A Data Mining Algorithm for Generalized Web Prefetching

Alexandros Nanopoulos, Dimitrios Katsaros, and Yannis Manolopoulos, *Member*, *IEEE Computer Society*

Abstract—Predictive Web prefetching refers to the mechanism of deducing the forthcoming page accesses of a client based on its past accesses. In this paper, we present a new context for the interpretation of Web prefetching algorithms as Markov predictors. We identify the factors that affect the performance of Web prefetching algorithms. We propose a new algorithm called WM_o, which is based on data mining and is proven to be a generalization of existing ones. It was designed to address their specific limitations and its characteristics include all the above factors. It compares favorably with previously proposed algorithms. Further, the algorithm efficiently addresses the increased number of candidates. We present a detailed performance evaluation of WM_o with synthetic and real data. The experimental results show that WM_o can provide significant improvements over previously proposed Web prefetching algorithms.

Index Terms—Prefetching, prediction, Web mining, association rules, data mining.

1 Introduction

In recent years, the Web has become the primary means for information dissemination. It is being used for commercial, entertainment, or educational purposes, and, thus, its popularity resulted in heavy traffic in the Internet. Since the Internet capacity is not keeping pace, the net effect of this growth was a significant increase in the user perceived latency, that is, the time between when a client issues a request for a document and the time the response arrives. Potential sources of latency are the Web servers' heavy load, network congestion, low bandwidth, bandwidth underutilization, and propagation delay.

An obvious solution would be to increase the bandwidth. This does not seem a viable solution since the Web's infrastructure (Internet) cannot be easily changed, without significant economic cost. Apart from this cost, higher bandwidth would ease users to create more sophisticated and "heavy" documents, "choking" again the network. Moreover, propagation delay cannot be reduced beyond a certain point since it depends on the physical distance between the communicating end points.

The first solution that was investigated toward the reduction of latency was the caching of Web documents at various points in the network (client, proxy, server) [5], [10], [37]. Caching capitalizes on the temporal locality [23]. Effective client and proxy caches reduce the client perceived latency, the server load, and the number of traveling packets, thus increasing the available bandwidth. Several caching policies have been proposed during the previous years, especially for proxy servers [5], [10], [37]. Nevertheless, the benefits reaped due to caching can be limited [25] when Web resources tend to change very frequently, resources cannot be cached (dynamically generated Web documents), they contain cookies (this issue matters only caching proxies), and when request streams do not exhibit high temporal locality. The negative effects of the first problem can be partially alleviated by employing some, though costly, cache consistency mechanism [13]. The second problem could be addressed by enhancing the cache with some of the respective server's query processing capabilities, so as to perform the necessary processing on data [16]. The third and fourth problems seem to not be tackled by caching at all.

As a further improvement in the situation, the technique of prefetching has been investigated. Prefetching refers to the process of deducing a client's future requests for Web objects and getting those objects into the cache, in the background, before an explicit request is made for them. Prefetching capitalizes on the spatial locality, that is, correlated references for different documents present in request streams [1], and exploits the client's idle time, i.e., the time between successive requests. The main advantages of employing prefetching is that it prevents bandwidth underutilization and hides part of the latency. On the other hand, an overaggressive scheme may cause excessive network traffic. Additionally, without a carefully designed prefetching scheme, several transferred documents may not be used by the client at all, thus wasting bandwidth. Nevertheless, an effective prefetching scheme, combined with a transport rate control mechanism, can shape the network traffic, reducing significantly its burstiness and, thus, can improve the network performance [9].

In general, there exist two prefetching approaches. Either the client will inform the system about its future requirements [32] or, in a more automated manner and transparently to the client, the system will make predictions based on the sequence of the client's past references [12], [33]. The first approach is characterized as *informed* prefetching and concentrates around the ideas developed as part of the so-called *Transparent Informed Prefetching* [32], where the application discloses its exact future requests

• *The authors are with the Department of Informatics, Artistotle University, Thessaloniki, 54124 Greece.*
E-mail: {alex, dimitris, manolopo}@delab.csd.auth.gr.

Manuscript received 4 Oct. 2000; revised 4 May 2001; accepted 6 Nov. 2001.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 113950.

and the system is responsible for bringing the respective objects into the buffer. In the design of a prefetching scheme for the Web, its specialties must be taken into account. Two characteristics seem to heavily affect such a design: 1) the client server paradigm of computing the Web implements and 2) its hypertextual nature. Therefore, informed prefetching seems inapplicable in the Web since a user does not know in advance its future requirements, due to the "navigation" from page to page by following the hypertext links. Source anticipation [38], [24], [18], i.e., the prefetching of all (or some part thereof) of the embedded links of the document, may work in some cases, but seems inappropriate in general, because there is no a priori information about which of a large set of embedded links the client is likely to request. On the other hand, the second approach, called *predictive* prefetching, is more viable, especially under the assumption that there is sufficient spatial locality in client requests because such a prefetching method could use the history of requests to make predictions.

Existing predictive prefetching algorithms examined in database, file systems, and recently on the Web can be categorized into two families: 1) those that use a graph, called *Dependency Graph (DG)*, to hold the patterns of accesses [21], [33] and 2) those that use a scheme adopted from the text compression domain [12], [19], [34], called *Prediction by Partial Match (PPM)*. Related work is described in detail in Section 3.

1.1 Motivation

Existing Web prefetching schemes differ from the corresponding ones proposed in the context of file systems only because they use techniques for the identification of user sessions. For the core issue in prefetching, i.e., prediction of requests, existing algorithms from the context of file-systems have been utilized. Consequently, existing Web prefetching algorithms do not recognize the specialized characteristics of the Web. More precisely, two important factors (identified in the present work) are:

- The order of dependencies among the documents of the patterns.
- The interleaving of documents which belong to patterns, with random visits within user sessions (i.e., noise).

These factors arise from both the contents of the documents and the site's structure (the links among documents) and are described as follows:

The choice of forthcoming pages can depend, in general, on a number of previously visited pages (see Section 2.2). The *DG* and the 1-order *PPM* algorithms consider only first order dependencies. Thus, if several past visits have to be considered and there exist patterns corresponding to higher dependencies, these algorithms do not take them into account in making their predictions. On the other hand, higher-order *PPM* algorithms use a constant maximum value for the considered orders. However, no method for the determination of the maximum order is provided in [34], [19]. A choice of a small maximum may have a similar disadvantage as in the former case, whereas a choice of a large maximum may lead to unnecessary computational cost, due to the maintenance of a large number of rules.

A Web user, obeying the navigation model of Section 2.2, may follow, within a session, links to pages that belong to one of several patterns. However, during the same session, the user may also navigate to other pages that do not belong to this pattern (or that may not belong to any pattern at all). Hence, a user session can contain both documents belonging to patterns and others that do not, and these documents are interleaved in the session. However, *PPM* prefetchers (of one or higher order) consider only subsequences of consecutive documents inside sessions. On the other hand, the *DG* algorithm does not require the documents, which comprise patterns, to be consecutive in the sessions. However, since the order of the *DG* algorithm is one, only subsequences with two pages (not necessarily consecutive in the sessions) are considered.

Consequently, none of the existing algorithms considers all the previously stated factors. Moreover, they have not yet been tested comparatively in order to examine the impact of these factors. This comparison requires the definition of a formal context for the description of existing approaches and the development of a new algorithm that considers all the factors.

1.2 Paper Contribution

In this paper, we focus on predictive prefetching. First, we identify two factors, i.e., the order of dependencies between page accesses and the noise which affects the method for calculating the appearance frequencies of user access sequences that characterize the performance of predictive Web prefetching algorithms. According to these factors, we present a framework which is used to describe prefetching algorithms in terms of Markov predictors. This framework allows for the formal examination of existing Web prefetching algorithms and the identification of equivalences or differences among them. Additionally, we develop a new algorithm that is formally proven to be a generalization of existing ones. An extensive analytical and experimental comparison of all algorithms, for the first time (the performance of existing algorithms have been examined only independently), indicates that the proposed algorithm outperforms existing ones by combining their advantages without presenting their deficiencies. Hence, the main contributions of this paper can be summarized as:

- a novel framework for a formal and unified description of Web prefetching algorithms,
- a new Web prefetching algorithm that generalizes existing ones, and
- a detailed (analytical and experimental) comparison of all algorithms.

The rest of the paper is organized as follows: Section 2 presents the necessary background information regarding the technique of prefetching and describes a model that characterizes the Web user navigation. Section 3 reviews related work and outlines the motivation of this work. Section 4 presents a framework for the unified treatment of all Web prefetching algorithms and Section 5 describes the proposed algorithm. Section 6 provides the experimental results and, finally, Section 7 contains the conclusions.

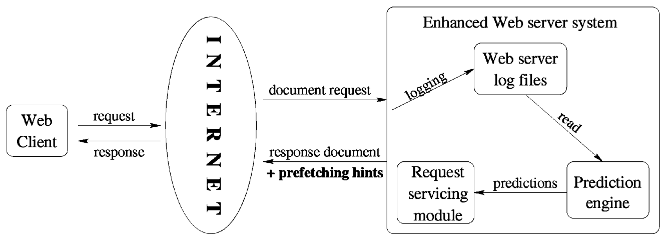


Fig. 1. Proposed architecture of a prediction-enabled Web server.

2 Background

2.1 Mechanism of Predictive Prefetching

Deduction of future references on the basis of predictive prefetching can be implemented by having an engine which, after processing the past references, derives the probability of future access for the documents accessed so far. The prediction engine can reside either in the client or in the server side. In the former case, it uses the set of past references to find correlations and initiates prefetching. No modifications need to be made to the current Web infrastructure (e.g., HTTP protocol, Web servers) nor to Web browsers if the prefetcher module runs as a proxy in the browser [24]. The main limitation of this approach is that the clients, in general, lack sufficient information to discover the correlations between documents since their requests cover a broad range of Web servers and an even broader range of documents. On the other hand, Web servers are in better position to make predictions about future references since they log a significant[1] part of requests by all Internet clients for the resources they own.

The main drawback of the latter approach is that additional communication between the server and the client is needed in order to realize the prefetching scheme. This scheme can be implemented by either the dissemination of predicted resources to the client [7] or exchange of messages between server and clients, having the server piggybacking information about the predicted resources onto regular response messages, avoiding establishment of any new TCP connections [11]. Such a mechanism has been implemented in [11], [18] and seems the most appropriate since it requires relatively few enhancements to the current request-response protocol and no changes to the HTTP 1.1 protocol.

In what follows in this article, we assume that there is a system implementing a server-based predictive prefetcher, which piggybacks its predictions as hints to its clients. Fig. 1 illustrates how such an enhanced Web server could cooperate with a prefetch engine to disseminate hints every time a client requests a document of the server.

2.2 User Navigation Model

The way users navigate in a Web site depends not only on their interests, but also on the site structure. More precisely, a user, who has currently selected document D, chooses the document to visit next mainly among the set of the links contained in D [22]. This choice, in general, is based on the previously visited documents (i.e., D and any other documents visited before D). Otherwise, the user is randomly exploring the site, not seeking for particular information. The former case induces dependencies between the visited documents of the site. If these dependencies correlate only pairs of documents, then they are called *first order dependencies*, otherwise, they are called *higher order dependencies*. This model of user navigation describes a Markovian process over the graph, whose nodes are the documents of the site and arcs are the links between the documents.

Users that are not randomly exploring a Web site usually visit pages according to a pattern. Therefore, a user access sequence contains pages that belong to one of several patterns. However, a user may visit other pages, as well, that do not belong to the pattern. Consequently, a user access sequence may contain pages which belong to a pattern and several other pages between them that do not.

The length of user access sequences, the dependencies between accesses, and the existence of page accesses (inside the sequences) that do not belong to patterns are parameters which depend on the type of the Web site. In small sites, the impact of these parameters may be small due to the limited navigational alternatives. In contrast, large sites, i.e., with a large number of documents and fairly high connectivity (that resemble the traditional hypertext databases), present navigational alternatives, hence, the impact of these parameters is significant. These kind of sites are expected to become more popular in the years to come when the site creation and maintenance process will become more automated by using tools like Araneus [2] and Strudel [20], that automatically generate sites based on the contents of underlying databases and find applications in service providing, like e-commerce. Since the performance requirements for this type of Web sites are significantly increased, prefetching can be very beneficial for them.

3 Related Work

Research on predictive Web prefetching has involved the important issue of log file processing and the determination of user transactions (sessions) from it.[2] Several approaches have been proposed toward this direction [14], [15]. Since it is a necessary step for every Web prefetching method, more or less similar approaches on transaction formation from log files have been proposed in [33], [34], [26]. However, the most important factor of any Web prefetching scheme is the prediction algorithm, which is used to determine the actual documents to be prefetched.

The prediction scheme described in [33] uses a prefetching algorithm proposed for prefetching in the context of file systems [21]. It constructs a data structure, called *Dependency Graph (DG)*, which maintains the pattern of access to different documents stored at the server. Fig. 2a illustrates an example of a dependency graph. The graph has a node for each document that has ever been accessed. There is an arc from node X to node Y (nodes correspond to documents), if and only if at some point in time, Y was accessed within w accesses after X, where w is the *lookahead window*, and both

1. They only miss the requests satisfied by browser or proxy caches.

2. This issue is not required for prefetching in the context of file systems.

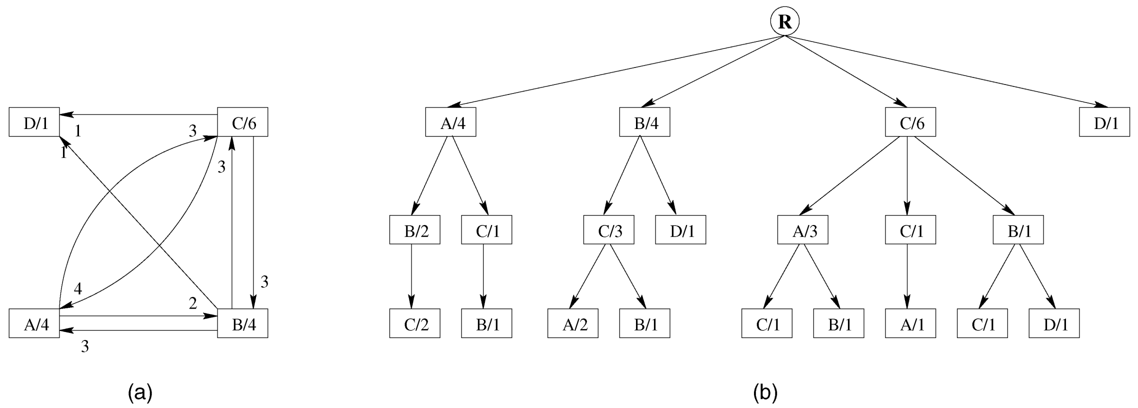


Fig. 2. (a) Dependency graph (lookahead window 2) for two request streams ABCACBD and CCABCBCA. (b) PPM predictor of 2-order for two request streams ABCACBD and CCABCBCA.

accesses were done by the same client. The weight on the arc is the ratio of the number of accesses from X to Y, to the number of accesses to X itself. The accuracy of prefetching is controlled with a user-defined cut-off threshold for the weights. For a complete description of the scheme, see [21], [33]. From the above description, it follows that *DG* considers only first-order dependencies.

The work described in [7] uses essentially the *Dependency Graph*, but makes predictions by computing the transitive closure of this graph. This method was tested and did not show significantly better results compared to the *Dependency Graph*. Moreover, it is very demanding in terms of computational time cost due to the closure calculation (which requires complexity of $O(n^3)$, where n is the number of vertices). This renders the method practically inapplicable for Web sites consisting of many pages.

The scheme described in [34], [19] also uses a prefetching algorithm from the context of file systems [12]. It is based on the notion of an *m*-order *Prediction-by-Partial-Match (PPM)* predictor. An *m*-order *PPM* predictor maintains Markov predictors of order *j*, for all $1 \leq j \leq m$ (cf., Section 4.1). This scheme is also called *All-m-Order Markov model* [17]. A *j*-order Markov predictor uses the preceding j "events" to calculate the probability of the next one to come. Fig. 2b illustrates a 2-order *PPM* predictor, where paths emanate from the tree root with maximum length equal to $m + 1 (= 3)$. Each root emanating path or subpath corresponds to a distinct sequence of requests seen so far. Each node of a path corresponds to a document of the server. The number associated with each node depicts the number of times this node was requested, after all nodes before it in the path, were requested. A user-defined cut-off threshold controls the documents that will be prefetched, with respect to the numbers in the corresponding nodes. More particularly, the approach in [19] is applied in the context of modem links, where prefetching is exploited only during idle modem time, in a more aggressive manner. For a complete description of the scheme, see [12], [34], [19]. Since *PPM* predictors use the preceding document accesses, they consider subsequences of consecutive documents within request streams. Hence, the way that the probability of patterns is counted by *PPM* is based on the assumption that they consist of documents which form consecutive subsequences within the pattern.

Recently, several algorithms have been proposed for mining patterns from Web logs [15], [8], [14], [31], [29]. Although these patterns can be characterized as descriptive since they indicate regularities discovered from user access information, algorithms for Web log mining and for predictive Web prefetching share the common objective of determining statistically significant user access sequences, i.e., *access patterns*. The Web prefetching strategy proposed in [26] develops a specialized association rule mining algorithm to discover the prefetched documents. It discovers dependencies between pairs of documents (association rules with one item in the head and one item in the body). The counting of support is done differently than in [3] since the ordering of documents is considered. Only consecutive subsequences (of length two) inside a user transaction are supported. For instance, the user transaction ABCACBD supports the subsequences: AB, BC, CA, AC, CB, and BD. Moreover, this algorithm uses support and confidence pruning criteria and maintains only rules with the highest confidence, for each rule head.[3] For the purpose of rule activation, i.e., the determination of prefetched documents after a given document request, the algorithm in [26] invokes a search to the rules that have been discovered. More particularly, in case document D_i is requested and there exists a rule $D_i \Rightarrow D_j$, then D_j is prefetched. Additionally, if a rule $D_j \Rightarrow D_k$ exists, then D_k can be also prefetched. The depth to which this search is continued during the activation of rules is specified by a user-defined parameter (called *pushed-length* [26]). However, this recursive activation presents an increase in the network traffic, as the experimental results in [26] illustrate. In [26], differently from other prefetching algorithms, rather high traffic increases are considered. Nevertheless, as described, the main aspect of any prefetching scheme is the rule discovery procedure. Therefore, recursive rule activation can be applied to any Web prefetching scheme. For this reason, the theoretical comparison of all schemes in this work is done with respect to rule discovery. However, experimental results (cf., Section 6) evaluate the use of recursive activation. As it will be described in the following sections, the algorithm in [26] can be classified in the context of 1-order *PPM*

3. In [26], an equivalent criterion is also proposed which considers the sizes of the documents.

prefetchers. Thus, it considers only first-order dependencies and assumes that patterns appear as subsequences of consecutive documents inside user transactions.

Other related work includes [27], which describes a prefetching algorithm that is also based on association rule mining. Similar to [26], rules with one document in both the head and the body are considered. However, the subject of that paper is Web-server caching and, more particularly, the prefetching of documents from the Web server's disk to its main memory. This approach differs from the Web prefetching, which concerns the prefetching of documents from the server into the client's cache. Besides prefetching based on association patterns, other types of Web log mining patterns can be used as well. For instance, path traversals [15] can be adapted to produce rules that can be used for Web prefetching. It will be shown (cf., Section 4) that the resulting scheme can be classified in the context of *m*-order *PPM*. The improvement in the efficiency of *PPM* is examined in [17] (which uses the name All-*m*th-Order Markov model for *PPM*). Three pruning criteria are proposed: 1) support-pruning, 2) confidence-pruning, and 3) error-pruning. The subject of [17] is mainly the efficiency, whereas its experimental results do not indicate significant improvement in the effectiveness of *PPM* prefetching. Nevertheless, support-pruning is a specialization of *PPM*, that is also examined in [26], [27] and in this paper as well. The other two criteria are used in a postprocessing step, on the set of discovered rules, and can be applied to any prefetching scheme, thus they are orthogonal issues to the subject examined in this paper. Finally, two variations of the *PPM* prefetcher are described in [35], [36]. The first one is a subset of the *PPM*, whereas in the second one, the selection of prefetching rules to activate is determined by "weights" assigned on them.

4 A Common Context for Predictive Web Prefetching

4.1 Markov Predictors

If $S = \langle p_1, \ldots, p_n \rangle$ is a sequence of accesses (called a transaction) made by a user, then the conditional probability that the next access will be p_{n+1} is $P(p_{n+1}|p_1, \ldots, p_n)$. Therefore, given a set of user transactions, rules of the form

$$p_1, \ldots, p_n \;\Rightarrow\; p_{n+1} \tag{1}$$

can be derived, where $P(p_{n+1}|p_1, \ldots, p_n)$ is equal to or larger than a user-defined cut-off value T_c. The left part of the rule is called *head* and the right part is called *body*. The body of the rule can also be of any length larger than one. Thus, rules of the form:

$$p_1, \ldots, p_n \;\Rightarrow\; p_{n+1}, \ldots, p_{n+m} \tag{2}$$

can be formed. In this case, $P(p_{n+1}, \ldots, p_{n+m}|p_1, \ldots, p_n)$ has to be larger than T_c.

The dependency of forthcoming accesses on past accesses defines a *Markov chain*. The number of past accesses considered in each rule for the calculation of the corresponding conditional probability is called the *order* of the rule. For instance, the order of the rule $A, B \Rightarrow C$ is 2.

Definition 1. *An n-order Markov predictor is defined to be a scheme for the calculation of conditional probabilities*

$$P(p_{n+1}, \ldots, p_{n+m}|p_1, \ldots, p_n)$$

between document accesses and the determination of rules of the form (2). The head of each rule has a size equal to n and the body has a maximum size equal to m.

A predictive prefetching algorithm can be defined as a collection of $1, 2, \ldots, n$-order Markov predictors (if several orders are considered). Additionally, an *activation* mechanism for finding the prefetched pages from the corresponding rules is required. Thus, the objectives of a predictive Web prefetching algorithm can be summarized as:

- the calculation of conditional probabilities based on user accesses,
- the determination of rules of the form (2), and
- the activation of rules for which their head contains the accesses a user has done up to a time point and the prefetching of pages which are in the body of the rules.

We present below how existing algorithms are described in this common context.

The *Dependency Graph (DG)* algorithm uses a *first order* (1-order) Markov predictor. It calculates conditional probabilities $P(p_i|p_j)$. It maintains a set of rules of the form $p_i \Rightarrow p_j$. For a user who has accessed the sequence of documents $S = \langle p_1, \ldots, p_n \rangle$, *DG* searches all rules with head p_n and prefetches all documents p_{n+1} for which there exists a rule $p_n \Rightarrow p_{n+1}$.

The *k*-order *PPM* algorithm uses $1, 2, \ldots, k$-order Markov predictors (k is a constant). These Markov predictors calculate conditional probabilities of the form

$$P(p_{n+1}|p_n), P(p_{n+1}|p_n, p_{n-1}), \ldots, P(p_{n+1}|p_n, \ldots, p_{n-k+1})$$

and determine the corresponding rules, which have head sizes equal to $1, 2, \ldots, k$. Since rules of several orders are activated from one user sequence, the same documents can appear in the body of different rules, hence duplicate elimination is performed.

The prefetching scheme in [26] uses a 1-order Markov predictor. The rules are of the form $p_i \Rightarrow p_j$. This is analogous to *DG* and 1-order *PPM*. The scheme in [26] uses two constraints: 1) support-based pruning and 2) only rules with the maximum conditional probability (confidence) are selected for each document which serves as head of a rule. These constraints do not affect the consideration of [26] as a Markov predictor.

Finally, path traversals [15], from sequences of the form $\langle p_1, \ldots, p_n \rangle$, derive rules of the form $p_1, \ldots, p_k \Rightarrow p_{k+1}, \ldots, p_n$, by considering conditional probabilities

$$P(p_{k+1}, \ldots, p_n|p_1, \ldots, p_k).$$

It is easy to see that path-traversals can use a set of $1, 2, \ldots, k$-order Markov predictors and are equivalent to *k*-order *PPM* (see also Section 4.2).

4.2 Calculation of Conditional Probabilities

For rules of the form (2), the probability

$$P(p_{n+1}, \ldots, p_{n+m} | p_n, \ldots, p_1)$$

is equal to $\frac{P(\langle p_1,\ldots,p_{n+m}\rangle)}{P(\langle p_1,\ldots,p_n\rangle)}$. Given a collection of user transactions, the probability $P(p_1, \ldots, p_n)$ of an access sequence $S = \langle p_1, \ldots, p_n \rangle$ is the normalized number of occurrences of S inside the collection of transactions. Thus, $P(S)$ is equal to $fr(S)$ divided by the total number of transactions, where $fr(S)$ denotes the number of occurrences of S, i.e., its frequency. We present next how each specific algorithm forms the transactions and counts the occurrence frequencies of accesses sequences.

4.2.1 Formation of User Transactions

The determination of user transactions first requires the identification of *user sessions* from the log file. This issue is handled in [14] and elsewhere, where accesses of each user are grouped into sessions according to their closeness in time. Then, user sessions can be further processed with the method proposed in [15]. Thus, user sessions are decomposed into a number of *maximal forward references*, using algorithm *MF* [15]. This filters out the effect of backward references (usually made by the use of the "Back" button in a browser), which are done only for navigation purposes. The cleansed user sessions constitute the user transactions. It is important to notice that, by this processing, the transactions contain only distinct pages. It can be assumed that a document initially duplicated in a user session (i.e., before processing with *MF*) will probably be contained in the local (client) cache after its first appearance. Thus, even if it was not removed by *MF*, there would be no need to prefetch it more than once.

The definition of user sessions has the objective of separating independent accesses made by different users or by the same user at distant points in time. Thus, false correlations during the calculation of conditional probabilities between page accesses are avoided. The concept of a user session is defined in all existing Web prefetching algorithms. In [34], it is called *browsing session*, whereas in [7], it is called *stride*. Although [33] uses the concept of user session with respect to separating accesses by different users, it does not explicitly separate accesses of the same user which have large time difference.[4] However, without loss of generality, we assume that the *DG* algorithm is based on user sessions which consider time differences as well. Throughout this paper, we use the term *transaction*, adopted from [14]. Thus, we assume that the information about user accesses is represented in a uniform way for all algorithms.

4.2.2 Frequency Counting

The calculation of conditional probabilities with respect to $\frac{P(\langle p_n,\ldots,p_{n+m}\rangle)}{P(\langle p_1,\ldots,p_n\rangle)}$ requires counting the frequency of the corresponding access sequences, i.e., $fr(\langle p_1, \ldots, p_{n+m}\rangle)$ and $fr(\langle p_1, \ldots, p_n\rangle)$. Given a set of user transactions, the frequency of an access sequence S, is equal to the number of transactions T, for which S is *contained* in T or, equivalently, S is a subsequence of T.[5] An abstract scheme for the frequency counting can be described as follows: Each transaction is read and the frequency of every *contained* subsequence is increased by one. After having read every transaction, the frequencies of the subsequences have been counted, thus the calculation of the corresponding probabilities and the formation of the rules (those having conditional probability larger than T_c) can be accomplished. The difference of each Web prefetching algorithm stems from the way the subsequence *containment* is defined. Therefore, for each transaction, each algorithm updates (i.e., increases by one) the frequencies of different subsequences. Notice that, although the algorithms for purposes of efficiency, may process transactions differently (e.g., in several passes [26], [15]), the result is equivalent to that produced by the abstract scheme. Thus, the reason for adopting it is that we focus on the results of the algorithms since it is more convenient to compare them in the sequel.

For each transaction $T = \langle p_1, \ldots, p_n\rangle$, *DG* increases by one all frequencies $fr(\langle p_i\rangle)$ and $fr(\langle p_i, p_j\rangle)$, for $1 \leq i < j \leq n$ and $j - i \leq \min\{w, n\}$, where w is the *lookahead window* size. Thus, *DG* calculates conditional probabilities as:

$$P(p_j|p_i) = \frac{fr(\langle p_i, p_j\rangle)}{fr(\langle p_i\rangle)}. \qquad (3)$$

For a transaction $T = \langle p_1, \ldots, p_n\rangle$, the k-order *PPM* updates the frequencies

$$fr(\langle p_i\rangle), fr(\langle p_i, p_{i+1}\rangle), \ldots, fr(\langle p_i, \ldots, p_{i+j}\rangle),$$

for each $1 \leq i \leq n$, and $i + j \leq n$ and $1 \leq j \leq k$. Thus, *PPM* calculates conditional probabilities as:

$$P(p_{i+j}|p_i, \ldots, p_{i+j-1}) = \frac{fr(\langle p_i, \ldots, p_{i+j-1}, p_{i+j}\rangle)}{fr(\langle p_i, \ldots, p_{i+j-1}\rangle)}. \qquad (4)$$

Notice that, although both 1-order *PPM* and *DG* form rules with one document in the head and one in the body, they are not equivalent. 1-order *PPM* (as *PPM* of all orders) requires that subsequences contain documents that are consecutive in the transaction, but the same does not hold for *DG*.

According to [26], a sequence $S = \langle p_i, p_j\rangle$ is *contained* in a transaction T, if p_i is immediately followed by p_j in T [26], i.e., the frequency of all subsequences with two consecutive documents are updated. The rules in [26] have one document in the head and the corresponding conditional probabilities are of the form (4), with k equal to one. Since both schemes calculate the same frequencies and determine the same conditional probabilities, they produce identical rules. Therefore, the scheme in [26] is equivalent to 1-order *PPM*.

For path-traversal patterns, the frequency of an access sequence S, is updated by a transaction T, if S is a *consecutive subsequence* in T [15], i.e., it updates the frequency of each subsequence that contains consecutive documents in a transaction. This is equivalent to the procedure followed by the k-order *PPM*. Therefore, a scheme based on path-traversal patterns can be categorized in the family of k-order *PPM* algorithms.

4. Accesses of the same user in [33] are contained in a ring buffer and each new entry replaces the oldest one in case the buffer is full.

5. Using data mining terminology, the frequency corresponds to the notion of *support*.

5 Generalized Algorithm

5.1 Overview of the Proposed Method

As it is evident from the framework in the previous section, a prefetching scheme that considers higher order dependencies requires the use of Markov predictors of higher orders, additionally to that of first order. Therefore, it should allow for rules with head sizes larger than one (and possibly for rules with body sizes larger than one). However, the maximum order depends on the way users navigate and on the site characteristics, thus it should vary and not be restricted to a constant value. Hence, a prefetching scheme should be able to adaptively select the appropriate maximum value for the order.

To take into account the existence of random accesses to documents within the transactions, the counting procedure of occurrence frequencies has to be able to neglect them. More particularly, given a transaction T, the frequency counting procedure should consider an access sequence S, which consists of documents that all belong to T, to be *contained* in T, even if it is not a subsequence of T with all the documents being consecutive in T. For example, $S = \langle A, B \rangle$ can be considered as being contained in $T = \langle A, X, B, C \rangle$, although A and B are not consecutive in T. Therefore, random accesses, like X, can be bypassed.

As it will be described in the following, the above requirements designate a scheme which corresponds to the discovery of associations among user accesses. However, the required scheme cannot be based on association rules of the form defined in [3] since the ordering,[6] these schemes correspond to the 1-order *PPM* algorithm.

The particular specifications described above present differences from existing approaches and call for the development of a new effective prefetching scheme. Additionally, the involved computational complexity requires the design of an efficient mining algorithm.

5.2 Proposed Type of Rules

Associations consider rules of several orders [3], and not of one only. The maximum order is derived from the data and it does not have to be specified as an arbitrary constant value [3]. For the frequency counting, a transaction T, supports sequences that do not necessarily contain consecutive documents in T. More specifically, an access sequence S is considered as a set of accesses and is *contained* in a transaction T, if it is a subset of it, that is, if $S \subseteq T$ [3]. Therefore, the use of Web log mining methods for the discovery of association rules among user accesses (denoted as *WM* method) seems to present the required specifications, which have been described previously.

However, it is clear that the ordering of document accesses inside a transaction is important for the purpose of prefetching. The problem of finding association rules was initially stated for basket data. Although each transaction contains the accessed documents in the right ordering, i.e., they are ordered with respect to their access time (due to the processing of transactions by *MF* algorithm), association rule discovery algorithms represent candidates as sets of documents, which do not consider this ordering.

For instance, let A and B be frequent documents (i.e., *large*, according to the terminology in [3]). From these documents, the candidate $c_1 = \{A, B\}$ will be produced, but the candidate $c_2 = \langle B, A \rangle$ will not since the consideration of candidates as sets does not distinguish between c_1 and c_2.[7] Moreover, the *containment* is defined by the subset operator, which also does not take the ordering of documents into account, thus, for a transaction $T = \langle B, C, A, D \rangle$, it holds that $c_1 \subseteq T$, although B precedes A in T. Since the frequency of c_1 is updated by such transactions, an association $A \Rightarrow B$ can be formed. According to this rule, when document A is requested, then a document B can be prefetched, although it is possible that, in the majority of transactions, B precedes A. Nevertheless, the rule $A \Rightarrow B$ does not reflect this fact and causes the incorrect prefetching of B. A large number of such incorrect rules introduces bandwidth waste. On the other hand, if there is a pattern such that the request of B induces the request of A, then the corresponding rule $B \Rightarrow A$ will be missed because it could be formed by c_2 (which is not considered). The same reasoning can be applied for candidates with larger length.

The required approach involves a new definition of the candidate generation procedure and the *containment* criterion. At the *k*th phase, the candidates are derived from the self-join $L_{k-1} \bowtie L_{k-1}$ [3]. However, in order to take the ordering of documents into account, the joining is done as follows: Let two access sequences, $S_1 = \langle p_1, \ldots, p_{k-1} \rangle$ and $S_2 = \langle q_1, \ldots, q_{k-1} \rangle$, both be in L_{k-1}. If $p_1 = q_1, \ldots, p_{k-2} = q_{k-2}$, then they are combined to form two candidate sequences, which are: $c_1 = \langle p_1, \ldots, p_{k-2}, p_{k-1}, q_{k-1} \rangle$ and $c_2 = \langle p_1, \ldots, p_{k-2}, q_{k-1}, p_{k-1} \rangle$. For instance, sequences $\langle A, B, C \rangle$ and $\langle A, B, D \rangle$ are combined to produce $\langle A, B, C, D \rangle$ and $\langle A, B, D, C \rangle$. The same holds for the second phase ($k = 2$). For instance, from $\langle A \rangle$ and $\langle B \rangle$, $\langle A, B \rangle$ and $\langle B, A \rangle$ are produced. The *containment* criterion is defined as follows:

Definition 2. *If $T = \langle p_1, \ldots, p_n \rangle$ is a transaction, an access sequence $S = \langle p'_1, \ldots, p'_m \rangle$ is contained by T iff:*

- *there exist integers $1 \leq i_1 < \ldots < i_m \leq n$ such that $p'_k = p_{i_k}$, for all k, where $1 \leq k \leq m$.*

A sequence S of documents contained in a transaction T with respect to the previous condition is called a *subsequence* of T and the containment is denoted as $S \preceq T$. In the sequel, the Web prefetching scheme that is based on the proposed type of rules is denoted as WM_o (o stands for ordering).

With respect to the framework presented in Section 4, WM_o algorithm uses a collection of $1, 2, \ldots, k$-order Markov predictors and produces rules of the form (2). This is done by calculating the respective conditional probabilities. Similar to [3], the maximum order does not have to be a prespecified maximum value. Moreover, WM_o differs from the k-order *PPM* in the calculation of conditional probabilities and, more specifically, in the counting of frequencies according to Definition 2.

6. The term *ordering* refers to the arrangement of documents in transactions and should not be confused with the term *order* of Markov-predictors.

7. Curly brackets denote sets (unordered) and angular brackets denote sequences (ordered).

5.3 Algorithm

The different candidate generation procedure of WM_o, due to the preservation of ordering (see Section 5.2), impacts the number of candidates. For instance, for two "large" documents $\langle A\rangle$ and $\langle B\rangle$, both candidates $\langle A,B\rangle$ and $\langle B,A\rangle$ will be generated in the second phase. Differently, a priori (and any other algorithm for association rules among basket data) would produce only one candidate, i.e., $\{A,B\}$. Following similar reasoning, the same argument can be stated for candidates with larger length, for each of which the number of different permutations is large. Nevertheless, ordering has to be preserved to provide correct prefetching. The work described in [4], [28] examines the problem of mining sequential patterns which consider ordering, as well. However, they do not take into account that user navigation is performed in a site which has a structure determined by its linkage. By not considering this factor, [4], [28] do not address the problem of dramatic increase in the number of candidates. Moreover, the work in [28] seeks patterns in a single large sequence of events using a sliding window over this sequence and, thus, it does not consider user sessions as is our case.

In order to reduce the number of candidates, pruning can be applied according to the site's structure [29], [30]. This type of pruning is based on the model of user navigation, presented in Section 2.2. The model assumes that navigation is performed following the hypertext links of the site, which form a directed graph. Therefore, an access sequence and, thus, a candidate, has to correspond to a path in this graph.

The determination of large paths can be performed in a level-wise manner, as in the a priori algorithm [3]. In each path of the algorithm, the database is scanned and the support of all candidates is counted. For each phase, all candidates have the same length. At the end of each pass, all candidates that become large are determined and they are used for the computation of the candidates for the next pass. The structure of the algorithm is given in [30], [3].

Candidates are stored in a trie structure. Each transaction that is read from the database is decomposed into the paths it contains and each one of them is examined against the trie, thus updating the frequency of the candidates. Candidate generation is performed by using a technique which extends candidates according to their outgoing edges in the graph, thus with respect to the site structure. Consequently, the ordering is preserved and, moreover, only paths in the graph are considered. Additionally, the support pruning criterion (if $T_s > 0$) of [3] has to be modified since, for a given candidate, only its subpaths have to be tested (i.e., subsets which correspond to a path in the graph) and not any arbitrary subset of documents, as it is designated for basket data [3]. Candidate generation is depicted in Fig. 3, where L_k denotes the set of all large paths of length k and G is the site graph.

The execution time required for the frequency counting procedure is significantly affected by the number of candidates [3], hence its efficiency is improved by this pruning criterion. Although several heuristics have been proposed for the reduction of the number of candidates for the Apriori algorithm, they involve basket data. The pruning with respect to the site structure is required for the particular problem, due to the ordering preservation and the large increase in the number of candidates. The effectiveness of pruning is verified by experimental results in Section 6. Details about the proposed algorithm can be found in [30], where extensive experimental results demonstrate its efficiency with respect to several parameters.

```
Algorithm genCandidates($L_k$, $G$)
begin
foreach $L = \langle \ell_1, \dots, \ell_k \rangle$, $L \in L_k$ {
    $N^+(\ell_k) = \{v | \exists \text{ arc } \ell_k \rightarrow v \in G\}$
    foreach $v \in N^+(\ell_k)$ {
        //apply modified apriori-pruning
        if $v \notin L$ and $L' = \langle \ell_2, \dots, \ell_k, v \rangle \in L_k$ {
            $C = \langle \ell_1, \dots, \ell_k, v \rangle$
            if $(\forall S \preceq C, S \neq L' \Rightarrow S \in L_k)$
                insert $C$ in the candidate-trie
        }
    }
}
end
```

Fig. 3. Candidate generation procedure.

5.4 Proof of Generalization

This section presents proofs that WM_o is a generalization of existing predictive Web prefetching algorithms, according to the framework presented in Section 4. The notion of generalization is based on showing that WM_o can be "reduced" to existing algorithms, whereas the inverse is not true. More precisely, for each case, it is shown that by applying a set of constraints, WM_o becomes equivalent to existing algorithms. Thus, existing algorithms are *constrained* instances of WM_o.

Proposition 1. *WM_o is a generalization of* DG.

Proof. We constrain WM_o to use only a 1-order Markov predictor. Let w be the length of the *lookahead* window.

Case 1. If $w = \infty$, for each transaction $T = \langle p_1, \dots, p_n\rangle$, WM_o will increase by one each $fr(p_i)$ and $fr(\langle p_i, p_j\rangle)$, where $1 \le i < j \le n$. These are exactly the frequencies that will be increased by *DG* also. Therefore, both WM_o and *DG* find the same values for conditional probabilities $P(p_j|p_i) = \frac{fr(\langle p_i,p_j\rangle)}{fr(\langle p_i\rangle)}$. It follows that *DG* and WM_o will produce the same rules, hence they become equivalent.

Case 2. If $w < \infty$ for *DG*, the same sliding window can be applied for WM_o also. This presents another constraint for WM_o, compared to the case where $w = \infty$. For each transaction $T = \langle p_1, \dots, p_n\rangle$, WM_o will increase by one each $fr(p_i)$ and $fr(\langle p_i, p_j\rangle)$, where $1 \le i < j \le n$ and $j - i \le \min\{w, n\}$. Again, the same rules are produced, hence, the algorithms become equivalent. □

Proposition 2. *WM_o is a generalization of PPM.*

Proof. We constrain WM_o to consider only subsequences with consecutive documents for frequency counting (notice that, if all documents of a subsequence S are consecutive in a transaction T, then $S \preceq T$). Let k be the order of *PPM*. Let also that WM_o is constrained to use the same maximum constant k for the order of rules it derives. In case rules of a higher order than k exist, if WM_o was not constrained, it would have found them.

With these constraints, for each transaction

$$T = \langle p_1, \ldots, p_n \rangle,$$

WM_o will update the frequencies $fr(\langle p_i \rangle)$,

$$fr(\langle p_i, p_{i+1} \rangle) \ldots, fr(\langle p_i, \ldots, p_{i+j} \rangle),$$

for each $1 \leq i \leq n$ and $i + j \leq n$ and $1 \leq j \leq k$. These are exactly the same frequencies that will be increased by *k-PPM* also (see Section 4.2.2). Consequently, they both find the same conditional probabilities, the same rules are produced, and the algorithms become equivalent. □

Regarding *DG* and *PPM*, none of them is a generalization of the other. *DG* uses maximum order equal to one, whereas the *k*-order *PPM* is equal to $k \geq 1$. On the other hand, inside a transaction, *DG* updates frequencies of sequences which do not necessarily have documents consecutive in the transaction. Differently, *PPM* requires that documents should be consecutive. However, there is a special case, given in the next corollary, where *DG* and *PPM* can be compared.

Corollary 1. *The 1-order PPM and the DG algorithm with sliding window size equal to one ($w = 1$) are equivalent. WM_o is a generalization of both these cases.*

Corollary 1 includes the scheme proposed in [26], which is categorized to the family of 1-order *PPM* (Section 4). The set of selected rules of [26] is a subset of the ones in the case where the highest confidence constraint is applied [26]. This constraint can also be applied to WM_o so as to become equivalent. The same reasoning can be followed for the support constraint [26].

6 Performance Results

This section presents the experimental results on the performance of predictive Web prefetching algorithms. We focus on *DG*, *PPM*, *WM*, and WM_o algorithms. Both synthetic and real data were used. The performance measures used are the following (their description can be found also in [34]):

- *Usefulness* (also called Recall or Coverage): the fraction of requests provided by the prefetcher.
- *Accuracy* (also called Precision): the fraction of the prefetched requests offered to the client that were actually used.
- *Network traffic*: the number of documents that the clients get when prefetching is used divided by the one when prefetching is not used.

First, we briefly describe the synthetic data generator. Then, we present the results and, finally, we provide a discussion.

6.1 Generation of Synthetic Workloads

In order to evaluate the performance of the algorithms over a large range of data characteristics, we generated synthetic workloads. Each workload is a set of transactions. Our data generator implements a model for the documents and the linkage of the Web site, as well as a model for user transactions.

According to the requirements about Web sites presented in Section 2.2, we choose so that all site documents have links to other documents, that is, they correspond to HTML documents. The fanout of each node, that is, the number of its outgoing links to other nodes of the same site, is a random variable uniformly distributed in the interval [1...*NFanout*], where *NFanout* is a parameter for the model. The target nodes of these links are uniformly selected from the site nodes. If some nodes have no incoming links after the termination of the procedure, then they are linked to the node with the greatest fanout. With respect to document sizes, following the model proposed in [6], we set the maximum size equal to 133KB and assign sizes drawn from a lognormal distribution[8] with mean value equal to 9.357KB and variance equal to 1.318KB.

In simulating user transactions, we generated a pool of P paths ("pattern paths," in the sequel). Each path is a sequence of links in the site and pairwise distinct Web server documents, and will be used as "seeds" for generating the transactions. Each of these paths is comprised of four nodes (documents), simulating the minimum length of a transaction. The paths are created in groups. Each group comprises a tree. The paths are actually the full length paths found in these trees. The fanout of the internal tree nodes is controlled by the parameter bf. Varying this parameter, we are able to control the "interweaving" of the paths. The nodes of these trees are selected using either the 80-20 fractal law or from the nodes that were used in the trees created so far. The percentage of these nodes is controlled by the parameter $order$, which determines the percentage of node dependencies that are nonfirst order dependencies. For example, 60 percent order means that 60 percent of the dependencies are nonfirst order dependencies. Thus, varying this parameter, we can control the order of the dependencies between the nodes in the path. The use of the fractal law results in some nodes being selected more frequently than others. This fact reflects the different popularity of the site documents, creating the so-called "hot" documents.

In order to create the transactions, we first associate a weight with each path in the pool. This weight corresponds to the probability that this path will be picked as the "seed" for a transaction. This weight is picked from an exponential distribution with unit mean, and is then normalized so that the sum of the weights for all the paths equals 1. A transaction is created as follows: First, we pick a path, say $\langle A, B, C, x \rangle$, tossing a P-sided weighted coin, where the weight for a side is the probability of picking the associated path. Then, starting from node A, we try to find a path leading to node B or with probability $corProb$ to node C, whose length is determined by a random variable, following a lognormal distribution, whose mean and variance are parameters of the model. This procedure is repeated for every node of the initial path except from those that, with probability $corProb$, were excluded from the path. The mean and variance of the lognormal distribution determine the "noise" inserted in each transaction. Low values for mean and variance leave the transaction practically unchanged with respect to its pattern path, whereas

8. Without loss of generality, we assume that HTML files are small files. Thus, according to [6], their sizes follow a lognormal distribution.

TABLE 1
The Parameters for the Generator

N	Number of site nodes	bf	Branching factor of the trees
$NFanout$	Maximum number of outgoing links	$order$	Order of the dependencies
T	Number of transactions	$noiseMean$	Mean value of the noise
P	Number of pattern paths	$noiseVar$	Variance of the noise
$corProb$	Probability of excluding a pattern path node		

larger values increase its length with respect to the pattern path. Table 1 summarizes the parameters of the generator.

6.2 Comparison of PPM Schemes

To simplify the comparison of all algorithms (and to clarify the illustrated charts), we separately examined the performance of *PPM* algorithms. Recall that the scheme proposed in [26] is categorized as 1-order *PPM*. We also examined the characteristics proposed in [26] (support pruning, selection of rules with the highest confidence for each document in the head, recursive activation), compared to the higher order *PPM* algorithm [34], [19]. Table 2 depicts the results. The synthetic data set that is used has the following values for the parameters: $N = 1,000$, $P = 1,000$, and 35,000 transactions were used for training, whereas 65,000 were used for evaluation. The *order* parameter was set to 50 percent. We examined two cases, one with low noise, i.e., when $meanNoise = 1.0$ (left part of Table 2), and one with high noise, i.e., when $meanNoise = 2.5$ (right part of Table 2). For both cases, *noiseVar* was set to 1.15. The order of (higher-order) *PPM* was set to five.

The names in the columns of Table 2 denote the following: C is the confidence value, A is the accuracy, U is the usefulness, and T is the network traffic. The confidence values for [26] are set according to the values in that paper (support threshold was set to 1 percent and *push-length* to 3). However, the corresponding confidence values for 5-order *PPM* are selected so that both schemes achieve the same network traffic (for the purpose of clarity in comparison). As it is shown, the scheme in [26] is less affected by increasing noise (when confidence equals 0.2), whereas 5-order PPM is significantly affected. (The best accuracy, i.e., 0.64, for the case of low noise is reduced to 0.21 for higher noise. Similarly, usefulness drops from 0.2 to 0.07.) Nevertheless, as it is illustrated, higher-order *PPM* clearly outperforms the scheme in [26] in all cases. Similar results, not shown here due to space restrictions, were obtained for other ranges of parameters and for real data sets. For the above reasons, in the sequel we use higher-order *PPM* as a representative of this category.

6.3 Comparison of all Algorithms

In order to carry out the experiments, we generated a number of workloads. Each workload consisted of $T = 100,000$ transactions. From these, 35,000 transactions were used to train the algorithms and the rest to evaluate their performance. The number of documents of the site for all workloads was fixed to $N = 1,000$ and the maximum fanout to $NFanout = 100$, so as to simulate a dense site. The branching factor was set to $bf = 4$ to simulate relatively low correlation between the paths. The number of paths of the pool for all workloads was fixed to $P = 1,000$. With several experiments, not shown in this report, it was found that varying the values of the parameters P and N does affect the relative performance of the considered algorithms. For all the experiments presented here, the *order* of the *PPM* algorithm was set equal to 5, so as to capture both low and higher order dependencies. For the experiments, the *lookahead window* of the *DG* algorithm was set equal to the length of the processed transaction, in order to be fair with respect to the other three algorithms. Also, in order to decouple the performance of the algorithms from the interference of the cache, we flushed it after the completion of each transaction. Each measurement in the figures that follow is the average of five different runs. At this point, we must note that the used performance measures are not independent. For example, there is a strong dependence between usefulness and network traffic. An increase in the former cannot be achieved without an analogous increase in the latter. This characteristic is very important for the interpretation of the figures to be presented in the sequel. In order to make safe judgments about the relative performance of the algorithms, we must always examine the two measures, while keeping fixed the value of the third metric, usually the network traffic, for all the considered algorithms.

In the first set of experiments, we tested the algorithms aiming at evaluating their performance when the confidence varies, as well as aiming at tuning them up. For

TABLE 2
Comparison of PPM Algorithms

scheme in [26]				k**-order PPM**			
C	**A**	**U**	**T**	**C**	**A**	**U**	**T**
0.10	0.10	0.08	1.8	0.17	0.17	0.13	1.8
0.15	0.10	0.03	1.3	0.37	0.23	0.07	1.3
0.20	0.15	0.05	1.3	0.47	0.64	0.20	1.3

scheme in [26]				k**-order PPM**			
C	**A**	**U**	**T**	**C**	**A**	**U**	**T**
0.10	0.10	0.08	1.8	0.17	0.17	0.13	1.8
0.15	0.10	0.03	1.3	0.37	0.23	0.07	1.3
0.20	0.11	0.01	1.3	0.47	0.21	0.07	1.3

Left: $meanNoise = 1.0$. *Right:* $meanNoise = 2.5$.

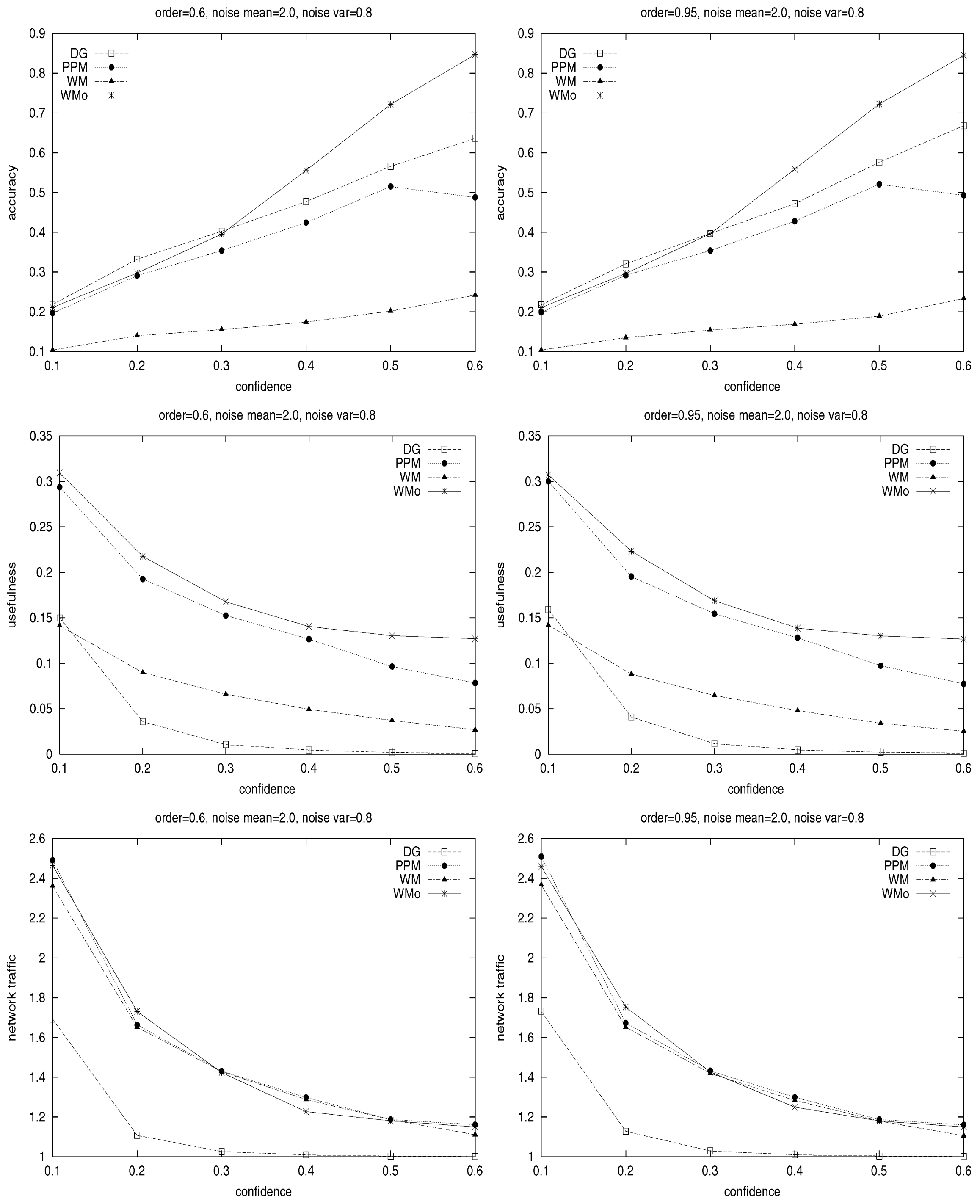


Fig. 4. Performance as a function of confidence for $order = 60\%$ (left) and $order = 95\%$ (right).

this set of experiments, we kept the noise relatively low, with a mean equal to 2.0 and variance equal to 0.8. This combination of values does not change the transactions with respect to their pattern paths very much, simulating a "neutral environment" for all algorithms. Keeping the noise low, we examined the algorithms for two different values of *order*, namely, 0.60 and 0.95. The results of this set of experiments are reported in Fig. 4. From these figures, as expected, it can be seen that usefulness and network traffic decrease with increasing confidence, whereas accuracy steadily increases. The decreasing usefulness asymptotically approaches a lowest value, which corresponds to the documents that are very accurately predicted and are always prefetched. The low network traffic corresponding to this value indicates that the number of these documents is rather small. The striking characteristic of these diagrams is the low performance the *DG* achieves. Regarding the rest of the algorithms, we can see that WM_o clearly outperforms the other two in terms of accuracy, and also in terms of usefulness for lower values of *order*.

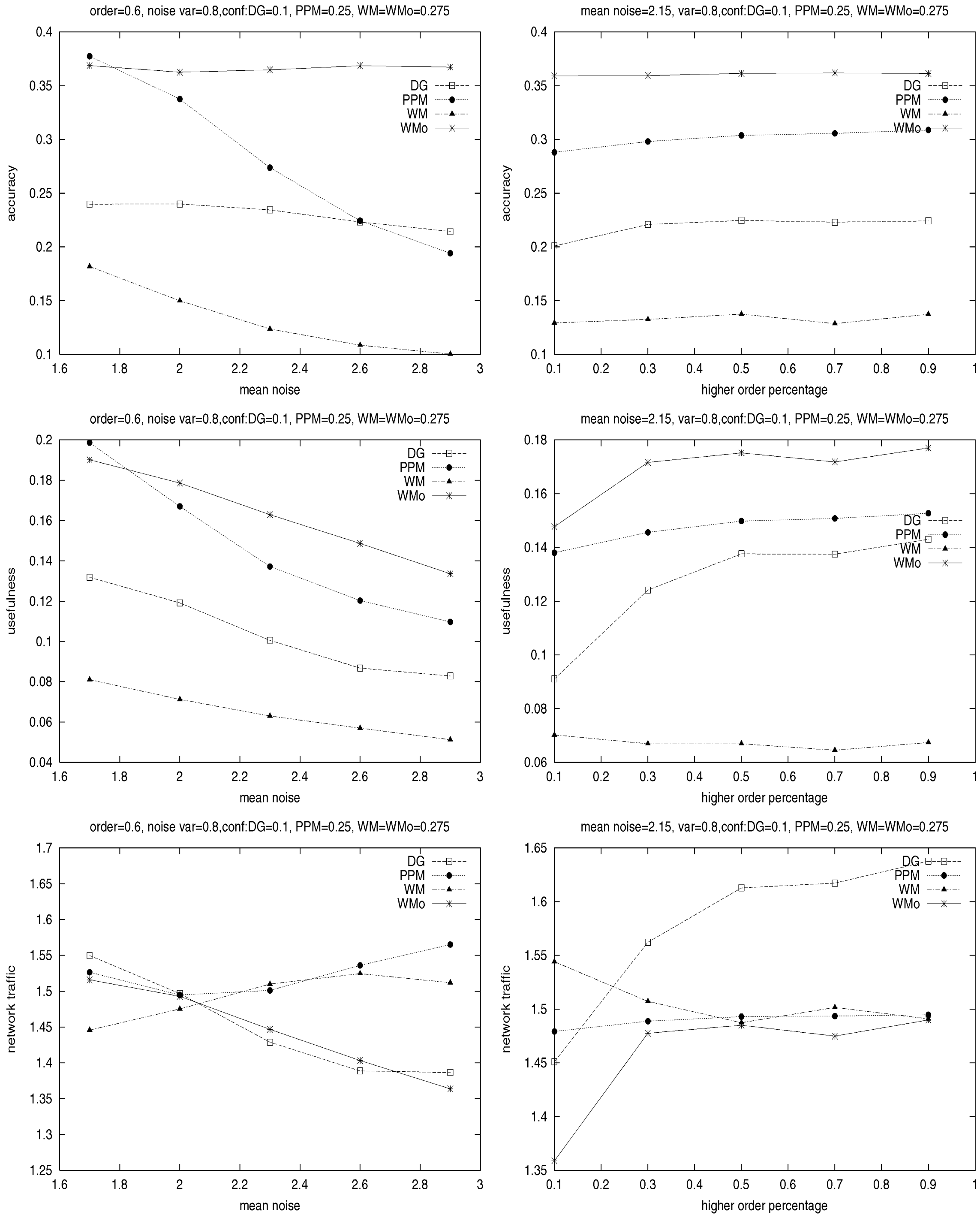


Fig. 5. Performance as a (left) function of noise and (right) $order$.

The second set of experiments assessed the impact of noise on the performance of the prediction schemes. For this set of experiments, we selected a confidence value such that each scheme incurred about 50 percent overhead in network traffic. The confidence value for *DG* was set equal to 0.10, for *PPM* equal to 0.25 and for the other two methods equal to 0.275. The results of this set of experiments are reported in the left part of Fig. 5. From these figures, it can be seen that WM_o clearly outperforms all algorithms in terms of accuracy and usefulness for the same network traffic. It is better than *PPM* by as much as 40 percent and than *DG* by as much as 100 percent, in terms of accuracy. *PPM* is better than WM_o for small values of noise, but it achieves this in expense of larger network traffic. It can also be seen that the accuracy of WM_o is not affected by the increasing noise at all, implying that WM_o makes correct predictions even at the presence of high noise. The usefulness for all algorithms drops with noise. In particular, the usefulness of *PPM* drops more steeply for noise mean values of up to 2.3. Beyond this value, usefulness for *PPM* seems to be stabilized, but this is achieved at the expense of higher network traffic.

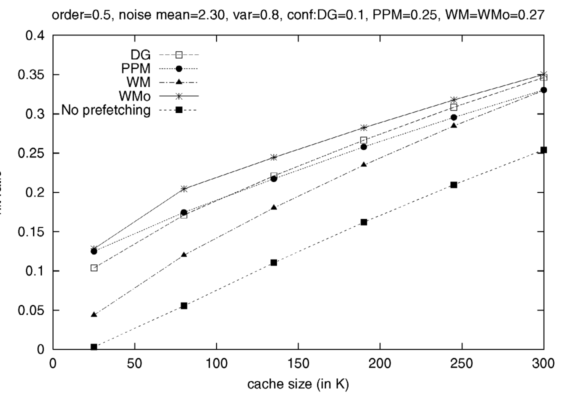


Fig. 6. Cache hits as a function of the cache size.

The third set of experiments evaluated the impact of the varying *order* on the performance of the methods. For this set of experiments, the confidence value for each method was the same as in the last set, whereas the mean value and variance of noise was set to 2.15 and 0.8, respectively. The results of this set of experiments are reported in the right part of Fig. 5. The general result is that only *DG* is affected from the varying *order*, since in order to keep its usefulness and accuracy in the same values, it increases its network traffic. The rest of the algorithms seem insensitive to the varying *order* with WM_o performing the best among them, in terms of both accuracy and usefulness.

Next, we evaluated the benefits of the prefetching algorithms for an LRU cache and compared it with the performance of the same cache with no prefetching at all. For this experiment, the range of cache size was selected to be in the range of a few hundred KBytes, to simulate the fact that not all, but only a small part of the Web client cache is "dedicated" to the Web server documents. The results of this experiment are reported in Fig. 6. From this figure, it is clear that prefetching is beneficial, helping a cache to improve its hit ratio by as much as 50 percent. The figure shows that the hit ratio increases steadily. This is due to the fact that, when the cache size becomes large enough to hold all the site documents, then any future reference will be satisfied by the cache and its hit ratio will approach 100 percent. The same figure shows that interference due to cache does not "blur" the relative performance of the prefetching algorithms. Therefore, WM_o outperforms all other algorithms. The performance gap would be wider in environments with a higher noise and higher order of dependencies between the accesses. For small cache sizes, WM_o, *PPM*, and *DG* have similar performance because for these sizes, the cache is not large enough to hold all prefetched documents and, thus, many of them are replaced before they can be used. On the other hand, for very large cache sizes, the performance of all algorithms converges since almost all the site documents are cached, as explained before.

6.4 Real Data Sets

We conclude the evaluation of the prefetching algorithms by reporting on some experiments conducted using real Web server traces. In the following, due to space limitations, we present only the results obtained from one server trace, namely, the *ClarkNet*, available from the site http://ita.ee.lbl.gov/html/traces.html. We used the first week of requests and we cleansed the log (e.g., by removing CGI scripts, staled requests, etc.). The user session time was set to six hours and 75 percent of the resulted transactions were used for training. The average transaction length was seven, with the majority of the transactions having length smaller than five, so we set the order of the *PPM* prefetcher to five and the lookahead window of the *DG* prefetcher to five. For WM_o, we turned off the structure-based pruning criterion since the site structure for this data set is not provided (however, for a real case application, the site structure is easily obtainable).

Table 3 presents the results from this experiment. The measurements were made so that the network traffic incurred was the same for all algorithms. As it is illustrated, WM_o achieves better performance than all the other algorithms, in all cases, in terms of accuracy and usefulness. This also verifies the performance results obtained from synthetic data.

6.5 Efficiency of WM_o

Finally, we examined the effectiveness of the pruning criterion, that is described in Section 5.3. We use a synthetic data set with the same characteristics as the ones used in the experiments of Section 6.3. This experiment compares the proposed WM_o algorithm with a version that does not use pruning with respect to the site structure, and is denoted as $WM_{o/wp}$ (WM_o without pruning). Moreover, we examined *WM* algorithm (Apriori algorithm for basket data), in order to provide a comparative evaluation of the result. Fig. 7 illustrates the number of candidates for these methods with respect to the support threshold (given as a percentage). As it is depicted, WM_o significantly outperforms both $WM_{o/wp}$ and *WM* for lower support thresholds, where the number of candidates is larger. For larger support thresholds, WM_o still outperforms $WM_{o/wp}$ and presents a comparative number of candidates with those produced by *WM*. Nevertheless, as shown by the previous experiments, the performance of *WM* for the overall prefetching purpose is significantly lower than that of WM_o.

The number of candidates significantly impacts the performance of this type of algorithms. This is in accordance with related work on association rule mining [3]. Therefore, the efficiency of WM_o is improved by the proposed pruning. Moreover, pruning is efficiently applied by considering the site structure in every step of the rule discovery algorithm (see Section 5.3). However, due to

TABLE 3
Comparison of Prefetchers with Real Data

	DG (w=5)		**5-order PPM**		**WM**		**WM$_o$**	
T	**A**	**U**	**A**	**U**	**A**	**U**	**A**	**U**
1.9	0.13	0.12	0.19	0.17	0.07	0.06	0.20	0.18
1.8	0.16	0.13	0.20	0.17	0.05	0.04	0.22	0.17
1.7	0.19	0.13	0.22	0.14	0.06	0.04	0.24	0.17
1.6	0.16	0.11	0.23	0.13	0.07	0.04	0.27	0.17
1.5	0.11	0.06	0.25	0.12	0.08	0.04	0.29	0.15

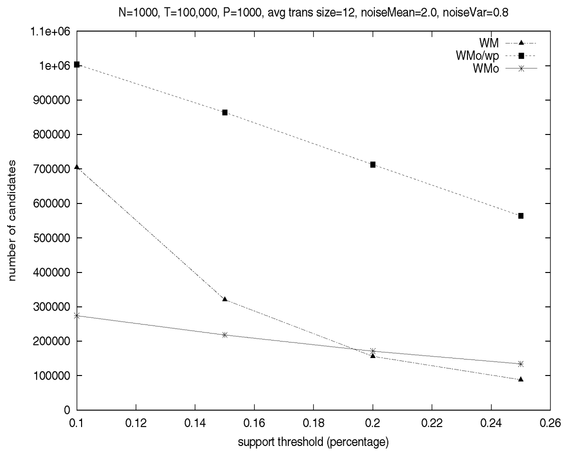


Fig. 7. Number of candidates with regard to support threshold.

space restrictions, a detailed description of this subject and extensive experimental results on the execution time of this procedure can be found in [30].

6.6 Discussion

From the experimental results, it is evident that WM_o, which is proposed as a generalization of existing algorithms, clearly presents the best performance. More precisely, WM_o prefetches more documents correctly than *PPM* and *DG* (i.e., shows better usefulness and accuracy) and this is achieved without additional cost in the network traffic. In the few cases where *PPM* performs slightly better, this happens at the cost of network traffic. At the same time, the accuracy of prefetching is constantly higher for WM_o compared to the other algorithms. In general, algorithm *WM* presents the worst performance because it does not consider the ordering of accesses and for this reason, it is not commented any further.

With respect to the two factors, order and noise, experiments indicate that *PPM* is affected by noise, whereas *DG* and WM_o are not. The increased noise significantly reduces the accuracy of *PPM*. On the other hand, its usefulness is also reduced and, at the same time, an increase in the network traffic is observed (which prevents a higher deterioration in usefulness). *DG* is sensitive to the factor of order, whereas *PPM* and WM_o are not. Additionally, the measurement of network traffic with respect to the factor of order verifies that the *PPM* algorithm uses a constant maximum value for its rules whereas, the WM_o algorithm dynamically adjusts the order of the rules, according to the order in the data. Thus, *PPM* requires a constant network traffic overhead with respect to the values of order (see Fig. 5). Additionally, experiments with real traces verify the previously described conclusions.

7 Conclusions

We considered the problem of predictive Web prefetching, that is, deriving users' future requests for Web documents based on their previous requests. Predictive prefetching suits the Web's hypertextual nature and significantly reduces the perceived latency.

We presented a new context for the interpretation of Web prefetching algorithms as Markov predictors. This context revealed the important factors that affect the performance of Web prefetching algorithms. The first factor is the order of dependencies between Web document accesses. The second is the interleaving of requests belonging to patterns with random ones within user transactions and the third one is the ordering of requests. None of the existing approaches has considered the aforementioned factors altogether.

We proposed a new algorithm called WM_o, which has proven to be a generalization of existing ones. It was designed to address their specific limitations and its characteristics include all the above factors. It compares favorably with previously proposed algorithms, like *PPM*, *DG*, and existing approaches from the application of Web log mining to Web prefetching. Further, the algorithm efficiently addresses the increased number of candidates.

Using a synthetic data generator, the performance of WM_o was compared against that of *PPM* and *DG*. Our experiments showed that, for a variety of order and noise distributions, WM_o clearly outperforms the existing algorithms. In fact, for many workloads WM_o achieved large accuracy in prediction with quite low overhead in network traffic. Additionally, WM_o proved to be very efficient in terms of reducing the number of candidates. These results were validated with experiments using real Web traces. In summary, WM_o is an effective and efficient predictive Web prefetching algorithm.

Acknowledgements

The authors thank the anonymous reviewers for suggesting several important improvements and, in particular, the first reviewer for his helpful comments. Intermediate results of this work were presented at the WebKDD '01 workshop ("Mining Log Data Across All Customer TouchPoints"). Work was supported by a national PABE project and an INTAS project (OPEN 97-11106).

References

[1] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira, "Characterizing Reference Locality in the WWW," *Proc. IEEE Conf. Parallel and Distributed Information Systems (IEEE PDIS '96)*, pp. 92-103, Dec. 1996.

[2] P. Atzeni, G. Mecca, and P. Merialdo, "To Weave the Web," *Proc. 23rd Conf. Very Large Data Bases (VLDB '97)*, pp. 206-215, Aug. 1997.

[3] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. 20th Conf. Very Large Data Bases (VLDB '94)*, pp. 487-499, Sept. 1994.

[4] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. IEEE Conf. Data Eng. (IEEE ICDE '95)*, pp. 3-14, Mar. 1995.

[5] C. Aggarwal, J. Wolf, and P. S. Yu, "Caching on the World Wide Web," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 1, pp. 95-107, Jan./Feb. 1999.

[6] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *Proc. ACM Conf. Measurement and Modeling of Computer Systems, (ACM SIGMETRICS '98)*, pp. 151-160, June 1998.

[7] A. Bestavros, "Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time," *Proc. IEEE Conf. Data Eng. (IEEE ICDE '96)*, pp. 180-189, Feb. 1996.

[8] B. Berendt and M. Spiliopoulou, "Analysis of Navigation Behavior in Web Sites Integrating Multiple Information Systems," *The VLDB J.*, vol. 9, no. 1, pp. 56-75, May 2000.

[9] M. Crovella and P. Barford, "The Network Effects of Prefetching," *Proc. IEEE Conf. Computer Comm. (IEEE INFOCOM '98)*, pp. 1232-1240, Mar. 1998.

[10] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," *Proc. 1997 USENIX Symp. Internet Technologies and Systems (USITS '97)*, pp. 193-206, Jan. 1997.

[11] E. Cohen, B. Krishnamurthy, and J. Rexford, "Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters," *Proc. ACM Conf. Applications, Technologies, Architectures and Protocols for Computer Comm. (ACM SIGCOMM '98)*, pp. 241-253, Aug. 1998.

[12] K.M. Curewitz, P. Krishnan, and J.S. Vitter, "Practical Prefetching via Data Compression," *Proc. ACM Conf. Management of Data (ACM SIGMOD '93)*, pp. 257-266, June 1993.

[13] P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the World Wide Web," *IEEE Trans. Computers*, vol. 47, no. 4, pp. 445-457, Apr. 1998.

[14] R. Cooley, B. Mobasher, and J. Srivastava, "Data Preparation for Mining World Wide Web Browsing Patterns," *Knowledge and Information Systems (KAIS)*, vol. 1, no. 1, pp. 5-32, Feb. 1999.

[15] M.S. Chen, J.S. Park, and P.S. Yu, "Efficient Data Mining for Path Traversal Patterns," *IEEE Trans. Knowledge and Data Eng.*, vol. 10, no. 2, pp. 209-221, Apr. 1998.

[16] P. Cao, J. Zhang, and K. Beach, "Active Cache: Caching Dynamic Contents on the Web," *Proc. IFIP Conf. Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, pp 373-388, Sept. 1998.

[17] M. Deshpande and G. Karypis, "Selective Markov Models for Predicting Web-Page Accesses," *Proc. SIAM Int'l Conf. Data Mining (SDM '01)*, Apr. 2001.

[18] D. Duchamp, "Prefetching Hyperlinks," *Proc. USENIX Symp. Internet Technologies and Systems (USITS '99)*, Oct. 1999.

[19] L. Fan, P. Cao, W. Lin, and Q. Jacobson, "Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance," *Proc. ACM Conf. Measurement and Modeling of Computer Systems, (ACM SIGMETRICS '99)*, pp. 178-187, June 1999.

[20] M.F. Fernandez, D. Florescu, A.Y. Levy, and D. Suciu, "Declarative Specification of Web Sites with Strudel," *The VLDB J.*, vol. 9, no. 1, pp. 38-55, May 2000.

[21] J. Griffioen and R. Appleton, "Reducing File System Latency Using a Predictive Approach," *Proc. 1994 USENIX Ann. Technical Conf. (USENIX '95)*, pp. 197-207, Jan. 1995.

[22] B. Huberman, P. Pirolli, J. Pitkow, and R. Lukose, "Strong Regularities in World Wide Web Surfing," *Science*, vol. 280, pp. 95-97, Apr. 1998.

[23] S. Jin and A. Bestavros, "Sources and Characteristics of Web Temporal Locality," *Proc. IEEE/ACM Symp. Modeling, Analysis and Simulation of Computer and Telecomm. Systems (MASCOTS '2000)*, Aug. 2000.

[24] R. Klemm, "WebCompanion: A Friendly Client-Side Web Prefetching Agent," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 4, pp. 577-594, July/Aug. 1999.

[25] T. Kroeger, D.E. Long, and J. Mogul, "Exploring the Bounds of Web Latency Reduction from Caching and Prefetching," *Proc. USENIX Symp. Internet Technologies and Systems (USITS '97)*, pp. 13-22, Jan. 1997.

[26] B. Lan, S. Bressan, B.C. Ooi, and Y. Tay, "Making Web Servers Pushier," *Proc. Workshop Web Usage Analysis and User Profiling (WEBKDD '99)*, Aug. 1999.

[27] B. Lan, S. Bressan, B.C. Ooi, and K. Tan, "Rule-Assisted Prefetching in Web-Server Caching," *Proc. ACM Int'l Conf. Information and Knowledge Management (ACM CIKM '00)*, pp. 504-511, Nov. 2000.

[28] H. Mannila, H. Toivonen, and I. Verkamo, "Discovery of Frequent Episodes in Event Sequences," *Data Mining and Knowledge Discovery (DMKD)*, vol. 1, no. 3, pp. 259-289, Sept. 1997.

[29] A. Nanopoulos and Y. Manolopoulos, "Finding Generalized Path Patterns for Web Log Data Mining," *Proc. East European Conf. Advances in Databases and Information Systems (ADBIS '00)*, pp. 215-228, Sept. 2000.

[30] A. Nanopoulos and Y. Manolopoulos, "Mining Patterns from Graph Traversals," *Data and Knowledge Eng. (DKE)*, vol. 37, no. 3, pp. 243-266, June 2001.

[31] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu, "Mining Access Patterns Efficiently from Web Logs," *Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD '00)*, Apr. 2000.

[32] H. Patterson, G. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed Prefetching and Caching," *Proc. ACM Symp. Operating Systems Principles (ACM SOSP '95)*, pp. 79-95, Dec. 1995.

[33] V. Padmanabhan and J. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency," *ACM SIGCOMM Computer Comm. Rev.*, vol. 26, no. 3, July 1996.

[34] T. Palpanas and A. Mendelzon, "Web Prefetching Using Partial Match Prediction," *Proc. Fourth Web Caching Workshop (WCW '99)*, Mar. 1999.

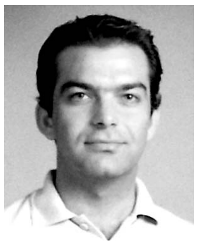
[35] J. Pitkow and P. Pirolli, "Mining Longest Repeating Subsequences to Predict World Wide Web Surfing," *Proc. USENIX Symp. Internet Technologies and Systems (USITS '99)*, Oct. 1999.

[36] R. Sarukkai, "Link Prediction and Path Analysis Using Markov Chains," *Computer Networks*, vol. 33, nos. 1-6, pp. 377-386, June 2000.

[37] J. Shim, P. Scheuermann, and R. Vingralek, "Proxy Cache Algorithms: Design, Implementation, and Performance," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 4, pp. 549-562, Aug. 1999.

[38] Z. Wang and J. Crowcroft, "Prefetching in World Wide Web," *Proc. IEEE Global Internet Conf.*, pp. 28-32, Nov. 1996.

Alexandros Nanopoulos received the bachelor's degree in computer science from the Aristotle University of Thessaloniki, Greece (1996). Currently, he is a PhD candidate in the Computer Science Department at Aristotle University. His research interests include data mining, databases for Web, and spatial databases.



Dimitrios Katsaros received the BSc degree in computer science from the Aristotle University of Thessaloniki, Greece (1997). He was a visiting researcher in the Department of Pure and Applied Mathematics at the University of L'Aquila, Italy. Currently, he is a PhD candidate in the Computer Science Department at Aristotle University. His research interests include Web databases, semistructured data, and mobile data management.

Yannis Manolopoulos received the BEng degree (1981) in electrical engineering and the PhD degree (1986) in computer engineering both from the Aristotle University of Thessaloniki. Currently, he is a professor in the Department of Informatics at Aristotle University. He has been with the Department of Computer Science at the University of Toronto, the Department of Computer Science at the Univertsity of Maryland at College Park, and the University of Cyprus. He has published more than 100 papers in refereed scientific journals and conference proceedings. He is a coauthor of a book entitled *Advanced Database Indexing* by Kluwer. He is also the author of two textbooks on data structures and file structures, which are recommended in the vast majority of the computer science/engineering departments in Greece. His research interests include spatiotemporal databases, data mining, databases for Web, and performance evaluation of storage subsystems. He is a member of the IEEE Computer Society.