USING A DATA MINING APPROACH FOR THE PREDICTION OF USER MOVEMENTS IN MOBILE ENVIRONMENTS

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Gökhan Yavaş
December, 2003

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Özgür Ulusoy (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Uğur Güdükbay

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. İbrahim Körpeoğlu

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray

Director of the Institute

ABSTRACT

USING A DATA MINING APPROACH FOR THE PREDICTION OF USER MOVEMENTS IN MOBILE ENVIRONMENTS

Gökhan Yavaş
M.S. in Computer Engineering
Supervisor: Prof. Dr. Özgür Ulusoy
December, 2003

Mobility prediction is one of the most essential issues that need to be explored for mobility management in mobile computing systems. In this thesis, we propose a new algorithm for predicting the next inter-cell movement of a mobile user in a Personal Communication Systems network. In the first phase of our three-phase algorithm, user mobility patterns are mined from the history of mobile user trajectories. In the second phase, mobility rules are extracted from these patterns, and in the last phase, mobility predictions are accomplished by using these rules.

The performance of the proposed algorithm is evaluated through simulation as compared to two other prediction methods. The performance results obtained in terms of *Precision* and *Recall* indicate that our method can make more accurate predictions than the other methods.

Keywords: Location prediction, data mining, mobile computing, mobility patterns, mobility prediction

ÖZET

MOBİL SİSTEMLERDE VERİ MADENCİLİĞİ KULLANILARAK KULLANICI HAREKETLERİNİN TAHMİNİ

Gökhan Yavaş
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Özgür Ulusoy
Aralık, 2003

Mobil bilgisayar sistemleri kapsamında yürütülen en önemli araştırma konularından biri de konum bilgisi yönetimidir. Konum bilgisi yönetimi, sistemden servis alan mobil kullanıcıların, zamana bağlı olarak değişen konum bilgilerinin uygun metodlar kullanarak güncellenmesi, saklanması ve gerektiğinde kullanılması konularını içerir. Son zamanlarda, hareket tahmini de konum bilgisi yönetimi alanında başlıca araştırma konularından biri haline gelmiştir.

Bu tezde, mobil kullanıcıların hareket modellerinin veri madenciliği kullanılarak çıkarılması ve bu modeller kullanılarak mobil kullanıcıların daha sonraki hareketlerinin tahmin edilmesi için yeni bir algoritma geliştirilmiştir. Üç aşamadan oluşan bu algoritmanın ilk aşamasında kullanıcı hareket modelleri, kullanıcıların önceden kaydedilmiş mobil yörüngelerinden veri madenciliği kullanılarak çıkarılmaktadır. İkinci aşamada bulunan hareket modellerinden hareket kuralları üretilmekte, son aşamada ise bu hareket kuralları kullanıcının bir sonraki hücreler arası hareketinin tahmini için kullanılmaktadır.

Sunulan algoritmanın performansı simülasyonlar yardımıyla iki farklı tahmin yöntemiyle karşılaştırılmıştır. Performans sonuçları algoritmamızın diğer metodlardan daha doğru tahminler yapabildiğini göstermektedir.

Anahtar Sözcükler: Yer tahmini, veri madenciliği, mobil bilgisayar sistemleri, hareket modelleri, hareket tahmini

Acknowledgements

First of all, I would like to express my gratitude to my supervisor Prof. Dr. Özgür Ulusoy for his guidance during my graduate study.

I would like to thank Assist. Prof. Dr. Uğur Güdükbay and Assist. Prof. Dr. İbrahim Körpeoğlu, for spending their time and effort to read and comment on this thesis.

I would also like to acknowledge the financial support of TÜBİTAK under the grant 102E201.

Finally, I would like to thank my family for their great support, love and patience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Personal Communication Systems (PCSs) are becoming more popular by the help of the recent developments in the computer and communication technologies. In the near future, PCSs will support a huge user population and offer services that will allow the users to access various types of data such as video, voice and images. A PCS allows dynamic relocation of mobile users since these systems are based on the notion of wireless access. Mobility of the users in PCSs gives rise to the problem of mobility management.

Mobility management in mobile computing environments covers the methods for storing and updating the location information of mobile users who are served by the system. A hot topic in mobility management research field is mobility prediction. Mobility prediction can be defined as the prediction of a mobile user's next movement where the mobile user is traveling between the cells of a PCS or GSM network. The predicted movement can then be used to increase the efficiency of PCSs. By using the predicted movement, the system can effectively allocate resources to the most probable-to-move cells instead of blindly allocating excessive resources in the cell-neighborhood of a mobile user.

Up until now, there has been a considerable amount of research on mobility management. Most of the research has focused on the problem of location update, which is concerned with the reporting of the up-to-date cell locations by the mobile users to the PCS network [4]. Location update should be performed whenever a mobile user moves to another cell in the network to be able to track the exact location of each mobile user. When an incoming call arrives, the network simply routes the call to the last reported location of the mobile user. Compared to the amount of work performed on location update, little has been done in the area of mobility prediction [1, 6, 7, 8, 10, 11]. These works have some deficiencies, which are explained in the following:

- Some of these works do not attempt to find mobility patterns. Instead, the patterns are assumed to be already available. These patterns are then used for mobility prediction.

- In some of these works, prediction is based on the probability distribution of the speed and direction of the mobile user. For collecting such information, highly sophisticated and expensive tools such as GPS (Global Positioning System) are needed.

- Most of the methods studied in these works are highly sensitive to a change in a mobile user's path. For this reason, the prediction accuracy drops in case of noisy data. These methods do not consider the difference between the randomness and the regularity in users' paths (i.e., they do not distinguish a random movement and a regular movement of a user). In general, users follow some path patterns when traveling in network and their random movements are relatively few when compared to regular movements. Therefore random and regular movements should not be treated equally.

Aiming to overcome the above deficiencies, we have developed an effective mobility prediction algorithm. In the first phase of this three-phase algorithm, movement data of mobile users is mined for discovering regularities in inter-cell movements. These regularities are called mobility patterns. Mobility rules are then extracted from the

mobility patterns in the second phase of our algorithm. In the third phase, the mobility rules, which match the current trajectory of a mobile user, are used for the prediction of the user's next movement. The first two phases of our prediction algorithm, which are user mobility pattern mining and mobility rule generation, are accomplished offline by the system. However, the last phase, i.e., the mobility prediction, is accomplished online. It means that whenever a user intends to make an inter-cell movement, a prediction request is sent to the system and the prediction is made by the system based on our mobility rule based prediction algorithm.

The rest of this thesis is organized as follows. In Chapter 2, we present the network model we have used in this work, formulate the problem that we deal with, and present the related work. Our method for the solution of the problem is proposed in Chapter 3. We present the experimental results in Chapter 4 and conclude our thesis in Chapter 5.

Chapter 2

Background Work

2.1 Problem Definition

In our work, we assume that the mobile users move in a wireless PCS network, which has an architecture similar to those used in *EIA/TIA IS41* and GSM standards [14]. The coverage area of the PCS network is partitioned into smaller areas which are called *cells*. In each cell in the PCS network, there is a base station (BS) which has the capability of broadcasting and receiving information. The base stations are connected to each other via a fixed wired network. Mobile users use radio channels to communicate with base stations.

The coverage area consists of a number of *location areas*. Each location area may consist of one or more cells but in our work we assume that each location area consists of only one cell. Base stations regularly broadcast the ID of the cell in which they are located. Therefore, the mobile users which are currently in this cell and listening to the broadcast channel will know in which cell they are now. The movement of a mobile user from his current cell to another cell will be recorded in a database which is called *home location register* (HLR). In addition, every base station keeps a database in which the profiles of the users located in this cell are recorded. This database is called *visitor*

location register (VLR). Therefore, in our system it is possible to get the movement history of a mobile user from the logs on its home location register.

Since mobile users may initiate calls to other users or receive incoming calls while moving in the coverage region, the ongoing calls should be transferred from one cell to another without call dropping. To avoid call dropping due to insufficient resources at the destination cell, apriori resource allocation could be employed at that cell.

In our work, we collect the movement trajectories of a user in the form of $T = <(id_1, t_1), (id_2, t_2) \ ...,(id_k, t_k)>$. Here id_1 denotes the ID number of the cell to which the user enters at time t_1. In this record it is clear that two consecutive ID numbers must be the ID numbers of two neighbor cells in the network. After the movement history of a user is collected in a predefined time interval in the above format, this record is partitioned into subsequences. This procedure is accomplished as follows: If the mobile user stays in a cell id_i more than a threshold value, before moving to another one id_{i+1} at t_{i+1}, we assume that his trajectory up until now $<id_1,... id_i>$ ends here, and at id_{i+1} a new trajectory is started. Therefore, the first subsequence is $<id_1, ...id_i>$. By continuing in this manner the record is partitioned into subsequences, and these subsequences are recorded to be used in our algorithm.

We name the trajectories obtained by the above procedure as *user actual paths* (UAPs). We consider the UAPs as a valuable source of information because the mobility of the users contains both regular and random patterns [8]. Therefore by using the UAPs, we may be able to extract the regular patterns and use them in prediction.

We assume that we have UAPs which have the form $U = <c_1, c_2, \ ..., c_n>$. In this notation, each c_k denotes the ID number of the k^{th} cell in the coverage region. In finding the trajectories that are frequently used by the mobile users, we generalize the pattern mining method presented in [2, 3], to be used in our domain. The method presented in that work was intended for mining the frequent user access patterns from web logs, and then using these access patterns for effective caching and prefetching.

We name the frequently followed trajectories as *user mobility patterns* (UMPs). Mining of the UMPs enables us to generate *mobility rules*. By considering the mobility rules and the trajectory of a user, we predict the next inter-cell movement of the user. In the next chapter, we describe the algorithm developed for accomplishing the above issues.

2.2 Related Work

Data mining can be defined as the process of discovering interesting knowledge, such as patterns, associations, changes, anomalies and significant structures, from large amounts of data stored in databases, data warehouses, or other information repositories. Association rule mining [13] is the data mining task in which association relationships between sets of items of a transactional database are discovered. The roots of our method for sequential pattern mining go back to the Apriori algorithm [13].

In the association rule mining terminology, an *itemset* is a collection of items, the *support* of an itemset is the fraction of transaction itemsets that contains the itemset and *large* is an itemset that has support larger than a user-defined threshold. The Apriori algorithm is a level wise algorithm and makes multiple passes over the data for discovering large itemsets. In each subsequent pass, the algorithm starts with a seed set of itemsets found to be large in the previous pass. This seed set is then used to generate new itemsets, which are potentially large. These itemsets are called the *candidates*. During the next pass over the database, the supports for these candidates are counted and at the end of the pass, the candidates, which are large, become the seed for the next pass. This process continues until there is no new large itemsets found.

Association rule mining differs from sequential pattern mining, because in association rule mining, the ordering of the items in an itemset is not considered. In the algorithms used for mining association rules, support counting is done with the *subset* criterion, which does not take into account the ordering of items inside a transaction. With respect to this criterion, if an itemset *A* is a subset of another itemset *B*, then it is

said that itemset *A* is supported by itemset *B*. However, in sequential pattern mining the ordering of the items in an itemset must be taken into consideration.

The sequential pattern mining problem is studied in [5]. For our domain, the mobile users are assumed to be moving between the cells of a PCS network. However, the algorithms proposed in [5] cannot be applied directly to our domain for mining mobility patterns. Because, these algorithms don't take into account the network topology while generating the candidate patterns. This weakness of the proposed algorithms gives rise to generation of candidate patterns, which can not exist as mobility patterns on the corresponding network, since only the sequence of neighboring cells of the network can be considered as a mobility pattern. Therefore, the number of candidates generated can be extremely high, and this factor can dramatically reduce the performance of the mining algorithm.

In [2, 3], sequential pattern mining is applied to the domain of predictive Web prefetching. Web prefetching can be defined as deriving users' future requests for Web documents based on their previous requests. For effectively predicting the users' future requests, user access patterns are mined from the Web logs of users' previous requests and then these patterns are used for prefetching. The method presented in [2, 3] extends existing algorithms for mining sequential patterns in order to take the graph structure of the corresponding Web site into account during support counting, candidate generation and pruning. As we describe in Section 3.1, in the first phase of our mobility prediction algorithm, we generalize the method presented in [2, 3] to be able to mine mobility patterns of users in mobile computing environments. In the latter stages of our algorithm, mobility rules are extracted from the mobility patterns, and by using these rules, user movements are predicted.

There has been a considerable amount of research in mobility prediction, as well. The work presented in [7] is among the pioneering research for predicting the mobile users' movement behavior. In this work, user's moving behavior is modeled as repetitions of some elementary movement patterns which are indeed circular and straight line patterns.

In order to estimate the future location of a user, a mobile motion prediction (MMP) algorithm is proposed. However, the MMP algorithm is highly sensitive to random movements of the user. It is reported in [7] that as the random movements of the user increase the performance of MMP decreases linearly.

The work in [8] proposes a two level scheme, which combines a local with a global prediction model. The top level is the *global mobility model* (GMM), whose resolution is determined in terms of the cells crossed by a mobile user during the lifetime of the connection. The bottom level is the *local mobility model* (LMM), whose resolution is determined in terms of a 3-tuple sample space (speed, direction, position) that varies with time. LMM is used to model the intra-cell movements of the mobile users. On the other hand, GMM is used to predict the inter-cell movement trajectory of a user by matching the user's actual path to one of the existing "mobility patterns". For this purpose pattern matching techniques are used. However, the weakness of the work is revealed at this point because there is no method presented in [8] to discover these mobility patterns.

In [6], a Gauss-Markov model is introduced, where a mobile user's current velocity and location is correlated in time to a various degree. Based on the Gauss-Markov model, a mobile user's future location is predicted by the network based on the information gathered from the user's last report of location and velocity. In [1], Aljadhai and Znati use a first-order autoregressive filter in order to determine the direction of movement of a user. It is claimed in that work that the proposed method guarantees that the predicted mobile direction is not affected by small deviations in the mobile user's direction.

In the work [10], for location prediction cell-to-cell transition probabilities of a mobile user is recorded, and based on this, resource allocation is done at the k most probable cells that are in the neighborhood of the current cell. Here k is a user-defined parameter.

In some of the other works such as [11, 9], data mining methods such as clustering and association rule mining are used for exploring mobility patterns. In [11], a new location tracking method called *behavior-based strategy* (BSS) is presented. The aim of this work is designing a better paging area for each mobile user for each time region. The moving behavior of each mobile is mined from long-term collection of the user's moving logs. Next, time varying probability of each mobile user is estimated by using user's moving behavior, and then optimal paging area of each time region is derived.

In [9], a method named dynamic *clustering based prediction* (DCP) of mobile user movements is presented. In this work, DCP is used for discovering user mobility patterns from collections of recorded mobile trajectories, and then these patterns are used for the prediction of movements and dynamic allocation of resources. Collected user trajectories are clustered according to their in-between similarity. Weighted edit distance measure [8] is used for determining the similarity between two trajectories. The clustering used in [9] is agglomerative. It means that initially every single trajectory forms a cluster itself. At each iteration of the clustering algorithm, two most similar clusters (i.e., clusters that are closest in terms of weighted edit distance) are merged to form a new cluster. Each cluster is represented by a number of cluster representative trajectories. After each merge operation, the representatives of new cluster are found to be the union of representative sets of the merged clusters. The merge operation continues until the number of the clusters is reduced to a predefined value. In the prediction phase, the representatives of the clusters are used. A mobile user's next trajectory is predicted by finding the best matching representative with its current trajectory. The best matching one has the minimum edit distance to the current trajectory. In case of more than one match, all matched representatives can be used for prediction.

Chapter 3

Mobility Prediction Based on Mobility Rules

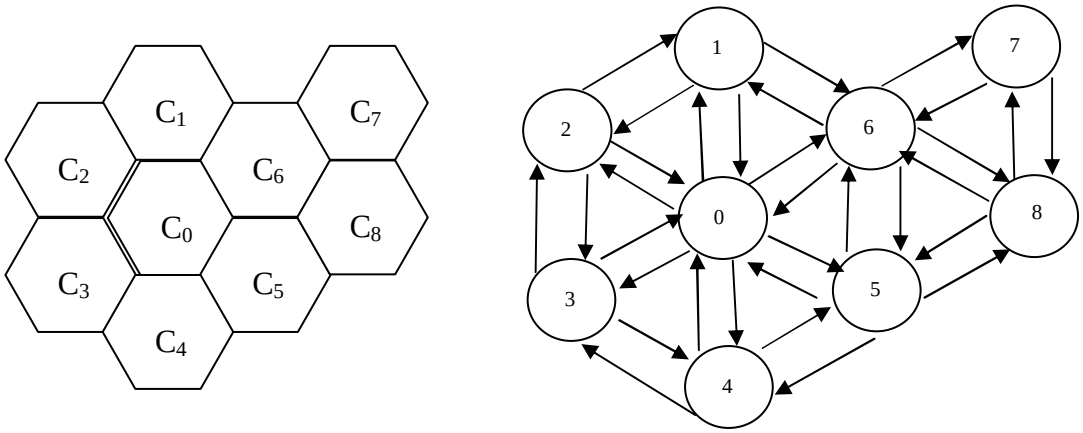
Our algorithm consists of three phases: user mobility pattern (UMP) mining phase, generation of mobility rules using the mined UMPs, and the mobility prediction phase. The next inter-cell movement of mobile users is predicted based on the mobility rules in the last phase. We examine each phase in detail in the following sections.

3.1 Mining User Mobility Patterns from Graph Traversals

In order to mine the UMPs from user actual paths (UAPs), *sequential pattern mining* [5] can be used. Sequential pattern mining has been previously used and examined in various research domains. One such work has been performed in the domain of web log mining [2, 3]. In that work, sequential pattern mining is used to mine the access patterns of a user while he is visiting the pages of web sites. This method assumes the web pages to be the nodes and the links between these pages to be the edges of an unweighted directed graph, *G*. Then, sequential pattern mining is applied to web logs by considering *G*.

In order to get a new method that is convenient for our domain, we generalize this method and apply it for UMP mining. In our method, we use a directed graph *G*, where

the cells in the coverage region are considered to be the vertices of *G*. The edges of *G* are formed as follows: If two cells, say A and B, are neighboring cells in the coverage region (i.e., A and B have a common border) then *G* has a directed and unweighted edge from A to B and also from B to A. These edges demonstrate the fact that a user can move from A to B or B to A directly. In Figure 3.1, an example coverage region and the corresponding graph *G* is presented.



(a) (b)

Figure 3.1: An example coverage region (a) and the corresponding graph *G* (b)

The algorithm we have developed for UMP mining is presented in Figure 3.2. To understand how the UMP mining algorithm works, assume that the set of candidate patterns each including *k* cells is found in the $(k\text{-}1)^{st}$ run of the while loop and this set is not empty (line 4, in Figure 3.2). The set of these patterns, denoted by C_k, is called *length-k candidate patterns*. Returning to the execution of our algorithm, from line 5 to line 12, first all the length-k subsequences of all UAPs are generated and these subsequences are used to count the supports of the length-k candidate patterns. In order to be more precise, the subsequence definition is given below:

Definition 1: Assume that we have two UAPs, $A = \langle a_1, a_2, \ldots, a_n \rangle$ and $B = \langle b_1, b_2, \ldots, b_m \rangle$. *B* is a ***subsequence*** of *A*, iff there exists integers $1 \le i_1 < \ldots < i_m \le n$ such that $b_k = a_{i_k}$, for all *k*, where $1 \le k \le m$.

UMPMining*()*

Input: All the UAPs in the database, D

Minimum value for support, $supp_{min}$

Coverage Region Graph, G

Output: User mobility patterns (UMPs), L

1. $C_1 \leftarrow$ the patterns which have a length of one
2. $k = 1$
3. $L = \varnothing$ // Initially the set of large patterns is empty
4. while $C_k \neq \varnothing$ {
5. foreach UAP $a \in D$ {
6. $S = \{s \mid s \in C_k$ and s is a subsequence of $a\}$
7. // S is the set of candidate length-k patterns which are also
8. // subsequences of UAP a
9. foreach $s \in S$ {
10. s.count = s.count + s.suppInc //increment the support of cands
11. }
12. }
13. // choose the candidates which has enough support
14. $L_k = \{ s \mid s \in C_k, s.\text{count} \geq supp_{min} \}$
15. $L = L \bigcup L_k$ // add these length-k large patterns to the set of all large patterns
16. // Generate length-(k+1) candidate patterns
17. $C_{k+1} \leftarrow$ CandidateGeneration(L_k, G), $\forall c \in C_{k+1}$ c.count = *0*
18. $k = k+1$
19. }
20. return L

Figure 3.2: User Mobility Pattern Mining Algorithm

In other words, *B* is a subsequence of *A*, iff all cells in *B* also exist in *A* while keeping their order in *B* (but they don't need to be consecutive in *A*).

Let's give an example by using the coverage region given in Figure 3.1: assume $A=<c_3, c_4, c_0, c_1, c_6, c_5>$, then $B=<c_4, c_5>$ will be a length-2 subsequence of *A*. In other words, the UAP *B* is ***contained*** by the UAP *A*.

In line 10 of the mining algorithm, we see that every candidate '*s*' has a *count* value and this value is incremented by '*s.suppInc*' value. The count value of a candidate keeps the support given to this candidate by the UAPs. This is the point where our algorithm extends the method presented in [2, 3]. The method presented in that work, increments the count value of a candidate by 1 if this candidate is *contained* by a UAP. By this method, the effect of possible noise in the data is minimized. Because the users who are following a UMP may follow random paths between the consecutive cells of this UMP. These paths can be characterized as noise and the UAPs containing noise are called corrupted. If the number of corrupted UAPs in the data is high, then a pattern may not have an adequate support and it will be missed.

However, this method of support counting treats a highly corrupted candidate pattern and a slightly corrupted (or even not corrupted at all) candidate pattern in the same way and assigns the support value of 1 to both patterns. Since this method is unfair for the context of mobile motion prediction, unlike the work in [2, 3], our support counting method considers the degree of corruption, i.e., we differentiate the support given to a slightly corrupted pattern and to a highly corrupted pattern. In our method, we calculate the support assigned to a candidate pattern *B* by an UAP *A* (i.e., *suppInc*) by using the following formula:

$$suppInc = \begin{cases} \dfrac{1}{1+totDist}, & \text{if pattern } B \text{ is contained by UAP } A. \\ 0, & \text{otherwise} \end{cases}$$

We can define the *totDist* value by means of the notion of string alignment [12]. Given two strings, where a string is a sequence of characters, Gusfield demonstrates methods for determining the similarity between these two strings by finding the optimal alignment between them. For instance, assume that the two strings are "*acbcdb*" and "*cadbd*". Here is one possible alignment of these two strings, where the special character "-" represents the insertion of a space.

a	c	-	-	b	c	d	b
-	c	a	d	b	-	d	-

Definition 2.1: If *x* and *y* are each single character or space, then $\delta(x, y)$ denotes the score of aligning *x* and *y*. In our case, the scoring function is defined as follows:

$$\delta(x, x) = 0 \text{ and } \delta(x, y) = \delta(x, -) = \delta(-, x) = 1$$

Definition 2.2: If *S* is a string, then $|S|$ denotes the length of *S* and $S[i]$ denotes the ith character of *S* (where the first character is $S[1]$ rather than, say $S[0]$).

Definition 2.3: Let *A* be a UAP and *B* be a pattern. A ***containment alignment*** X' maps *A* and *B* into strings A' and B' that may contain space characters, where

1. $|A'| = |B'|$,
2. the pattern *B* is contained by the UAP *A*, and
3. the removal of all spaces from A' and B' leaves *A* and *B*, respectively.

The total score of the alignment X' is

$\sum_{i=k}^{m} \delta(A'[i], B'[i])$, where *k* is the index of first and *m* is the index of last non-space character in B'.

The above definition of containment alignment is an adaptation of the string alignment definition given in [12]. For any two patterns, there are many possible

containment alignments. For instance, assume that A=<c_3, c_4, c_0, c_1, c_6, c_5, c_8> and B=<c_4, c_5>. Then, two possible containment alignments for these patterns are:

1. A' = c_3 c_4 c_0 c_1 c_6 c_5 c_8
   B' = - c_4 - - - c_5 -

2. A' = c_3 - c_4 c_0 c_1 c_6 c_5 c_8
   B' = - - c_4 - - - c_5 -

Definition 2.4: An ***optimal containment alignment*** of UAP A and pattern B is one that has the minimum possible value for these two patterns. We call the containment alignment with minimum value optimal by the nature of our scoring function that was presented in Definition 2.1. As one can see, our scoring function gives a penalty of 1 for each mismatch in the alignment, and in Definition 2.3 the value of an alignment is defined as the sum of penalties which are naturally the result of mismatches. Therefore, the optimal alignment will have the minimum value, which denotes the minimum number of mismatches, and we call this value *totDist* for these patterns. Indeed, *totDist* gives us the exact number of cells which exist between the consecutive cells of B in A.

For instance, an optimal containment alignment for the patterns A and B will be:

A' = c_3 c_4 c_0 c_1 c_6 c_5 c_8
B' = - c_4 - - - c_5 -

The value of this optimal containment alignment is 3 and by Definition 2.4 *totDist=3*. Actually, it can be said that the pattern B is 3 cells corrupted with respect to pattern A. Therefore, the support value given to B by A is $\text{suppInc} = \frac{1}{1+3} = \frac{1}{4}$. It is easily seen that the quality of the patterns will improve since this method is a more accurate way of support counting. The improvement in the pattern quality will give rise to more accurate mobility rules. Therefore, the prediction accuracy by using these rules will be higher when compared to the accuracy by using the rules that are generated with the former way of support counting [2, 3]. Indeed, our support counting method is a generalization of the support counting method of [2, 3]. If we simply take *totDist* as 0, without

considering the degree of corruption, we will end up with the support counting method of that work. Therefore, we can claim that applying different methods for calculating *totDist* will affect the quality of the rules obtained. For this reason, an appropriate method, such as ours, should be selected for calculating this value.

For support counting and storing large patterns, a trie data structure is used as in the work [2, 3] instead of the hash-tree data structure recommended to be used for [13]. In the hash-tree the candidates exist only in the leaves of the tree. On the other hand, every trie node sequence, from root to any node, can represent a pattern in the trie data structure. This property of the trie structure provides efficiency in support counting procedure. Furthermore the trie data structure grows dynamically as its leaves are extended and there is no need to build repeatedly a new hash-tree for every iteration.

To count the supports of length-k candidate patterns, first all candidate patterns are inserted into the trie. Next the database of UAPs is scanned. For each UAP *A* of length *n*, all possible length-k subsequences and their *totDist* values should be determined (If n < k, then UAP *A* is skipped and not used in this phase of support counting). Then, each of these subsequences are searched in the trie and for those who exist in the trie, their support count is increased by *suppInc* value, which is calculated with the *totDist* value of the corresponding subsequence.

After counting the supports of all the candidates, the candidates which have a support smaller than the threshold value ($supp_{min}$) are eliminated. The remaining candidates are called the *length-k large patterns* (L_k). Then, L_k is added to the set in which all the large patterns are maintained.

The next step in the mining algorithm is the generation of *length-(k+1) candidate patterns*, C_{k+1}. For this step, the *CandidateGeneration()* function, presented in Figure 3.3, is used.

CandidateGeneration *()*

Input: Length-k large patterns, L_k

Coverage Region Graph, *G*

Output: Length-(k+1) candidate patterns, *Candidates*

1. *Candidates* = $\varnothing$ // Initially the candidates set is empty
2. foreach $L = <l_1, l_2, ..., l_k>,\ L \in L_k$ { // for each length-k large pattern *L*
3. // determine all the cells which are neighbors of l_k in *G*
4. $N^+ = \{v \mid$ there is an edge in *G* such as $l_k \rightarrow v\}$
5. foreach $v \in N^+(l_k)$ { // for each of these neighbor cells, *v*
6. // generate a candidate by attaching *v* to end of *L*
7. $C' = < l_1, l_2, ..., l_k, v>$
8. $S = \{s \mid s$ is a length-k subsequence of C' and *s* is a path in *G*$\}$
9. if $\forall s \in S \Rightarrow s \in L_k$
10. // if all k-length subsequences of new C' is large then add it to cands set
11. $Candidates \leftarrow Candidates \bigcup C'$
12. }
13. }
14. return *Candidates*

Figure 3.3: Generation of length-(k+1) candidates

To illustrate how the candidate generation algorithm works, assume that there exists a pattern $C = <c_1, c_2, ..., c_k>$ in L_k which is given as the input to this algorithm. To generate the possible candidates from *C*, all the nodes in *G* which have an incoming edge from the cell c_k are assigned to a set which is denoted by $N^+(c_k)$. This is the set of all the cells to which a mobile user can move from c_k. Next, a cell, *v*, from $N^+(c_k)$ is attached to the end of the pattern C in order to generate a possible candidate $C' = <c_1, c_2, ..., c_k, v>$. If all the length-k subsequence patterns of C', which can exist as

paths in the corresponding network graph G, are elements of L_k, then C' is added to the length-(k+1) candidates set. This procedure is repeated for all the cells in the set $N^+(c_k)$.

Example: An example database of UAPs is given in Table 3.1.

UAP ID	UAP
1	<3, 0, 6, 7>
2	<3, 4, 5, 6, 7>
3	<2, 1, 6, 8>
4	<4, 5, 8, 7>
5	<0, 5, 8>
6	<4, 5, 6, 7>

Table 3.1: Database of User Actual Paths (UAPs)

In Tables 3.2, 3.3, 3.4, 3.5, the execution of the UMP mining algorithm with $supp_{min}$= 2 and graph *G* which is given in Figure 3.1 is illustrated on an example using the database of UAPs which is given in Table 3.1. In Table 3.2, set of length-1 candidate patterns (C_1) and set of length-1 large patterns (L_1) are given.

C_1	
CAND	SUPP
<0>	2
<1>	1
<2>	1
<3>	2
<4>	3
<5>	4
<6>	4
<7>	4
<8>	3

L_1	
PATTERN	SUPP
<0>	2
<3>	2
<4>	3
<5>	4
<6>	4
<7>	4
<8>	3

Table 3.2: Length-1 candidate patterns (C_1) and length-1 large patterns (L_1)

Next, C_2 is generated by using the candidate generation algorithm given in Figure 3.3 and, L_1 is used in this process. Then, the supports of these candidates are counted and the patterns which have a support value larger than $supp_{min}$ are assigned to set L_2. The sets C_2 and L_2 are presented in Table 3.3.

C_2			
CAND	SUPP	CAND	SUPP
<0, 3>	0	<5, 6>	2
<0, 4>	0	<5, 8>	2
<0, 5>	1	<6, 0>	0
<0, 6>	1	<6, 5>	0
<3, 0>	1	<6, 8>	1
<3, 4>	1	<6, 7>	3
<4, 0>	0	<7, 6>	0
<4, 3>	0	<7, 8>	0
<4, 5>	3	<8, 5>	0
<5, 4>	0	<8, 6>	0
<5, 0>	0	<8, 7>	1

L_2	
PATTERN	SUPP
<4, 5>	3
<5, 6>	2
<5, 8>	2
<6, 7>	3

Table 3.3: Length-2 candidate patterns (C_2) and length-2 large patterns (L_2)

Having L_2, C_3 is generated using *CandidateGeneration()* function, and then the large patterns in C_3 are assigned to the set L_3. These sets are shown in Table 3.4.

C_3	
CAND	SUPP
<4, 5, 8>	1
<4, 5, 6>	2
<5, 6, 7>	2

L_3	
PATTERN	SUPP
<4, 5, 6>	2
<5, 6, 7>	2

Table 3.4: Length-3 candidate patterns (C_3) and length-3 large patterns (L_3)

C_4 and L_4 contain only the pattern <4, 5, 6, 7>. By using L_4, *CandidateGeneration()* function can not generate any length-5 candidates. Therefore, the UMP mining algorithm terminates with the set of large candidates, L, which is shown in Table 3.5.

L			
PATTERN	SUPP	PATTERN	SUPP
<0>	2	<4, 5>	3
<3>	2	<5, 6>	2
<4>	3	<5, 8>	2
<5>	4	<6, 7>	3
<6>	4	<4, 5, 6>	2
<7>	4	<5, 6, 7>	2
<8>	3	<4, 5, 6, 7>	2

Table 3.5: The set of all large patterns

3.2 Generation of Mobility Rules

In the second phase of our movement prediction algorithm, the mobility rules which will be used in the next phase (i.e., the prediction phase) are generated. Having the UMPs mined in the previous phase, we can now produce the set of the mobility rules from these UMPs. Assume that we have a UMP $C = < c_1, c_2, ..., c_k >$, where k>1. All the possible mobility rules which can be derived from such a pattern are:

$< c_1 > \rightarrow < c_2, \ldots c_k >$

$< c_1, c_2 > \rightarrow < c_3, \ldots c_k >$

…

$< c_1, c_2, \ldots, c_{k-1} > \rightarrow < c_k >$

For a mobility rule, we call the part of the rule before the arrow the *head* of the rule, and the part after the arrow the *tail* of the rule. Moreover, when these rules are generated, a confidence value is calculated for each rule. For a mobility rule *R*: $< c_1, c_2, \ldots, c_{i-1} > \rightarrow < c_i, c_{i+1}, ... c_k >$, the confidence is determined by using the following formula:

$$confidence(R) = \frac{< c_1, c_2, \ldots, c_k > .count}{< c_1, c_2, \ldots, c_{i-1} > .count} \times 100$$

By using the mined UMPs, all possible mobility rules are generated and their confidence values are calculated. Then the rules which have a confidence higher than a predefined confidence threshold ($conf_{min}$) are selected. These rules are used in the next phase of our algorithm, which is the mobility prediction.

Example: All possible mobility rules and their confidence values for the UMPs given in Table 3.5 are demonstrated in Table 3.6.

Mobility Rules			
Rule	Conf	Rule	Conf
<4> → < 5>	100	<5> → <6, 7>	50
<5> → <6>	50	<5, 6> → <7>	100
<5> → < 8>	50	<4> → <5, 6, 7>	66.6
<6> → <7>	75	<4, 5> → <6, 7>	66.6
<4> → <5, 6>	66.6	<4, 5, 6> → <7>	100
<4, 5> → <6>	66.6		

Table 3.6: All possible mobility rules

If the threshold confidence value, $conf_{min}$, is assumed to be 50, then the rules having a confidence bigger than or equal to $conf_{min}$ will be the same as the rules in Figure 10 since all these rules have a confidence bigger than $conf_{min}$.

3.3 Mobility Prediction

This is the third and the last phase of our algorithm. The pseudo-code for the mobility prediction phase of our algorithm is presented in Figure 3. In this phase, the next movement of the mobile user is predicted. The prediction procedure can be summarized as follows: Assume that a mobile user has followed a path P=< c_1, c_2, …, c_{i-1} > up to now. Our algorithm finds out the rules whose *head* parts are *contained* in path P, and also the last cell in their *head* is c_{i-1}. We call these rules the *matching rules*. We store the first cell of the tail of each matching rule along with the confidence of the rule in an array of such tuples. The tuples of this array are then sorted in descending order with respect to their confidence.

Then, we define another parameter, *m*, which is the maximum number of predictions that can be made each time the user moves. For prediction, we select the first *m* tuples from the sorted tuples array. Then the cells of these tuples are our predictions for the next movement of the mobile user. It means that we use the first *m matching* rules that have the highest confidence for predicting the user's next movement.

MobilityPrediction*()*

Input: Current trajectory of the user, $P=< c_1, c_2, \ldots, c_{i-1} >$

Set of mobility rules, R

Maximum predictions made each time, m

Output: Set of predicted cells, *PCells*

1. *PCells* = ∅ // Initially the set of predicted cells is empty
2. $k = 1$
3. foreach rule $r : <a_1, a_2, ..., a_j> \rightarrow <a_{j+1}, ..., a_t> \in R$ { // check all the rules in R
4. // find the set of matching rules
5. if $<a_1, a_2, ..., a_j>$ is contained by $P=< c_1, c_2, \ldots, c_{i-1} >$ and $a_j = c_{i-1}$ {
6. // Add the rule into the set of matching rules
7. *MatchingRules* ← *MatchingRules* ⋃ *r*
8. // Add the (a_{j+1}, *r.confidence*) tuple to the *Tuples* array
9. *TupleArray[k]* = (a_{j+1}, *r.confidence*)
10. $k = k+1$
11. }
12. }
13. // Now sort the *Tuples* array w.r.t. the second element of the tuples
14. // (which is the confidence of the corresponding rule) in descending order
15. *TupleArray* ← *sort*(*TupleArray*)
16. *index* = 0
17. // Select the first *m* elements of the *Tuples* array
18. while (*index* < *m* && index < *TupleArray*.length){
19. *PCells* ← *PCells* ⋃ *TupleArray*[*index*]
20. *index* = *index*+1
21. }
22. return *PCells*

Figure 3.4: Mobility Prediction Algorithm

Example: Assume that a mobile user is traveling through the cells of the coverage region shown in Figure 1. Also the UAPs that the user has followed in its mobility history are given in Figure 4. Then, the mobility rules that are given in Figure 10 will be used in mobility prediction for this user. Moreover, suppose that the user has followed a path P=<2, 3, 4, 5> up to now and he is currently in cell 5. Our algorithm will find the rules <5> → <6>, <5> → < 8>, <4, 5> → <6>, <5> → <6, 7>, and <4, 5> → <6, 7> as the *matching rules*. The first cell in each rule's tail will be stored along with the rule's confidence in an array of *(cell, confidence)* tuples. If there are more than one tuple for a cell in the array, then the one which has the biggest confidence is kept and the others are deleted. Then, these tuples are sorted with respect to their confidence values in descending order. For our example, the sorted tuple array will be: *TupleArray* = [(6, 66.6), (8, 50)]. If *m* is equal to 1, then only cell 6 will be used for the prediction of user's next movement. If *m* is equal to 2, then both cells 6 and 8 are the predicted cells for the next movement.

Chapter 4

Experimental Results

4.1 Simulation Design

For simulation, we have adapted the simulation model which is presented in our earlier work [9]. In this model, it is assumed that a mobile user travels on a 15 by 15 hexagonal shaped network which gives a total of 225 base stations.

In order to generate the *user actual paths* (UAPs), first a number of *user mobility patterns* (UMPs) is generated. The length of a UMP is determined by a uniform distribution with a mean length *l*. Each UMP is taken as a random walk over the hexagonal network. There are two types of UAPs generated. The first type consists of UAPs that follow a UMP and the second type consists of *outliers* (i.e., those which don't follow a pattern). The ratio of the number of outliers to the number of UAPs that follow a UMP is denoted by *o*. For each new UAP we decide whether it is going to be an outlier or not, according to the value *o*. If it is an outlier, then it is formed as a random walk over the hexagonal network. Otherwise, a UMP is selected randomly that will correspond to the generated UAP. We also use a corruption mechanism to distinguish the UAP from its corresponding UMP. We insert random cells between the consecutive cells of the UMP. In order to accomplish this, we define a corruption ratio *c*, which denotes the ratio of the number of such random cells to the number of cells in the corresponding UMP.

Total number of UAPs is 10,000 and from these, we construct the training and test sets. The number of UAPs in training set is 9,000 and the number of UAPs in test set is 1,000. UMPs are mined from the UAPs in the training set and then the mobility rules that will be used in prediction are generated by using these UMPs. The UAPs in the test set are used for evaluating the prediction accuracy of our algorithm. There are two performance measures used for the evaluation of the proposed algorithm:

- Recall: the number of correctly predicted cells divided by the total number of requests (i.e., the total number of inter-cell movements that the user makes).

- Precision: the number of correctly predicted cells divided by the total number of predictions made.

The parameters used in the experiments and their default values are given in Table 4.1. The default values of *l*, *c* and *o* are adapted from [9].

Symbol	**Definition**	**Default values**
m	Maximum number of predictions made each time	2
l	Average length of UAPs	5
c	Corruption factor	0.4
o	Outlier Percentage	30 %
$supp_{min}$	Minimum support percentage	0.05 %
$conf_{min}$	Minimum confidence percentage	70 %

Table 4.1: Symbol table for the parameters used in our experiments

4.2 Algorithms Used for Comparison

We compared our UMP-Based mobility prediction method with two different prediction methods. The first method is *Mobility Prediction based on Transition Matrix* (TM). In this method, a cell-to-cell transition matrix is formed by considering the previous inter-cell movements of mobile users. The predictions are based on this transition matrix by selecting the *m* most probable cells as the predicted cells. The second prediction method is the *Ignorant Prediction*, which is presented in [15]. Ignorant Prediction method disregards the information available from movement history. To predict the next inter-

cell movement of a user, this method assigns equal transition probabilities to the neighboring cells of the user's currently residence cell. It means that prediction is performed by randomly selecting *m* neighboring cells of the current cell.

The first experiment is conducted for choosing the *m* (the maximum number of predictions made each time) value which is appropriate for all the methods. The next two experiments are conducted for tuning the parameters of our method, which are: $supp_{min}$ (the minimum support threshold used in UMP mining algorithm) and $conf_{min}$ (the minimum confidence threshold used in mobility rule generation algorithm). In these experiments, we search for the best values for each parameter that make both recall and precision good. The last two experiments are to measure the performance of our method as compared to the performance of other methods.

4.3 Impact of Maximum Number of Predictions

In the first experiment, we examine the performance impact of parameter *m*, maximum number of predictions made at each move of user. As Figure 4.1 indicates, the precision obtained by our method and the precision obtained by TM decrease as *m* increases. The decrease in precision obtained by TM is more dramatic when compared to that obtained by our method. The decrease in both precision values is due to the fact that as the number of predictions made at each movement of the user increases, the probability of having some incorrect predictions gets higher. Therefore, the number of correct predictions made by our method and TM doesn't increase in the same rate with the number of predictions.

On the other hand, the precision obtained by Ignorant Prediction method remains almost constant as *m* increases, ignoring some statistical variations. It is around 0.2 for all *m* values. As *m* increases, the total number of predictions and the number of correct predictions for this method increase at the same rate. This explains why the precision of the Ignorant method is fixed at 0.2. This value is very low when compared to the value obtained by our method. Moreover, if the hexagonal simulation network is perfect (i.e., all the cells in the network have 6 neighbors), we would expect that the precision value

of the Ignorant method should be fixed at $\frac{1}{6} = 0.1\overline{6}$. Our simulation network is not perfect because the cells that are at the corners have 2 or 3 neighbors. Besides, the cells which are along the left, right, top and bottom sides of the simulation network have less than 6 neighbors.

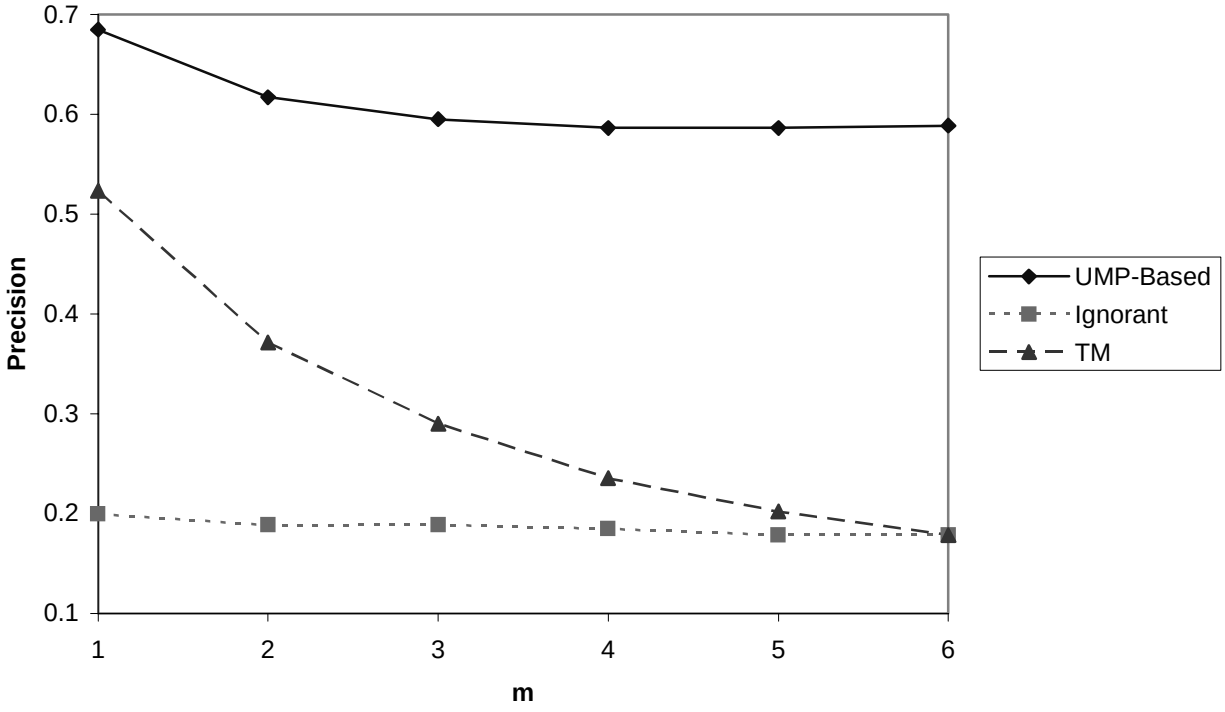


Figure 4.1: Precision as a function of the Maximum Number of Predictions made each time

The recall values for all methods increase with increasing *m* as shown in Figure 4.2. This observation can be explained by the fact that as the number of predictions made at each move of the mobile increases, the probability of predicting the correct cell increases. The increase in recall values with TM and Ignorant methods are more significant when compared to the increase with our method. For our method, beginning from *m*=3, recall values don't increase significantly and become almost fixed at around 0.54. This is because the number of matching rules is the same for all *m* values. Beginning from some *m* value, the number of correct predictions doesn't increase because the *m* value exceeds the number of matching rules. Therefore, the number of

correct predictions becomes stable making the recall value stable. For TM and Ignorant methods, recall values increase steadily, finally reaching to 1.

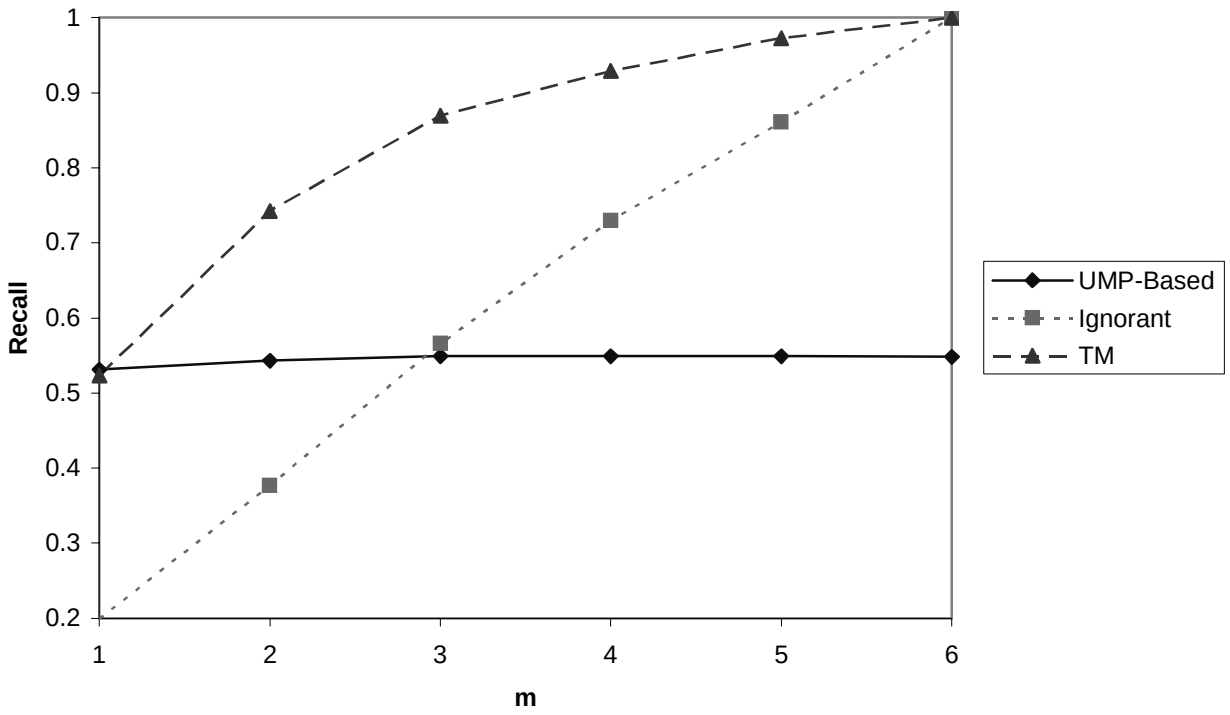


Figure 4.2: Recall as a function of the Maximum Number of Predictions made each time

By considering the above results, one can easily see that there is a trade-off between recall and precision measures. Therefore a middle ground should be found for the *m* value. The increase in recall with our method is not very significant when compared to that obtained with TM which is the actual competitor to our method. Thus, setting *m* as small as possible would be appropriate for our method since we don't want the precision to drop with increasing *m* because we don't gain anything in recall with higher *m* values. In addition, we can say that setting *m*=2 could be considered as a good choice for TM as well, because the increase rate in the recall value from *m* values 1 to 2 is maximum for TM. Since the precision value decreases with increasing *m* for TM, making *m* bigger than 2 does not increase the recall value so much that it would be worth to decrease the precision value. Moreover, if we set m bigger than 3, this would cause excessive

network resource waste. Therefore we will set m=2 for all the methods at the rest of the performance experiments.

4.4 Impact of Minimum Support Value

Next, we investigate the effect of increasing minimum support ($supp_{min}$) value on the recall and precision values obtained by our method. It is shown in Figures 4.3 and 4.4 that as the $supp_{min}$ increases, the precision and recall values decrease. This is due to the fact that the increase in the $supp_{min}$ value leads to a decrease in the number of mined mobility rules. Therefore, the number of correct predictions is reduced. This causes the recall and the precision values to decrease.

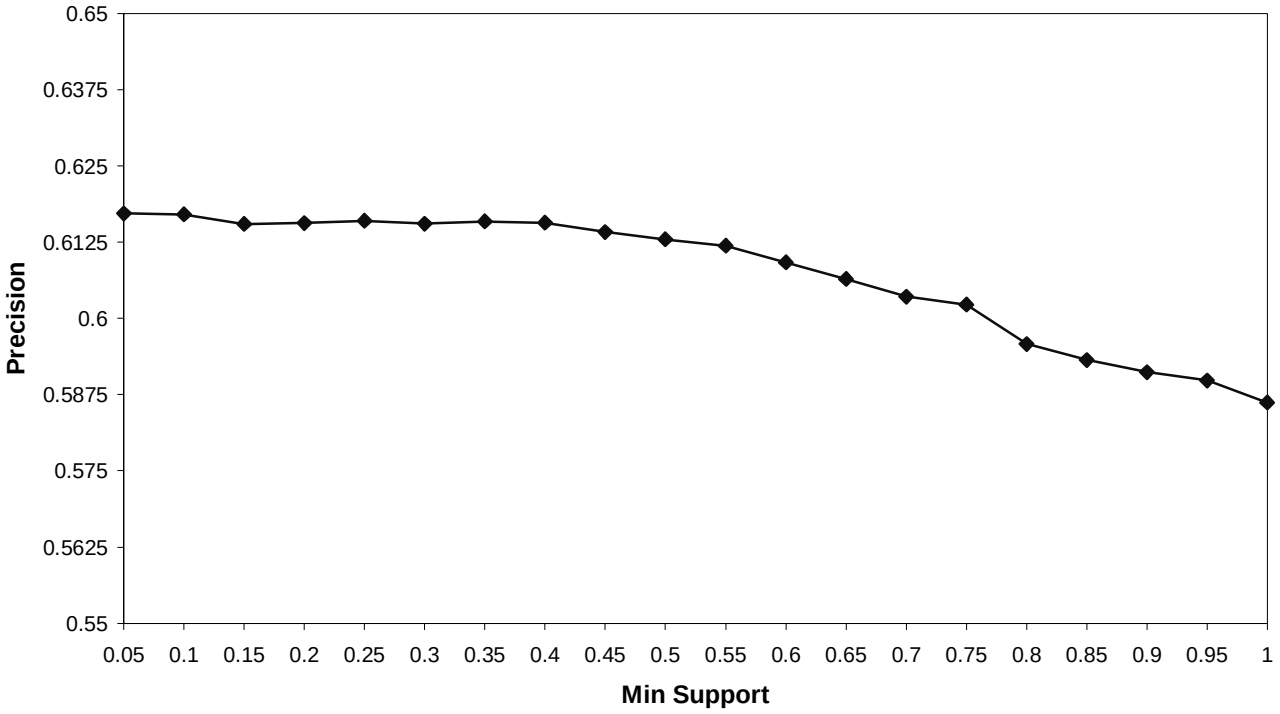


Figure 4.3: Precision as a function of the Minimum Support for UMP-Based Prediction algorithm

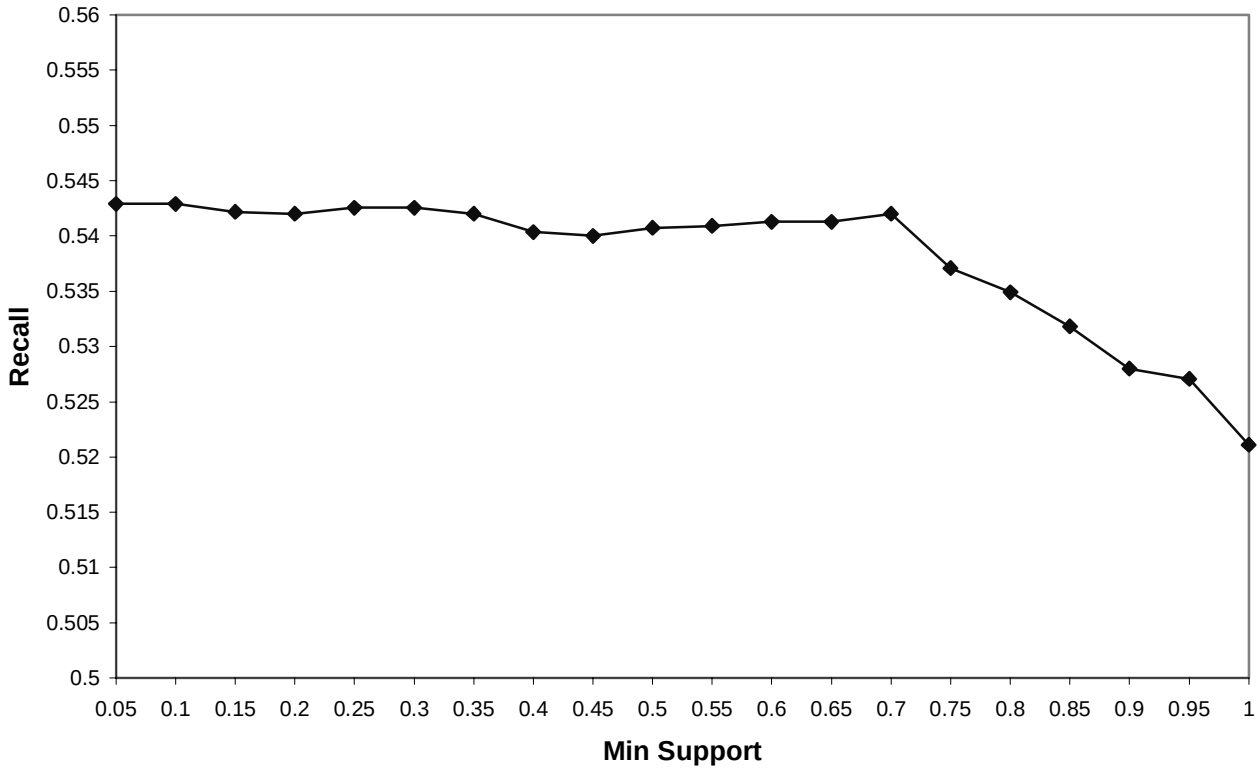


Figure 4.4: Recall as a function of the Minimum Support for UMP-Based Prediction algorithm

Since both recall and precision values decrease for increasing $supp_{min}$, it would be most appropriate to choose $supp_{min}$=0.05 which is the smallest value used in the experiments. We have observed that recall and precision values do not increase considerably (it can even be said that the values do not increase at all) for the $supp_{min}$ values smaller than 0.05.

4.5 Impact of Minimum Confidence Value

In this experiment, we examine the effect of increasing minimum confidence ($conf_{min}$) values on the recall and precision of our method. Figure 4.5 indicates the impact of minimum confidence on the precision. As one can realize, the precision increases as $conf_{min}$ increases. Even, the precision reaches to very high values such as 0.98 at $conf_{min}$ = 100. Because, at high $conf_{min}$ values, only the rules that have high confidence values are used for prediction. As a result, the number of rules used for prediction is reduced and their quality gets higher with the increasing $conf_{min}$. This leads to a higher

decrease rate in the number of predictions when compared to the decrease rate in the number of correct predictions. Therefore, the precision value improves as $conf_{min}$ increases.

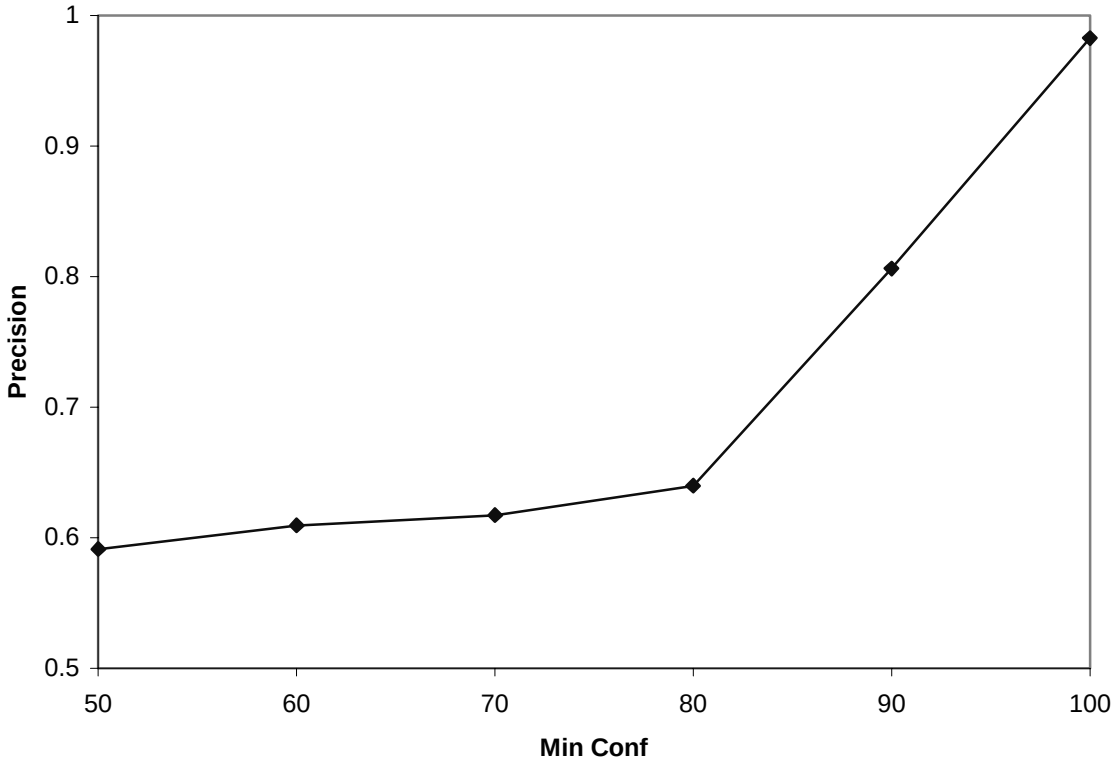


Figure 4.5: Precision as a function of the Minimum Confidence for UMP-Based Prediction algorithm

On the other hand, we observe the opposite effect on the recall as shown in Figure 4.6. As the $conf_{min}$ value increases, the number of mined rules is reduced. The decrease in the rules negatively affects the number of correct predictions. Therefore, the recall decreases as $conf_{min}$ increases.

Once again a trade-off between recall and precision is observed with increasing $conf_{min}$ values. This case is similar to the one observed with the experiment evaluating the impact of parameter *m*. Using a similar approach, a middle ground value of 70 has been chosen for $conf_{min}$.

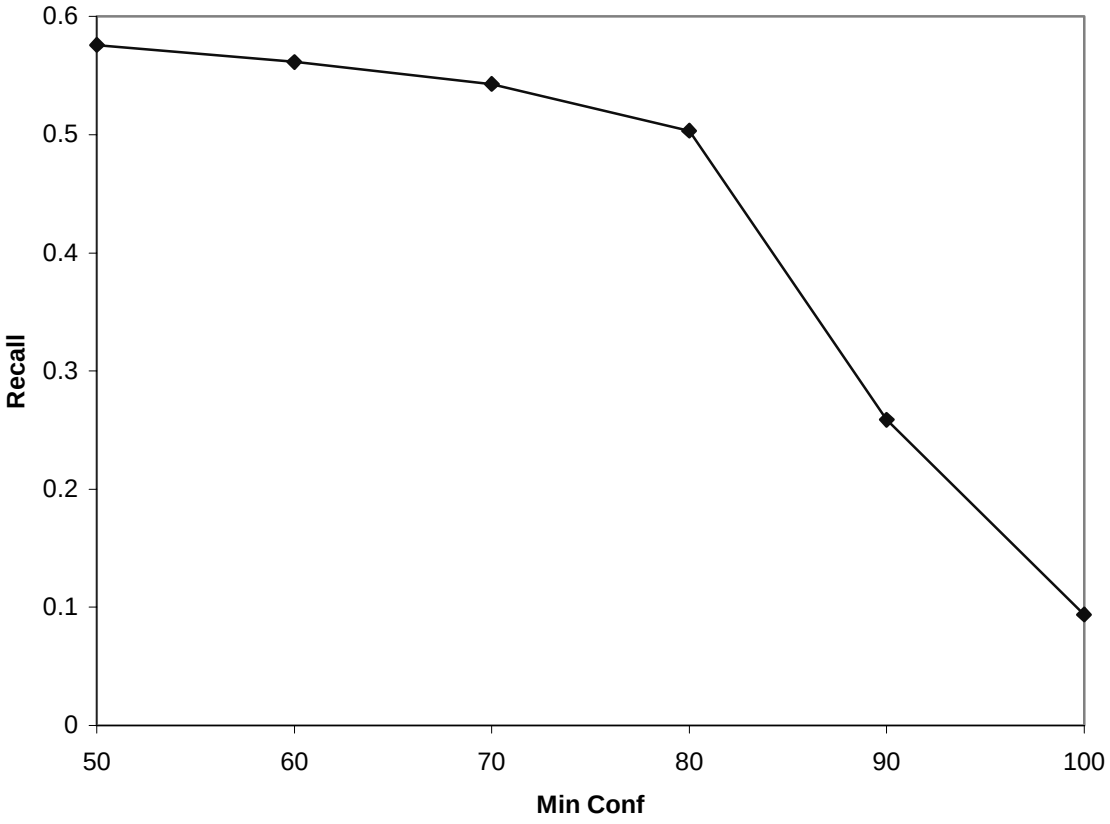


Figure 4.6: Recall as a function of the Minimum Confidence for UMP-Based Prediction algorithm

4.6 Impact of Corruption Factor

Next, we examine the effect of corruption on the precision and recall values. The impact of increasing corruption factor is illustrated in Figures 4.7 and 4.8. As one can observe in Figure 4.7, the precision value is very high for our method when the corruption is zero. However, this is not a realistic case because there is no possibility of absence of corruption. A realistic corruption value would be 0.4 which is the default value used in our experiments. As the corruption increases, the precision is reduced since the number of mobility rules that are determined by our algorithm decreases. But the precision values, which are never less than 0.45 can be considered good for such high corruption factors. TM is also affected by corruption but the decrease in precision for TM is less

significant when compared to that of our method. However, the precision obtained by our method is better than the precision obtained by its closest competitor, which is TM. This is true for all corruption values. Although the Ignorant Prediction method demonstrates a stable precision value, it presents the worst performance for precision. This indicates the ineffectiveness of the Ignorant Prediction method.

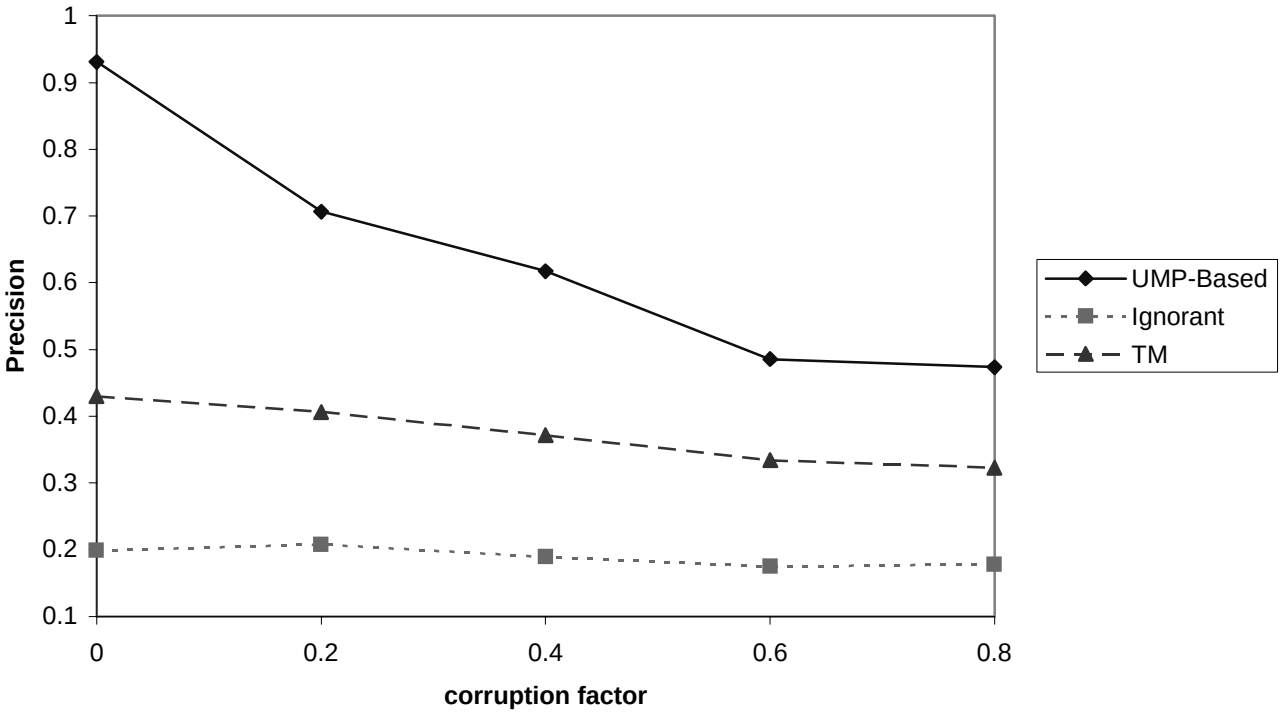


Figure 4.7: Precision as a function of the Corruption Factor

The recall value also drops for both our method and TM with the increasing corruption factor. For our method, we can explain this by the decreasing number of mined rules. As the corruption in the data increases, the UAPs will provide less support to large patterns. This leads to a decrease in the number of UMPs mined by our algorithm. As a result, the number of mobility rules which are determined by our algorithm decreases. There is another reason for the performance reduction of our method. As a result of the corruption, our prediction algorithm will match less or even will not match any mobility rules to the current trajectory of a mobile user. Therefore, no prediction can be accomplished in many cases when the corruption gets very high.

Increasing the corruption does not reduce the performance of Ignorant Prediction significantly. This is an expected result because Ignorant Prediction disregards the historical inter-cell movement of users and every prediction of this method is random. Therefore the corruption factor does not affect the performance of Ignorant Prediction.

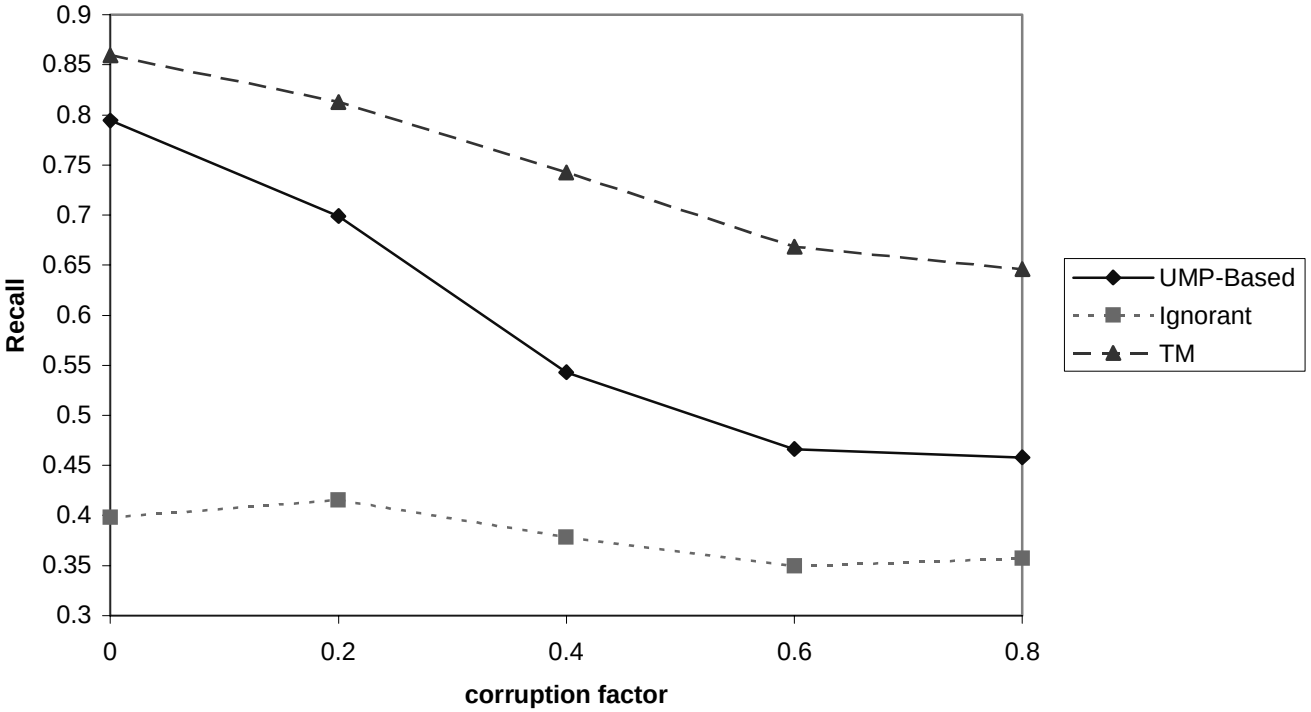


Figure 4.8: Recall as a function of the Corruption Factor

4.7 Impact of Outlier Percentage

In the last experiment, we examine the impact of outlier percentage in the data set. The results are presented in Figures 4.9 and 4.10. When we increase the outlier percentage, we observe a slight decrease in the recall. On the other hand, the precision of our method is not affected by the increasing outlier percentage. This can be explained by the fact that the rules which are mined from outlier UAPs are not used in predicting the next trajectory in most of the predictions made. Because, these rules are supported by the outliers and they are not common. When a user is following a UMP, these rules are not used for prediction. Therefore, the precision is not reduced.

The recall and precision values obtained by TM behave similarly when compared to the values obtained by our method. The recall of TM experiences a slight decrease but it is better than the recall of our method for all outlier percentages. However the precision of our method is always better than the precision of TM.

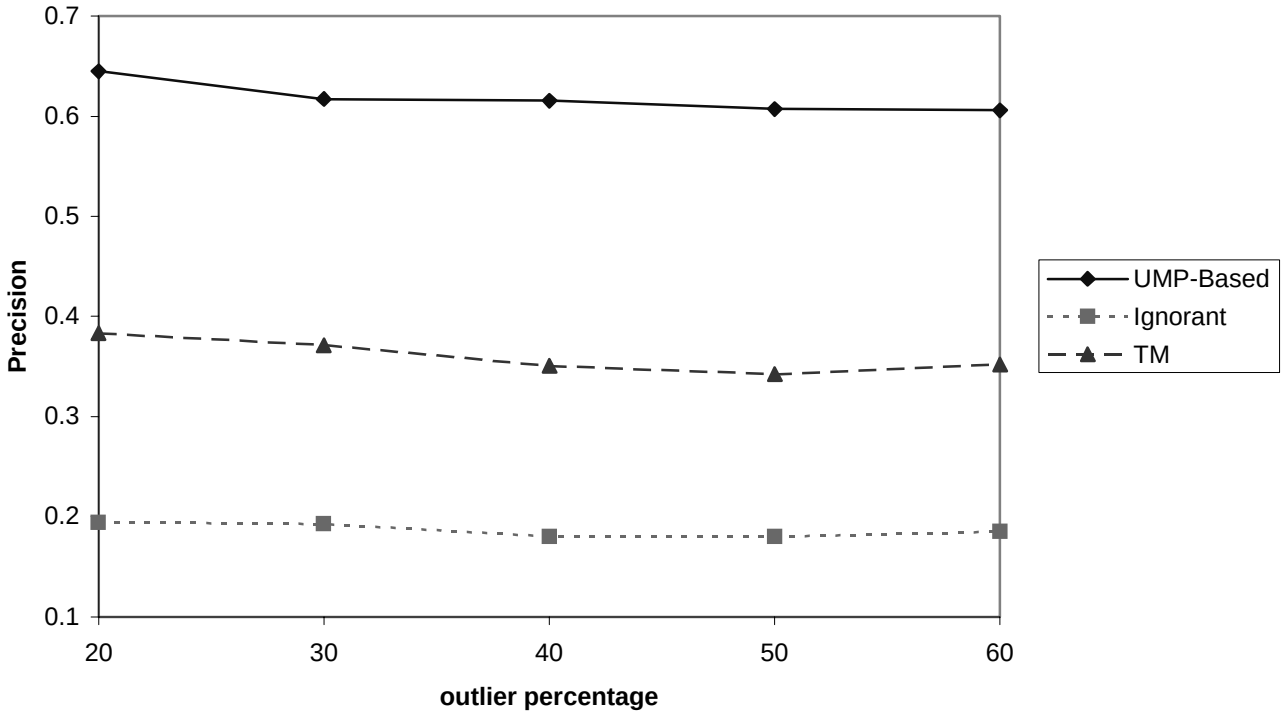


Figure 4.9: Precision as a function of the Outlier Percentage

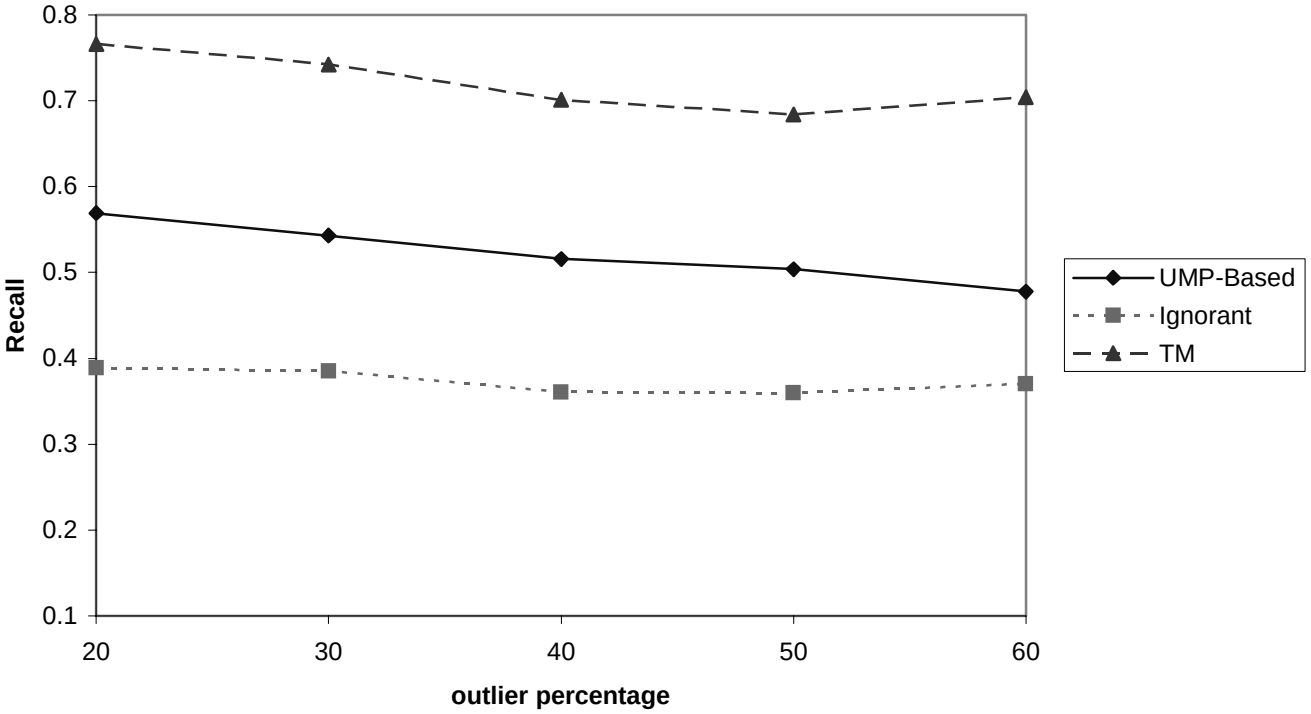


Figure 4.10: Recall as a function of the Outlier Percentage

Chapter 5

Conclusion and Future Work

In this thesis, we present a data mining algorithm for the prediction of user movements in a mobile computing system. The algorithm proposed is based on mining the mobility patterns of users, forming mobility rules from these patterns, and finally predicting a mobile user's next movements by using the mobility rules.

We have evaluated the performance of our algorithm using simulation and compared the obtained results with the performance of two other prediction methods. These methods are Mobility Prediction based on Transition Matrix (TM) and Ignorant Prediction. In TM, mobility prediction is based on the cell-to-cell transition probability matrix. The Ignorant method does not take any historical information into account when making prediction. In this method, randomly selected neighbors of the current cell are used as the predicted cells. This method can be considered as a baseline algorithm for comparison.

Our method has performed well with a variety of corruption factor and outlier percentage values. We have observed that although an increase in the corruption in the data decreases the recall and precision, an increase in the outlier percentage has no significant effect on the recall and precision. When compared to the performance of the baseline method, which is Ignorant Prediction, our method provides a very good performance in terms of precision and recall.

When we compare its performance with the performance of TM, it can be seen that the precision obtained with our method is better than that observed with TM. This result indicates that our method makes more accurate predictions. Most of its predictions made at each request are correct. On the other hand, the recall values obtained with TM are higher than those obtained with our method for most of the experiments. This is due to the nature of our method, which may not make prediction in response to some of the requests. The reason is that there may not be any matching rule for the current trajectory of the user when a prediction request is made. Thus, our method does not make any prediction in that case. On the other hand, TM makes prediction at most of the requests because it only keeps the transition probabilities of the cells. Therefore, even if there has been only one transition from a cell, say *A*, then it will use this information to make a prediction when the user is in cell *A*. It will have a higher potential to make predictions at every request, resulting in higher probability to make a correct prediction. Since the number of requests in the test set is the same for both methods and the number of correct predictions is higher for TM, TM produces higher recall values.

The algorithm presented in this thesis can be extended in the following directions.

- As we explain in Section 3.1, the support counting method used in our prediction algorithm is a generalization of the pattern mining approach presented in [2, 3]. For calculating the *totDist* value, our method takes the degree of corruption in the patterns into account. This is accomplished by decreasing the support given to a pattern by a UAP as the number of corrupted cells in the pattern increases. By giving small support to more corrupted patterns, this method improves the quality of mined patterns. Although our method for calculating this value is a good choice, some other methods can also be employed for this calculation and the results can be compared with our method.

- In our work, we do not consider the time domain of the mobility patterns and mobility rules. There are no time stamps associated with the collected mobile user trajectories. However, in real life, the mobility patterns of the users might be

closely related to the time. In some specific time period of the day, users might be following some specific movement paths. Therefore, different sets of rules might be associated with different time intervals. As a future work, it may be a good idea to extend our algorithm to include the time domain of mobility rules.

Bibliography

[1] A. Aljadhai and T. Znati. Predictive Mobility Support for QoS Provisioning in Mobile Wireless Environments. *IEEE Journal on Selected Areas in Communications*, 19(10):1915-1930, 2001.

[2] A. Nanopoulos, D. Katsaros, Y. Manolopoulos, Effective Prediction of Web User Accesses: A Data Mining Approach, In *Proceedings of the WebKDD Workshop (WebKDD'01)*, 2001.

[3] A. Nanopoulos, D. Katsaros, Y. Manolopoulos, A Data Mining Algorithm for Generalized Web Prefetching, *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 5, Sep./Oct. 2003.

[4] I. F. Akyildiz, S. M. Ho, and Y.-B. Lin. Movement-Based Location Update and Selective Paging for PCS Networks. *IEEE/ACM Trans. on Networking*, 4(4):629-639, Aug. 1996.

[5] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proceedings of the IEEE Conference on Data Engineering (ICDE'95)*, pages 3–14, 1995.

[6] B. Liang and Z. Haas. Predictive Distance-Based Mobility Management for PCS Networks. In *Proceedings of the IEEE Conference on Computer and Communications (IEEE INFOCOM'99)*, pages 1377-1384, 1999.

[7] G.Y. Liu and M.Q. Gerald. A Predictive Mobility Management Algorithm for Wireless Mobile Computing and Communications. In *Proceedings of the IEEE International Conference on Universal Personal Communications*, pages 268-272, 1995.

[8] T. Liu, P. Bahl, and I. Chlamtac. Mobility Modeling, Location Tracking, and Trajectory Prediction in Wireless ATM Networks. *IEEE Journal on Selected Areas in Communications*, 16(6):922-936, 1998.

[9] D. Katsaros, A. Nanopoulos, M. Karakaya, G. Yavas, O. Ulusoy, Y. Manolopoulos, Clustering Mobile Trajectories for Resource Allocation in Mobile Environments, *Intelligent Data Analysis Conference (IDA'2003), Lecture Notes in Computer Science (Springer Verlag)*, vol.2810, 2003.

[10] S. Rajagopal, R.B. Srinivasan, R.B. Narayan, and X.B.C. Petit. GPS-Based Predictive Resource Allocation in Cellural Networks. In *Proceedings of the IEEE International Conference on Networks (IEEE ICON'02)*, pages 229-234, 2002.

[11] H.-K. Wu, M.-H. Jin, J.-T. Horng, and C.-Y. Ke. Personal Paging Area Design Based on Mobile's Moving Behaviors. In *Proceedings of the IEEE Conference on Computer and Communications (IEEE INFOCOM'01)*, pages 21-30, 2001.

[12] D. Gusfield, Algorithms on Strings, Trees, and Sequences. *Cambridge University Press*, 1997.

[13] R. Agrawal, R. Srikant, Fast Algorithms for Mining Association Rules. In *Proceedings of Very Large Databases Conference (VLDB'94)*, pages 487-499, 1994.

[14] S. Mohan and R. Jain, Two User Location Strategies for Personal Communication Systems. *IEEE Personal Communications Magazine*, pages 42-50, First Quarter 1994.

[15] Amiya Bhattacharya and Sajal K. Das. LeZi-- Update: An Information-Theoretic Approach to Track Mobile Users in PCS Networks. *ACM Wireless Networks*, 8(2-3), pages 121-135, 2002.