2-26-2004

Evaluating Location Predictors with Extensive Wi-Fi Mobility Data

Libo Song
Dartmouth College

David Kotz
Dartmouth College

Ravi Jain
DoCoMo USA Labs

Xiaoning He
DoCoMo USA Labs

Dartmouth Digital Commons Citation

Libo Song, David Kotz, Ravi Jain, and Xiaoning He. Evaluating location predictors with extensive Wi-Fi mobility data. In Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), March 2004. 10.1145/965732.965747

Evaluating next-cell predictors with extensive Wi-Fi mobility data

Libo Song, David Kotz
Dartmouth College
{*lsong,dfk*}*@cs.dartmouth.edu*

Ravi Jain, Xiaoning He
DoCoMo USA Labs
{*jain,xiaoning*}*@docomolabs-usa.com*

Dartmouth College Computer Science Technical Report TR2004-491

February 26, 2004

Abstract

Location is an important feature for many applications, and wireless networks can better serve their clients by anticipating client mobility. As a result, many location predictors have been proposed in the literature, though few have been evaluated with empirical evidence. This paper reports on the results of the first extensive empirical evaluation of location predictors, using a two-year trace of the mobility patterns of over 6,000 users on Dartmouth's campus-wide Wi-Fi wireless network.

We implemented and compared the prediction accuracy of several location predictors drawn from four major families of domain-independent predictors, namely Markov-based, compression-based, PPM, and SPM predictors. We found that low-order Markov predictors performed as well or better than the more complex and more space-consuming compression-based predictors. Predictors of both families fail to make a prediction when the recent context has not been previously seen. To overcome this drawback, we added a simple fallback feature to each predictor and found that it significantly enhanced its accuracy in exchange for modest effort. Thus the Order-2 Markov predictor with fallback was the best predictor we studied, obtaining a median accuracy of about 72% for users with long trace lengths. We also investigated a simplification of the Markov predictors, where the prediction is based not on the most frequently seen context in the past, but the most recent, resulting in significant space and computational savings. We found that Markov predictors with this recency semantics can rival the accuracy of standard Markov predictors in some cases. Finally, we considered several seemingly obvious enhancements, such as smarter tie-breaking and aging of context information, and discovered that they had little effect on accuracy. The paper ends with a discussion and suggestions for further work.

Index Terms

Network measurements, Experimentation with real networks, location-aware computing, location prediction.

I. Introduction

A fundamental problem in mobile computing and wireless networks is the ability to track and predict the location of mobile devices. An accurate location predictor can significantly improve the performance or reliability of wireless network protocols, the wireless network infrastructure itself, and many applications

A shorter, preliminary version of this paper is to appear at IEEE Infocom, March 7-11, 2004. Compared to the preliminary conference version of the paper, this version has been significantly expanded and revised to include several new experiments, results and insights. First, we have experimented with all 6000 traces, far more than the 2000 long traces we focused on previously. Second, we have also implemented two additional recent predictors, PPM and SPM, thus essentially evaluating all well-known families of predictors. Finally, we have also considered coarser-grain prediction (building prediction), a new metric (running median accuracy) to investigate the effect of trace length, and considered the correlation with trace entropy further.

in pervasive computing. These improvements lead to a better user experience, to a more cost-effective infrastructure, or both.

For example, in wireless networks the SC (Shadow Cluster) scheme can provide a consistent QoS support level before and after the handover [LAN95]. In the SC scheme, all cells neighboring a mobile node's current cell are requested to reserve resources for that node, in case it moves to that cell next. Clearly, this approach ties up more resources than necessary. Several proposed SC variations [SG98] attempt to predict the device's movement so the network can reserve resources in only certain neighboring cells.

Location prediction has been proposed in many other areas of wireless cellular networks as a means of enhancing performance, including better mobility management [LJ96], improved assignment of cells to location areas [DS99], more efficient paging [BD02], and call admission control [YL02].

Many pervasive computing applications include opportunities for location-based services and information. With a location predictor, these applications can provide services or information based on the user's next location. For example, consider a university student that moves from dormitory to classroom to dining hall to library. When the student snaps a photo of his classmates using a wireless digital camera and wishes to print it, the camera's printing application might suggest printers near the current location and near the next predicted location, enabling the photo to finish printing while the student walks across campus.

Both of these examples depend on a predictor that can predict the next wireless cell visited by a mobile device, given both recent and long-term historical information about that device's movements. Since an accurate prediction scheme is beneficial to many applications, many such predictors have been proposed, and recently surveyed by Cheng et al. [CJv03].

To the best of our knowledge, no other researchers have evaluated location predictors with extensive mobility data from real users. Das et al. use two small user mobility data sets to verify the performance of their own prediction schemes, involving a total of five users [DCB+02].

In this paper we compare the most significant domain-independent predictors using a large set of user mobility data collected at Dartmouth College. In this data set, we recorded for two years the sequence of wireless cells (Wi-Fi access points) frequented by more than 6000 users. It is important to note that the data records the registration of devices to access points, and does not directly record the physical movement of human users. Thus, for example, a mobile device located at the boundary between two cells may register alternately with one access point and then the other in response to changing radio conditions, even if the user does not change locations. Nonetheless, from the point of view of many system applications,

such as those attempting to improve mobility or QoS management, the user's physical location may not matter; more important is the cell in which the device is located and consuming resources. In this paper we focus on predicting the next cell visited by the mobile device.

After providing more background on the predictors and our accuracy metric, we present the detailed results we obtained by running each predictor over each user's trace. We found that even the simplest classical predictors can obtain a median prediction accuracy of about 63% over all users, and about 72% over all users with sufficiently long location histories (1000 cell crossings or more), although accuracy varied widely from user to user. This performance may be sufficient for many useful applications. On the other hand, many applications will need higher accuracy.

We found that the simple Markov predictors performed as well or better than the more complicated LZ, PPM, and SPM predictors, with smaller data structures. We also found that many predictors often fail to make any prediction, but our simple fallback technique provides a prediction and improves overall accuracy. We conclude the paper with a more extensive summary of our conclusions.

II. Background

In this section, we discuss the nature of next-cell prediction, and summarize several predictors from the literature. We define the metrics we used to evaluate the predictors, and how we collected the empirical data set we used in our evaluation.

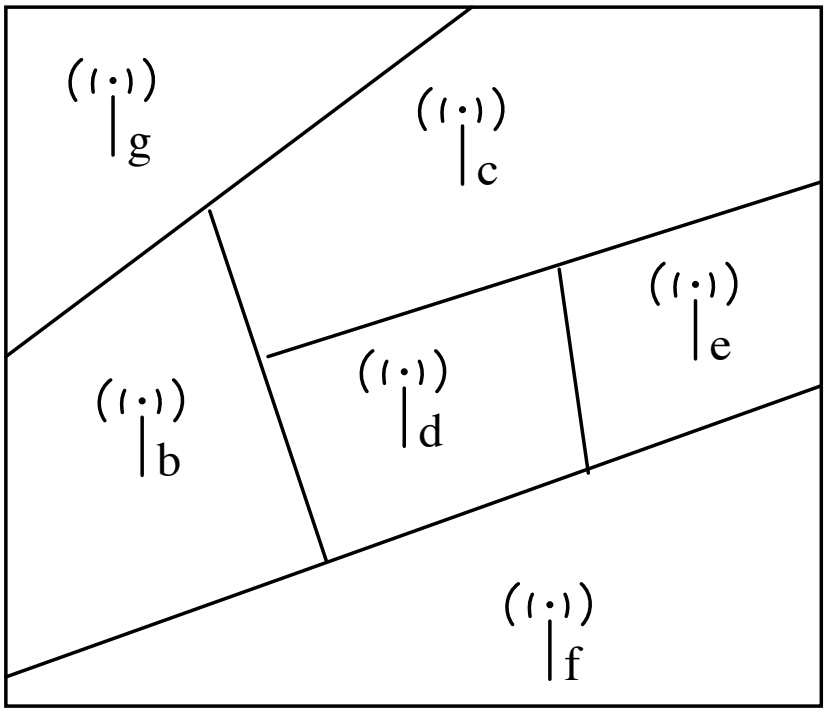
A. Location

In the context of this work we assume that a user resides at a given discrete location at any given time; sample locations include "room 116" within a building, "Thayer Dining Hall" within a campus, "cell 24" within a cellular network, or "berry1-ap" access point within an 802.11 wireless network.

In our data, as we discuss below, we have available the name of the access point with which the user's device is associated. We also have the geographical location of each access point, and the name of the building the access point is located in (or the nearest building). For simplicity we call the access point where the user's device is registered a *location*, although this should not be taken to imply that we know the user's precise geographical location in, say, latitude and longitude.

We list all possible locations in a finite alphabet $\mathcal{A}$, and can identify any location as a symbol a drawn from that alphabet. For a given user we list the sequence of locations visited, its *location history* L, as a string of symbols. If the history has n locations, $L_n = a_1 a_2 \ldots a_n$.

The location history may be a sequence of location *observations*, for example, the user's location recorded once every five seconds, or a sequence of location *changes*. In the latter case, $a_i \neq a_{i+1}$ for all



Sample location history $L = gbdcbgcefbdbde$

Figure 1. Sample cell map and location history

$0 < i < n$. All of the predictors we consider in this paper are agnostic to this issue. It happens that our data is a sequence of location changes. We refer to a location change as a *move*.

All of the predictors we consider in this paper are domain-independent and operate on the string L as a sequence of abstract symbols. They do not place any interpretation on the symbols. For that reason, our location history does not include any timing information, or require any associated information relating the symbols such as geographic coordinates. As an example, though, consider the environment with six wireless cells (labeled b through g) diagrammed in Figure 1, accompanied by one possible location history.

B. Domain-independent predictors

In this paper we consider only domain-independent predictors; we will examine domain-dependent predictors in future work. We are interested in *on-line predictors*, which examine the history so far, extract the current context, and predict the next location. Once the next location is known, the history is now one symbol longer, and the predictor updates its internal tables in preparation for the next prediction.

During the rest of this section, we discuss two families of domain-independent predictors, Order-k

Markov predictors and LZ-based predictors. We also discuss two more recent predictors, PPM and SPM.

Markov family

The order-k (or "$O(k)$") Markov predictor assumes that the location can be predicted from the current *context*, that is, the sequence of the k most recent symbols in the location history ($a_{n-k+1}, \ldots, a_n$). The underlying Markov model represents each state as a context, and transitions represent the possible locations that follow that context.

Consider a user whose location history is $L = a_1 a_2 \ldots a_n$. Let substring $L(i,j) = a_i a_{i+1} \ldots a_j$ for any $1 \leq i \leq j \leq n$. We think of the user's location as a random variable X. Let $X(i,j)$ be a string $X_i X_{i+1} \ldots X_j$ representing the sequence of random variates $X_i, X_{i+1}, \ldots X_j$ for any $1 \leq i \leq j \leq n$. Define the context $c = L(n-k+1, n)$. Let $\mathcal{A}$ be the set of all possible locations. The Markov assumption is that X behaves as follows, for all $a \in \mathcal{A}$ and $i \in \{1, 2, \ldots, n\}$.

$$\begin{aligned} &P(X_{n+1} = a | X(1,n) = L) \\ &\quad = P(X_{n+1} = a | X(n-k+1, n) = c) \\ &\quad = P(X_{i+k+1} = a | X(i+1, i+k) = c) \end{aligned}$$

where the notation $P(X_i = a_i | \ldots)$ denotes the probability that X_i takes the value a_i. The first two lines indicate the assumption that the probability depends only on the context of the k most recent locations. The latter two lines indicate the assumption of a stationary distribution, that is, that the probability is the same anywhere the context is the same.

These probabilities can be represented by a *transition probability matrix* M. Both the rows and columns of M are indexed by length-k strings from $\mathcal{A}^k$ so that $P(X_{n+1} = a | X(1,n) = L(1,n)) = M(s, s')$ where $s = L(n-k+1, n)$, the current context, and $s' = L(n-k+2, n)a$, the next context. In that case, knowing M would immediately provide the probability for each possible next symbol of L.

Since we do not know M, we can generate an estimate $\widehat{P}$ from the current history L, the current context c, and the equation

$$\widehat{P}(X_{n+1} = a | L) = \frac{N(ca, L)}{N(c, L)} \tag{1}$$

where $N(s', s)$ denotes the number of times the substring s' occurs in the string s.

Given this estimate, we can easily define the behavior of the $O(k)$ Markov predictor. It predicts the symbol $a \in \mathcal{A}$ with the maximum probability $\widehat{P}(X_{n+1} = a | L)$, that is, the symbol that most frequently followed the current context c in prior occurrences in the history. Notice that if c has never occurred before, the above equation evaluates to $0/1 = 0$ for all a, and the $O(k)$ Markov predictor makes no prediction.

If the location history is not generated by an order-k Markov source, then this predictor is, of course, only an approximation.

Fortunately, $O(k)$ Markov predictors are easy to implement. Our implementation maintains an estimate of the (sparse) matrix M, using Equation 1. To make a prediction, the predictor scans the row of M corresponding to the current context c, choosing the entry with the highest probability for its prediction. After the next move occurs, the predictor updates the appropriate entry in that row of M, and updates c, in preparation for the next prediction.

Vitter and Krishnan [VK96] use $O(k)$ predictors to prefetch disk pages, and prove interesting asymptotic properties of these predictors. Other variations of Markov predictors can be found in the survey [CJv03].

LZ family

LZ-based predictors are based on a popular incremental parsing algorithm by Ziv and Lempel [ZL78] often used for text compression. This approach seems promising because (a) most good text compressors are good predictors [VK96] and (b) LZ-based predictors are like the $O(k)$ Markov predictor except that k is a variable allowed to grow to infinity [BD02]. We briefly discuss how the LZ algorithm works.

Let γ be the empty string. Given an input string s, the LZ parsing algorithm partitions the string into distinct substrings $s_0, s_1, \ldots, s_m$ such that $s_0 = \gamma$, and for all $j > 0$ substring s_j without its last character is equal to some $s_i, 0 \leq i < j$, and $s_0 s_1 \ldots s_m = s$. Observe that the partitioning is done sequentially, i.e., after determining each s_i the algorithm only considers the remainder of the input string. Using the example from Figure 1, $s = gbdcbgcefbdbde$ is parsed as $\gamma, g, b, d, c, bg, ce, f, bd, bde$.

Associated with the algorithm is a tree, the *LZ tree*, that is grown dynamically during the parsing process. Each node of the tree represents one substring s_i. The root is labeled γ and the other nodes are labeled with a symbol from $\mathcal{A}$ so that the sequence of symbols encountered on the path from the root to that node forms the substring associated with that node. Since this LZ tree is to be used for prediction, it is necessary to store some statistics at each node. The tree resulting from parsing the above example is shown in Figure 2; each node is labeled with its location symbol and the value of its statistic, a counter, after parsing.

To parse a (sub)string s we trace a path through the LZ tree. If any child of the current node (initially the root) matches the first symbol of s, remove that symbol from s and step down to that child, incrementing its counter; continue from that node, examining the next symbol from s. If the symbol did not match any child of the current node, then remove that symbol from s and add a new child to the current node, labeled with that symbol and counter=1; resume parsing at the root, with the now shorter string s.

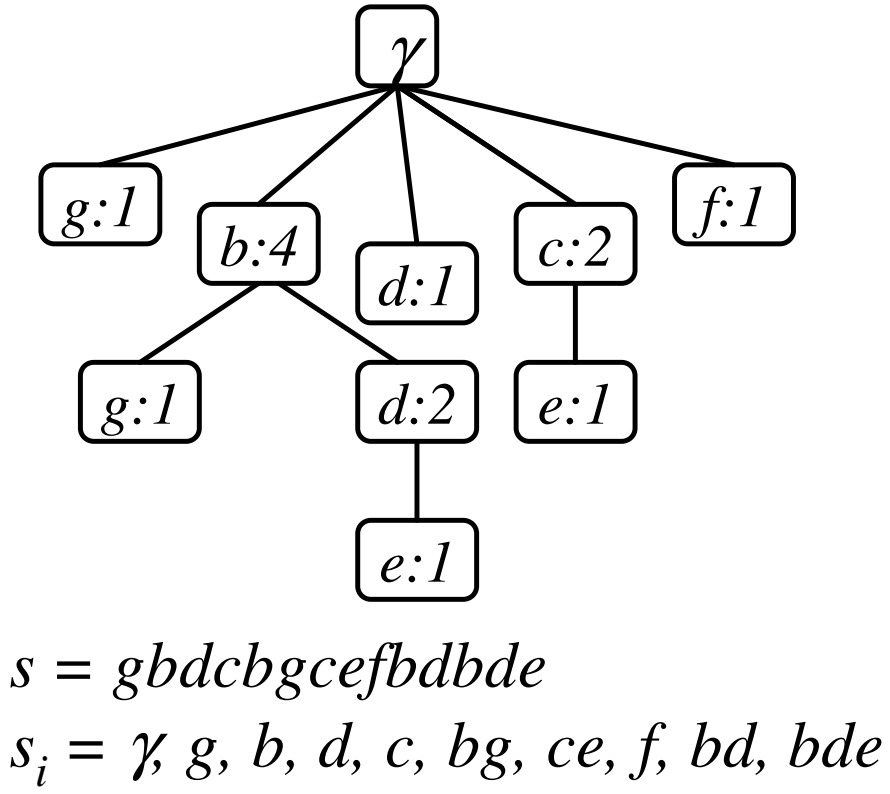


Figure 2. Example LZ parsing tree

Based on the LZ parsing algorithm, several predictors have been suggested in the past [VK96], [KV98], [FMG92], [BD02], [YL02]. We describe some of these below and then discuss how they differ. For more detailed information, please refer to [CJv03].

LZ predictors. Vitter and Krishnan [VK96] considered the case when the generator of L is a finite-state Markov source, which produces sequences where the next symbol is dependent on only its *current state*. (We note that a finite-state Markov source is different from the $O(k)$ Markov source in that the states do not have to correspond to strings of a fixed length from $\mathcal{A}$.) They suggested estimating the probability, for each $a \in \mathcal{A}$, as

$$\widehat{P}(X_{n+1} = a|L) = \frac{N^{LZ}(s_m a, L)}{N^{LZ}(s_m, L)} \tag{2}$$

where $N^{LZ}(s', s)$ denotes the number of times s' occurs as a prefix among the substrings $s_0, \ldots, s_m$ which were obtained by parsing s using the LZ algorithm.

It is worthwhile comparing Equation (1) with Equation (2). While the former considers how often the string of interest occurs in the entire input string, the latter considers how often it occurs in the partitions s_i created by LZ. Thus, in the example of Figure 2, while *dc* occurs in L it does not occur in any s_i.

The LZ predictor chooses the symbol a in $\mathcal{A}$ that has the highest probability estimate, that is, the highest value of $\widehat{P}(X_{n+1} = a|L)$. An on-line implementation need only build the LZ tree as it parses the string, maintaining node counters as described above. After parsing the current symbol the algorithm rests at a node in the tree. It examines the children of the current node, if any, and predicts the symbol labeling the child with the highest counter, which indicates the highest frequency of past occurrence. If the current node is a leaf, the LZ predictor makes no prediction.

LZ-Prefix and PPM. Bhattacharya and Das propose a heuristic modification to the construction of the LZ tree [BD02], as well as a way of using the modified tree to predict the most likely cells that contain the user so as to minimize paging costs to locate that user.

As we mention above, not every substring in L forms a node s_i in the LZ parsing tree. In particular, substrings that cross boundaries of the s_i, $0 < i \leq m$, are missed. Further, previous LZ-based predictors take into account only the occurrence statistics for the prefixes of the leaves. To overcome this, they proposed the following modification. When a new leaf is created for s_i, all the proper suffixes of s_i (i.e., all the suffixes not including s_i itself) are also inserted into the tree. If a node representing a suffix does not exist, it is created; and the occurrence counter for every prefix of every suffix is incremented.

Example. Suppose the current leaf is $s_m = bde$ and the string de is one that crosses boundaries of existing s_i for $1 \leq i < m$. Thus de has not occurred as a prefix or a suffix of any s_i, $0 < i < m$. The set of proper suffixes of s_m is $S_m = \{\gamma, e, de\}$, and since there is no node representing the substring for de, it is created. Then the occurrence frequency is incremented for the root labeled γ, the first-level children b and d, and the new leaf node de. Figure 3 shows the tree after this transformation, which we call "LZ+prefix" or "LZP" for short.

We observe that this heuristic only discovers substrings that lie within a leaf string. Nonetheless, at this point it is possible to use the modified LZ tree and apply one of the existing prediction heuristics, e.g., use Eq. 2 and the Vitter-Krishnan method. Indeed, we include the LZP predictor in our comparison.

Bhattacharya and Das [BD02] propose a further heuristic that uses the LZP tree for prediction. This second heuristic is based on the Prediction by Partial Match (PPM) algorithm for text compression [BCW90]. (The PPM algorithm essentially attempts to "blend" the predictions of several $O(k)$ Markov predictors; we describe it briefly below.) Given a leaf string s_m, construct its set of proper suffixes S_m and update the LZP tree as described above. Then, for each such suffix, the heuristic considers the subtree rooted at the suffix and finds all the paths in this subtree originating from this subtree root. The PPM algorithm is then applied to this set of paths. PPM first computes the predicted probability of each path, and then uses these probabilities to compute the most probable next symbol(s) based on their weights (number of

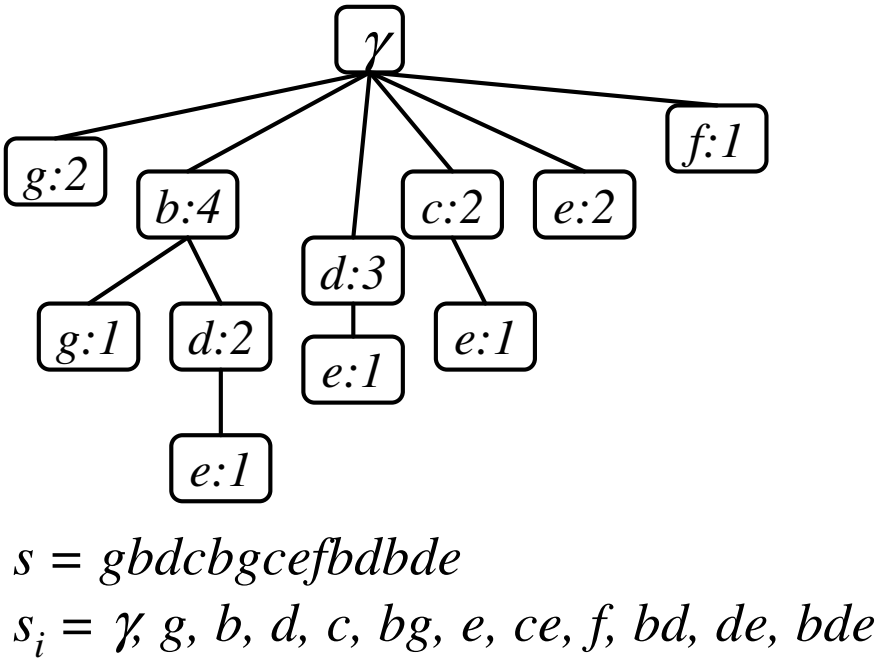


Figure 3. Example LZP parsing tree

occurrences) in the path. We include the "LZ+prefix+PPM" predictor, nicknamed "LeZi" [BD02], in our comparison.

PPM

Prediction by Partial Matching (PPM) is a data compression scheme, often used in text compression [CW84]. As with Markov predictors, the basic idea of PPM is to use the last few symbols in the input stream to predict the next one. Order-k PPM uses the k immediately preceding symbols to update the model for prediction. The PPM predictor blends a set of different order context models, from 0 to k. Each fixed-order context model is built as an order-k Markov model. The combination of the different models are achieved through the use of "escape" probabilities. The probability of encountering a previously unseen symbol is called the escape probability. There are three major methods to determine the escape probabilities [BCW90], although we do not have space here to describe them. The method used in our implementation is called "Method C," which gives a count to the escape event equal to the number of different symbols that have been seen in the context so far. Thus the escape probability is the escape count divided by the sum of all counts of symbols that have been seen in the context and the escape count. If E_k is the escape probability, then for all $a \in \mathcal{A}$, the order-k PPM probability is

$$P(x_n + 1 = a|L) = P_k(a) + \sum_{i=1}^{k} P_{i-1}(a)E_i, \tag{3}$$

where $P_k(a)$ is the probability of a being the next symbol, as predicted by the order-k Markov model.

SPM

Jacquet, Szpankowski and Apostol [JSA00] proposed a predictor called the *Sampled Pattern Matching Algorithm* (SPM). This algorithm is of interest as it considers much larger classes of sources (in our case, movement traces) than $O(k)$ Markov predictors. In particular, it addresses what are called strongly mixing sources, which contain fixed-order Markov sources as a special case. SPM was shown theoretically to be asymptotically optimal, that is, to have a prediction accuracy the same as the best possible predictor, for this class of sources. SPM is similar to $O(k)$ Markov; rather than using a fixed context length k, the length is determined by a fixed fraction α of the longest context that has been previously seen, as described below.

Start by finding the longest suffix $X_{n-t+1}X_{n-t+2}\ldots X_n$ that has occurred previously in the sequence $X_1X_2\ldots X_n$; this is called the maximal suffix. Here, n is the length of the whole sequence, and t is the length of the maximal suffix. Instead of considering the entire maximal suffix, however, the algorithm considers only a fraction of the suffix. Choose a fixed constant α, where $0 < \alpha < 1$. The suffix of interest is now $c = X_{n-k+1}, X_{n-k+2}, \ldots, X_n$ where $k = \lceil \alpha t \rceil$. The next predicted character is

$$\text{argmax}_{a \in A} N(ca, L),$$

where $N(s, L)$ is the number of times the string s occurred in the history L.

Example: Consider the sequence of length 40 below. The maximal suffix is YGSJSLJZ; if we set $\alpha = 0.5$ then the fractional suffix is SLGZ. Its occurrences are shown below marked by a box:

$$\boxed{\text{SLJZ}}\text{GGDL}\boxed{\text{YGSJ}\boxed{\text{SLJZ}}}\text{KGS}\boxed{\text{SLJZ}}\text{ID}\boxed{\text{SLJZ}}\text{GGZ}\boxed{\text{YGSJ}\boxed{\text{SLJZ}}}.$$

The symbols that followed the fractional suffix SLGZ in the sequence are G, K, I, G, respectively. SPM will predict G.

C. Breaking ties

In our descriptions of the predictors, above, we indicate that each predicts the symbol with the highest probability of occurring next, given the current state of its data structure. It is quite possible, though, for there to be a tie for first place, that is, several symbols with equal probability and none with higher probability. Generally the literature does not indicate how to break a tie, yet our implementations must make a specific choice. We implemented 3 different tie-break methods:

- *First added:* among the symbols that tie, we predict the first one added to the data structure, that is, the first one seen in the location history.

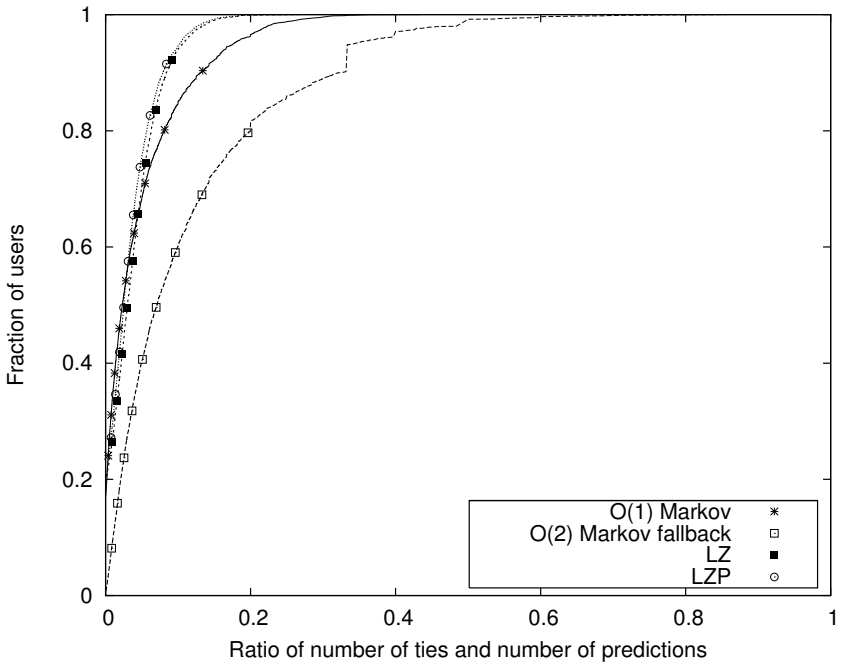


Figure 4. Ratio of number of ties and the number of predictions

- *Most recently added:* among the symbols that tie, we predict the one that was most recently added to the data structure.
- *Most recent:* among the symbols that tie, we predict the one most recently seen in the history.

In Figure 4, we show the ratio of the number of ties and the number of predictions, using a cumulative distribution function (CDF) across all users for different predictors (the O(2) Markov fallback predictor will be described later in the next section). Fortunately, Figure 4 shows that most of the users had few ties, less than about 10% of all predictions. Our experiments in the next section showed that the results were not significantly affected by the choice of a tie-breaking method.

D. Accuracy metric

During an on-line scan of the location history, the predictor is given a chance to predict each location. There are three possible outcomes for this prediction, when compared to the actual location:

- The predictor correctly identified the next location.
- The predictor incorrectly identified the next location.
- The predictor returned “no prediction.”

All predictors encounter situations in which they are unable to make a prediction; in particular, all realistic predictors will have no prediction for the first location of each user trace.

We define the *accuracy* of a predictor for a particular user to be the fraction of locations for which the predictor correctly identified the next location. Thus, "no prediction" is counted as an incorrect prediction. In the future we plan to examine other metrics that can better distinguish the two forms of incorrect prediction (there may be some applications that may prefer no prediction to a wild guess). For now, this metric best reflects the design of predictors common in the literature.

E. Entropy metric

We believe that there are some intrinsic characteristics of a trace that ultimately determine its predictability and hence the performance of different predictors.

In general, the entropy $H(X)$ of a discrete random variable X is defined as

$$H(X) = -\sum_{x \in \mathcal{X}} P(x) \log_2 P(x) \tag{4}$$

where $\mathcal{X}$ is the set of all possible of values of x.

In this paper, we compute the entropy of a given user under the $O(k)$ Markov predictor. We obtain the probabilities $P(x)$ from the data structures constructed by that predictor on that user's trace.

If C is the set containing all the context strings encountered in parsing a location history L, then the conditional entropy is

$$H(X|C) = -\sum_{c \in \mathcal{C}} \frac{N(c, L)}{n - k + 1} \sum_{a \in \mathcal{A}} P(x = a|c) \log_2 P(x = a|c) \tag{5}$$

In the results section, we compare the accuracy metric with the entropy metric, for each user, on several predictors.

F. Data collection

We have been monitoring usage on the Wi-Fi network at Dartmouth College since installation began in April 2001. Installation was largely complete by June 2001, and as of spring 2003 there were 543 access points providing 11 Mbps coverage to the entire campus. Although there was no specific effort to cover outdoor spaces, the campus is compact and the interior APs tend to cover most outdoor spaces. The wireless population is growing fast. As of May 2003 there are between 2,500 and 3,500 users active in any given day.

The access points transmit a "syslog" message every time a client card associated, re-associated, or disassociated; the message contains the unique MAC address of the client card. Although a given card might be used in multiple devices or a device used by multiple people, in this paper we think of the wireless card as a "network user" and thus the term "user" refers to a wireless card. As mentioned

previously, the locations in a trace are the names of the access points with which a user is associated and do not necessarily correspond to the precise geographical locations of humans. Similarly, changes in location, i.e., moves, do not necessarily correspond to physical movement of humans.

We have a nearly continuous, two-year record of these syslog messages from April 2001 through March 2003. Our data has some brief “holes” when the campus experienced a power failure, or when a central syslog daemon apparently hung up. Also, since syslog uses UDP it is possible that some messages were lost or misordered. In March 2002, we installed two redundant servers to record the data, so that holes in one of the servers can be filled by the data recorded by the other server. For more information about Dartmouth’s network and our data collection, see our previous study [KE02] [KE03].

After we cleaned the data of some glitches, and merged the data from redundant data-collection servers (where available), we extracted user traces from the data. Each user’s trace is a series of locations, that is, access-point names. We introduce the special location “OFF” to represent the user’s departure from the network (which occurs when the user turns off their computer or their wireless card, or moves out of range of all access points). The traces varied widely in length (number of locations in the sequence), with a median of 494 and a maximum of 188,479; most are shown in Figure 5. Users with longer traces were either more active (using their card more), more mobile (thus changing access points more often), or used the network for a longer period (some users have been on the network since April 2001, and some others have only recently arrived on campus). Some users have visited many APs during the period of measurement, while some others have only visited a small number of APs. Figure 6 shows the distribution of the number of visits for each user. We are also interested in the density of visit per AP for each user. Some users have visited many APs, but only a few visits for each AP. However, some other users have intensively visited a small number of APs. Figure 7 presents the distribution of average visits per AP for each user.

G. Evaluation experiments

To evaluate a predictor, we run each user’s trace independently through our implementation of that predictor. For each location in the trace, the predictor updates its data structure and then makes a prediction about the next location. Our experimental framework tracks the accuracy of the predictor.

III. RESULTS

We evaluated the location predictors using our Wi-Fi mobility data. In this section we examine the results.

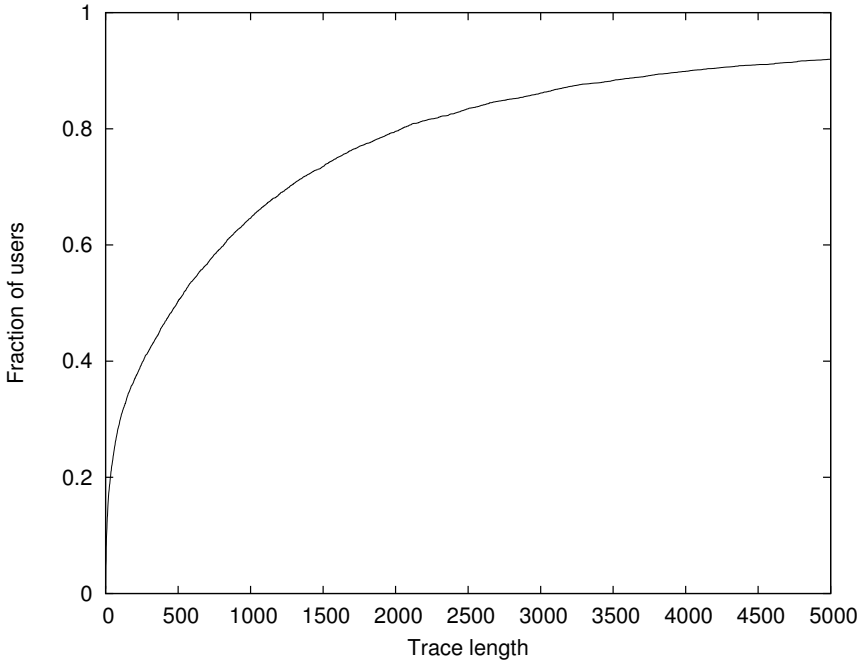


Figure 5. Length of user traces

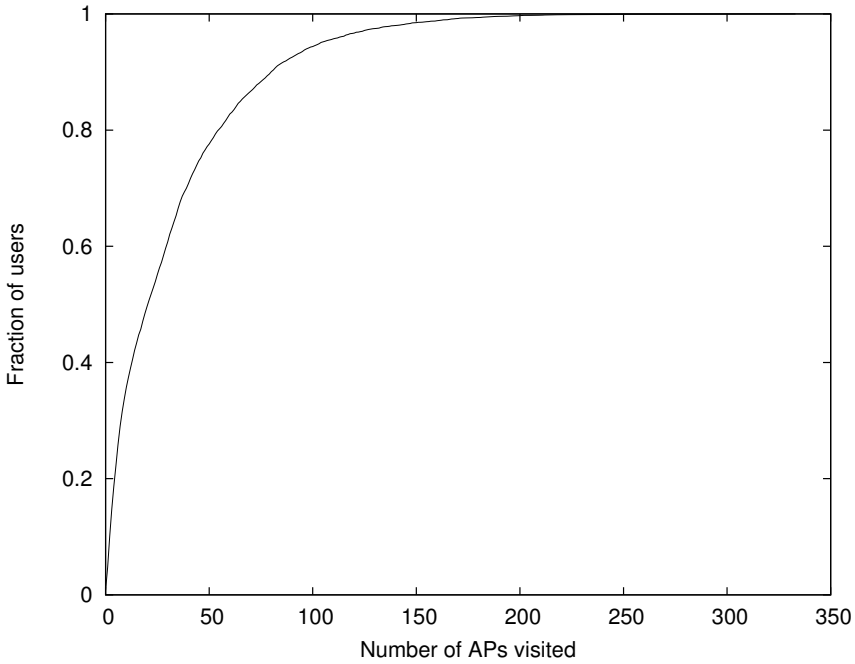


Figure 6. Number of visits

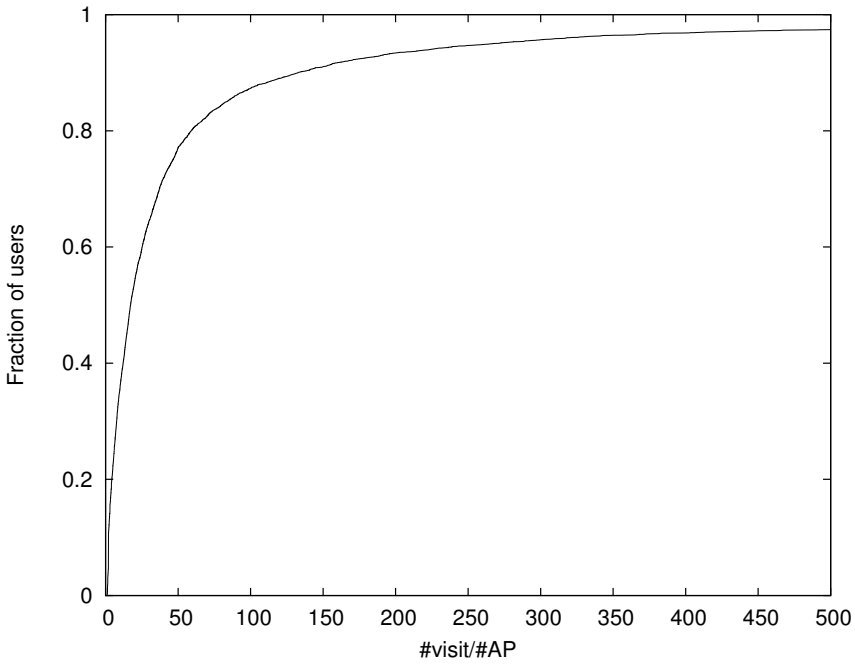


Figure 7. Visits per AP

A. *Markov*

Recall that accuracy is defined as the number of correct predictions divided by the number of attempts (moves). As we step through the locations in a single trace, attempting to predict each location in sequence, we can examine the accuracy up to that point in the trace. Figure 8 shows how the accuracy metric varies over the course of one user's history, using the $O(1)$ Markov predictor. Ultimately, for comparison purposes, we record the accuracy over the entire trace, the rightmost value in this plot. In most subsequent graphs we use this overall accuracy as the performance metric.

Of course, we have several thousand users and the predictor was more successful on some traces than on others. In Figure 9 we display the accuracy of the $O(1)$ Markov predictor in CDF curves, one for each of three groups of traces: *short*, *medium*, and *long* traces, defined respectively as those with 100 or fewer moves, 101-1000 moves, and over 1000 moves. The predictor was clearly unsuccessful on most short traces, because its curve is far to the left. Ultimately, we found that all predictors fared very poorly on short traces and somewhat poorly on medium traces. In previous work [SKJH04] we thus focused on long traces; there were 2,195 such users, out of 6,202 total users traced. In the remainder of this paper, for the sake of completeness, we generally consider all 6,202 traces. In some cases we do only consider long traces, when doing so makes trends clearer; in that case we note so explicitly.

We proposed three different tie-break methods, namely, first added, recently added, and recently seen.

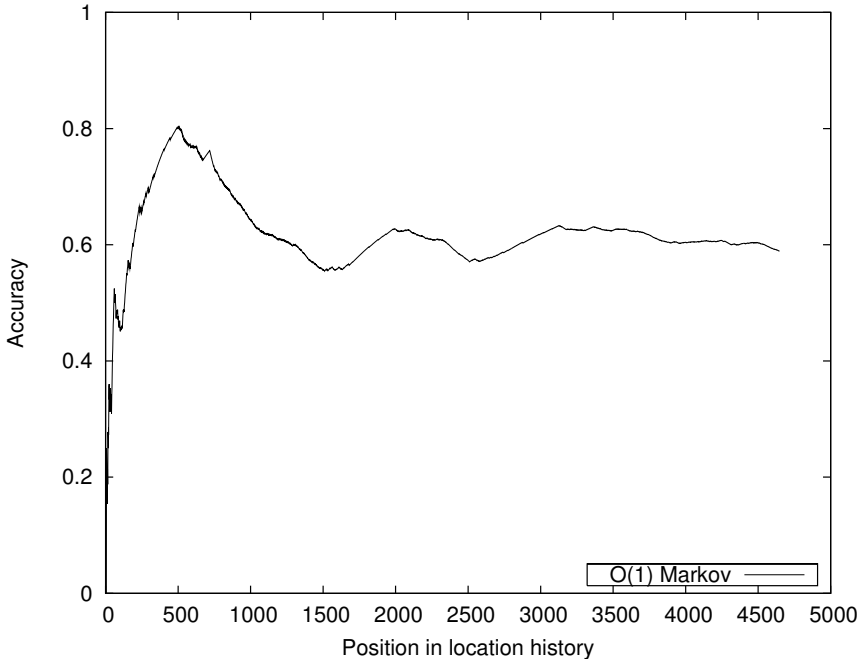


Figure 8. Prediction accuracy for a sample user

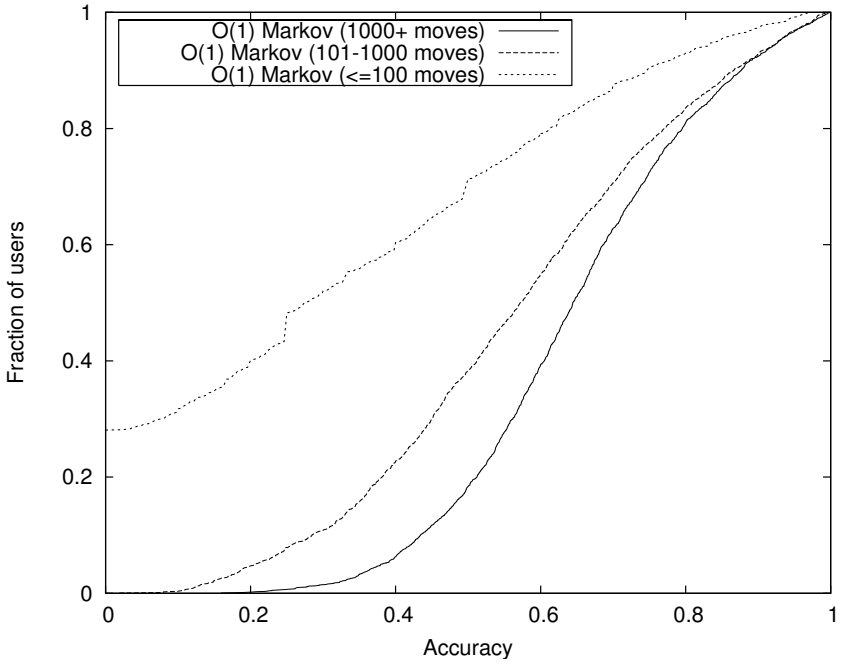


Figure 9. Accuracy of $O(1)$ Markov predictor

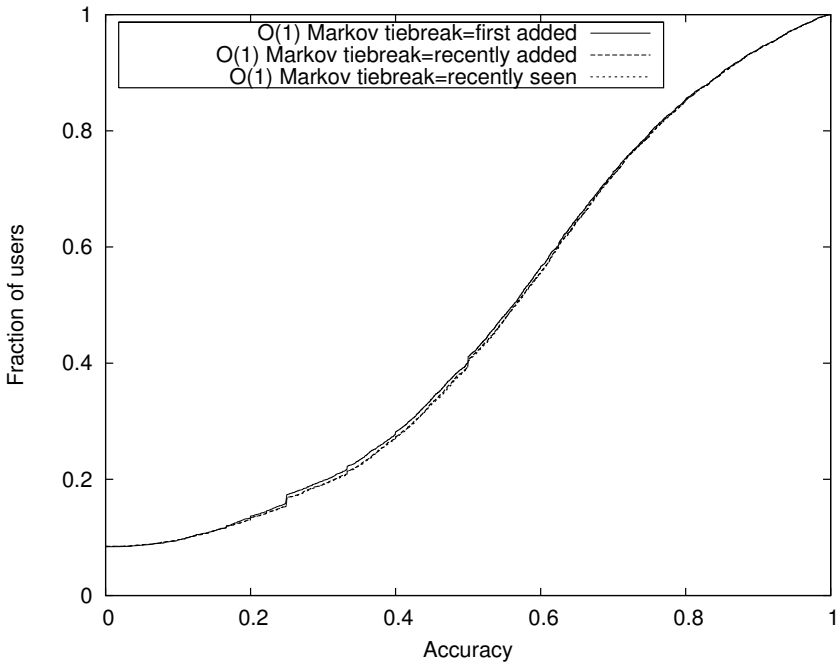


Figure 10. Different tie-break methods of $O(1)$ Markov predictor

Figure 10 showed that the results of the O(1) Markov predictor were improved slightly (the median of the accuracy increased less than 1% using either "recent" method). Although the distribution of accuracy did not change much, the individual prediction accuracy changed for many users. There were 941 users whose prediction accuracies increased more than 1% when using the "recently added" tie-break method from using the "first added" tie-break method, while only 334 users' prediction accuracies decreased more than 1%. The recent seen tie-break method had a similar result. Figure 11 shows the ratio of the accuracy when using "recently added" and "recently seen" methods over the "first added" method. Our results also showed that the tie-break methods had little effect on the accuracy distribution of higher order Markov predictors, which are not shown on the plots.

Since the results were not significantly affected by the choice of a tie-breaking method, we used the "first added" method throughout the results below, and do not consider tie-breakers further.

Intuitively, it should help to use more context in each attempt to predict the next location. In Figure 12 we compare the accuracy of $O(1)$ Markov with that of $O(2)$, $O(3)$, and $O(4)$ Markov predictors. As an aside, we include an "order 0" Markov predictor, in which no context is used in the prediction of each move. This predictor simply predicts the most frequent location seen so far in that user's history. Although it represents a degenerate form of Markov prediction, it helps to show the benefit of context in the $O(1)$ predictor.

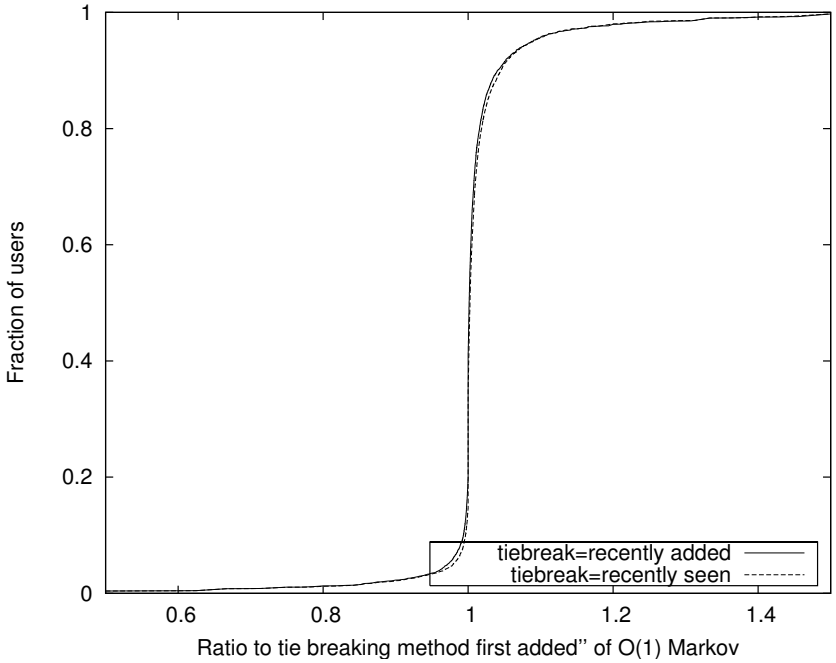


Figure 11. Ratio of different tie-break methods for $O(1)$ Markov predictor

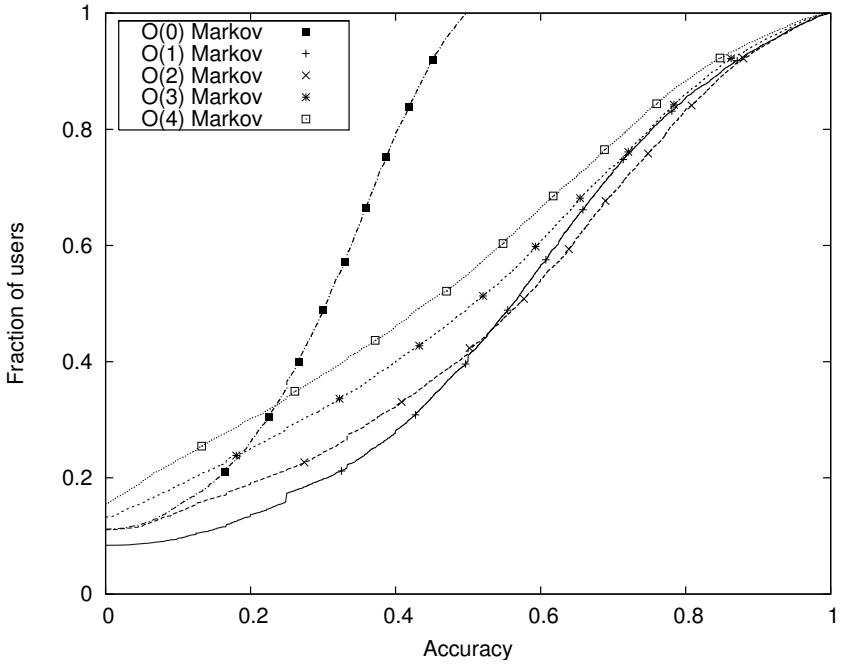


Figure 12. Comparing Markov predictors (order 0 - 4) for all traces

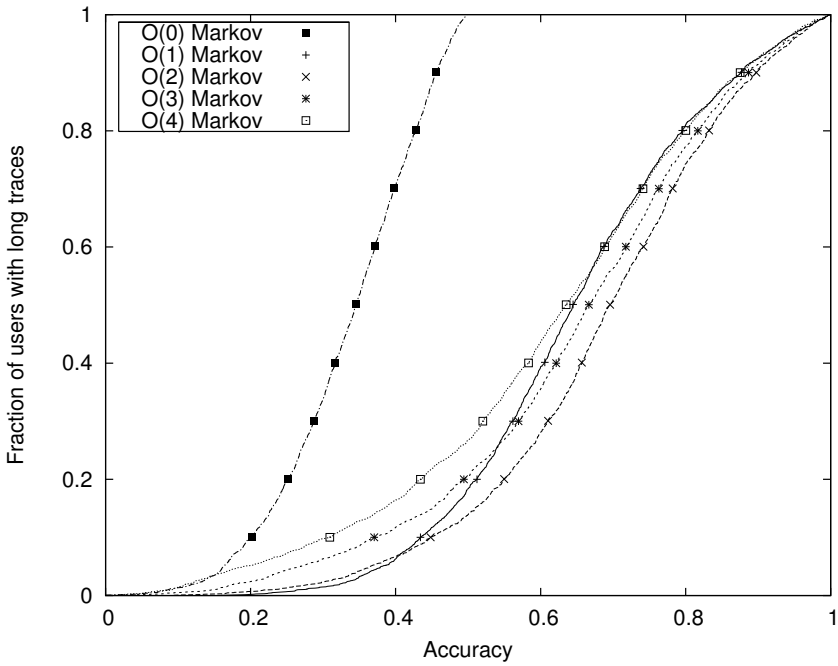


Figure 13. Comparing Markov predictors (order 0 - 4) for long traces

Not surprisingly, $O(2)$ often outpredicted $O(1)$, and its curve is further to the right. (In fact, when considering only long traces, $O(2)$ generally outpredicts $O(1)$, see Figure 13). The high-order $O(3)$ and $O(4)$ predictors were, however, worse than $O(2)$. Although these predictors use more information in the prediction process, they are also more likely to encounter a context (a three- or four-location string) that has not been seen before, in which case they are unable to make a prediction. A missing prediction is not a correct prediction, and these unpredicted moves bring down the accuracy of the higher-order predictors. In Figure 14 we display the *conditional accuracy* of these same predictors: the number of correct predictions divided by the number of predictions. In this metric, we ignore unpredicted moves, and it becomes clear that longer context strings did lead to better prediction, where possible, although with diminishing returns. This trend becomes clearer when we consider only long traces in Figure 15.

Returning to our original accuracy metric, we now consider a meta-predictor based on the Markov predictor family. Figure 16 displays the performance of the $O(2)$ Markov predictor with "fallback," which uses the results of the $O(2)$ predictor when it makes a prediction, or the $O(1)$ predictor if the $O(2)$ predictor has no prediction. In general, the $O(k)$ fallback predictor recursively uses the result of the $O(k-1)$ predictor (with $k = 0$ as the base of the recursion) whenever it encounters an unknown context. Fallback indeed helped to improve the $O(2)$ Markov predictor's performance. $O(3)$ and $O(4)$ Markov also improved with fallback, but the curve of $O(4)$ with fallback is nearly identical to the $O(3)$

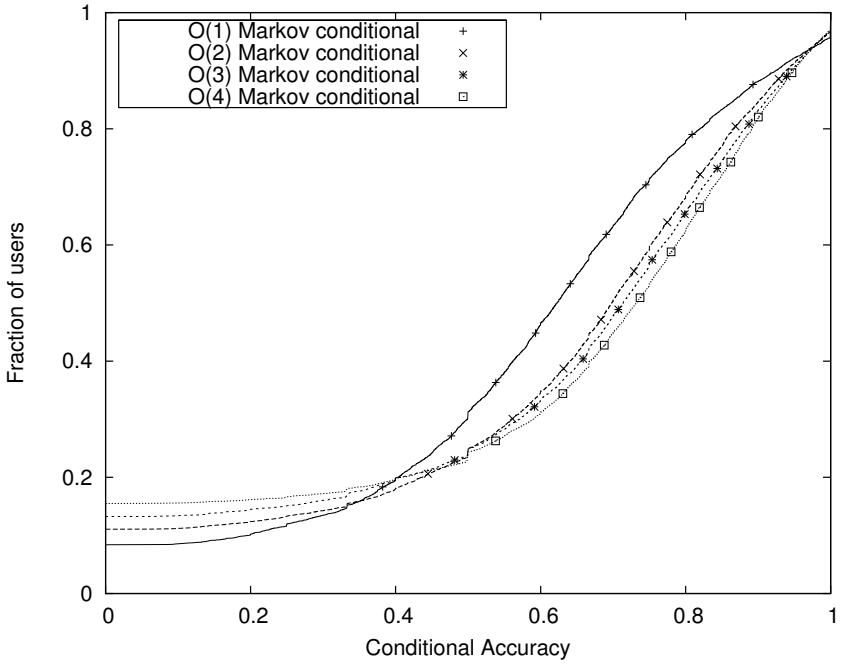


Figure 14. Conditional accuracy metric for all traces

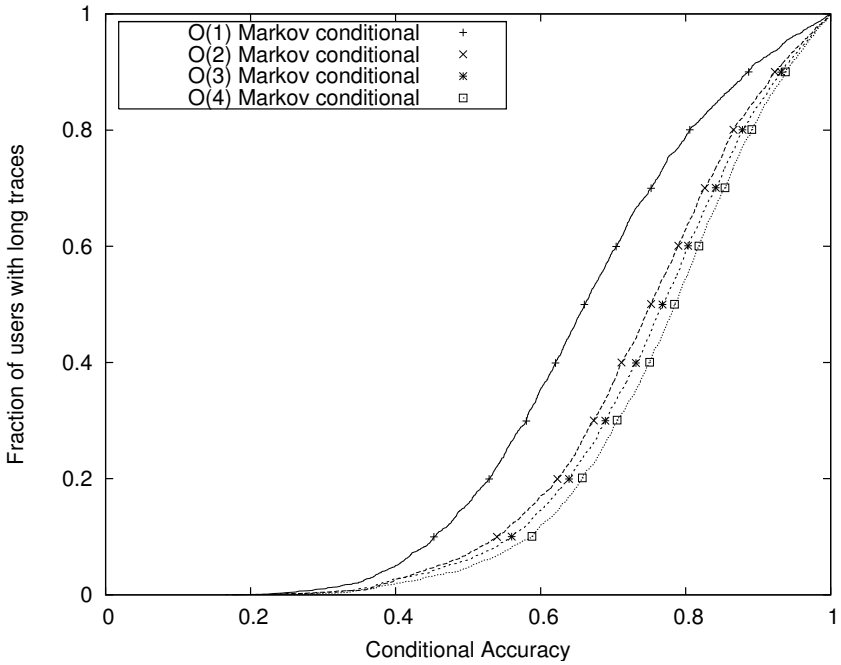


Figure 15. Conditional accuracy metric for long traces

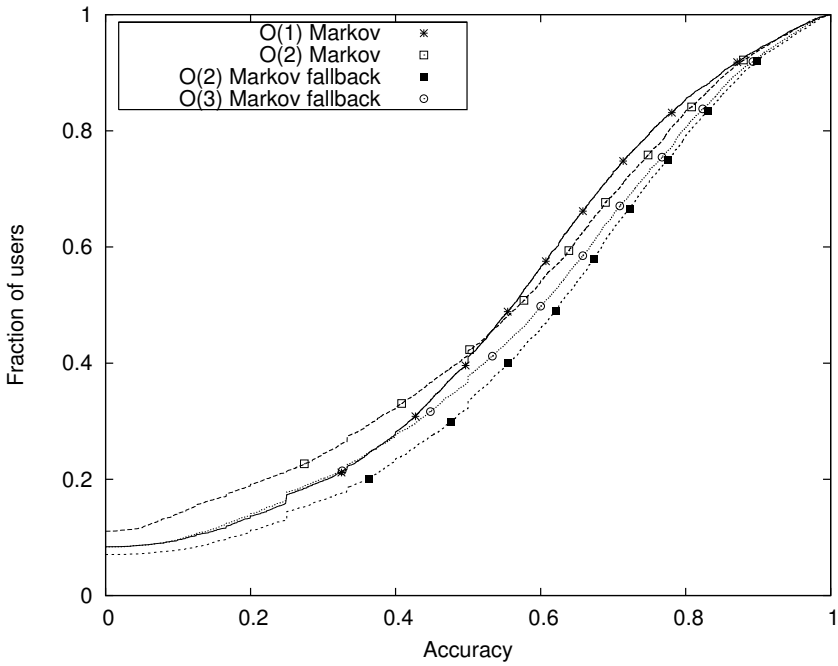


Figure 16. Markov predictors with fallback

Markov with fallback and are thus not shown. Markov predictors with fallback gain the advantage of the deeper context but without losing accuracy by failing to predict in unknown situations.

In the plots so far we examine the performance of $O(k)$ Markov predictors that used the most recent k locations in making a prediction, weighting the potential next locations by their frequency of occurrence in the past. An alternative approach assigns weights according to the recency of occurrence in the past; thus, one transition (the most recent seen for this context) has weight 1 and the others have weight 0. Observe that the Order-k Markov prediction using the most-frequent location in general takes $O(n^{k+1})$ space, while using the most-recent location it takes only $O(n^k)$ space, a potentially significant decrease in storage. Figure 17 shows the quality of Markov predictors based on this approach. Curiously, here $O(2)$ did worse than $O(1)$, although fallback made up some of the difference. We are still exploring the reason behind this difference.

In Figure 18 we compare the recency-weighted approach with the original frequency-weighted approach. The best recency-weighted Markov predictor, $O(1)$, was better than the corresponding frequency-weighted predictor. This result implies that recent history was a better predictor of the immediate future than was an accumulated probability model, when considering only the current location. On the other hand, recall from Figure 17 that among the recency-weighted predictors the extra context of $O(2)$ lowered prediction accuracy. Thus, among $O(2)$ predictors, the frequency-weighted approach beats the recency-

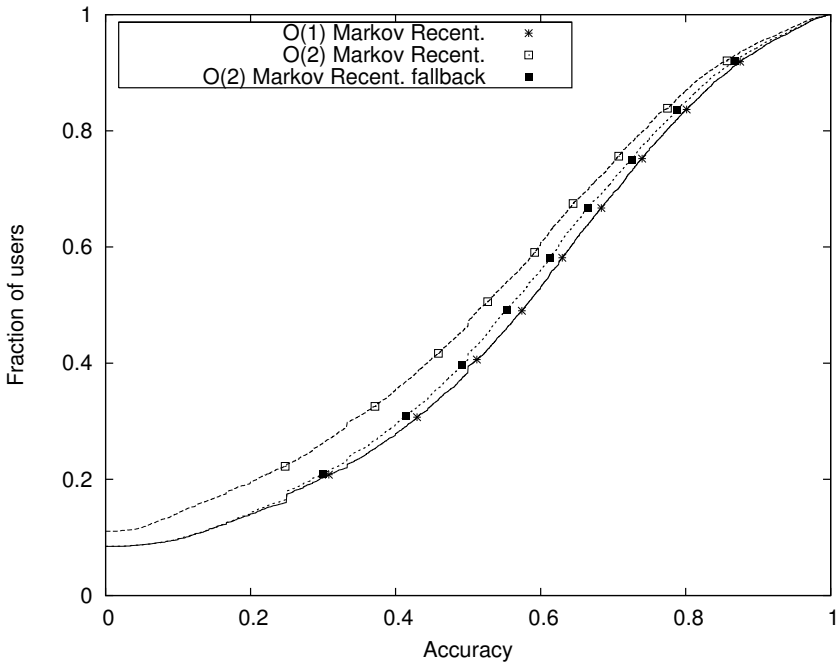


Figure 17. Markov using "most recent" rather than "most frequent" next choice

weighted approach (not shown in Figure 18). Ultimately, the $O(2)$ frequency-weighted Markov predictor with fallback has the best outcome.

Some of our user traces span a period of hours or days, but some span weeks or months. Clearly it is possible for a user's mobility patterns to change over time; for example, a student may move to a new dormitory, or attend different classes. The transition weights constructed by the frequency-weighted Markov model may become ineffective after such a change; for users with a long history, these weights will adapt too slowly. In another series of experiments, we added an exponential decay to the frequency weights so that more recent locations have a larger impact. We apply the following function to decay the weights when we update the frequency count in the transition table of the Markov predictor. The function is

$$c = \begin{cases} c+1 & \text{for current place} \\ c \cdot \lambda & \text{others} \end{cases},$$

where c is the frequency count of places, and λ is the age factor ($0 \leq \lambda \leq 1$).

We show the results of various age factors in Figure 19 for the $O(1)$ Markov predictor. It is clear that aging improved the $O(1)$ Markov predictor, though different age factors had varied improvement. We found that an age factor λ of 0.7 led to best improvement. In fact, when the age factor $\lambda = 0$, the frequency $O(1)$ Markov predictor becomes the recency $O(1)$ Markov predictor, since all the history information is eliminated (of course, this implementation will take more memory space and more computation than the

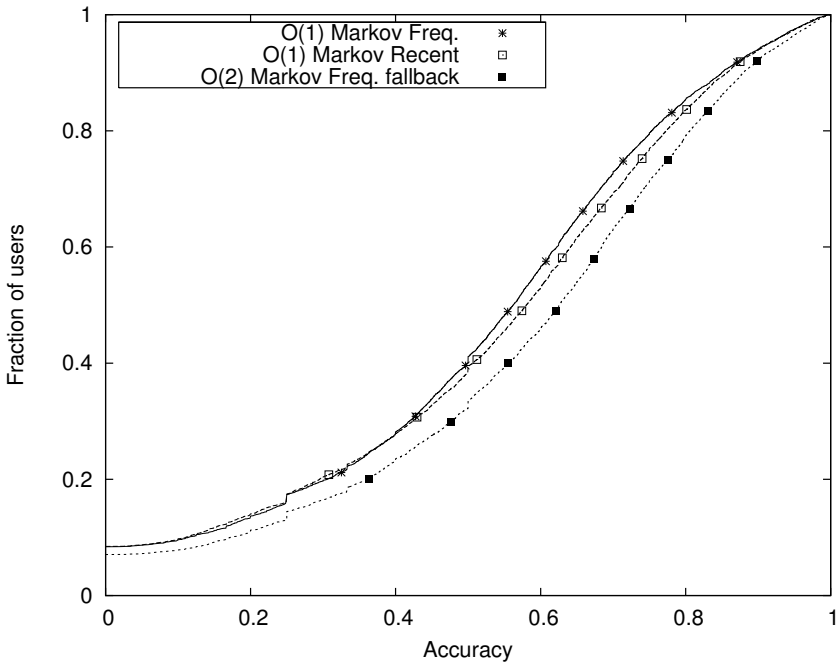


Figure 18. Markov: “most recent” vs. “most frequent”

simple recency Markov predictor). Figure 20 shows the ratio of accuracy with aging over the accuracy without aging of each user for the $O(1)$ Markov predictor. We see that about 70% of users' accuracies improved, while about 20% of users' accuracies decreased. But the aging methods did not improve the $O(2)$ Markov predictor, as shown in Figure 21. In the $O(2)$ Markov fallback predictor (Figure 22), we see that the aging actually made the accuracy worse than without aging. In a closer look at the ratio plot in Figure 23, we see that aging caused more users to have decreased accuracy than those who had increased accuracy. It is unclear why aging helped $O(1)$ Markov prediction but hurt most users of $O(2)$ Markov with fallback.

B. LZ-based

We first consider the LZ predictor, shown in Figure 24. Since LZ makes no prediction whenever the current context string leads to a leaf in the tree, the plot includes two LZ variants. As an alternative to no prediction (the left curve), we can use the statistics at the root (the middle curve) to make a prediction based on the probability of each location. That is, when the predictor encounters a leaf, it behaves as if it is at the root and simply predicts the most frequently seen child of the root. Given our accuracy metric, it is always better to make some prediction than no prediction, and indeed in this case the accuracy improved significantly. An even better approach (the right curve) is to fallback to progressively shorter substrings of the current context, much as we did with the Markov predictors, until a substring leads to a

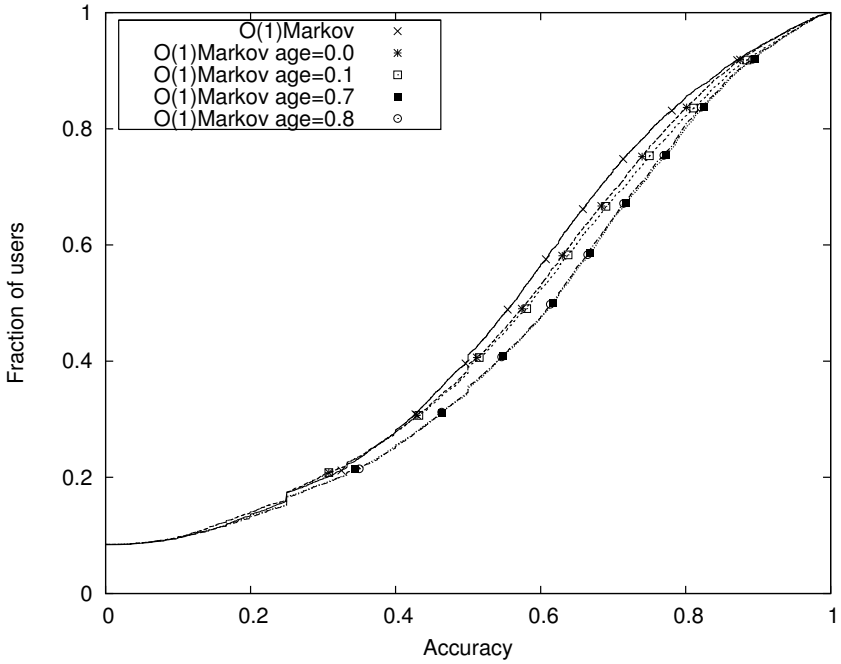


Figure 19. Aging the history of O(1) Markov predictor

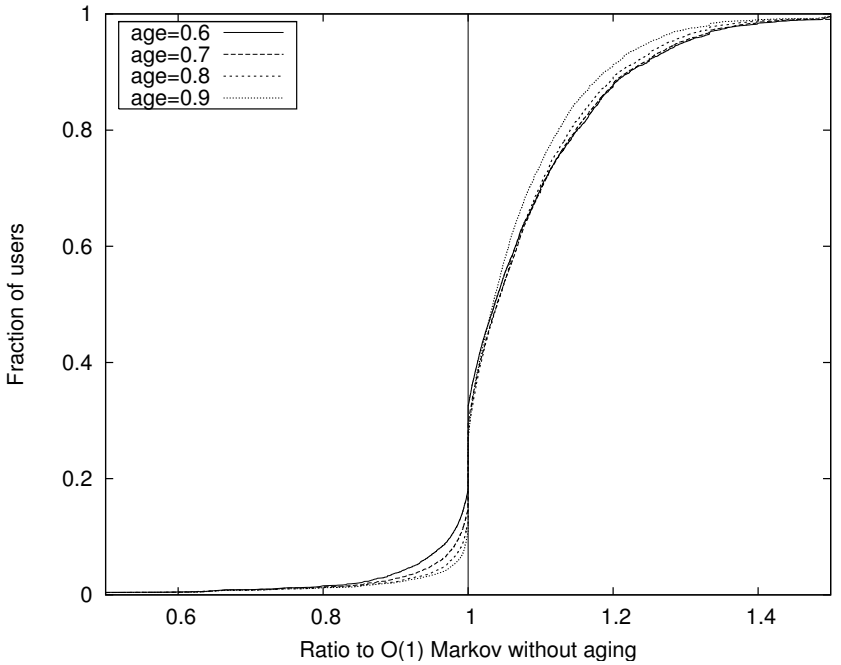


Figure 20. CDF of aging the history of O(1) Markov predictor

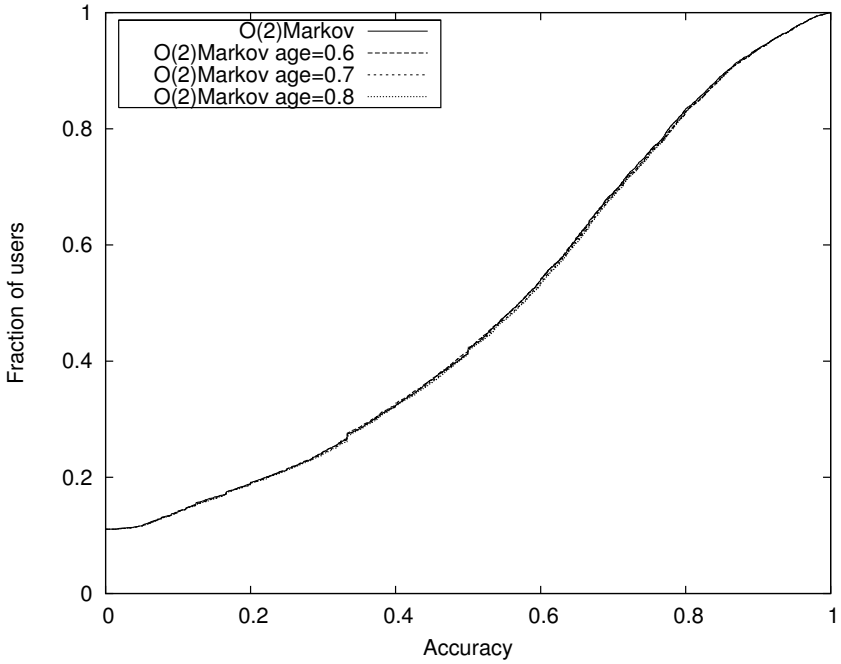


Figure 21. Aging the history of O(2) Markov predictor

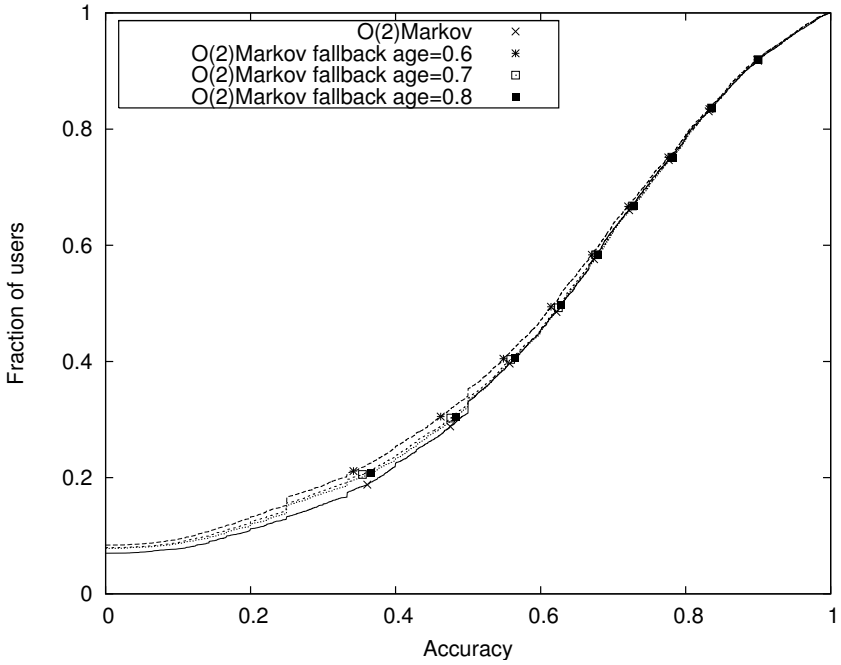


Figure 22. Aging the history of O(2) Markov fallback predictor

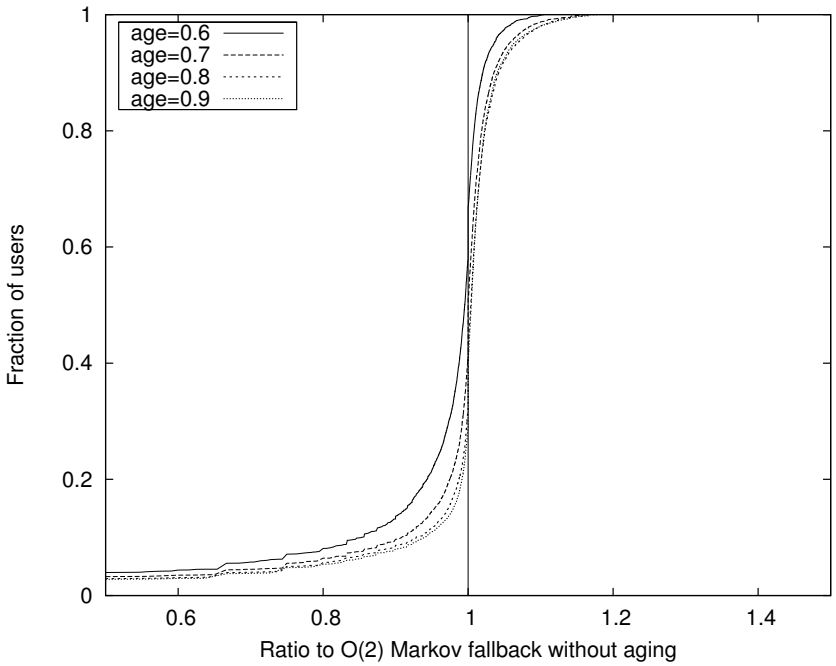


Figure 23. CDF of Aging the history of O(2) Markov fallback predictor

non-leaf node from which we can make a prediction. This fallback ability is critical to allow prediction to occur while the tree grows, since the trace often leads to a leaf just before adding a new leaf.

Bhattacharya and Das [BD02] proposed two extensions to the LZ predictor. Figure 25 displays the performance of the first extension, LZP. Once again, this predictor (as defined in [BD02]) makes no prediction when the context leads to a leaf. We modified LZP to use statistics at the root, which helps, and (better) to try fallback.

The second extension produces a LZ+prefix+PPM predictor nicknamed “LeZi.” Figure 26 compares the performance of LZ with LZP and LeZi, showing that each extension did improve the accuracy of the LZ predictor substantially.

When we compare the best variant of each LZ form, in Figure 27, it becomes clear that the simple addition of our fallback technique to the LZ predictor did just as well as the prefix and PPM extensions combined. (LeZi automatically includes fallback by adding suffix substrings into the tree, so there is no fallback variant.) This trend is even more pronounced when considering only long traces. LZ with fallback is a much simpler predictor than LZP or LeZi, and since the accuracy is similar (and as we show below, the LZ data structure was smaller), among the LZ predictors we recommend LZ with our fallback technique.

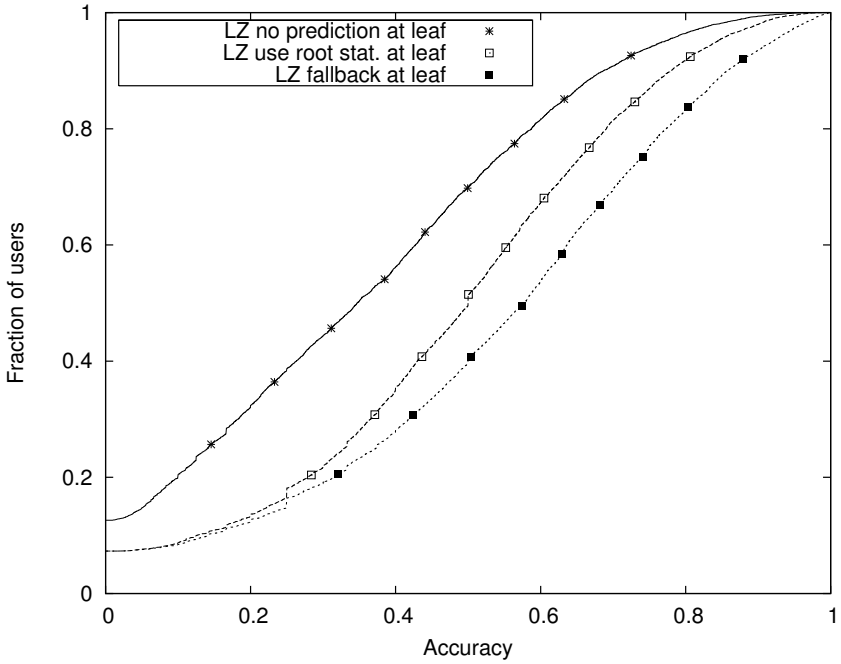


Figure 24. LZ predictors

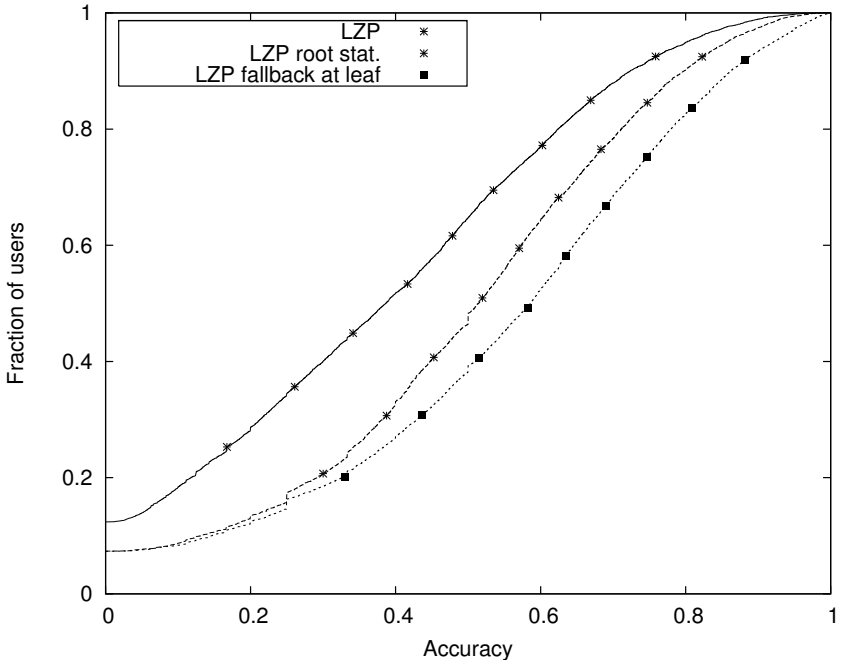


Figure 25. LZP predictors

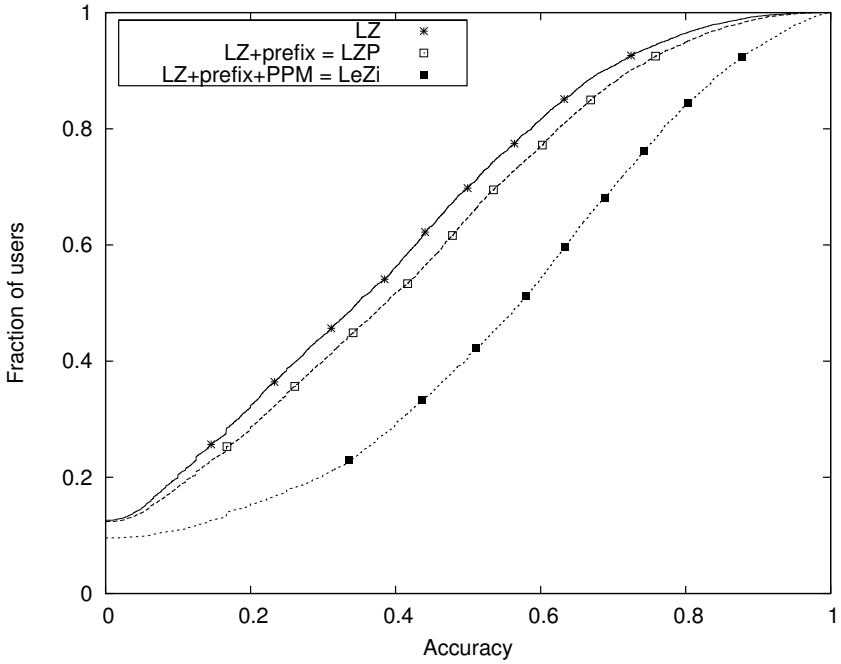


Figure 26. LZ, LZP, LeZi predictors

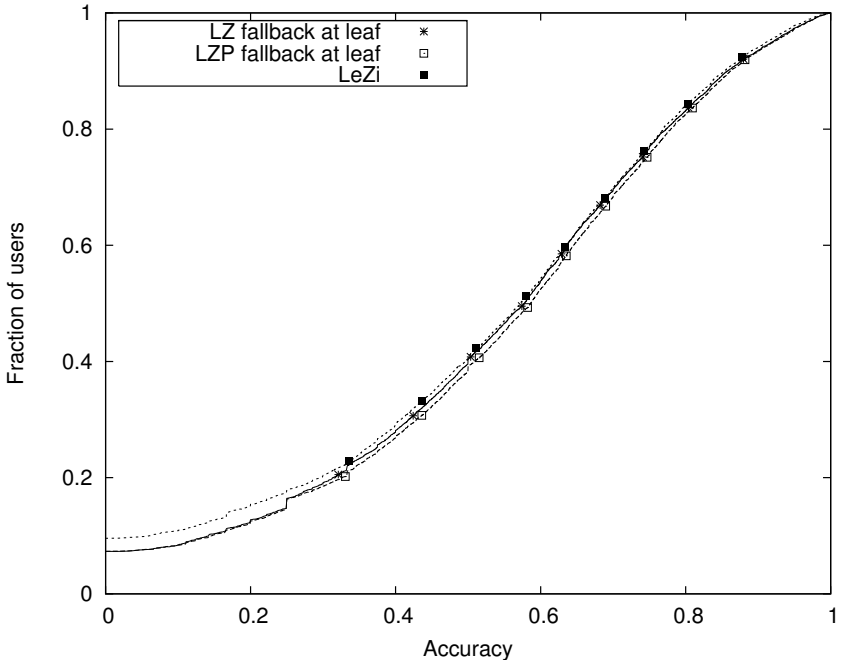


Figure 27. LZ predictors with fallback

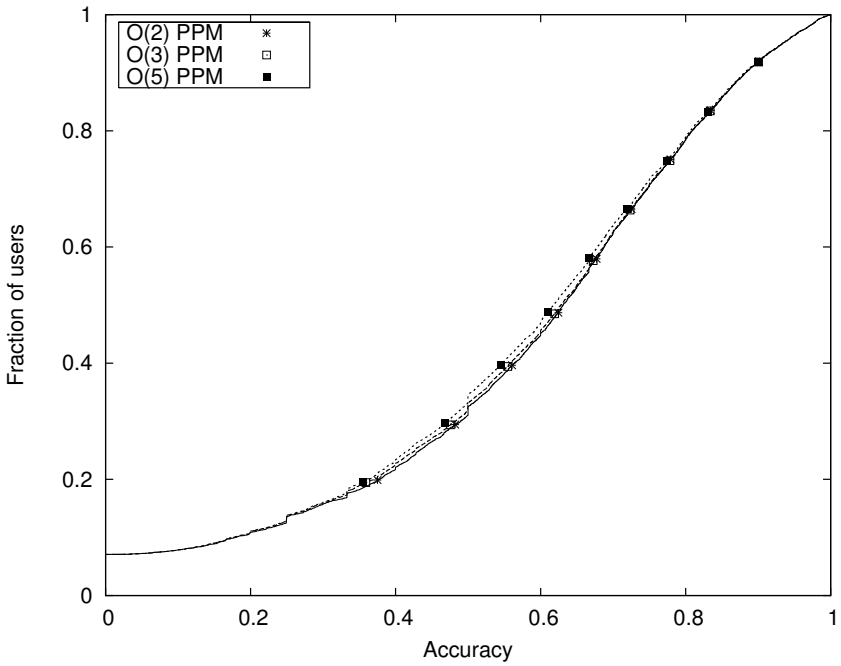


Figure 28. PPM predictors

C. PPM and SPM

Prediction by Partial Matching (PPM) uses a finite context like Markov predictors, but blends together several fixed-order context models to predict the next character [CT97]. Figure 28 shows that the amount of context does not matter much when the order is 2 or above. The higher-order PPM predictors do not have better prediction accuracy.

The SPM predictor [JSA00] is, in a sense, an expansion of the PPM predictor, in which there is no limit on order. Theoretically, the SPM predictor should outperform any finite-order PPM predictor. However, Figure 29 shows that both PPM and SPM have an accuracy only slightly better than the $O(2)$ Markov predictor with fallback. For long traces (not shown), we found the difference to be negligible. Here we choose $\alpha = 0.5$. We also tried to set $\alpha = 0.25$ and $\alpha = 0.75$; both of the results are worse than $\alpha = 0.5$ as shown in Figure 30.

D. AP prediction overall

We compare the best Markov predictors with the best LZ predictors in Figure 31 and Figure 32. It is difficult to distinguish the LZ family from the recency-based $O(1)$ Markov, which all seem to have performed equally well. In general, the $O(2)$ Markov predictor with fallback was the best overall. For long traces (Figure 32) the median accuracy with this predictor is 72%. It is striking that the extra complexity, and the theoretical aesthetics, of the LZ, PPM, and SPM predictors apparently gave them no

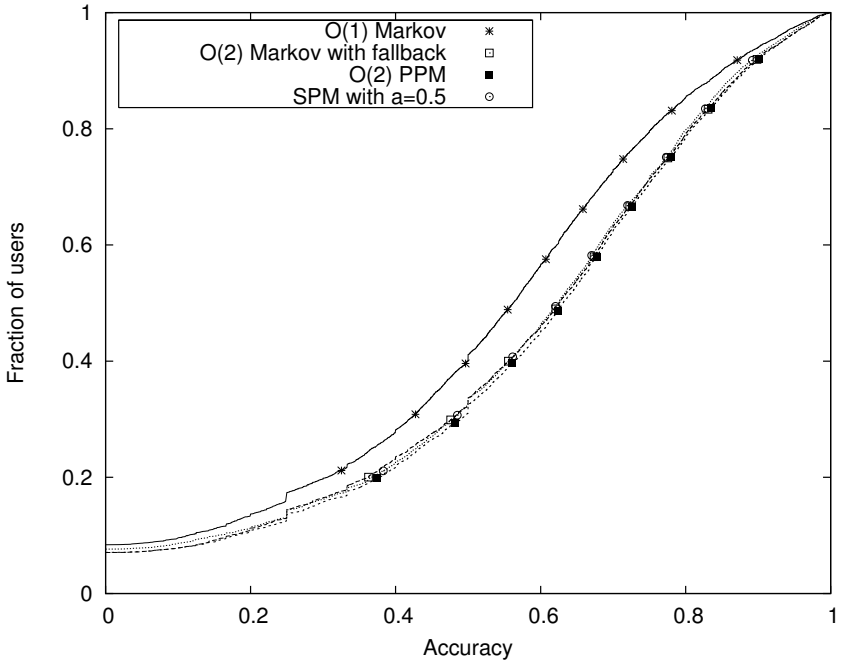


Figure 29. SPM predictors with $\alpha = 0.5$

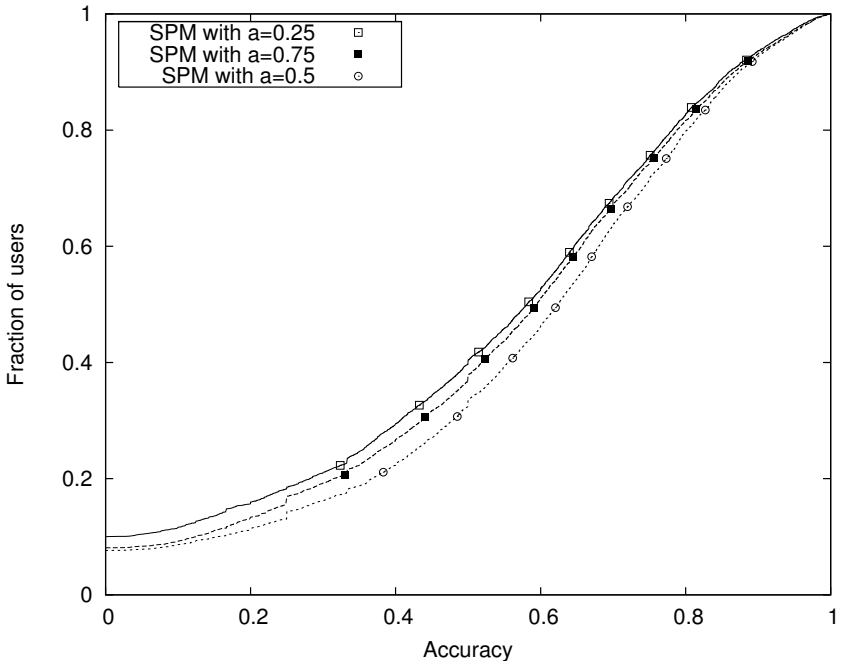


Figure 30. SPM predictors with different fractional contexts

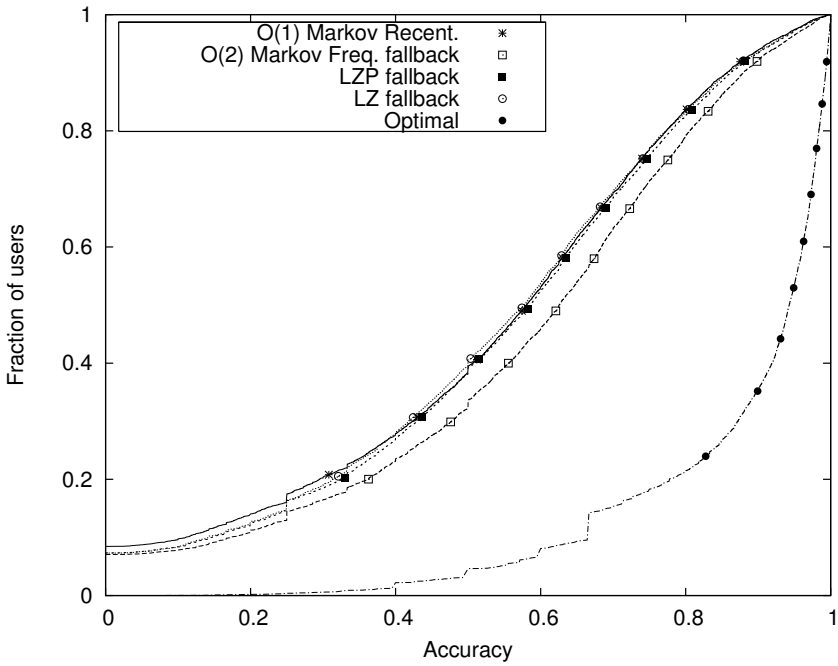


Figure 31. The best predictors, compared

advantage.

We include an "Optimal" curve in Figure 31, as a simple upper bound on the performance of history-based location predictors. In our definition, the "optimal" predictor can accurately predict the next location, except when the current location has never been seen before. Although it should be possible to define a tighter, more meaningful upper bound for domain-independent predictors like those we consider here, it seems clear that there is room for better location-prediction algorithms in the future.

E. Building prediction

In some applications, we may need only a coarser sense of the user's location. We extracted building-level traces from the AP-level traces; a building trace show the sequence of buildings visited by the user and is necessarily shorter than the original sequence of APs visited. Then we used our predictors to predict the next building in each user's trace. Figure 33 shows that the building prediction was more accurate than AP prediction. Here we focus only on long traces, and show that the best predictor can approach a median accuracy of 80%. One reason is that there is a smaller number of choices in the building trace than in the AP trace. Also, the sets of users of the two different forms of prediction are not the same; there are 2190 users in the AP-level prediction who have more than 1000 transitions in the trace history, while in the building-level prediction there are only 1501 users who have more than 1000 transitions in the building-level trace.

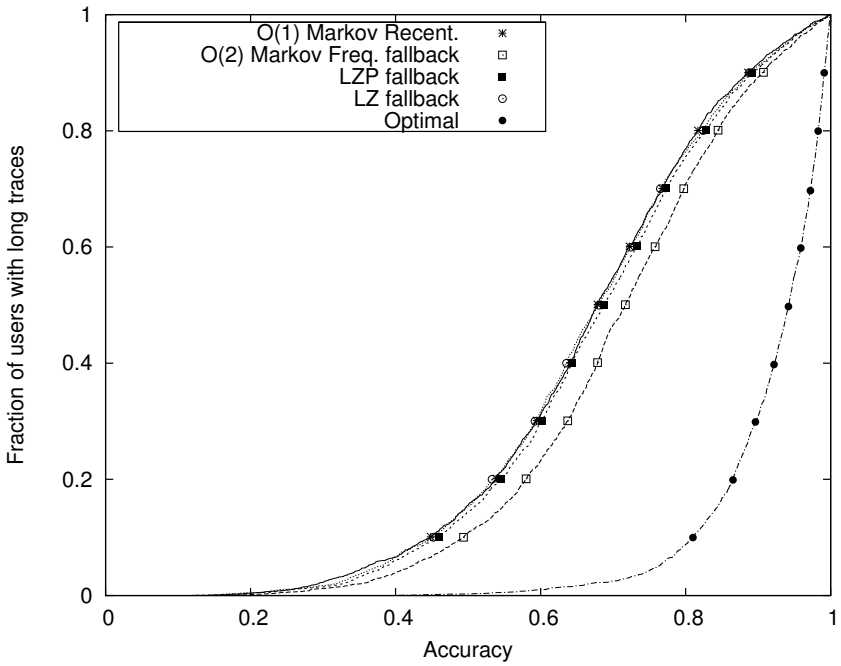


Figure 32. The best predictors, compared (long traces only)

In the building-level prediction, the $O(2)$ Markov with fallback predictor also outperformed the $O(1)$ Markov predictor, by about the same amount as in the AP-level prediction.

F. Correlation with Entropy

It may be that some user traces are simply less predictable than others. Some intrinsic characteristics of a trace may determine its predictability. We explored several such characteristics.

Entropy

The conditional entropy, as defined in section II-E, seems like a good indicator of predictability. We computed the conditional entropy for each user on their entire trace using Equation 5. In Figure 34 we compare the entropy of each user, based on the probability table built by the $O(1)$ Markov predictor, with the accuracy of the $O(1)$ predictor for long traces. The correlation is striking, and indeed the correlation coefficient is -0.95 (a coefficient of 1.0 or -1.0 represents perfect correlation). This strong correlation indicates that some users with high entropy are doomed to poor predictability. On the other hand the correlation is not surprising because the entropy was computed from the same tables used for prediction.

Figure 35 shows the correlation of the conditional entropy and the prediction accuracy for all traces. This correlation is not as striking as for long traces. For short traces, the entropy can be low since the amount of information is small and the accuracy can also be low (see below for our comparison of accuracy with trace length).

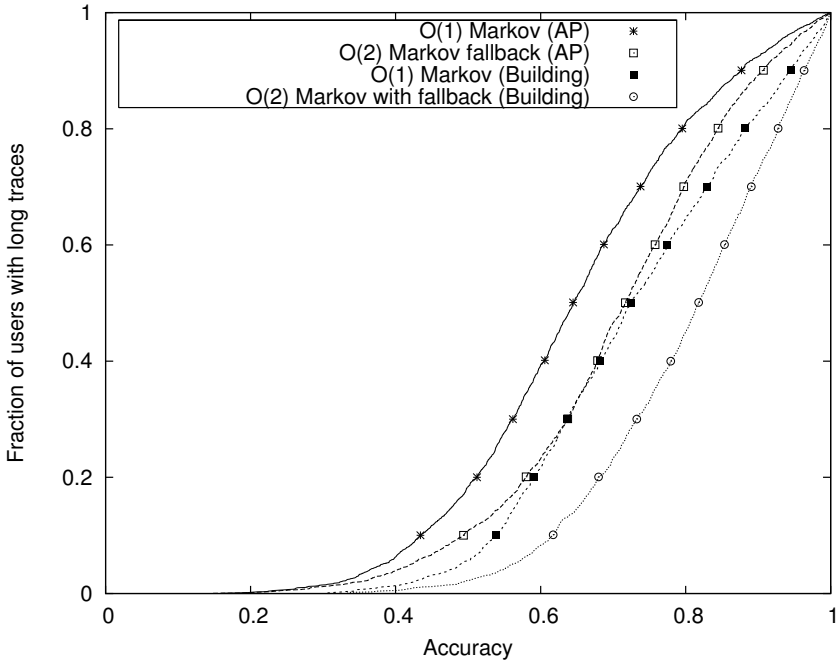


Figure 33. Building Prediction on uses with long traces

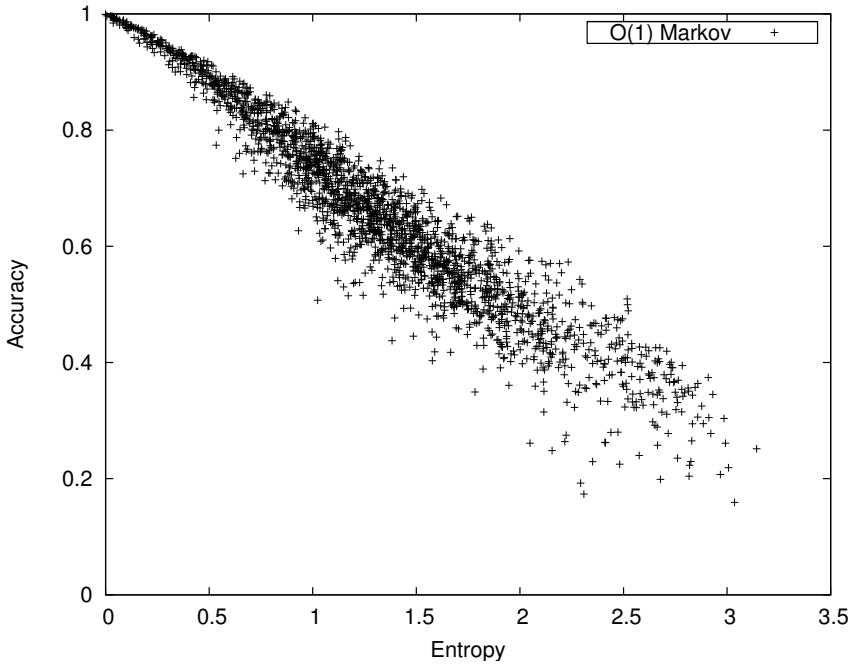


Figure 34. Correlating O(1) Markov prediction accuracy with entropy (long traces only)

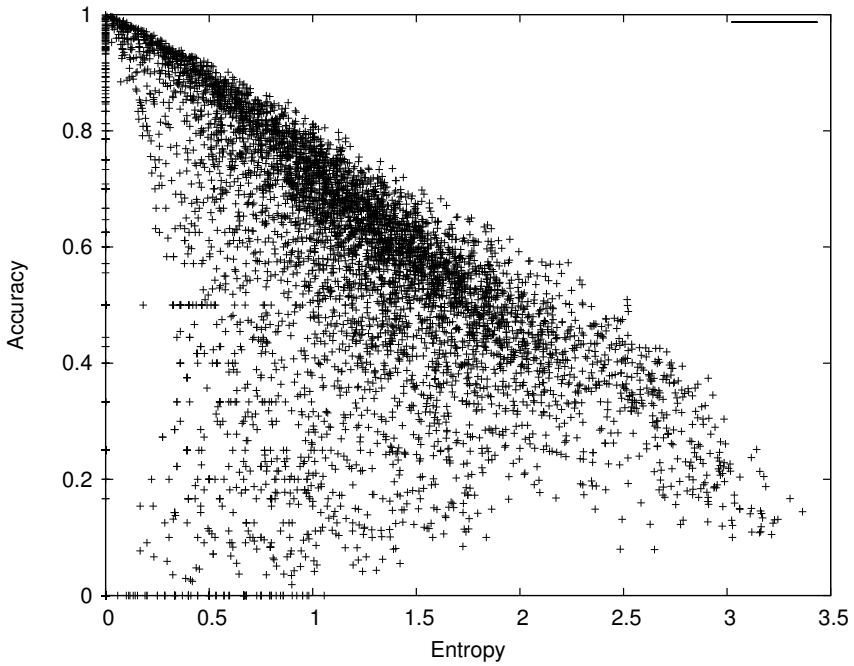


Figure 35. Correlating O(1) Markov prediction accuracy with entropy (all traces)

Calculating the relevant entropy in the context of the $O(2)$ Markov with fallback predictor is slightly more involved; the formula is derived in an Appendix. Figure 36 shows the correlation between entropy and the accuracy of the $O(2)$ Markov predictor for all traces; the correlation is visually similar to the $O(1)$ Markov correlation.

Trace Length

As indicated earlier, trace length has an impact on the accuracy of all the predictors. So far we have regarded accuracy for a given trace as simply the final average accuracy at the end of the trace. At each step i, for each trace that has length at least i, the running average accuracy up to step i is found. There are generally several traces that have length at least i; we find the median accuracy at step i among all such traces. This metric, called the *median running accuracy*, enables us to consider how, over all traces, accuracy changes as trace length increases.

Figure 37 shows how the median running accuracy increases with trace length for $O(1)$ Markov as well as $O(2)$ Markov with fallback. Clearly the median running accuracy increases rapidly with trace length up to short traces, and then increases only relatively slowly. Note that for this metric $O(2)$ Markov with fallback outpredicts $O(1)$ for all trace lengths.

We tried several equations to fit the experimental data, such as power functions and log functions. We found that the log functions fit best. These curves can be fit to the equations:

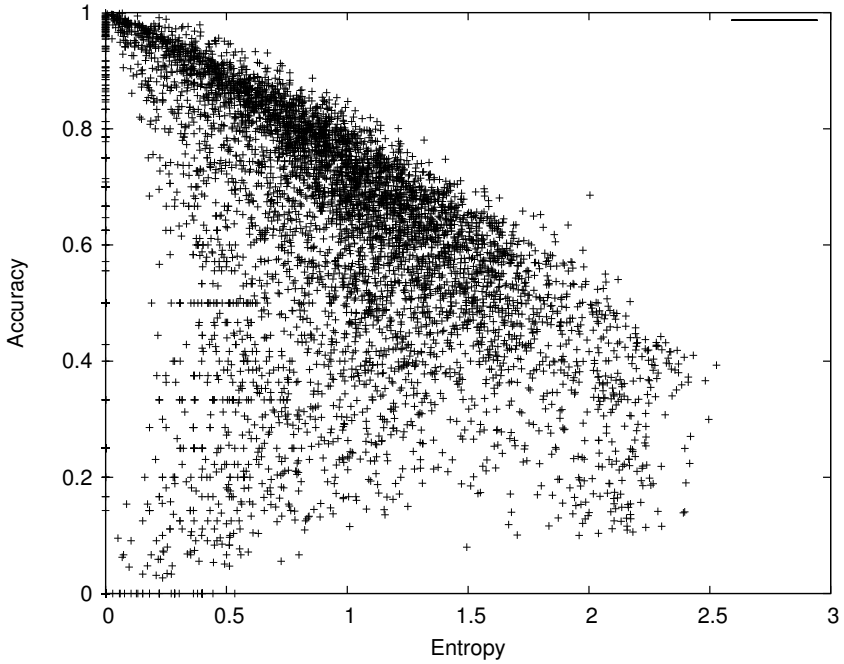


Figure 36. Correlating O(2) Markov fallback prediction accuracy with entropy (all traces)

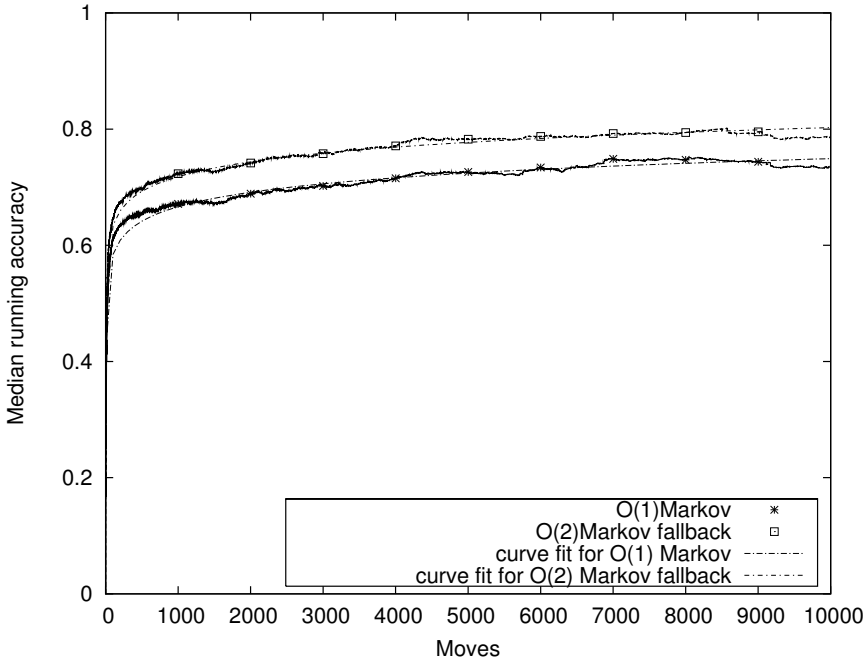


Figure 37. Correlating accuracy with trace length

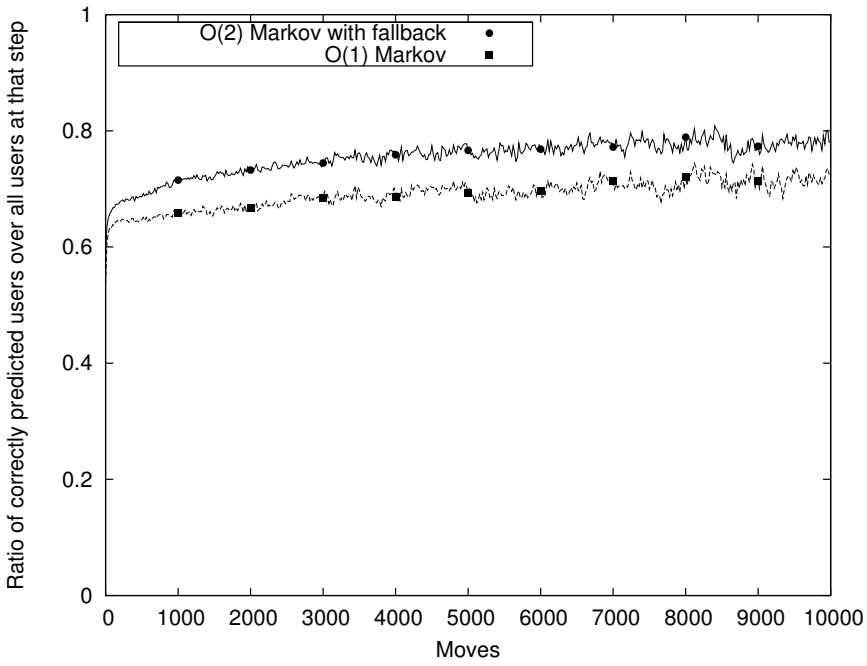


Figure 38. Correlating ratio of correct prediction with step

for $O(1)$ Markov,

$$a(i) = 0.036 \log(i) + 0.4176 \ , \tag{6}$$

for $O(2)$ Markov with fallback,

$$a(i) = 0.0367 \log(i) + 0.4647 \ . \tag{7}$$

The R^2 of the $O(1)$ Markov fit is 0.8717, and the R^2 for the $O(2)$ Markov fallback is 0.8956. The fits are good for long traces ($i > 100$), but have larger errors for short traces ($i < 100$).

We also investigated the correlation of accuracy and trace length in another way. We have 6202 users, and each user has a different trace length. Let $u(i)$ be the number of users whose trace has at least i moves. At each step, some users are predicted correctly, while the other users are predicted incorrectly. Let $A(i)$ be the number of users whose prediction at move i was correct. The ratio of correct prediction at move i is $A(i)/u(i)$. Figure 38 shows the ratio for $O(1)$ Markov and $O(2)$ Markov with fallback. Actually, this plot smoothes each curve by plotting the median ratio within each bucket of 20 moves. The resulting curves look similar to those in Figure 37. Again, the ratio of correct prediction increased rapidly for short traces, then increased slightly for longer traces.

In general, prediction accuracy was quite good for traces with at least several hundred moves, with minimal improvement after that.

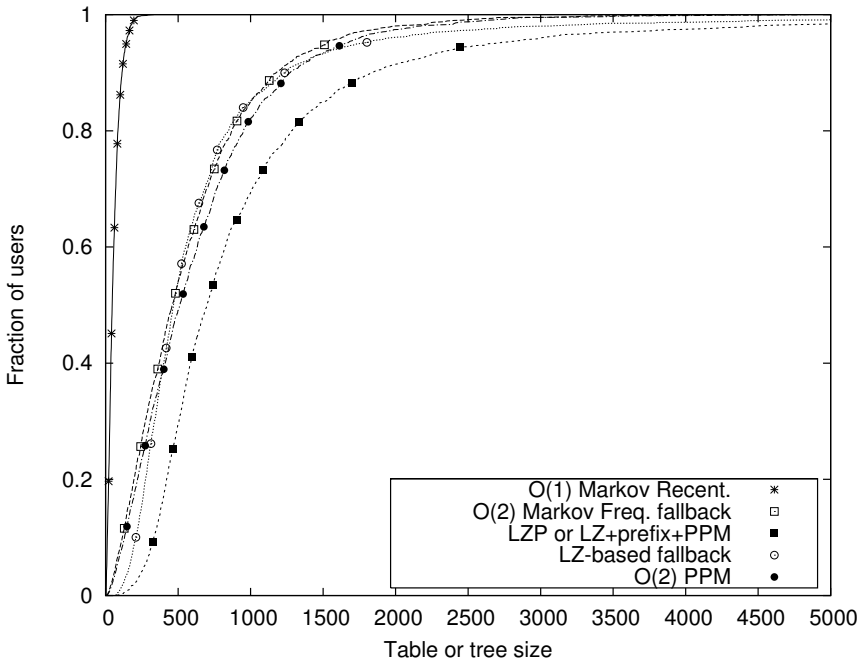


Figure 39. Distribution of final table sizes

G. Other metrics

Of course, prediction accuracy is not the only metric by which to compare location predictors. In Figure 39 we show the size of the predictors' data structures, at the end of simulating each user.[1] As with the accuracy metric, we plot the table size for each predictor as a distribution across all users. For Markov predictors, we define the size to be the number of entries in the (sparse) transition matrix, except for the recency-based $O(1)$ Markov, which simply needs to record the location of the latest transition for each location ever visited. For LZ predictors, we define the size to be the number of tree nodes. (Since the size of each table entry or tree node is implementation dependent, we do not measure table sizes in bytes.)

Clearly the recency-based $O(1)$ Markov had the simplest and smallest table, by far. In second place are the $O(2)$ Markov and LZ predictors. PPM used more space, and the LZP and LeZi predictors used by far the most space.

Although the medians of $O(2)$ Markov and LZ predictors appear to be similar, on a closer examination it is clear that the LZ predictor has a much higher maximum. Indeed, this plot is truncated at the right side; for one user, LZ used 17,374 nodes. LeZi used as many as 23,361 nodes! The Markov predictor,

[1]The table size of SPM predictor is not shown in the plot. To speed up the software development time, we used an inefficient data structure to represent the user trace. It is not fair to compare this data structure with the table size of other predictors. A more efficient data structure can be found at [JSA00].

on the other hand, never used more than 5,000 table entries.

Computational complexity is another important metric, in some applications; we leave the asymptotic analysis to the theoretical literature. Furthermore, we have not yet evaluated the running time of these predictors on our data (e.g., average microseconds per prediction), as we have not yet tuned our implementation. Nonetheless, it is intuitively clear that the simpler Markov predictors are likely to be faster than the LZ predictors.

IV. Conclusions

In this paper we compared four major families of domain-independent on-line location predictors (Markov, LZ, PPM, and SPM) by measuring their accuracy when applied to the mobility traces we collected from 6,202 users of Dartmouth College's wireless network.

Many of the traces in our collection were short, less than 100 movements, and all predictors fared poorly on most of those users. These predictors typically required a fair amount of history to initialize their probability tables to a useful state. In general we found that accuracy increases rapidly with trace length for short traces (less than 100 moves), slowly for medium traces (101-1000 moves) and slowly for long traces (over 1000 moves). Nonetheless in this paper we show results over all traces, sometimes focusing on long traces to highlight trends.

In general, the simple low-order Markov predictors worked as well or better than the more complex compression-based predictors, and better than high-order Markov predictors. In particular, $O(3)$ (or above) Markov predictors did not improve over $O(2)$ Markov, and indeed reduced accuracy by failing to make predictions much of the time.

Most of the predictors, as defined in the literature, fail to make a prediction when faced with a context (recent history) that has never been encountered in the full user's history. We found it was simple to add "fallback" to most predictors, allowing the predictor to use shorter and shorter context until it was able to make a prediction, and that this fallback often improved the predictor's accuracy substantially.

In particular, $O(2)$ Markov with fallback beat or equaled all of the other predictors, even though the latter have better theoretical asymptotic behavior [CJv03]. We found $O(2)$ Markov with fallback to be the best overall predictor, was simple to implement, had relatively small table size, and had the best overall accuracy.

We introduced and evaluated a simple alternative to the frequency-based approach to Markov predictors, using recency (probability 1 for most recent, 0 for all other) to define the transition matrix. Although this recency approach was best among $O(1)$ Markov predictors, it was worst among $O(2)$ Markov predictors, and we are still investigating the underlying reason.

We found most of the literature defining these predictors to be remarkably insufficient at defining the predictors for implementation. In particular, none defined how the predictor should behave in the case of a tie, that is, when there was more than one location with the same most-likely probability. We investigated a variety of tie-breaking schemes within the Markov predictors, although we found that the accuracy distribution was not sensitive to the choice for individual users it often made a big difference in accuracy.

Since some of our user traces extend over weeks or months, it is possible that the user's mobility patterns do change over time. All of our predictors assume the probability distributions are stable. We briefly experimented with extensions to the Markov predictors that "age" the probability tables so that more recent movements have more weight in computing the probability, but the results were mixed. The accuracy distribution of $O(1)$ Markov were improved significantly with most "age" factors, while $O(2)$ Markov were not affected much. However, the $O(2)$ Markov with fallback predictor had worse prediction accuracy distribution when we applied the aging method.

We examined the original LZ predictor as well as two extensions, prefix and PPM. LZ with both extensions is known as LeZi. We found that both extensions did improve the LZ predictor's accuracy, but that the simple addition of fallback to LZ did just as well, was much simpler, and had a much smaller data structure. To be fair, LeZi tries to do more than we require, to predict the future path (not just the next move). We also examined both the PPM and SPM predictors, but found that they did had little or no improvement compared to $O(2)$ Markov with fallback. This result is interesting since the LZ-based predictors and SPM have better theoretical behavior (asymptotically optimal for a larger class of sources). However, for our real data, with its large number of short and medium traces (as well as other phenomena which are hard to capture theoretically), we find that $O(2)$ Markov with fallback performs better in practice.

We stress that all of our conclusions are based on our observations of the predictors operating on about 6000 users, and in particular whether a given predictor's accuracy distribution seems better than another predictor's accuracy distribution. For an individual user the outcome may be quite different than in our conclusion. We plan to study the characteristics of individual users that lead some to be best served by one predictor and some to be best served by another predictor.

There was a large gap between the predictor's accuracy distribution and the "optimal" accuracy bound (a rather coarse upper bound), indicating that there is substantial room for improvement in location predictors. On the other hand, our optimal bound may be overly optimistic for realistic predictors, since it assumes that a predictor will predict accurately whenever the device is at a location it has visited before.

We suspect that domain-specific predictors will be necessary to come anywhere close to this bound.

Overall, the best predictors had an accuracy of about 65–72% for the median user. On the other hand, the accuracy varied widely around that median. Some applications may work well with such performance, but many applications will need more accurate predictors; we encourage further research into better predictors, as long as they are experimentally verified.

We continue to collect syslog data, extending and expanding our collection of user traces. We plan to evaluate domain-specific predictors, develop new predictors, and develop new accuracy metrics that better suit the way applications use location predictors. We also plan to extend the predictors to predict the time of the next move, as well as the destination.

ACKNOWLEDGMENTS

The Dartmouth authors thank DoCoMo USA Labs, for their funding of this research, and Cisco Systems, for their funding of the data-collection effort. We thank the staff of Computer Science and Computing Services at Dartmouth College for their assistance in collecting the data.

REFERENCES

[BCW90] Timothy C. Bell, John G. Cleary, and Ian H. Witten. *Text Compression*. Prentice Hall, 1990.

[BD02] Amiya Bhattacharya and Sajal K. Das. LeZi-Update: An information-theoretic approach to track mobile users in PCS networks. *ACM/Kluwer Wireless Networks*, 8(2-3):121–135, March–May 2002.

[CJv03] Christine Cheng, Ravi Jain, and Eric van den Berg. Location prediction algorithms for mobile wireless systems. In M. Illyas and B. Furht, editors, *Handbook of Wireless Internet*. CRC Press, 2003.

[CT97] John G. Cleary and W. J. Teahan. Unbounded length contexts for PPM. *The Computer Journal*, 40(2/3), 1997.

[CW84] John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, April 1984.

[DCB+02] Sajal K. Das, Diane J. Cook, Amiya Bhattacharya, Edwin Heierman, and Tze-Yun Lin. The role of prediction algorithms in the MavHome smart home architecture. *IEEE Wireless Communications*, 9(6):77–84, 2002.

[DS99] Sajal K. Das and Sanjoy K. Sen. Adaptive location prediction strategies based on a hierarchical network model in a cellular mobile environment. *The Computer Journal*, 42(6):473–486, 1999.

[FMG92] Meir Feder, Neri Merhav, and Michael Gutman. Universal prediction of individual sequences. *IEEE Transactions on Information Theory*, 38(4):1258–1270, July 1992.

[JSA00] Philippe Jacquet, Wojciech Szpankowski, and Izydor Apostol. An universal predictor based on pattern matching, preliminary results. In Daniele Gardy and Abdelkader Mokkadem, editors, *Mathematics and Computer Science: Algorithms, Trees, Combinatorics and Probabilities*, chapter 7, pages 75–85. Birkhauser, 2000.

[KE02] David Kotz and Kobby Essien. Analysis of a campus-wide wireless network. In *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking*, pages 107–118, September 2002. Revised and corrected as Dartmouth CS Technical Report TR2002-432.

[KE03] David Kotz and Kobby Essien. Analysis of a campus-wide wireless network. *ACM Mobile Networks and Applications (MONET)*, 2003. Accepted for publication.

[KV98] P. Krishnan and Jeffrey Scott Vitter. Optimal prediction for prefetching in the worst case. *SIAM Journal on Computing*, 27(6):1617–1636, 1998.

[LAN95] David A. Levine, Ian F. Akyildiz, and Mahmoud Naghshineh. The shadow cluster concept for resource allocation and call admission in ATM-based wireless networks. In *Mobile Computing and Networking*, pages 142–150, 1995.

[LJ96] George Liu and Gerald Maguire Jr. A class of mobile motion prediction algorithms for wireless mobile computing and communications. *ACM/Baltzer Mobile Networks and Applications (MONET)*, 1(2):113–121, 1996.

[SG98] William Su and Mario Gerla. Bandwidth allocation strategies for wireless ATM networks using predictive reservation. In *Proceedings of Global Telecommunications Conference (IEEE Globecom)*, volume 4, pages 2245–2250, November 1998.

[SKJH04] Libo Song, David Kotz, Ravi Jain, and Xiaoning He. Evaluating location predictors with extensive wi-fi mobility data. In *Proceedings of The 23rd Conference of the IEEE Communications Society (INFOCOM)*, Hong Kong, China, March 2004.

[VK96] Jeffrey Scott Vitter and P. Krishnan. Optimal prefetching via data compression. *Journal of the ACM*, 43(5):771–793, 1996.

[YL02] Fei Yu and Victor C. M. Leung. Mobility-based predictive call admission control and bandwidth reservation in wireless cellular networks. *Computer Networks*, 38(5):577–589, 2002.

[ZL78] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, September 1978.

APPENDIX

A. Definition of the Entropy

The standard definitions relating to entropy are as follows:

Definition 1: The entropy H(X) of a discrete random variable X is defined as

$$H(X) = -\sum_{x \in S} p(X = x) \log p(X = x)$$

The entropy is used to describe how uncertain of a random variable is. In other words, how much information on average is needed if we want to determine the value of X. The more uncertain a random variable is, the higher the entropy.

Definition 2: The joint entropy H(X,Y) of a pair of discrete random variable (X,Y) is defined as

$$H(X, Y) = -\sum_{x \in S} \sum_{y \in T} p(X = x, Y = y) \log p(X = x, Y = y)$$

Similarly, the joint entropy describes how much information on average is needed if we want to determine the both values of X and Y.

Definition 3: If $(X, Y) \sim p(x, y)$, the conditional entropy $H(X|Y)$ is defined as

$$\begin{aligned} H(X|Y) &= \sum_{x \in S} p(X = x) H(Y|X = x) \\ &= -\sum_{x \in S} p(X = x) \sum_{y \in T} p(Y = y|X = x) \log(p(Y = y|X = x)) \end{aligned}$$

Conditional entropy describes how much more information on average is needed if we want to determine the value of X when the Y is known.

B. Some Discussion

In our prediction research, the input to a predictor is a random variable. Entropy is used to describe the characteristic of a random variable. As a result, it is independent of prediction algorithm.

However, for each different predictor, the random variable fed into the predictor is different. As a result, we have different entropy calculations for different predictor.

For example, assume that we have a sequence which is $c = \{a, b, a, b, a\}$.

For a order-1 Markov predictor, the input random variable, h, which has 2 possible values, i.e. a, b; and $p(h = a) = 60\%$ and $p(h = b) = 40\%$ in this example.

For an order-2 Markov predictor however, the input random variable is different from the order-1 Markov, we call it h'. h' also has 2 possible values, i.e. $\{ab\}$, $\{ba\}$ and $p(h' = ab) = 50\%$ and $p(h' = ba) = 50\%$ in our example.

For random variables h and h', we can calculate entropy and they are different. The result tells us that order-1 has more uncertainty than order-2. Similarly, we can write down the entropy for the Order-k Markov predictor. Again, the entropy is different not because the predictor is different, but because the input random variable to a predictor is different even though these random variables are extracted from the same movement sequence.

C. Entropy of an Order-k Markov predictor

For an order-k Markov predictor, we predict the output based on the following metric.

$$\hat{p}(y = a|h(k)) = p\{y = a, h(k)\} = \frac{n_{\{a,h(k)\}}}{n_{total}}$$

where $h(k)$ is a sequence of k symbols and y is the prediction result. We always choose $y = a$ which leads to maximum $\hat{p}(y = a|h(k))$ as the prediction.

The entropy is then calculated as

$$\begin{aligned} H(Y|h(k)) &= -\sum_{y \in A} \hat{p}(y = a|h(k)) \log \widehat{p}\,(y = a|h(k)) \\ &= -\sum_{y \in A} p(y = a, h(k)) \log p(y = a, h(k)) \end{aligned}$$

In this calculation, since $h(k)$ is specified, the only random variable is y and it is a one-dimension random variable. In the predictor, the calculation described in this equation is performed on each row or column.

Since $h(k)$ is also a random variable, we can calculate the conditional entropy using the following equation

$$H(Y|h(k)) = \sum_{h(k) \in H_k} p(h = h(k)) H(Y|h = h(k))$$

where H_k is all the k-symbol sequence we have seen before. In our predictor, this means we will sum the results of each row using this equation.

D. Entropy of input sequence of an Order-K fall back Markov predictor

First of all, we have to deduce the probability of each output of an order-k Markov fall-back predictor. Assume we have seen a k-symbol sequence, $c = \{c_k, c_{k-1}, ..., c_2, c_1\}$, we use $h(k) = \{c_k, c_{k-1}, ..., c_2, c_1\}$, $h(k-1) = \{c_k, c_{k-1}, ..., c_2\}$,...,$h(1) = \{c_k\}$ to represent the input at each fall back stage. Note that for a given c, the $(h(i), i = 1....k)$ are deterministic.

We use $h(i), i = 1....k$ to represent all the i-symbol sequences we have observed. $p(h(k) \in H_k)$ is the probability that we have observed the $h(k)$ before. And $p(h(k) \notin H_k)$ is the probability that we have NOT observed the $h(k)$ before. It is clear that $p(h(k) \in H_k) + p(h(k) \notin H_k = 1$ and $p(h(k) \notin H_k)$ has only two possible values, 0 or 1.

$$\begin{aligned} &\hat{p}(Y = a|c) \\ &= \hat{p}(Y = a|h(k)) \times p(h(k) \in H_k) + \hat{p}(Y = a|h(k-1)) \times p(h(k) \notin H_k, h(k-1) \in H_{k-1}) \\ &+ ... + \hat{p}(Y = a|h(1)) \times p(h(k) \notin H_k, h(k-1) \notin H_{k-1}, ..., h(1) \in H_1) \\ &= p(Y = a, h(k)) \times p(h(k) \in H_k) + p(Y = a, h(k-1)) \times p(h(k) \notin H_k, h(k-1) \in H_{k-1}) \\ &+ ... + p(Y = a, h(1)) \times p(h(k) \notin H_k, h(k-1) \notin H_{k-1}, ..., h(1) \in H_1) \end{aligned}$$

If we assume that $h(i) \in H_i$ is independent of $h(j) \in H_j$, $\forall i \neq j$, we have

$$\begin{aligned} &\hat{p}(Y = a|c) \\ &= p(Y = a, h(k)) \times p(h(k) \in H_k) + p(Y = a, h(k-1)) \times p(h(k) \notin H_k) \times p(h(k-1) \in H_{k-1}) \\ &+ ... + p(Y = a, h(1)) \times p(h(k) \notin H_k) \times p(h(k-1) \notin H_{k-1}) \times ... \times p(h(1) \in H_1) \end{aligned}$$

when c is given, the equation above outputs all one term. When we calculate the entropy as follows:

$$
\begin{aligned}
&H(Y|c)\\
&= -\sum_{a\in A}\{[p(Y=a,h(k))\times p(h(k)\in H_k)+\ldots+p(Y=a,h(1))\times p(h(k)\notin H_k)\times p(h(k-1)\notin H_{k-1})\times\ldots\times p(h(1)\in H_1)]\\
&\times\log[p(Y=a,h(k))\times p(h(k)\in H_k)+\ldots+p(Y=a,h(1))\times p(h(k)\notin H_k)\times p(h(k-1)\notin H_{k-1})\times\ldots\times p(h(1)\in H_1)]\}\\
&=-\{\sum_{a\in A}\{p(Y=a,h(k))\times p(h(k)\in H_k)\times\log[p(Y=a,h(k))\times p(h(k)\in H_k)]\}\\
&+\ldots+\\
&\sum_{a\in A}\{p(Y=a,h(1))\times p(h(k)\notin H_k)\times\ldots\times p(h(1)\in H_1)\times\log[p(Y=a,h(1))\times p(h(k)\notin H_k)\times\ldots\times p(h(1)\in H_1)]\}\}
\end{aligned}
$$

Note that many terms are canceled. We can reduce the equation to

$$
\begin{aligned}
&\sum_{a\in A}p(Y=a,h(k))\times p(h(k)\in H_k)\times\log[p(Y=a,h(k))\times p(h(k)\in H_k)]\\
&=\sum_{a\in A}p(Y=a,h(k))\times p(h(k)\in H_k)\times\{\log[p(Y=a,h(k))]+\log[p(h(k)\in H_k)]\}\\
&=\sum_{a\in A}p(Y=a,h(k))\times p(h(k)\in H_k)\times\log[p(Y=a,h(k))]\\
&\underbrace{+\sum_{a\in A}p(Y=a,h(k))\times p(h(k)\in H_k)\times\log[p(h(k)\in H_k)]}_{=0}\\
&=-H(Y=a|h(k))\times p(h(k)\in H_k)
\end{aligned}
$$

As a result, the equation can be reduced to

$$
\begin{aligned}
&H(Y|c)\\
&=H(Y|h(k))\times p(h(k)\in H_k)+\ldots+H(Y|h(1))\times p(h(k)\notin H_k)\times\ldots\times p(h(1)\in H_1
\end{aligned}
$$

Since c is also a random variable, we calculate the conditional entropy $H(Y|C)$

$$H(Y|C)=\sum_{c\in R}p(C=c)H(Y|C=c)$$

Since $p(h(k)\in H_k)\sim c$, for each different c, the $H(Y|C=c)$ is different.