Mining Temporal Moving Patterns in Object Tracking Sensor Networks

Vincent S. Tseng
Department of Computer Science
and Information Engineering
National Cheng Kung University
Tainan, Taiwan, R.O.C.
tsengsm@mail.ncku.edu.tw

Kawuu W. Lin
Department of Computer Science
and Information Engineering
National Cheng Kung University
Tainan, Taiwan, R.O.C.
linwc@idb.csie.ncku.edu.tw

Abstract **Advances in wireless communication and microelectronic devices technologies have enabled the development of low-power micro-sensors and the deployment of large scale sensor networks. With the capabilities of pervasive surveillance, sensor networks can be very useful in a lot of commercial and military applications for collecting and processing the environmental data. One of the very interesting research issues is the energy saving in object tracking sensor networks (OTSNs). However, most of the past studies focused only on the aspect of movement behavior analysis or location tracking and did not consider the temporal characteristics, which are very critical in OTSNs. In this paper, we propose a novel data mining method named TMP-Mine with a special data structure named TMP-Tree for discovering temporal moving patterns efficiently. To our best knowledge, this is the first study that explores the issue of discovering temporal moving patterns that contain both movement and time interval simultaneously. Through empirical evaluation on various simulation conditions, TMP-Mine is shown to deliver excellent performance in terms of accuracy, execution efficiency, and scalability.**

Index Terms— **Object tracking, temporal moving patterns, sensor data, sensor networks, data mining.**

1. Introduction

Advances in wireless communication and microelectronic devices technology have enabled the development of low-power micro-sensors and the deployment of large scale sensor networks. With the capabilities of pervasive surveillance, sensor networks can be very useful in a lot of commercial and military applications for collecting and processing the environmental data, in which each sensor node is composed of sensing, data processing, and communication components [15]. Sensor nodes form *ad hoc* networks [3, 8], and the nodes can communicate with each other by RF radios in the networks without any infrastructure. However, the intrinsic limitations such as power constraints, synchronization, deployment, and data routing bring numerous research challenges [19]. Comparing with standard ad hoc networks, sensor nodes are static and carry scarcer battery and less processing power. Replenishing the battery charge is expensive or infeasible due to the environmental conditions, so the energy is the most important system resource that should be reserved [4, 13]. One of the very interesting issues is the energy saving in object tracking sensor networks (OTSNs) [18].

The past studies for energy saving in sensor networks focused on optimizing the communication cost by inactivating the RF radios of idle sensor nodes [6, 7, 9, 11]. However, although the sensing and computing components consume relative less energy than radios [15], these studies did not consider the energy saving issues for these components [18]. Economizing on energy resource can extend the lifetime of OTSNs. One of the best ideas is to put a sensor node to sleep when there are no objects in its sensing region, and wake a sensor node up again when an object enters its sensing region. Based on this idea, the studies for energy saving in OTSNs can be further divided into two categories: non-prediction based tracking and prediction based tracking. The intuitive energy saving method of non-prediction based tracking is periodically turn the sensor nodes off and only wake up the sensor nodes when it is time to monitor their sensing regions. In [5], the authors proposed the Frisbee scheme that only a limited zone (namely a Frisbee) of the network that is close to the event is kept in its fully active state. However, it is difficult to choose a good radius of the Frisbee. In [18], the authors proposed the prediction-based energy saving schemes (PES) to conserve the scarce energy resource by exploiting the latest detected or average velocity of an object to predict the next node(s) that the object might visit. Because the sensor nodes might lose the object when it is in sleeping mode, the latest detected velocity can not be the absolutely correct and this directly causes the scarce energy suffering from the incorrect prediction. Note that both of the non-prediction based and prediction based tracking methods considered the moving behavior of objects as randomness. In real applications, the behavior of the moving objects is often based on certain underlying events instead of randomness completely. Central to this issue is the problem of discovering the moving behavior of objects.

Over the past few years a considerable number of studies have been made on using data mining techniques to discover this kind of interesting rules/patterns from WWW [14], transaction database [1,2], and mobility data [10,12,16,17]. Most of these past studies focused only on the aspect of movement behavior analysis or location tracking and did not consider the temporal characteristics that are very important in OTSNs. For example, for an object tracking application in museums, if we know from the network server log that most of the travelers arrived at the region of exhibit A at some time, several of the travelers will moved to exhibit B after 20 minutes while several of them stayed at exhibit A for 30 minutes and then moved to exhibit C. Later, if a new traveler stays at the region of exhibit A for 30 minutes and the sensor node loses the traveler, the node may have strong confidence to wake up the sensor node in the region of exhibit C instead of the region of exhibit B. By integrating this kind of patterns into the prediction schemes, the error of predictions can be substantially reduced.

To our best knowledge, however, no studies have explored the issue of discovering temporal moving patterns in OTSNs so as to consider both movement and time interval simultaneously. In this paper, we propose a novel data mining method named TMP-Mine with a special data structure named TMP-Tree for efficiently discovering temporal moving patterns in OTSNs . The object moving patterns discovered by TMP-Mine can be incorporated into the existed prediction schemes so as to reduce the error of predictions for energy savings. Through empirical evaluation on various simulation conditions, TMP-Mine is shown to deliver excellent performance in terms of accuracy, execution efficiency, and scalability.

The rest of this paper is structured as follows. In Section 2, we give the problem formulation. In Section 3, we describe the proposed method and the underlying data structures. The empirical evaluation for performance study is made in Section 4. The conclusions and future directions are given in Section 5.

2. Problem Formulation

Let S=< (l_1, rt_1) (l_2, rt_2) ... (l_m, rt_m) > be a temporal moving sequence of an object with length equal to m, namely m-sequence, where l_i represents the object's location at time rt_i and $rt_i < rt_{i+1} \ \forall 1 \le i < m$. Notice that the value of real time rt is unique in a sequence, i.e., rt_i will never be equal to rt_j $\forall i \ne j$. The ascending order of elements of sequence is sorted by using the real time as the key. In order to discover the *temporal* moving patterns, we use time slot to uniformly segment the time dimension of sequence for its real number characteristic. If the time slot is set to b, we will obtain a transformed sequence S_t=< (l_1, t_1) (l_2, t_2) ... (l_m, t_m) >, where $t_i = \frac{rt_i}{b}$.

Definition 1. A temporal moving pattern P'=<(l_1', t_1') (l_2', t_2') ... (l_m', t_m')> is a *sub-pattern* of another access pattern P=<(l_1, t_1) (l_2, t_2) ... (l_n, t_n)>, written as $P' \subset P$, if $m \le n$ and there exists a strictly increasing sequence ($i_1, i_2, \ldots, i_m$) of indices such that for all $j=1, 2, \ldots, m$, $l_j' = l_{i_j}$ and $t_{j+1}' - t_j' = t_{i_{j+1}} - t_{i_j}$. Here, P is called the *super-pattern* of P'.

Definition 2. Given a database $D = \{ S_1, S_2, \ldots, S_N\}$ that contains N temporal moving sequence, the *support* of pattern P is defined as

$$\sup(P) = \frac{|\{S_i \mid P \subset S_i \text{ and } 1 \le i \le N\}|}{N}.$$

Definition 3. P is called a *frequent temporal moving pattern* (*F-TMP*) if sup(P) is greater than a specified support threshold δ.

With the above definitions, the problem of discovery of temporal moving patterns is defined as follows. Given a database D containing the logs of object's moving sequence and a specified support threshold, the problem is to discover all the F-TMPs existed in this database.

3. Proposed Method

3.1 TMP-Tree Construction

For the purpose of discovering the F_TMPs efficiently, it is required to construct a TMP-Tree in advance. The purpose of constructing TMP-Tree is to aggregate the temporal moving patterns into the memory in a compact form so that the mining of frequent patterns can be done efficiently. The main advantages of TMP-Tree are 1) only one physical database scan is needed to mine all of the large patterns, and 2) the TMP-Tree is compact so that the huge amount of data can be handled efficiently.

The node of TMP-Tree is termed as location node (LNode) for it semantically represents the location the object traversed. Each LNode of TMP-Tree has the following structure:

LNode := { l, c, parent-link, next-link, children table, ITree }

The label of LNode is stored in the variable l, and the times the LNode traversed is stored in variable c. The parent-link and next-link are two pointers linking to the parent LNode and next LNode with the same label, respectively. All of the children LNodes of the node are tabulated in the children table. Each LNode is attached an interval tree, namely ITree, for discovering the temporal patterns.

Figure 1 shows the TMP-Tree construction function. The input of this function is the object moving log, and the function returns the TMP-Tree after inserting every tuple in log into the TMP-Tree. At the beginning, the TMP-Tree T is initialized (line 1). The function then retrieves the tuples from the log one by one (line 2), and extracts the location path, namely LPath, and interval path, namely IPath, from each tuple (line 3 and line 4). For example, given an object moving path {(L_a, 2)(L_b, 5)(L_c, 12)}, the LPath is {(L_a), (L_b), (L_c)} while the IPath is {(0), (5-2), (12-2)}={(0), (3), (10)}. The details of ExtractLPath() and ExtractIPath() are shown in Figure 2 and Figure 3, respectively. Afterwards, the extracted LPath is inserted into TMP-Tree by invoking InsertLPath() and the function returns the last location node of T that corresponds to the last llabel of

```
function TMP_Tree_Construct ( $D$ )
1.    $T \leftarrow \Phi$ , $D' \leftarrow$ Database_Filtering( D )
2.    foreach tuple $\tau \in D'$
3.        lp $\leftarrow$ ExtractLPath($\tau$)
4.        ip $\leftarrow$ ExtractIPath($\tau$)
5.        lnode $\leftarrow$ InsertLPath( T, lp , 1)
6.        InsertIPath( lnode, ip , 1)
7.    end foreach
8.    return T;
```

Figure 1. Construction function for TMP-Tree.

```
function ExtractLPath ( $\tau$ )
1.    lp $\leftarrow \Phi$
2.    for i = 1 to length( $\tau$ )
3.        lp [ i ] $\leftarrow$ getLPart( i , $\tau$ )
4.    end for
5.    return lp;
```

Figure 2. LPath extraction function.

```
function ExtractIPath ( $\tau$ )
1.    ip $\leftarrow \Phi$
2.    shiftedInterval = getIPart( 1 , $\tau$ )
3.    ip[1] = 0
4.    for i = 2 to length( $\tau$ )
5.        ip [ i ] $\leftarrow$ getIPart( i , $\tau$ ) – shiftedInterval
6.    end for
7.    return ip;
```

Figure 3. IPath extraction function.

LPath (line 5). The function then invokes InsertIPath() to insert IPath into the interval tree (ITree) on that returned LNode (line 6).

An LLabel table is maintained together with a TMP-Tree to record the appearing times and the last appearing node with this label of each label. The logic structure of each tuple of LLabel table is represented as following:

LLabel := { l, c, last appearing node}

Figure 4 shows the function that inserts an LPath into a TMP-Tree with specified count. The function first fetches the root node of T storing in a temporary node, namely lnode, and LLabel table namely lt (line 1) and discretizes the input LPath into an array. The function then inserts each label into TMP-Tree *in order* (line 2). For a label l, if lnode has a child with label = l (line 3) by checking children table of lnode, the function will look up the children table and assign the entry with label = l to lnode (line 4), and increase the count of lnode by the specified count (line 5). If lnode has no child with label = l, the function will create a new node with label l (line 7) and set the count of lnode to the specified count (line 8). Because lnode is a new node on the path, several structure modifications should be made. We retrieve the last appearing pointer that points to the last LNode with label = l from LLabel table lt (line

```
function InsertLPath ( T , lp , c)
1.    lnode $\leftarrow$ root ( T ), lt $\leftarrow$ getLLabelTable(T)
2.    for i = 1 to length( lp )
3.        if ( getChildren( lnode , lp [ i ] )) $\neq \Phi$ )
4.            lnode $\leftarrow$ getChildren( lnode , lp [ i ] ))
5.            increase the count of lnode by c
6.        else
7.            lnode $\leftarrow$ InsertLChild( lnode , lp [ i ])
8.            set the count of lnode to c
9.            tmpnode $\leftarrow$ getLNode( lp[i] , lt )
10.           setNextLink( lnode , tmpnode)
11.           set the last appearing link to lnode
12.       end if
13.       increase the count of lp [ i ] in LLabel
          table lt by c
14.   end for
15.   return lnode;
```

Figure 4. LPath insertion function.

```
procedure InsertIPath ( lnode , ip , c)
1.    inode $\leftarrow$ ITree_root ( lnode ),
      it $\leftarrow$ getILabelTable(lnode )
2.    for i = 1 to length( ip )
3.        if ( getChildren( inode , ip [ i ] )) $\neq \Phi$ )
4.            inode $\leftarrow$ getChildren( inode , ip [ i ] ))
5.            increase the count of inode by c
6.        else
7.            inode $\leftarrow$ InsertIChild( inode , ip [ i ])
8.            set the count of INode to c
9.            tmpnode $\leftarrow$ getINode( i , it )
10.           setPeerLink( inode , tmpnode )
11.           set the level - link to inode
12.       end if
13.       increase the count of ip [ i ] in ILabel
          table it by c
14.   end for
```

Figure 5. IPath insertion function.

9), namely tmpnode, and set lnode's next-link to tmpnode (line 10). By setting the last appearing node for current label in LLabel table to lnode (line 11), all nodes with same label will form a link list, and hence we can keep track of all nodes with a specified label via the LLabel table. In the final step, the count (appearing times) of current label in LLabel table is increased (line 13), and the function returns the lnode (line 15).

Figure 5 shows the procedure that inserts an IPath into the ITree, which is attached on an LNode. The node of ITree is termed as interval node (INode) for it semantically represents the time interval between the physical traversed locations. Each INode of ITree has the following structure:

INode := { l, c, parent-link, peer-link, children table }

The time label of INode is stored in the variable l, and the times the INode traversed is stored in variable c. The parent-link points to the parent INode. The peer-link points to the next INode on the same level. All of the children INodes of the node are tabulated in the children table as LNode structure.

The insertion procedure is some similar to InsertLPath(). Notice that the major difference is that this procedure uses *level-link* structure for each interval node (INode) instead of next-link (line 9). By keeping track of the level-link, it is easy to sum the counts of labels on certain level of ITree. Figure 8 shows the TMP-Tree constructed by using the log in Table 1.

3.2 TMP-Mine

Figure 6 shows the detailed algorithm for TMP-Mine, which is based on the depth-first search (DFS) approach extending from [14]. However, the method proposed in [14] focused on the mining of single dimensional sequential patterns. We here devise a new algorithm to manipulate the two dimensional moving patterns including location and time attributes simultaneously. It recursively constructs the TMP-Trees and mines the trees till termination condition is met. First, we list all the labels with count greater than the support threshold δ by scanning the LLabel table of current TMP-Tree, and store the labels into a temporary set (line 1), namely FreqL. If FreqL is empty (line 2), the prefix pattern of current TMP-Tree is output as one of the F-TMPs and the procedure returns (line 3). Otherwise, for each label llabel in L_L1 (line 5), we fetch all the nodes with label equal to llabel from current TMP-Tree into a temporary set (line 6), namely LNode_tmp. Based on the prefix llabel, we also accumulate the count of each ancestor label by extracting the ancestor lpaths of lnodes in LNode_tmp (line 7). Function getFrequentLIPair() returns the set of frequent location-interval pair (LIPair) where every LIPair in this set is with count greater than the support threshold δ (line 8). If FreqLIPair is empty (line 9), the prefix pattern is output as

```
procedure TMP_Mine ( T , $\delta$ , Ptn)
1.   FreqL ← getFrequentLabel( getLLabelTable ( T ), $\delta$ )
2.   if (FreqL == Φ )
3.       output prefix pattern Ptn and return
4.   end if
5.   foreach llabel ∈ FreqL
6.       LNode_tmp ← getNodesByLabel( llabel )
7.       FreqAncestorLabels ← getFrequentAncestorLabels \
                              ( LNode_tmp )
8.       FreqLIPair ← getFrequentLIPair( LNode_tmp , \
                      FreqAncestorLabels )
9.       if ( FreqLIPair == Φ )
10.          output prefix pattern Ptn and return
11.      endif
12.      foreach lipair ∈ FreqLIPair
13.          T' = Reconstruct_TMP_Tree \
                 ( T , Ptn , lipair , LNode_tmp)
14.          TMP_Mine ( T' , $\delta$ , Ptn + llabel + lipair)
15.      end foreach
16.  end foreach
```

Figure 6. The algorithm of TMP-Mine.

one of the F-TMPs and the procedure ends (line 10). Otherwise, we reconstruct TMP-Trees for each lipair in FreqLIPair (line 13), and the mining procedure is invoked recursively (line 14) to discover all the F-TMPs.

3.3 TMP-Tree Reconstruction

In section 3.2, we detail the TMP-Mine algorithm. The line 13 in Figure 6 invokes Reconstruct_TMP_Tree() to reconstruct TMP-Tree according to the given prefix pattern. Figure 7 is the TMP-Tree reconstruction function. This reconstruction function begins with initializing a TMP-Tree T' and extracting the interval part from lipair (line 1). For each lnode in LNode_tmp (line 2), we get its cross-peer nodes by iprefix (line 3). All of the cross-peer nodes with label = iprefix will be

```
function Reconstruct_TMP_Tree ( T , Ptn , cur_ilabel, LNode_tmp)
1.   T' ← Φ
2.   foreach lnode ∈ LNode_tmp
3.       INode_tmp ← getCrossPeerNodesByLabel( lnode , iprefix)
4.       c ← sum the count
5.       endnode ← InsertLPath( T' , getLPath(lnode) , c)
6.       foreach inode ∈ INode_tmp
7.           InsertIPath( endnode, getIPath( inode ), getCount(inode) )
8.       end foreach
9.   end foreach
10.  return T'
```

Figure 7. TMP-Tree reconstruction function.

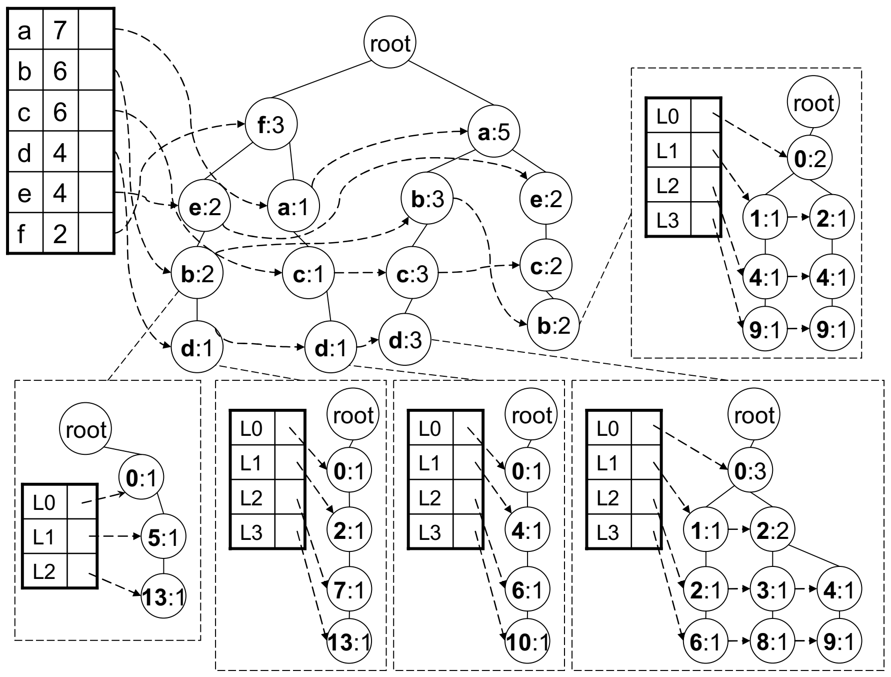

Figure 8. The TMP-Tree constructed from Table 1.

Table 1. An integrated log of temporal moving sequences.

Object ID	Temporal Moving Sequence
1	<(a,1)(e,3)(c,5)(b,10)>
2	<(a,3)(b,5)(c,7)(d,12)>
3	<(a,1)(e,2)(c,5)(b,10)>
4	<(f,0)(e,5)(b,13)>
5	<(a,4)(b,6)(c,7)(d,12)>
6	<(f,0)(a,4)(c,6)(d,10)>
7	<(a,0)(b,1)(c,2)(d,6)>
8	<(f,1)(e,3)(b,8)(d,14)>

returned, and the sum of count of every cross-peer node will be assigned to c (line 4). Then the function InsertLPath() is invoked and it returns the last node, endnode, corresponding to the parent path (line 5). For each inode in INode_tmp, we retrieve the parent path and count of inode, and invoke InsertIPath() to insert it into T' (line 6-8). After all of the lnodes in LNode_tmp are processed, the function returns the TMP-Tree T' (line 10).

An Elaborate Example

We here demonstrate an example to show the way that the F-TMPs are discovered. Figure 8 is the TMP-Tree constructed from the log in Table 1. As shown in Figure 8, each LNode is represented as the form "L:C", where L is the LLabel and C is the count. For illustration, every ITree is surrounded by a dotted rectangle. The representation of an INode is the same as an LNode, denoted as "I:C", where the I is the ILable and C is the count associated with this ILabel.

Take the first tuple, <(a,1)(e,3)(c,5)(b,10)>, in Table 1 for insertion example. The LPath and IPath extracted from this tuple are <(a)(e)(c)(b)> and <(1-1)(3-1)(5-1)(10-1)>, i.e., <(0)(2)(4)(9)>, respectively. Because the TMP-Tree is initialized as an empty one, four LNodes for this LPath are created in the current TMP-Tree. Thesbe four llabels are also tabulated in the LLable table. Besides, a new ITree is constructed with the IPath <(0)(1)(4)(9)> on the last lnode that corresponds to this LPath, i.e., the LNode with llabel = b. As stated in section 3.1, the level-links of the entries in ILabel table should be set to the corresponding INodes. When inserting the second tuple, <(a,3)(b,5)(c,7)(d,12)>, from Table 1 into the TMP-Tree, the first llabel of the LPath, <(a)(b)(c)(d)>, is a, which is already the child of the root, and the count of the LNode is increased by 1 instead of creating new LNode for this llabel. On inserting the next llabel b into the TMP-Tree, a new LNode will be created for this current LNode has only one child labeled as e, not b. Notice that llabel b already have appeared in the LLabel table. It is required to break the link of last appearing node of entry b in LLabel table, and set it to the newly created LNode. At the same time, the next-link of the newly created LNode is set to the original LNode, which is pointed by the last appearing node of entry b. By maintaining this linking list, we can keep track of all the lnodes with llabel = b. The same procedure will be performed on the remaining two llabels, c and d. At the last step, the IPath, <(0)(2)(4)(9)>, is inserted into the ITree of the last LNode corresponding to llabel d. The TMP-Tree in Figure 8 is constructed by inserting the tuples in Table 1 in sequence.

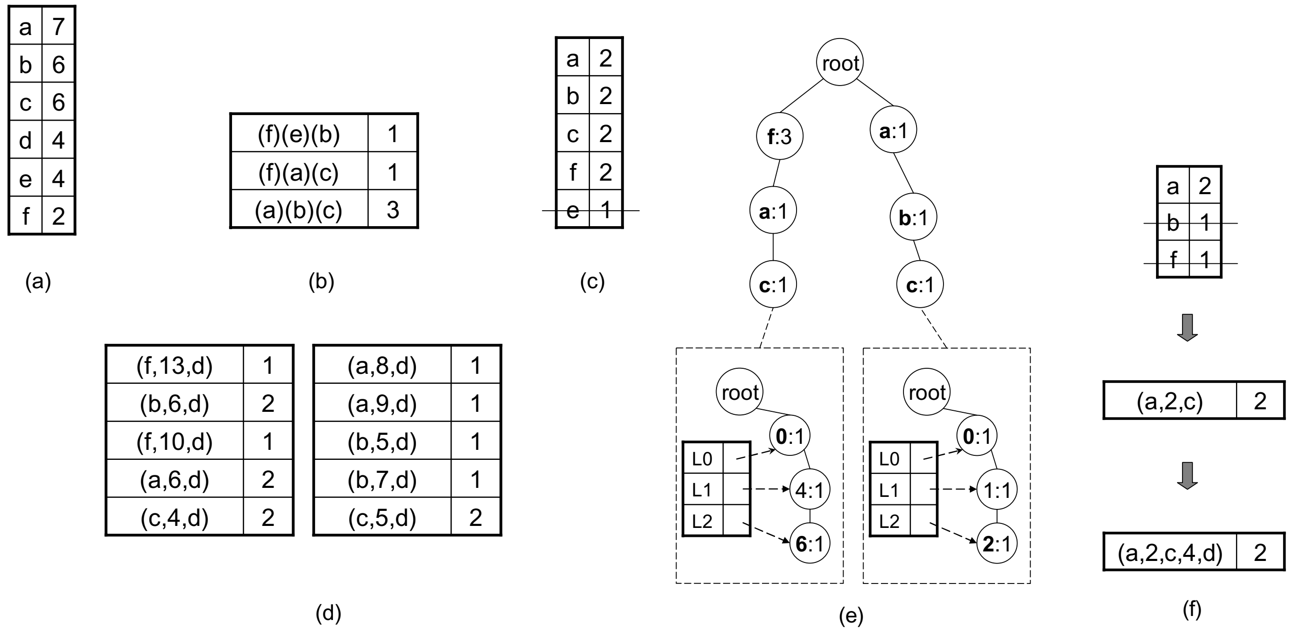

Figure 9. Part of the F-TMPs discovering example.

Table 2. The discovered F-TMPs from Table 1.

F-TMPs	F-TMPs
(a,2,b)	(b,1,c)
(a,2,c,4,d)	(b,6,d)
(a,2,c)	(c,4,d)
(a,4,c)	(c,5,b)
(a,4,c,5,b)	(c,5,d)
(a,6,d)	(e,8,b)

Once the TMP-Tree is prepared, we are able to discover the F-TMPs. Figure 9 illustrates part of the mining process. In the beginning of the mining process, the llabels that are greater than the specified support will be fetched after scanning the LLabel table. The example is demonstrated with the support threshold set to 20%. It is to say that the pattern with support greater than or equal to 2 (8*20%=1.6) can be one of the F-TMPs. In Figure 9-a, the count of each llabel is greater than 2, and therefore all of the llabels should be considered in the following process. If we take the llabel d as pattern base, as shown in Figure 9-b, three LPaths, <(f)(e)(b)>:1, <(f)(a)(c)>:1 and <(a)(b)(c)>:3, are obtained, where the last number denotes the count of the LPath that ends with llabel d. The count of distinct llabels is accumulated as shown in Figure 9-c. In Figure 9-c, the llabel e is pruned due to the count of llabel e is less than the support threshold under the current pattern base, <(d)>.

The left four llabels are called frequent llabels. For each frequent llabel, called $llabel_i$, we fetch all the LNodes with llabel = $llabel_i$ by tracking the next-link and last appearing node pointers. Each LNode forms several distinct intervals between it and base LNode by subtracting the ILabel value of LNode from the ILabel value of base LNode. Take the path <(f)(e)(b)> for example. The ILabel of b is 8 while the ILabel of the base LNode, d, is 14. The interval between b and d is 6 obtained by subtracting 8 from 14, and it is denoted as (b, 6, d), which is called an LIPair. The accumulating result of the LIPairs is shown in Figure 9-d. The LIPairs with count less than 2 are pruned.

The algorithm advances by choosing an arbitrary LIPair. If the chosen LIPair is (c, 4, d), we obtain a reconstructed TMP-Tree as in Figure 9-e, in which the two LPaths both end with (c, 4, d) in Figure 9-e. Simultaneously, the algorithm constructs two ITrees. Under the prefix pattern (c, 4, d), only llabel a is frequent, which is as shown in Figure 9-f. Again, the algorithm accumulates the LIPairs, and obtains the only one LIPair, i.e., (a, 2, c). Under the pattern base for the *reconstructed TMP-Tree*, no llabel has count greater than 2. Therefore, (a, 2, c) and (c, 4, d) are concatenated to form an F-TMP, and the sub-procedure returns.

4. Experimental Results

We conduct three experiments to evaluate the performance of TMP-Mine under different system conditions by varying the parameters in terms of number of objects, support threshold, and segmentation unit, respectively. Meanwhile, the effects of varying these system parameters were also studied. All of the experiments were conducted on a P4-2.4GHz machine with 1G MB main memory. The algorithm and simulation are implemented in Java with JDK 1.5.0 virtual machine running on Fedora Core 2.

4.1 Simulation Model

To evaluate the performance of SMAP-Mine, we used a simulator that simulates an OTSN to generate the workload data. In the base experimental model, the network is modeled as a mesh network with size |W| = 20*20, and there are 100,000 objects in this network. Initially, each object arrives at the network on an arbitrary detecting sensor node at some time. We choose exponential distribution with parameter Ι (defaulted as 0.25) to decide the stay interval. Each object in the mesh network may move back by the parameter P_b (defaulted as 0.1) or randomly move to other nodes by parameter P_n (defaulted as 0.2). The average length of moving sequence is modeled by parameter L (defaulted as 10). We believe that in the OTSN, the behavior of the moving object is based on certain events instead of randomness completely. For this reason, we use two parameters l_e and P_e to model the average length and the probability of the events, respectively. The length of each event is modeled by a Poisson distribution and the probability of events is modeled as a normal distribution. The parameters l_e and P_e are defaulted as 4 and 0.05, respectively.

4.2 Impact of the number of objects

This experiment analyzes the required execution time and number of discovered patterns when varying the number of objects. The execution time is composed of loading time and mining time. The loading time includes the database scanning time and the time for initial TMP-Tree construction. The mining includes all of the time that the recursive reconstruction needs. In Figure 10, we see that with the number of objects increasing, the required time increases almost linearly even under the low support threshold (defaulted as 0.5%), demonstrating the scalability of TMP-Mine.

Figure 11 shows the amounts of discovered patterns under different object numbers. As indicated, the relative variation is small but there exists an increasing trend. By having a check on the datasets, we find that some patterns with support slightly smaller than the specified threshold turn to be frequent when the number of objects increases.

4.3 Impact of the support threshold

This experiment analyzes the required execution time and number of discovered patterns when varying the support threshold from 0.1% to 0.5%. In the experiment, as shown in Figure 12, the loading time is approximately the same because that the five support thresholds on the identical dataset make all of the nodes in OTSN be frequent and hence generate almost the same initial TMP-Tree in size.

The larger support threshold produces the less number of frequent LLabels or ILabels, which directly decreases the required mining time. For this reason, as shown in Figure 13, the mining time decreases with the support threshold increased.

4.4 Impact of the segmentation unit

This experiment analyzes the required execution time and number of discovered patterns when varying the segmentation unit. If the segmentation unit is too large, we may lose the information of time interval. On the contrary, if the segmentation unit is too small, it can not discretize the time dimension effectively. Take the sequence S=< (a,0)(b,3)(c,5)> for example. If the segmentation unit is set to 10, we obtain the

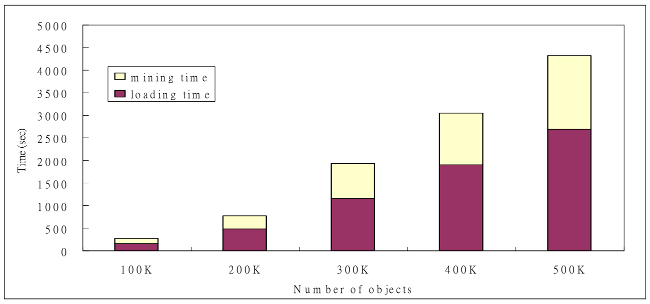

Figure 10. Execution time with number of objects varied.

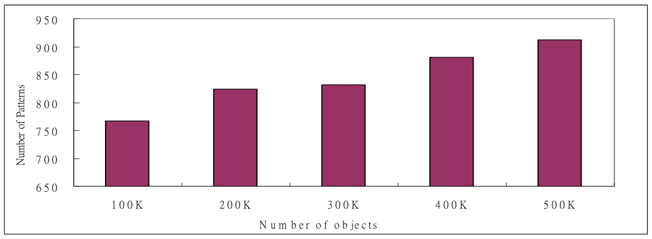

Figure 11. Number of patterns with number of objects varied.

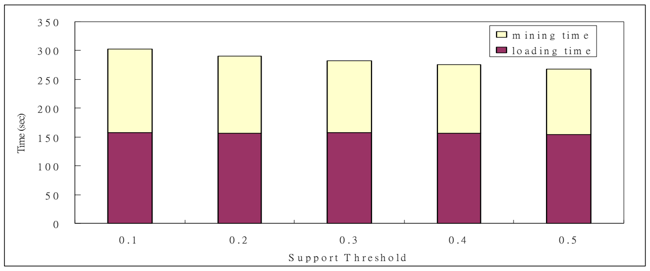

Figure 12. Execution time with support threshold varied.

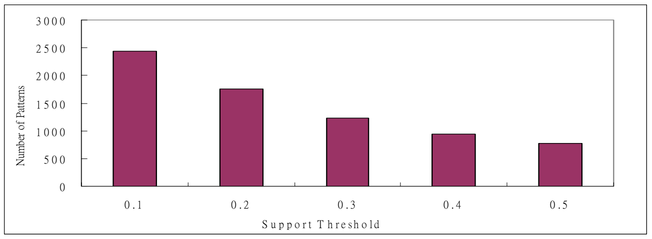

Figure 13. Number of patterns with support threshold varied.

transformed sequence S' =< (a,0)(b,0)(c,0)>, in which we lose the interval information, (a, 3, b) and (b, 2, 5). Notice that the average staying time is defaulted as 1.0 in the base model. Consequently, the number of generated sub-patterns is almost the same while the segmentation unit is less than 1.0. Moreover, if the segmentation unit is greater than 1.0, each sub-pattern will be supported by more sequences, and hence the number of patterns increases under the fixed support threshold, as shown in Figure 15. Figure 14 is the measurement of required execution time. The mining time is decided by the number of TMP-Tree reconstructions. To vary the segmentation unit will not change the required number of reconstructions directly or indirectly, and therefore the mining time is almost the same.

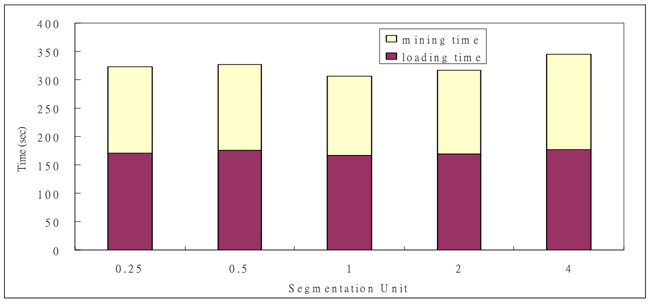

Figure 14. Execution time with segmentation unit varied.

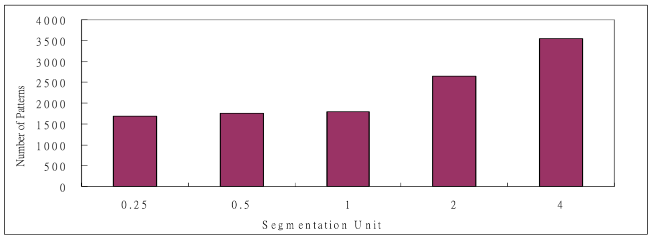

Figure 15. Number of patterns with segmentation unit varied.

5. Conclusions and Future work

In this paper, we have proposed a novel data mining method named TMP-Mine with a special data structure named TMP-Tree for discovering temporal moving patterns efficiently. To our best knowledge, no studies have explored the issue of discovering temporal moving patterns that contain both movement and time interval simultaneously. Through empirical evaluation on various simulation conditions, TMP-Mine is shown to deliver excellent performance in terms of accuracy, execution efficiency, and scalability.

For the future work, we will apply the discovered F-TMPs to the field of energy saving in OTSNs, and evaluate the performance under real OTSNs. Besides, since the discovered F-TMPs can be exploited in wide applications, we will integrate the TMP-Mine method with several applications like data allocation, data replication and location-based services, with the aim to enhance the quality of new applications in sensor networks.

Acknowledgement

This research was partially supported by National Science Council, Taiwan, R.O.C., under grant number NSC 93-2213-E006-030.

References

[1] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," Proceedings of 20th International Conference on Very Large Data Bases, Santiago de Chile, 1994, pp. 487-499.

[2] R. Agrawal, R. Srikant, "Mining Sequential Patterns," Proceedings of the 11th International Conference on Data Engineering, Taipei, Taiwan, 1995, pp. 3-14.

[3] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, "Wireless sensor networks: a survey," Computer Networks 38(4), 2002, pp. 393-422.

[4] J. Carle, D. Simplot, "Energy-Efficient Area Monitoring for Sensor Networks," IEEE Computer 37(2), 2004, pp. 40-46.

[5] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, J. Zhao, "Habitat Monitoring: Application Driver for Wireless Communications Technology," Proceedings of the First ACM SIGMOMM Workshop on Data Communications in Latin America and the Caribbean, 2001.

[6] D. Estrin, R. Govindan, J.S. Heidemann, S. Kumar: Next Century Challenges, "Scalable Coordination in Sensor Networks," Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking, 1999, pp. 263-270.

[7] S. Goel and T. Imielinski, "Prediction-based monitoring in sensor networks: taking lessons from MPEG," ACM ComputerCommunication Review, 31(5), October 2001.

[8] T. Hara, N. Murakami, S. Nishio, "Replica allocation for correlated data items in ad hoc sensor networks," SIGMOD Record 33(1), 2004, pp. 38-43.

[9] W.R. Heinzelman, A. Chandrakasan, H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," Proceedings of the 33rd Hawaii International Conference on System Sciences, 2000.

[10] J.L. Huang, M.S. Chen, W.C. Peng, "Exploring group mobility for replica data allocation in a mobile environment," Proceedings of the 2003 ACM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, 2003, pp. 161-168.

[11] Y.B. Ko, N.H. Vaidya, "Location-Aided Routing (LAR) in mobile ad hoc networks," Wireless Networks 6(4), 2000, pp. 307-321.

[12] M. Kyriakakos, S. Hadjiefthymiades, N. Frangiadakis, L.F. Merakos, "Multi-user Driven Path Prediction Algorithm for Mobile Computing," Proceedings of 14th International Workshop on Database and Expert Systems Applications, Prague, Czech Republic, 2003, pp. 191-195.

[13] S. Pattem, S. Poduri, B. Krishnamachari, "Energy-Quality Tradeoffs for Target Tracking in Wireless Sensor Networks," Proceedings of the Second Information Processing in Sensor Networks, 2003, pp. 32-46.

[14] J. Pei, J. Han, B. Mortazavi-Asl, H. Zhu, "Mining Access Patterns Efficiently from Web Logs," Proceedings of Fourth Pacific Asia Conference on Knowledge Discovery and Data Mining, Kyoto, Japan, April 2000, pp. 396-407.

[15] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, "Energy aware wireless microsensor networks," IEEE Signal Processing Magazine, 19(2), 2002, pp. 40-50.

[16] S.M. Tseng, W.C. Lin, "Mining Sequential Mobile Access Patterns Efficiently in Mobile Web Systems," Proceedings of International Workshop on Mobile Computing (held with ICS), Taiwan, 2004.

[17] S.M. Tseng, C.F. Tsui, "Mining Multi-Level and Location-Aware Associated Service Patterns in Mobile Environments," IEEE Transactions on Systems, Man and Cybernetics: Part B, Vol. 34, No. 6, 2004.

[18] Y. Xu, J. Winter, W.C. Lee, "Prediction-Based Strategies for Energy Saving in Object Tracking Sensor Networks," Proceedings of the Fifth IEEE International Conference on Mobile Data Management, 2004, pp. 346-357.

[19] H. Yang, B. Sikdar, "A protocol for Tracking Mobile Targets using Sensor Networks," Proceedings of IEEE Workshop on Sensor Networks Protocols and Applications, 2003.