A Suffix Tree Based Prediction Scheme for Pervasive Computing Environments

Dimitrios Katsaros[1] and Yannis Manolopoulos[2]

[1] Department of Informatics, Aristotle University, Thessaloniki, 54124, Hellas
dimitris@skyblue.csd.auth.gr
http://skyblue.csd.auth.gr/~dimitris/
[2] Department of Informatics, Aristotle University, Thessaloniki, 54124, Hellas
manolopo@skyblue.csd.auth.gr
http://skyblue.csd.auth.gr/~manolopo/yannis.html

Abstract. Discrete sequence modeling and prediction is a fundamental goal and a challenge for location-aware computing. Mobile client's data request forecasting and location tracking in wireless cellular networks are characteristic application areas of sequence prediction in pervasive computing, where learning of sequential data could boost the underlying network's performance. Approaches inspired from information theory comprise ideal solutions to the above problems, because several overheads in the mobile computing paradigm can be attributed to the randomness or uncertainty in a mobile client's movement or data access. This article presents a new information-theoretic technique for discrete sequence prediction. It surveys the state-of-the-art solutions and provides a qualitative description of their strengths and weaknesses. Based on this analysis it proposes a new method, for which the preliminary experimental results exhibit its efficiency and robustness.

1 Introduction

The new class of computing, termed *location-aware computing*, which emerged due to the evolution of location sensing, wireless networking, and mobile computing presents unique challenges and requires high performance solutions to overcome the limitations of current wireless networks stemming from the scarcity of wireless resources. A location-aware computing system must be cognizant of its user's state, and must modify its behavior according to this information. A user's state usually consists of its physical location and information needs. If a human were given such context, s/he would make decisions in a proactive fashion, anticipating mobile client's user needs. In making these proactive decisions, the system must be able, among other things, to deduce future data requests and also to record and predict the positions of roaming clients.

The issues of data request prediction and location tracking/prediction, although diverse in nature, are simply different facets of the same coin; they can both be described in terms of a *discrete sequence prediction* problem formulation. From a qualitative point of view, this problem can be described as follows: *given a history of events, forecast the next one to come.*

P. Bozanis and E.N. Houstis (Eds.): PCI 2005, LNCS 3746, pp. 267–277, 2005.

Drastic solutions to the aforementioned problems have direct impact on the underlying wireless network performance. Accurate data request prediction results in effective data prefetching [18], which, combined with a caching mechanism [13], can reduce user-perceived latencies as well as server and network loads. Also, effective solutions to the mobility tracking problem can reduce the update and paging costs, freeing the network from excessive signaling traffic [4].

1.1 Motivation and Paper Contributions

The problem of discrete sequence prediction has received a lot of attention in various fields of computer science; prediction techniques have been developed in the context of Web/database prefetching [5,7,9,15,19], computational biology [1,3], mobile location tracking [4,8,17,21], machine learning [16,20]. All these techniques are related to some *lossless compression scheme*, due to the classical result about the duality between the lossless compression and the prediction of discrete sequences [11]. These algorithms can be classified in four families: a) the $\mathcal{LZ}78$ family (acronym for Lempel-Ziv-78) comprised by the works [4,7,8,15,17,21], b) the $\mathcal{PPM}$ family (Prediction by Partial Match) comprised by the works [5,7,9,19], c) the $\mathcal{PST}$ family (acronym for Probabilistic Suffix Tree) comprised by the works [1,3,16,20], and d) the $\mathcal{CTW}$ family (acronym for Context Tree Weighting) comprised by the works [24,23,22].

Each of these works has been developed in the context of a specific application field (computational biology, Web, etc) and reflects the characteristic of this field. The pervasive computing environment requires for the prediction method to posses some very specific features. The prediction method

- should be online and need not rely on time-consuming preprocessing of the available historical data in order to build a prediction model,
- should present low storage overhead,
- refrain from using administratively tunable or statistically estimated (from historical data) parameters, because they are not reliable and/or they are frequently changing.

The aforementioned prediction models do not posses all the above characteristics, as it will become evident from the discussion on the relevant research work (see Section 2). Table 1 summarizes the weaknesses of the relevant models with respect to the requirements described earlier.

Therefore our motivation stems from seeking for an online, self-tuning and with low storage requirements prediction model. Evidently, such a model should be supported by an appropriate data structure.

The present paper's purpose is to introduce the ideas of a novel prediction scheme and not to perform an exhaustive performance evaluation. In this context, it makes the following contributions. Firstly, it presents a classification of the state-of-the-art prediction methods into families and gives a qualitative comparison of their characteristics. It describes a new method for discrete sequence prediction, which meets the requirements set by the pervasive computing

Table 1. A qualitative comparison of discrete sequence prediction models

Prediction		Overheads		
Family	Model	Training	Parameterization	Storage
$\mathcal{LZ}78$	[7]	on-line	moderate	moderate
	[4]	on-line	moderate	moderate
	[21]	on-line	moderate	moderate
$\mathcal{PPM}$	[5]	off-line	heavy	large
	[7]	on-line	moderate	large
	[9]	off-line	heavy	large
	[19]	off-line	moderate	large
$\mathcal{PST}$	[1]	off-line	heavy	low
	[3]	off-line	heavy	low
	[16]	off-line	heavy	low
	[20]	off-line	heavy	low
$\mathcal{CTW}$	[24,23]	on-line	moderate	large
	[22]	on-line	moderate	large

environement. This method is based on the ideas described in [10]. Finally, it presents a preliminary performance evaluation of the proposed method to prove its effectiveness and robustness without delving into an exhaustive comparison with all the competing techniques.

2 Relevant Work on Discrete Sequence Prediction

All the predictors, we present in the sequel, are based on the assumption that the next event to come depends on a number of previous (seen in the past) events. The "size" of the past (i.e., number of preceding events) defines the "order" of the context or the *order of the predictor*. Ideally, we would like to have predictors that do not impose any constraint on the order, but such a constraint helps the current predictors to reduce the storage requirements of the underlying data structure, which supports their operation. Suppose that the following sequence has been seen $abcdefgabcdklmabcdexabcd$; we will show the prediction models constructed by each predictor.

2.1 The $\mathcal{PPM}$ Predictor

The most famous predictor is based on the $\mathcal{PPM}$ compression algorithm [6]. The algorithm requires an upper bound on the number of consecutive events it will model. Suppose that this bound is 3, then the maximum context that $\mathcal{PPM}$ can model consists of 3 events/symbols. The predictor is supported by a *trie* and its content is illustrated in Figure 1 for our sample sequence. The trie is constructed by sliding (symbol by symbol) a window of size equal to the maximum context upon the sequence, and recording the substring inside this window in the trie.

Apparently, the need of a predetermined maximum context size is a drawback and if the sequence of events contains dependencies of larger size, then they

cannot be modeled. The numbers beside each symbol record its frequency of occurrence with respect to its context. Apart from this basic $\mathcal{PPM}$ scheme several variants of it have appeared in the literature. The work in [9] presented some *selective* $\mathcal{PPM}$ models that prune some states of the predictor in case they do not appear very frequently. Similar in spirit is the work of [5]. For instance, a *frequency-pruned* $\mathcal{PPM}$ model [9] with frequency threshold equal to $\frac{1}{10}$ is illustrated in Figure 2.

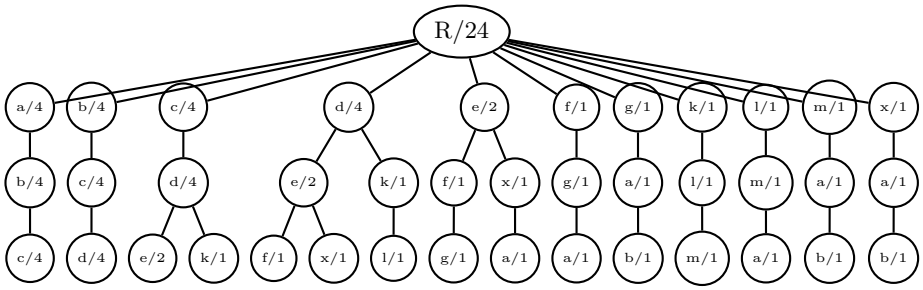

Fig. 1. The $\mathcal{PPM}$ predictor for the sequence *abcdefgabcdklmabcdexabcd*

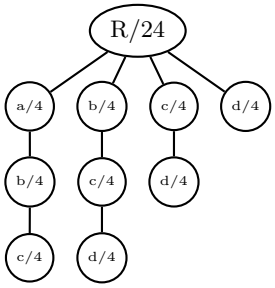

Fig. 2. The frequency-pruned $\mathcal{PPM}$ predictor with frequency threshold equal to $\frac{1}{10}$ for the sequence *abcdefgabcdklmabcdexabcd*

2.2 The $\mathcal{LZ}78$ Predictor

Another popular predictor is the $\mathcal{LZ}78$ predictor [7], which parses the input sequence into distinct substrings, such that, for all substrings, the prefix of each substring (i.e., all characters but the last one) is equal to some substring already encounter and stored into the *trie* that supports the $\mathcal{LZ}78$ predictor. Therefore the sample sequence will be parsed into the following substrings: $a, b, c, d, e, f, g, ab, cd, k, l, m, abc, de, x, abcd$. The contents of the corresponding trie are illustrated in Figure 3.

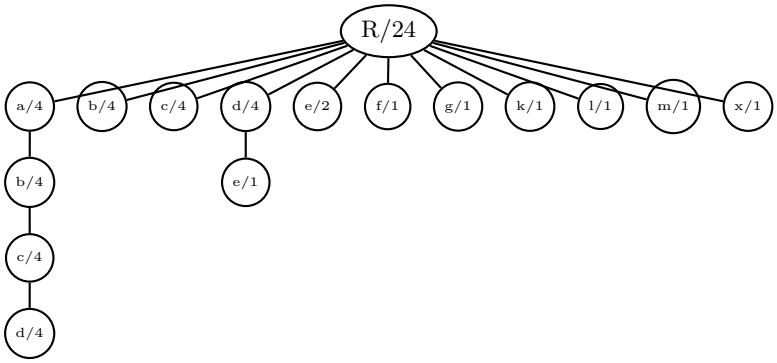

Fig. 3. The $\mathcal{LZ}78$ predictor for the sequence $abcdefgabcdklmabcdexabcd$

The drawback of this predictor is that the "decorrelation" process it uses constructs patterns only if it see them at least twice. For instance, only after seeing three times the ab pattern it is able to predict that with high probability it will be followed by a c character. This has a direct impact on the confidence it pays to some patterns. For instance, the confidence of the pattern de is only $\frac{1}{4}$, instead of the value of $\frac{2}{4}$ assigned by $\mathcal{PPM}$. Although, for very large training sequences this problem will not affect the prediction quality significantly, for short training sequences the $\mathcal{LZ}78$ will yield sparse and noisy statistics [2]. Compared with the original $\mathcal{PPM}$ predictor, the branches of the $\mathcal{LZ}78$ predictor are not of the same length, which in general is a desirable characteristic of the predictor, and also it does not use any predetermined parameters for the maximum length of the context it models.

An enhancement to the original $\mathcal{LZ}78$ method is proposed in [4,8,21], where the trie is augmented with *every prefix of every suffix* of a newly recorded pattern, but this enhancement is still not enough to compensate for the drawbacks mentioned above.

2.3 The $\mathcal{PST}$ Predictor

The $\mathcal{PST}$ predictor is very similar to $\mathcal{PPM}$ but it attempts to construct the best possible prediction model given a specific maximum context length. For this purpose it maintain in total five user defined parameters (including the context length), whose tuning is quite difficult and application-specific. Specifically, it maintains a) the threshold P_{min}, which defines the minimum occurence probability of a subsequence in order to be included into the $\mathcal{PST}$ tree, i.e., no symbol occurring with probability less than P_{min} can be encoded into the $\mathcal{PST}$, b) r, which is a simple measure of the difference between the prediction of the candidate (to be included into the $\mathcal{PST}$ tree) and its direct father node, c) γ_{min} the smoothing factor, d) a, a parameter that together with the smooting probability defines the significance threshold for a conditional appearance of a symbol.

The $\mathcal{PST}$ for our sample sequence, assuming that the maximum represented sequence length is $L = 3$, $P_{min} = \frac{1}{12}$, $r = 1$, $\gamma_{min} = 0.0001$, $a = 0$, is shown in Figure 4. This value for the threshold P_{min} means that only substrings consisting of symbols appearing at least twice in the original sequence will be encoded into the $\mathcal{PST}$ tree. Note that we do not draw the probability vector corresponding to each node.

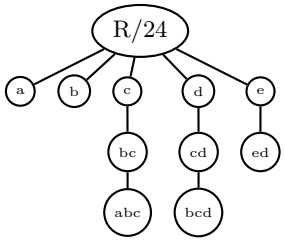

Fig. 4. The $\mathcal{PST}$ predictor for the sequence *abcdefgabcdklmabcdexabcd*

It should be noted that $\mathcal{PST}$ differ from the classical suffix tree [12], which contains all the suffixes of a given string. The two data structures have the following relation: the skeleton (nodes, edges and labels) of a $\mathcal{PST}$ for a given input string is simply a subtree of the suffix tree associated with the *reverse* of that string.

2.4 The $\mathcal{CTW}$ Predictor

This prediction technique was initially introduced for binary alphabets [24,23] and thus is not very popular in the applications's domain. Some efforts to extend it for multi-symbol alphabets [22] suffer from exponentially-growing computational cost and, in addition (as reported in chap. 4 of [22]), they perform poorly. Thus, we do not examine it further here.

3 The $\mathcal{STP}$ Prediction Method

Before proceeding to describe the proposed prediction algorithm, we provide a formal definition of the discrete sequence prediction problem.

Definition 1 (Discrete Sequence Prediction problem). *Let us assume that a sequence $s_1^n = s_1, s_2, \ldots, s_n$ of events is given. Each symbol of s_1^n belongs to a finite alphabet. Given this sequence, the goal is to predict the next event to come, i.e., the $\hat{s}_{n+1}$.*

The algorithm works as follows: a) It finds the largest suffix of s_1^n, call it ss_i^n, whose copy appears somewhere inside s_1^n. Then, it takes a suffix of ss_i^n (the

Table 2. The $\mathcal{STP}$ algorithm

Algorithm *STP*
// Current sequence is $s_1^n = s_1, s_2, \ldots, s_n$.
// Predict the symbol after s_n.
begin
STEP 1.
Find the *largest* suffix of s_1^n, whose copy appears somewhere inside s_1^n.
Let this suffix be named ss_1^l. Its length is l and starts at
the position i in s_1^n, i.e.,
$ss_1^l = (s_{n-l+1}, s_{n-l+2}, \ldots, s_n) = (s_{n-i-l+1}, s_{n-i-l+2}, \ldots, s_{n-i})$.
STEP 2.
Take a suffix of ss_1^l of length k with $k = \lceil \alpha * l \rceil$, where α is a parameter.
Let this suffix be named sss_1^k, where $sss_1^k = (s_{n-k+1}, s_{n-k+2}, \ldots, s_n)$.
Suppose that ss_1^l appears m times inside s_1^n.
Each such occurence defines a *marker* and the m positions after
each market are called *marked positions*.
STEP 3.
The predicted symbol is the symbol that appears
the most times in the marked positions.
(In case of ties, the prediction consists of multiple symbols.)
end

length of this suffix is a parameter of the algorithm) and locates its appearances inside s_1^n. The symbols that appear after the appearances of it are the candidate predictions of the algorithm. The final outcome of the prediction algorithms is the symbol which appears most times. In pseudocode language, this is expressed as follows:

To explain how $\mathcal{STP}$ algorithm works, we present a simple example in the sequel.

Example 1. Suppose that the sequence of symbols seen so far is the following: $s_1^{24} = abcdefgabcdklmabcdexabcd$. The largest suffix of s_1^{24} which appears somewhere in s_1^{24} is the $ss_1^4 = abcd$. Let $\alpha = 0.5$. Then $sss_1 2 = cd$. The appearances of cd inside s_1^{24} are located at the positions $3, 10, 17, 23$. Therefore, the marked positions are the $5, 12, 19, 25$. Obviously the last one is not null, since it "contains" the symbol we want to predict. In the general case, all marked positions will contain some valid symbol. Thus, the sequence of candidate predicted symbols is e, k, e. Since the symbol that appears most of the times in this sequence is the e, the output of the $\mathcal{STP}$ algorithm, i.e., the predicted symbol at this stage, is e.

Theorem 1. *The $\mathcal{PST}$ algorithm is generalization of both $\mathcal{PPM}$ and $\mathcal{LZ}78$ with respect to the patterns it can discover.*

Proof. For a proof see [14].

Implementation Details. The implementation of the algorithm requires an appropriate data structure to support its basic operations, which are the following:

a) determination of the maximal suffix (at step 1) and, b) substring matching (at steps 1 and 2). These two operations can be "optimally" supported by a *suffix tree* [12]. The suffix tree of a string $x_1, x_2, \ldots, x_n$ is a trie built from all suffixes of $x_1, x_2, \ldots, x_n\$$, where *mathdollar* is a special symbol not belonging to the alphabet. External nodes of a suffix tree contain information about the suffix positions in the original string and the substring itself that leads to that node (or a pair of indexes to the original string, in order to keep the storage requirement linear in the string length). It is a well known result that the suffix tree can be built in linear (optimal) time (in the string length), and can support substring finding in this string also in linear (optimal) time (in the length of the substring). Therefore, the substring searching operation of our algorithm can be optimally be implemented. As for the maximal suffix determination operation, if we keep pointers to those external nodes that contain suffixes ending with the $ symbol (since on of them will be the longest suffix we are looking for), then we can very efficiently support this operation, as well.

From the above discussion, we conclude the following: a) the $\mathcal{STP}$ algorithm is online; it needs no training or preprocessing of the historical data, b) the storage overhead of the algorithm is low, since it is implemented upon the suffix tree and finally, c) it has only one tunable parameter, α, which fine-tunes the algorithm's accuracy. Therefore, it meets all the requirements we set in Subsection 1.1 for the features of a good predictor for pervasive environments.

4 Performance Evaluation of $\mathcal{STP}$

We conducted some preliminary performance evaluation tests in order to examine the prediction capabilities of the $\mathcal{STP}$ method. At this stage we are not interested in its comparison with other competing algorithms. We simply aimed at examining its prediction accuracy and the impact of the α parameter on its performance. We are currently implementing all the competing approaches to perform an exhaustive comparison. At this paragraph we will present only one experiment with real data as proof of concept of the algorithm and its ability to carry out prediction.

We examined the prediction performance of the algorithm using a real web server trace, namely the *ClarkNet*, available from the site http://ita.ee.lbl.gov/html/traces.html. We used both weeks of requests and we cleansed the log (e.g., by removing CGI scripts, staled requests, etc). The user session time was set to 6 hours. As performance measures we used the prediction *precision* and *overhead*, which are defined as follows:

Definition 2. *The ratio of symbols returned by the predictor that indeed match with the next event/symbol in the sequence, divided by the total number of symbols return by the predictor defines the prediction* precision.

Definition 3. *The total number of symbols return by the predictor divided by the total number of events/symbols of the sequence defines the prediction* overhead.

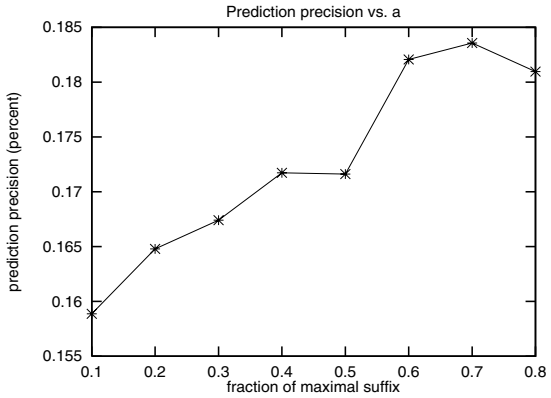

Fig. 5. Performance of the $\mathcal{STP}$ method. Prediction precision.

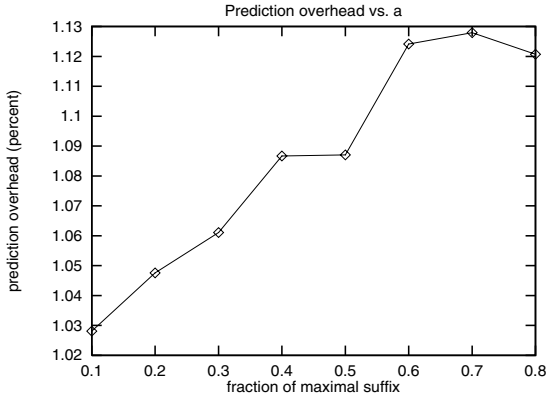

Fig. 6. Performance of the $\mathcal{STP}$ method. Prediction overhead.

The results are illustrated in Figures 5 and 6. Apparently, larger values for α increase the precision of the algorithm, since the predictor makes use of longer, and thus of more selective, context.

5 Future Work

This paper presents only the basic idea of the $\mathcal{STP}$ algorithm. Currently, we elaborate on it by developing some truncated versions of it, in order to reduce its size and to exploit any changing patterns in the client's behavior. For instance, by expelling some suffixes (e.g., the longer ones) from the suffix tree we can take advantage of a user whose data interests or habits (trajectories) evolve over time. Additionally, we are implementing all the state-of-the-art prediction algorithms in order to perform an exhaustive performance comparison to draw important

conclusions under what cases each algorithm performs the best and also to verify in practice the validity of Theorem 1.

6 Concluding Remarks

Discrete sequence prediction is an effective means to reduce access latencies and location uncertainty in wireless networking applications. Due to the unique features of the corresponding mobile applications, the employed prediction scheme should be online, lightweight and accurate; though the existing prediction schemes do not satisfy all these requirements. To address all of them, we presented a new prediction scheme, named $\mathcal{STP}$. This new scheme is based on the suffix tree data structure, which guarantees low storage overhead, fast and online construction. We showed the viability of the method through some preliminary experimental results.

References

1. A. Apostolico and G. Bejerano. Optimal amnesic probabilistic automata or how to learn and classify proteins in linear time and space. *Journal of Computational Biology*, 7(3–4):381–393, 2000.
2. R. Begleiter, R. El-Yaniv, and G. Yona. On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004.
3. G. Bejerano and G. Yona. Variations on probabilistic suffix trees: Statistical modeling and prediction of protein families. *Bioinformatics*, 17(1):23–43, 2001.
4. A. Bhattacharya and S. K. Das. LeZi-Update: An information-theoretic framework for personal mobility tracking in PCS networks. *ACM/Kluwer Wireless Networks*, 8(2–3):121–135, 2002.
5. X. Chen and X. Zhang. A popularity-based prediction model for Web prefetching. *IEEE Computer*, 36(3):63–70, 2003.
6. J. G. Cleary and I. H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984.
7. K. M. Curewitz, P. Krishnan, and J. S. Vitter. Practical prefetching via data compression. In *Proceedings of the ACM International Conference on Data Management (SIGMOD)*, pages 257–266, 1993.
8. S. K. Das, A. Cook, D. J. Bhattacharya, Heierman E. O., and T.-Y. Lin. The role of prediction algorithms in the MavHome smart home architecture. *IEEE Wireless Communications*, 9(6):77–84, 2002.
9. M. Deshpande and G. Karypis. Selective Markov models for predicting Web page accesses. *ACM Transactions on Internet Technology*, 4(2):163–184, 2004.
10. A. Ehrenfeucht and J. Mycielski. A pseudorandom sequence – How random is it? *American Mathematical Monthly*, 99(4):373–375, 1992.
11. M. Feder and N. Merhav. Relations between entropy and error probability. *IEEE Transactions on Information Theory*, 40(1):259–266, 1994.
12. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer science and computational biology*. Cambridge University Press, 1997.
13. D. Katsaros and Y. Manolopoulos. Web caching in broadcast mobile wireless environments. *IEEE Internet Computing*, 8(3):37–45, 2004.

14. D. Katsaros and Y. Manolopoulos. Prediction in wireless networks by variable length Markov chains. Manuscript in preparation, 2005.
15. P. Krishnan and J. S. Vitter. Optimal prediction for prefetching in the worst case. *SIAM Journal on Computing*, 27(6):1617–1636, 1998.
16. C. Largeron-Leténo. Prediction suffix trees for supervised classification of sequences. *Pattern Recognition Letters*, 24(16):3153–3164, 2003.
17. A. Misra, A. Roy, and S. K. Das. An information-theoretic framework for optimal location tracking in multi-system 4*G* wireless networks. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, volume 1, pages 286–297, 2004.
18. A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. A data mining algorithm for generalized Web prefetching. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1155–1169, 2003.
19. J. Pitkow and Pirolli P. Mining longest repeating subsequences to predict World Wide Web surfing. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 139–150, 1999.
20. D. Ron, Y. Singer, and N. Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25(2–3):117–149, 1996.
21. A. Roy, S. K. Das, and A. Misra. Exploiting information theory for adaptive mobility and resource management in future wireless cellular networks. *IEEE Wireless Communications*, 11(4):59–65, 2004.
22. P. Volf. *Weighting techniques in data compression: Theory and algorithms*. PhD thesis, Technische Universiteit Eindhoven, 2002.
23. F. J. Willems. The context-tree weighting method: Extensions. *IEEE Transactions on Information Theory*, 44(2):792–798, 1998.
24. F. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens. The context-tree weighting method: Basic properties. *IEEE Transactions on Information Theory*, 41(3):653–664, 1995.