class of 2D-episodes. Although the algorithm works for any class of 2D-episode we focus in experiments on a class of frequent parallel 2D-episodes. By a 2D-parallel episode we mean the same unordered set of events (parallel episode) that occurs within the same window in all streams simultaneously. We compare the quality of approximate answers by computing 10 most frequent 2D-episodes from the class and

comparing them to actual 10 most frequent episodes discovered using the standard Apriori-like algorithm.

12.5.4 Frequent parallel 2D-episodes

In this experiment we compute an approximate answer to a query computing 10 most frequent 2D-parallel episodes of length 5 within a window of size 15. Figure 12.10 shows the class of parallel 2D-episodes of size 5. The results are presented in Figure 12.11, where the computed (approximate) i-the most frequent parallel 2D-episode is compared with its actual rank using bar graphs. The computed and actual rank is the same for the first most frequent parallel 2D-episode and differs by one for the next nine most frequent parallel 2D-episode. This clearly shows that we can obtain exact answers to a query computing the most frequent parallel 2D-episodes by using our approximate answers as long as the approximate frequencies are consistent with the true ordering of the actual frequencies.

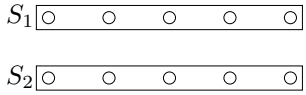

Figure 12.10.: The class of parallel 2D-episodes of length 5

12.6 Stream query processing

There has been an extensive research on stream query processing however there was a comparably little work on approximate ad hoc query processing. We start by defining types of queries considered in stream query processing.

With respect to the knowledge of future queries to be issued, we distinguish two types of queries: *predefined queries* and *ad hoc queries*. A predefined query is supplied to the data stream management system before any data has arrived, and

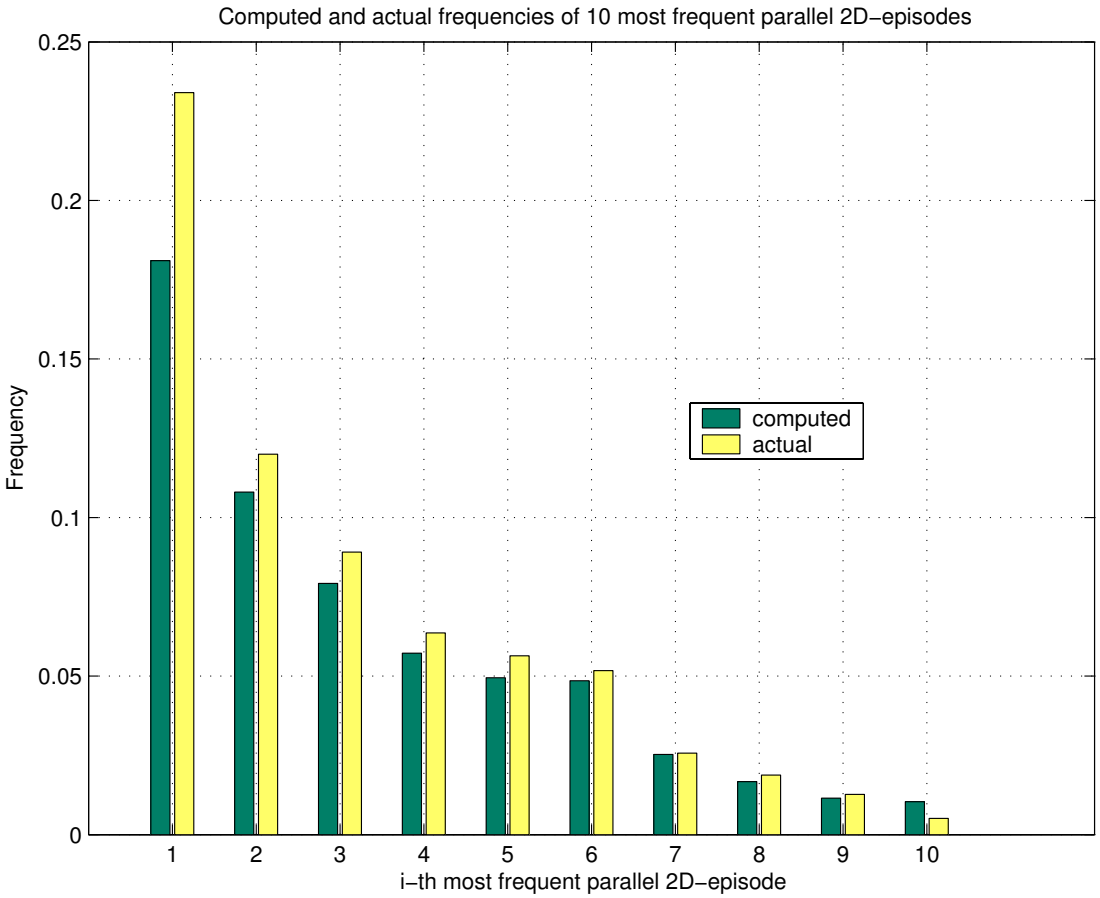

Figure 12.11.: Analytic 10 most frequent parallel 2D-episodes and their actual rank

the system can select an appropriate summarization method for that given query. An ad hoc query, on the other hand, is issued on-line after some number of events has already arrived. Ad hoc queries complicate the design of a data steam management system, because they are not known in advance and a summarization method for this type of queries must use the available main memory resources to store appropriately general information to be able to provide good approximate answers to a broad range of possible future queries.

With respect to the duration of time a query is processed, we distinguish *continuous queries* and *one-time queries*. One-time queries (a class that includes traditional

DBMS queries) are queries that are evaluated once over a point-in-time snapshot of the data stream. Continuous queries are evaluated continuously as events arrive.

The work on approximate query answering led to many general methods for data reduction and summarization. Example data reduction methods include:

- sliding window: reduces the amount of input data by evaluating a query over the sliding window instead over the entire history of the stream. Thus, the most recent elements in the window are called active data elements while the rest are called expired and they no longer contribute to query answers or statistics on the data set. In general sliding windows have two distinct purposes: either they are an important part of the query semantics or they are an approximation scheme to improve efficiency by considering only a limited past.

- random sampling: reduces the amount of input data by sampling

Data summarization methods are data structures that approximate stream data using main memory synopsis or sketches of the exact data. The difficulty of designing a summarization method in general stems from the fact that it must adapt to rapidly changing characteristics of the data streams. Example summarization methods, studied in the literature, include:

- sketches: summary of data stream using a set of parameters that allow to estimate the answer to certain queries [37, 38]

- histograms: capture the distribution of values in a data stream [39, 40]

- wavelets: wavelets with Haar bases were used to build the space efficient histograms on the underlying data distribution that estimate the fraction of records in the database that satisfy a query (selectivity estimation) [41]

In order to use the window paradigm we need an ordering of events within the window. There are two conventions used: *implicit timestamps* when the elements arrive sequentially in order and *explicit timestamps* when the elements have an explicit

timestamp attribute in which case the elements may arrive unordered. Also there are two methods of measuring the window size: *physical units* and *logical units*. Physical units measure a window size as the number of elements in the window. Logical units, on the other hand, measure a window size as a range of time covered by the window. An in-depth review of the state of the art in the research on stream query processing is presented in [42].

13 SUMMARY

In Chapters 8-10 we presented a solution to the problem of identification of significant sets of subsequence patterns. We provided computational formulas for the probability of existence $P^{\exists}(w)$ for an arbitrary set of serial episodes in an event sequence represented with a variable-length Markov model. Using the formulas we showed how to compute the significance thresholds and how to select the window size to guarantee that an occurrence of a set of subsequence patterns is meaningful. In particular for the case of a 0-th order event sequence we presented efficient algorithms for computing $P^{\exists}(w)$. The importance of the efficient algorithms for $P^{\exists}(w)$ for the 0-order Markov model stems from the following facts: (1) in many situations when a model of the reference sequence is not known in order to find significant episodes in a monitored sequence we will choose the 0-order model of the monitored sequence as a reference sequence; and (2) even for non-memoryless reference sequences, as the experimental results suggest, the formulas may provide a sufficient prediction accuracy to guarantee a sharp detection of the thresholds.

In Chapter 11 we presented experimental results that confirmed a remarkable accuracy of our formulas. In experiments conducted on non-memoryless sources: the English text and web access data we showed that, even for these cases, $P^{\exists}(w)$ for the 0-order model closely approximated $\overline{\Omega^{\exists}}(w, n)$, which demonstrated that the memoryless assumption did not limit the practical usefulness of the formulas. We also showed that the threshold mechanism indeed provided a sharp detection of significant episodes. In experiments conducted on *Wal-Mart* transactions we showed an exceptional closeness between $P^{\exists}(w)$ for a parallel episode for the 0-order model and $\overline{\Omega^{\exists}}(w, n)$. We also gave an example of a lower-frequency episode that was more significant than a higher-frequency episode. In experiments conducted on DNA sequences we showed that higher order Markov models outperformed the 0-order

model in terms of accuracy in predicting occurrences of episodes. The drawback of using higher order Markov models is the high computational cost of computing the thresholds. This could be overcome by using a combination of the 0-order model and a higher order model. In such a technique we could use the higher model for small values of w where the accuracy of the prediction would be crucial and the 0-order model for large w, where $P^{\exists}(w)$ converges to 1 for both models.

In Chapter 12 we presented a novel method for ad hoc sliding window query answering with probabilistic approximation guarantees for a class of queries computing the frequency of past windows containing a given join condition between multiple data streams. To specify the join conditions we introduced the concept of a 2D-episode. As a summarization technique we proposed a variable-length Markov model continually learned while monitoring multiple streams. We also showed that using our method we could answer iceberg queries without the use of external memory. In experiments, conducted on streams of *Wal-Mart* transactions, we confirmed that our approximate method provided quality answers for the real-life data. There are many extensions of the work on query answering. Examples include: (1) to design efficient algorithms for computing the probability of a join for the equijoin constraint and the case of dependent streams (2) to implement the change of distribution mechanism and characterize its usefulness in improving the quality of answers for fast varying stream data (3) to use formulas for the probability of a join for providing fast answers to continuous queries.

LIST OF REFERENCES

[1] H. Mannila, H. Toivonen, and A. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

[2] L. Boasson, P. Cegielski, I. Guessarian, and Y. Matiyasevich. Window-accumulated subsequence matching problem is linear. In *Proceedings of the Principles of Database Systems*, pages 327–336, 1999.

[3] P. Flajolet, Y. Guivarc'h, W. Szpankowski, and B. Vallée. Hidden pattern statistics. *Lecture Notes in Computer Science*, 2076:152–165, 2001.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference Very Large Data Bases*, pages 487–499, September 1994.

[5] M. Atallah, R. Gwadera, and W. Szpankowski. Detection of significant sets of episodes in event sequences. In *Fourth IEEE International Conference on Data Mining*, pages 67–74, October 2004.

[6] G. Das, R. Fleischer, L. Gąsieniec, D. Gunopulos, and J. Kärkkäinen. Episode matching. In *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching*, pages 12–27, 1997.

[7] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, 1998.

[8] K. Julisch and M. Dacier. Mining intrusion detection alarms for actionable knowledge. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining*, 2002.

[9] M. Qin and K. Hwang. Frequent episode rules for internet anomaly detection. In *Proceedings of the IEEE International Symposium on Network Computing and Applications*, pages 161–168, August 2004.

[10] J. Luo, S. Bridges, and R. Vaugham. Fuzzy frequent episodes for real-time intrusion detection. In *IEEE International Conference on Fuzzy Systems*, pages 368–371, December 2001.

[11] M. Klementtinen. *A Knowledge Discovery Methodology for Telecommunication Network Alarm Data*. PhD thesis, University of Helsinki, 1999.

[12] G. Ramstein, P. Bunelle, and Y. Jacques. Discovery of ambiguous patterns in sequences: Application to bioinformatics. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 581–586, 2000.

[13] R. Gwadera. A sliding window ad hoc query answering with probabilistic guarantees. Technical report, Purdue University, Department of Computer Science, 2005.

[14] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 265–276, May 1997.

[15] R. Gwadera, M. Atallah, and W. Szpankowski. Reliable detection of episodes in event sequences. In *Third IEEE International Conference on Data Mining*, pages 67–74, November 2003.

[16] P. Bilingsley. *Probability and Measure*. John Wiley and Sons, 1995.

[17] R. Bradley. Basic properties of strong mixing conditions: A survey and some open questions. *Probability Surveys*, 2:107–144, 2005.

[18] G. Jones. On the Markov chain central limit theorem. *Probability Surveys*, 1:299–320, 2004.

[19] J. Rissanen. A universal data compression system. *IEEE Transactions on Information Theory*, IT-29(5):656–664, 1983.

[20] A. Martin, G. Seroussi, and M. Weinberger. Linear time universal coding and time reversal of tree sources via FSM closure. *IEEE Transaction on Information Theory*, 50(7):1442–1468, 2004.

[21] J. Rissanen. Fast universal coding with context models. *IEEE Transactions on Information Theory*, 45(4):1065–1071, 1999.

[22] P. Bühlmann and A. Wyner. Variable length Markov chains. *Annals of Statistics*, 27:480–513, 1999.

[23] P. Bühlmann. Model selection for variable length Markov chains and tuning the context algorithm. *Annals of the Institute of Statistical Mathematics*, 52:287–315, 2000.

[24] M. Mächler and P. Bühlmann. Variable length Markov chains: Methodology, computing and software. *Journal of Computational and Graphical Statistics*, 13:435–455, 2004.

[25] F. Willems, Y. Shtarkov, and T. Tjalkens. The context-tree weighting method: Basic properties. *IEEE Transactions on Information Theory*, IT-41:653–664, 1995.

[26] S. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. Technical Report 10-98, Computer Science Group, Harvard University, 1998.

[27] S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3):400–401, 1987.

[28] F. Jelinek and R. Mercer. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of Workshop on Pattern Recognition in Practice*, pages 381–397, May 1980.

[29] E. Ristad and R. Thomas. Nonuniform Markov models. In *International Conference on Acoustics, Speech, and Signal Processing*, page 791, April 1997.

[30] S. Salzberg, A. Delcher, S. Kasif, and O. White. Microbial gene identification using interpolated Markov models. *Nucleic Acids Research*, 26(2), 1998.

[31] W. Li. DNA segmentation as a model selection process. In *RECOMB '01: Proceedings of the Fifth Annual International Conference on Computational Biology*, pages 204–210, 2001.

[32] W. Szpankowski, W. Ren, and L. Szpankowski. An optimal DNA segmentation based on the MDL principle. In *IEEE Computer Society Bioinformatics Conference*, pages 541–546, 2003.

[33] R. Gwadera, M. Atallah, and W. Szpankowski. Markov models for discovering significant episodes. In *SIAM International Conference on Data Mining*, pages 404–414, April 2005.

[34] W. Szpankowski. *Average Case Analysis of Algorithms on Sequences*. John Wiley, 2001.

[35] R. Azad and M. Borodovsky. Probabilistic methods of identifying genes in prokaryotic genomes: Connections to the HMM theory. *Briefings in Bioinformatics*, 5(2):118–130, 2004.

[36] M. Hammad, W. Aref, and A. Elmagarmid. Stream window join: Tracking moving objects in sensor-network databases. In *Proceedings of the 15th International Conference on Scientific and Statistical Database Management*, pages 75–84, July 2002.

[37] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 61–72, 2002.

[38] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 13–24, 2001.

[39] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, pages 471–475, 2001.

[40] H. Jagadish, V. Poosala, N. Koudas, K. Sevcik, S. Muthukrishnan, and T. Suel. Optimal histograms with quality guarantees. In *Proceedings of the 24th International Conference on Very Large Data Bases*, pages 275–286, 1998.

[41] Y. Matias, J. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 448–459, June 1998.

[42] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1–16, 2002.

VITA

Robert Gwadera received the M.S. degree in Electrical and Computer Engineering from the Department of Electronics, Telecommunication and Informatics, Technical University of Gdansk, Poland in 1995, where in 1996 he started a postgraduate program in Electronics, Telecommunication and Informatics and continued until December 1998. In January 1999 he joined the Department of Computer Science, Purdue University to pursue a Ph.D. in Computer Science. In 2003 he received the M.S. in Computer Sciences from Purdue University. In September 2005 he joined the Laboratory of Computer and Information Science at Helsinki University of Technology, Finland. He received the Ph.D. in December, 2005. His research interests are data mining, databases and security.