A Dynamic Mobility Histogram Construction Method Based on Markov Chains

Yoshiharu Ishikawa
Nagoya University
ishikawa@itc.nagoya-u.ac.jp

Yoji Machida*
University of Tsukuba
y-machida@kde.cs.tsukuba.ac.jp

Hiroyuki Kitagawa
University of Tsukuba
kitagawa@cs.tsukuba.ac.jp

Abstract

With the recent progress of spatial information technologies and communication technologies, it has become easier to track positions of a large number of moving objects in real-time. Mobility statistics *plays an important role in the interactive analysis of a large collection of moving objects trajectories and its use of movement pattern prediction. The development of an effective mobility statistics measure and its efficient computation method are critical issues. Thus, we propose an approach for constructing a* mobility histogram *to summarize a number of moving object trajectories. The histogram is based on a mobility statistics model called* the Markov chain model. *To facilitate an interactive analysis performed by a user, we provide a mobility histogram data cube-like logical representation and support an OLAP-style analysis. Since trajectory data is often received continuously as a* trajectory stream, *we have to support dynamic histogram construction and maintenance. We introduce a tree structure as the physical representation of a histogram and present histogram construction and maintenance methods that work efficiently within the given upper-bound size. We evaluate the performance and the precision of the proposed method by means of experiments.*

1. Introduction

Owing to the development of mobile computing technologies, GPS and positioning devices, and wireless network facilities, tracking the positions of moving objects [10] has become relatively easier today. The collected trajectory data can be used to monitor the behavior of moving objects and to analyze their movement patterns. The latter activity is often called *mobility analysis* [27]. By analyzing movement patterns, we can observe an overall movement tendency and predict future movement patterns.

Analysis of movement patterns are also important for query processing in spatio-temporal databases that store trajectories of moving objects [10, 15, 16]. One of the topics relevant to our study is *selectivity estimation* for spatio-temporal databases. There exist some approaches for estimating selectivities of queries on spatio-temporal databases [6, 7, 25].

* Current affiliation: Hitachi Information Systems, Ltd.

In this paper, we focus on the summarization of a large number of moving object trajectories for the efficient calculation of *mobility statistics* [27], which in turn is based on the *Markov chain model*. In context to spatio-temporal data analysis, the Markov chain model is used to describe a spatio-temporal movement tendency between spatial areas under the assumption that the movement patterns can be described by the Markov chain model. The model has been used in the analysis of various kinds of movement data such as car traffic and demographic transition (i.e., how people travel from one area to another within a specified period).

In our approach, summarized trajectory data is represented as a special kind of histogram called *mobility histogram*. For the logical representation of the mobility histogram, we employ data cube-like representation so that the user can perform an OLAP-style interactive data analysis on the trajectory data. In mobility analysis, moving objects are monitored and tracked in real-time. Since the positions of moving objects are obtained continuously as a *trajectory stream* in such a situation, a dynamic histogram construction and maintenance facility is required. Considering this requirement, we propose a tree-based physical representation of a mobility histogram that supports efficient and adaptive maintainance. In addition, a histogram requires a compact data structure that fits in a limited memory space. The proposed physical histogram representation is designed to growth within the given upper-bound size.

The remaining part of the paper is organized as follows. Section 2 describes the related work. Section 3 introduces the notion of mobility statistics based on the Markov chain model. Section 4 introduces the logical histogram representation that is based on the data cube concept and Section 5 presents its physical structure and the construction/maintenance method. Section 6 shows the experimental results based on the proposed approach. Finally, Section 7 concludes the paper.

2. Related work

2.1. Aggregation in spatio-temporal databases

Aggregate computation in spatio-temporal databases has recently gained much interest. [17] surveys the techniques to evaluate aggregate queries on spatial, temporal, and spatio-temporal datasets, but moving objects are beyond its scope.

Database query optimization often requires statistical information such as selectivity estimates of the target database. For spatial databases, there exist several approaches for the estimation of query selectivities (e.g., [2]). For spatio-temporal databases, however, there are only a few proposals. [6, 7] present an approach to selectivity estimation for spatial queries on a spatio-temporal database that stores moving points. The approach in [25] calculates selectivities for spatio-temporal queries that change their query ranges with respect to time. In contrast to the above approaches, which aim to estimate query selectivities on spatio-temporal databases, our method focuses on summarizing a large number of moving objects using the Markov chain model. The summarized statistics can be used for the description and prediction of the overall trend of moving objects.

[24] proposes an approach of incrementally updatable multi-dimensional histogram. Their method, *adaptive multi-dimensional histogram* (*AMH*), is used to approximate the processing of present-time spatial queries. In addition to the incremental histogram maintance method, a method to archive old histogram buckets for historical queries and a smoothing method for future queries are proposed.

For the extraction of statistics from a spatio-temporal database, we have already proposed a method to compute Markov chain-based mobility statistics from an indexed spatio-temporal database [14]. The approach assumes that trajectories of moving objects are stored in a spatial index R-tree. We proposed an efficient method for the calculation of mobility statistics using an index. In contrast to this approach, the method proposed in this paper focuses on a dynamic environment in which trajectory data is delivered in a streamed fashion, thus, a completely different approach is required.

2.2. Multi-dimensional and dynamic histograms

A *histogram* is an important concept in database query planning and optimization to approximate data distributions in a database [13]. So far, various histogram construction techniques have been proposed, but most of them treat one-dimensional cases. There exist several approaches to treat multi-dimensional cases involving spatial databases (e.g., [2, 4, 22]). It is known that the construction of a multi-dimensional histogram is NP-hard [20]; thus, most the techniques alleviate the problems using heuristics and/or greedy approaches.

Besides the treatment of multi-dimensional cases, dynamic histogram construction and maintenance methods are proposed (e.g., [4, 9, 19, 26]). Among them, multi-dimensional cases are treated in [4, 26], but spatio-temporal databases are not considered.

The problem we tackle here can be classified into a dynamic and multi-dimensional case. However, in contrast to the previous approaches, our approach handles highly correlated data appearances due to the nature of moving objects and the Markov chain model (i.e., moving objects in the real world cannot move to an arbitrary area within a limited time). In addition, to support an exploratory mobility analysis performed by a user, we provide histograms with *multiple granularities*, namely, histograms with multiple spatio-temporal coarseness. By using mobility histograms with different spatio-temporal granularities, a user would be able to achieve a detailed analysis of the movement patterns.

3. Markov chain-based mobility statistics

In this section, we introduce the notion of Markov chain model with reference to the spatio-temporal mobility analysis. Table 1 shows the symbols and their definitions.

Suppose that a target two-dimensional space is partitioned into spatial *regions* as shown in Fig. 1. Each dimension is equally divided in 2^P ranges such that $R = 2^{2P}$ regions in total. The figure shows the case of $P = 2$. We call a partitioning shown in the figure as *level-P partitioning*. For each region, a $2P$ bit region number that obeys the *Z-ordering method* [21] is assigned.* The figure shows that object A located in region 9 at $t = \tau$ moves to region 12 at $t = \tau + 1$ and then moves to region 6 at $t = \tau + 2$.

Table 1. Symbols and their definitions

Symbol	Definitions
P	parameter to specify partitioning per dimension
R	total number of spatial regions
n	order of Markov chains
M	maximal partitioning level
N	maximal number of histogram nodes
W	window length

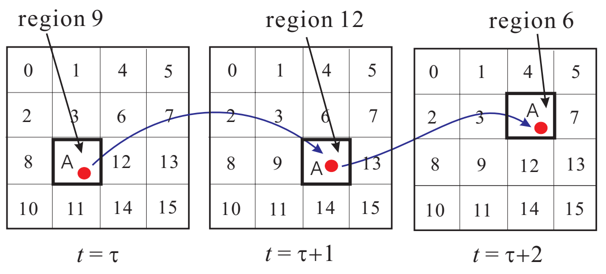

Figure 1. Notion of Markov chain model

Suppose that another moving object B located in region 9 moves to region 12 in unit time later. Consider that we want to know the probability that object B moves to region 6 next; we denote the probability by $\Pr(6|9, 12)$. If we assume the transition between spatial regions obeys the *Markov chain model* [27], we can say that the probability is a second-order *Markov transition probability*.

In order to generalize the above definition, we denote an order-n Markov transition probability by

*We can use other numberling schemes that have better clustering properties, but the Z-ordering method is used since it can be implemented using relatively simple algorithms.

$\Pr(r_n | r_0, \ldots, r_{n-1})$. That is, the probability of an object, that traveled to regions $r_0,\, r_1, \ldots, r_{n-1}$ in this order at unit time will next visit region r_n. Note that $r_0,\, r_1, \ldots, r_n$ ($r_i \in \{0, \ldots, R-1\}$, $0 \le i \le n$) can contain duplicates.

Consider that there is a dataset of moving object trajectories. Let the start time of recording the moving trajectories be $t = 0$, and let the current time be $t = T$. If we know the objects contained in each region at each unit time $t = 0,\, 1, \ldots, T$, we can estimate the probability $\Pr(r_n | r_0, \ldots, r_{n-1})$ as follows:

$$\Pr(r_n | r_0, \ldots, r_{n-1}) = \frac{\sum_{t=0}^{T-n} |\bigcap_{i=0}^{n} \mathrm{objs}(r_i,\, t+i)|}{\sum_{t=0}^{T-n} |\bigcap_{i=0}^{n-1} \mathrm{objs}(r_i,\, t+i)|}, \quad (1)$$

where $\mathrm{objs}(r_i,\, t)$ is a function that returns the set of objects contained in region r_i at time t. We could regard this probability as a special kind of an *association rule* [12].

The simplest probability estimation method would be to store all the trajectories in the database, and upon estimation request, we compute the probability using Eq. (1). Although the method is simple and clear, the database is required to store large volume of trajectory data, and the computational overhead is prohibitively large for an interactive analysis.

To tackle this problem, we propose a *mobility histogram* that approximately represents the distribution of movement patterns to reduce the data size and the computational cost in the analysis. In the next section, we describe its logical representation.

4. Logical histogram representation

4.1. Accumulating transition sequences in data cube

We assume that the incoming trajectory stream contains tuples with the form $\langle id, x, y, t\rangle$, where id is the id of an object, and x and y are the x and y coordinate values at time t. Given such stream data, we can easily transform it into an order-n Markov transition stream for a user-specified n. Each entry of the stream is an order-n transition sequence such as $9 \rightarrow 12 \rightarrow 6$ for $n = 2$. Here forth, we assume that order-n Markov transition sequences are continuously arriving from the stream. A *mobility histogram* is constructed for the stream to accumulate and summarize transition sequences. We employ a *data cube* as the logical representation of a mobility histogram.

To represent order-n Markov chain-based statistics, a histogram is constructed as an $(n+1)$-dimensional data cube. Figure 2 shows a sample of a data cube for $n = 2$. The data cube corresponds to the level-1 space partitioning ($P = 1$). Since the two-dimensional target space is partitioned into $R = 2^{2P} = 4$ spatial regions, the data cube contains $R^{n+1} = 64$ cells. For each dimension of the data cube, steps 0, 1, and 2 corresponds to each step of a second-order Markov chain. For instance, when the sequence $1 \rightarrow 1 \rightarrow 2$ is received from the transition sequence stream, the corresponding cell value is incremented. The row entitled *Sum* represents the total for each dimension.

In [5], a data cube is also used for spatio-temporal databases. However, they used a data cube as an index structure for moving objects and their cube consists of three dimensions (two spatial axes and a temporal axis).

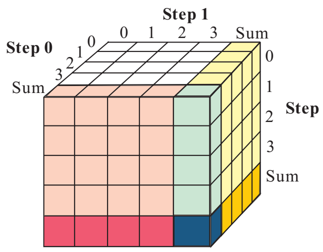

Figure 2. Logical histogram representation

In general, object movement patterns change as time passes so that a histogram constructed for a long time span may not be able to appropriately represent dynamically changing mobility statistics. Therefore, we construct a histogram for each set of W consecutive transition sequences, where W is called the *window length*. Old histograms are written in files and later utilized for further analyses. This approach to save old statistics for historical queries is inspired from [24].

4.2. Query processing using data cube

Consider a data cube for order-2 Markov chains. The probability that an object that has moved from region 1 to region 2 and then moves to region 4 is calculated as $\Pr(4|1,\, 2) = val(1,\, 2,\, 4)/val(1,\, 2,\, *)$, where $val(1,\, 2,\, 4)$ is the value of the cube cell $(1,\, 2,\, 4)$ and $val(1,\, 2,\, *) = \sum_{i=0}^{2^P-1} val(1,\, 2,\, i)$.

If we have a data cube for order-n Markov chains, we can also calculate the probabilities for order 1 to $n-1$ transitions. For example, the probability that an object in region 1 subsequently goes to region 2 is calculated as $\Pr(2|1) = val(1,\, 2,\, *)/val(1,\, *,\, *)$ using an order-2 data cube. In addition, a data cube can be used for various types of queries such as

- the probability that an object in region 1 at $t = \tau$ and in region 3 at $t = \tau + 2$ is in region 2 at $t = \tau + 1$ (τ is an arbitrary time): $val(1,\, 2,\, 3)/val(1,\, *,\, 3)$, and
- the probability that an object which is in region 2 at $t = \tau + 1$ and in region 3 at $t = \tau + 2$ is in region 1 at $t = \tau$: $val(1,\, 2,\, 3)/val(*,\, 2,\, 3)$.

Using the data cube representation, we can support other kinds of queries. For example, *density queries* [11], which discover dense areas from spatio-temporal databases, can be supported by a data cube using aggregation and selection.

4.3. Roll-up and drill-down operations

When we analyze object movement patterns, we sometimes need to see the underlying data in different granularities. For example, a user analyzing trajectories using a histogram constructed for one minute unit time basis may want to see a rough overview of the patterns using a histogram for 30 seconds unit time. This situation is similar to the *drill-down* operation [12] for *temporal dimension*. Of course, the opposite operation to use a coarser histogram (e.g., two minutes unit time) corresponds to the *roll-up* operation. To support drill-downs and roll-ups in our approach, we simply create multiple data cubes based on different unit times, e.g., one minute, two minutes, etc. If the histogram construction cost is low, the construction of multiple histograms is not critical.

We describe how to represent roll-up and drill-down operations in a *spatial sense* using Fig. 3, where first-order Markov chains are considered. The figure on the left-hand side shows the level-1 spatial partitioning and that on the right-hand side shows the level-2 partitioning. We can regard the data cube constructed from the level-1 (level-2) partitioning as the "roll-up" ("drill-down") version of the level-2 (level-1) data cube.

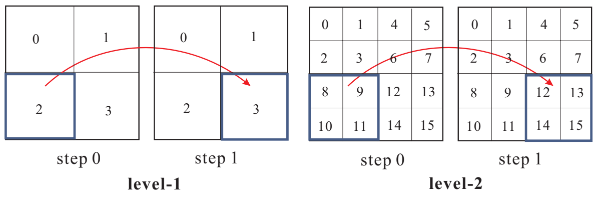

Figure 3. Roll-up and drill-down

In practice, we do not construct multiple histograms for different spatial resolutions but create only *one* histogram. As described in the next section, we represent a histogram physically as a compact approximated tree structure. Different spatial resolutions are adaptively handled by partial tree expansion.

5. Physical histogram representation

The direct implementation of the logical data cube representation has a huge overhead. As shown before, the total number of spatial regions is $R = 2^{2P}$, and a data cube for order-n Markov chains has R^{n+1} cells. When $P = 5$ and $n = 2$, for example, we have approximately one billion cells since $R^{n+1} = 1024^3$. Even if we represent each cell with two bytes, it requires 2 GB of memory. To reduce the size and enable dynamic maintenance of the histogram structure, we propose a tree-based physical representation.

5.1. Histogram structure and its construction

5.1.1 Basic structure

A physical histogram is represented as a tree structure that is a combination of a *quad-tree* and a *k-d tree* [23]. In this subsection, we describe the basic structure. The concrete structure is shown in the next subsection.

Suppose that input transition sequences are based on level-M partitioning such as $2^{(M)} \rightarrow 10^{(M)} \rightarrow 12^{(M)}$, where the notation $r^{(M)}$ is used to specify the partitioning level of region r explicitly. Here, M is called the *maximal partitioning level* since we cannot represent more accurate statistics beyond this level.

Each node in a histogram tree has zero to four child nodes each node containing a *counter* for the corresponding transition sequences. For example, consider an order-2 transition sequence $r_0^{(M)} \rightarrow r_1^{(M)} \rightarrow r_2^{(M)}$ is inserted in a tree, where $r_0^{(M)}, r_1^{(M)}$, and $r_2^{(M)}$ are region numbers. We call $r_0^{(M)}, r_1^{(M)}$, and $r_2^{(M)}$ step-0 region, step-1 region, and step-2 region, respectively. The sequence is processed as follows:

1. Start from the root of the tree.
2. Translate $r_0^{(M)}$ into a binary number, then extract its first two-bits. Depending on the value, that is, `00` (= $0^{(1)}$), `01` (= $1^{(1)}$), `10` (= $2^{(1)}$), or `11` (= $3^{(1)}$), follow the corresponding edge and then visit the child node. If the child node is not instantiated, create the node and an edge from the parent (root) to the node.
3. Extract the first two bits from $r_1^{(M)}$ and follow the corresponding edge. A non-instantiated node is treated as in Step 2.
4. Process $r_2^{(M)}$ in a similar manner. Steps 2 to 4 correspond to transitions in level-1 coarse resolution.
5. Use the next two bits from $r_0^{(M)}$, $r_1^{(M)}$, and $r_2^{(M)}$, respectively, to traverse following each edge. This step corresponds to level-2 partitioning.
6. Repeat these steps until a leaf node where all bits are consumed.

Note that there is a *counter* in each node; when we visit a node we increment its count. The idea of alternative expansion for each transition step is borrowed from k-d trees. The approach of iterative four-way decomposition of the space is inspired from quad-trees.

Figure 4 shows an example to add a transition sequence $3^{(2)} \rightarrow 6^{(2)} \rightarrow 12^{(2)}$ for a tree with $M = 2$. A dotted edge indicates that a corresponding transition sequence has not yet arrived so that the edge is not allocated.

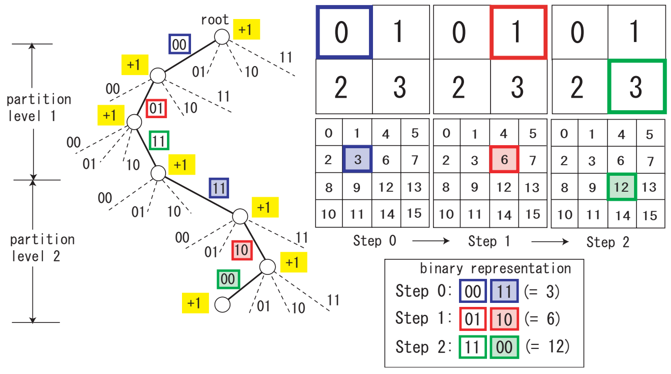

Figure 4. Physical histogram structure

5.1.2 Approximated representation

The physical histogram structure shown in the previous subsection is accurate but requires a large amount of storage space. Therefore, we propose its approximated representation. Under the given upper bound of the number of nodes N, we construct an approximated tree structure.

The idea is to expand the tree adaptively considering the statistical uniformity the nodes. Figure 5 is used as an example. It shows an image of an order-2 histogram tree after hundreds of insertions of transition sequences. Consider the black node, which is a leaf node. It corresponds to the transition sequence $1^{(1)} \rightarrow 3^{(1)} \rightarrow *$ since it has `01` for the first two bits of step 0 and `11` for those of step 1. This node has its own buffer that contains all the transition sequences with the transition sequence pattern shown above.

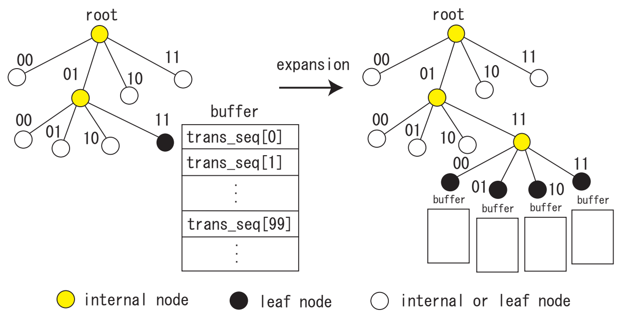

Figure 5. Examination for node expansion

Now, suppose that the buffer contains 100 transition sequences. We classify these sequences depending on their first two bits in step 2, which are `00`, `01`, `10`, or `11`, and then count their numbers. Assume that the counts are 25, 24, 26, and 24, respectively. This implies that an object which was in $1^{(1)}$ in step 0 and $3^{(1)}$ in step 1 goes to one of the four regions ($0^{(1)}, 1^{(1)}, 2^{(1)}$, and $3^{(1)}$) in step 2 with almost equal probabilities. In contrast, suppose that the counts are 55, 5, 40, and 0. In this case, the count distribution is skewed. Our approach to this situation is to "expand" this node and create four child nodes. The transition sequences contained in the current buffer are distributed to the child nodes and stored in their buffers. Using this approach, the tree can grow adaptively depending on the distribution of mobility patterns.

A histogram tree continues to expand until the number of nodes reaches N, the limit of the node numbers. We then count the number of transition sequences contained in each leaf node buffer. The counts are stored in the leaf nodes and then the buffers are finally deleted. In contrast to the basic method described above, the approximated tree contains counts only in the leaf nodes. After this point, the structure of the tree is fixed. When a new transition sequence arrives, we only increment the counter in the corresponding leaf node. When the number of arrived transition sequence becomes W, the window length, we store the histogram tree in a file and then start to create a new histogram. The approach to expand a tree adaptively is also found in decision tree construction [8]. The paper proposes the VFDT method that constructs a decision tree from stream data.

5.1.3 Checking non-uniformity

The problem faced by the approximated tree described above is how to detect skewness in count values. We need an appropriate method such that

- it is based on a statistically clear criteria, and
- it has efficient implementation.

The latter is quite important for dynamic environments required for stream data processing. For this purpose, we utilize the χ^2 *test for goodness of fit* [28]. Let the counts of `00`, `01`, `10`, and `11` during tree expansion be x_{00}, x_{01}, x_{10}, and x_{11}, respectively. The *null hypothesis* H_0 here is that the distribution of transition sequences is uniform and the probability that a transition sequence is classified into one of the four regions is $1/4$. The χ^2 value is calculated as follows:

$$\bar{x} = \frac{x_{00} + x_{01} + x_{10} + x_{11}}{4} \tag{2}$$

$$\chi^2 = \sum_{c \in \{00,01,10,11\}} \frac{(x_c - \bar{x})^2}{\bar{x}}. \tag{3}$$

It obeys the χ^2 distribution with $4 - 1 = 3$ degrees of freedom. When the significance level is 5%, we can assume that the distribution is non-uniform if $\chi^2 > 7.815$ holds; the constant is taken from the χ^2 statistics table.

When we utilize the χ^2 test for goodness of fit, we must consider the case when one or more values of x_c ($c \in \{00, 01, 10, 11\}$) in Eq. (3) are too small. The situation is common in our situation since transition sequences arrive in a streamed fashion and we first encounter small occurrence numbers. To solve this problem, we utilize the nonparametric statistical test instead of the χ^2 test when the total number of transition sequences is small. The method is an extended version of the *binomial test* [28], a well-known robust nonparametric method. The detail is shown in the appendix.

5.2. Query processing

We describe the query processing method to estimate the count of a given transition sequence using a histogram. For example, if we use a histogram for order-2 ($n = 2$) Markov chains, a generic query is given as follows:

> Estimate the count of the transition sequence $r_0^{(m_0)} \rightarrow r_1^{(m_1)} \rightarrow r_2^{(m_2)}$.

Queries to a logical histogram, namely a data cube, can be processed by the combination of queries in this form. We illustrate the overview of the process using simple examples.

Suppose that a query $3^{(2)} \rightarrow 6^{(2)} \rightarrow 9^{(2)}$ is given. Since it can be translated into a binary representation `0011` $\rightarrow$ `0110` $\rightarrow$ `1001`, we try to traverse the tree as

$$\underset{(1,0)}{\texttt{00}} \rightarrow \underset{(1,1)}{\texttt{01}} \rightarrow \underset{(1,2)}{\texttt{10}} \rightarrow \underset{(2,0)}{\texttt{11}} \rightarrow \underset{(2,1)}{\texttt{10}} \rightarrow \underset{(2,2)}{\texttt{01}}$$

from its root, where 00, 01, 10, and 11 represent the labels of the edges to be followed. The notation (l, s) means that the move corresponds to the partition level l and step s. The actual query processing depends on the situation and is performed as follows:

1. If the node visited by the final move 01 (2, 2) is a leaf node, we return the count contained in the node.
2. When the final move 01 (2, 2) still visits a non-leaf node, we traverse the descendants of this node, accumulate the counts in the leaf nodes, and then return the summation of the counts.
3. When we meet a leaf node in the middle of the traversal, it means that the histogram only contains coarse statistics about the transition sequence. Therefore, we return an approximated result. For example, if we reach a leaf node when we move as

$$\underset{(1,0)}{\texttt{00}} \rightarrow \underset{(1,1)}{\texttt{01}} \rightarrow \underset{(1,2)}{\texttt{10}} \rightarrow \underset{(2,0)}{\texttt{11}},$$

we estimate the count as $C/4^2 = C/16$, where C is the count contained in the leaf node.

We can process a query in which the resolutions of query regions are not equal. Consider the query $3^{(2)} \rightarrow 1^{(1)} \rightarrow 9^{(2)}$ as an example: it uses a coarse resolution only for step 1. The query can be represented in a binary format as $\texttt{0011}^{(2)} \rightarrow \texttt{01}^{(1)} \rightarrow \texttt{1001}^{(2)}$, but note that we can expand it as $\texttt{0011}^{(2)} \rightarrow \texttt{01}**^{(2)} \rightarrow \texttt{1001}^{(2)}$, where "**" represents any of 00, 01, 10, and 11. Therefore, we can simply issue four queries then take their summation to answer the given query.

A more complex query can be solved using a similar approach; we decompose a query into simpler queries and then integrate their query results. Since a query can be processed quite efficiently as shown later, the execution of multiple queries is not really an overhead. Of course, we can develop a sophisticated algorithm that can traverse a tree with a smaller number of queries; however, this will result in a complex algorithm.

5.3. Use of a bitmap

The approximated tree is compact, but it has an error. We propose an extended method for the approximated representation to improve its accuracy using a *bitmap*. The basic idea is to construct an additional data cube in a coarse resolution, but each cube cell contains a binary value. If there is one or more transition sequences, the value is one, otherwise it is zero. For example, if the order of Markov chains is $n = 2$ and the partitioning parameter is $P = 3$, since $R = 2^{2\times3} = 64$, the size of the bitmap becomes $R^{n+1} = R^3 = 262,144$ bits = 32 KB.

A bitmap is utilized in query processing as follows. Given a query transition sequence, we first determine whether the count of the sequence is zero using the bitmap. We return zero if the bit is zero, otherwise an estimated count is returned using the method described above. If we use a more detailed bitmap (e.g., $P = 4, 5, \ldots$), we can provide more accurate results; the bitmap can also be used for queries in coarser levels. However, this requires additional storage space and there is a tradeoff between accuracy and efficiency.

6. Experiments

In this section, we evaluate the proposed methods based on the experiments. We compare three methods, the basic naive method (BASE), the approximated method (APR), and the approximated method enhanced with a bitmap (APR-BM). In Subsection 6.1, we present the experimental settings. Subsection 6.2 compares histogram construction costs of the three methods in terms of storage size and construction time. Subsection 6.3 shows the result of the query processing time, and Subsection 6.4 evaluates the accuracy of the method.

6.1. Experimental settings

We perform experiments using the trajectory data generated by the moving object simulator made by Brinkoff [3]. The system simulates the traffic of moving objects on a real city road network. The data used in the experiments is generated from the core part of Oldenburg city, Germany (about 2.5 km × 2.8 km in area). Figure 6 shows the map of the city. The maximal partitioning is $M = 10$; namely, the entire map is decomposed into $1,024 \times 1,024$ regions in the most detailed resolution.

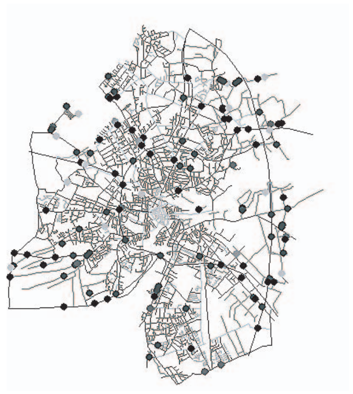

Figure 6. Simulation area

We set up the parameters such that there were 1,000 moving objects for every moment of the simulation. The simulation time length was 50 unit times. The experiments were performed on a 3.2 GHz Pentium 4 PC with 1 GB memory.

6.2. Histogram construction

We extracted 50,000 order-2 ($n = 2$) Markov chain trajectory sequences from the simulation data. We used the top 1,000 entries and 10,000 entries in some experiments. Although we have described the use of multiple time windows in Subsection 4.1, we utilize only one time window and construct one histogram for each method in the following experiments. The simulation dataset used in the experiments is considered to be based on a stational process so that it is reasonable to use one time window.

6.2.1 Histogram size

Table 2 shows the histogram sizes for three methods BASE, APR, and APR-BM. 1K, 10K, and 50K imply that the number of the trajectory sequence is 1,000, 10,000, and 50,000, respectively. For APR, we need to specify the upper bound of the allocatable nodes, we have specified 1,000, 10,000, and 50,000 for each case of 1K, 10K, and 50K. The numbers of actually allocated nodes are 724, 6,692, and 33,844. That is, our approximated histogram construction algorithm decided that full node allocatoin is not necessary for each case. For APR-BM, the same upper bounds are specified so that the actual numbers of nodes are the same as APR. APR-BM also needs a bitmap; we have allocated a bitmap with the partitioning level $P = 3$.

Table 2. Histogram size (MB)

Data Size	BASE	APR	APR-BM
1K	0.35	0.01	0.04
10K	2.7	0.10	0.13
50K	9.4	0.52	0.55

As shown in the table, the naive method BASE has a huge storage overhead because it does not use any data reduction. Thus, BASE is not an appropriate method for the accumulation of mobility statistics in a compact data structure. APR-BM utilizes a bitmap of the partition level $P = 3$. Although it requires 32 KB of storage, the additional cost is relatively small compared to the histogram size of APR. Therefore, we can say that the approximated method APR and its extension APR-BM have good features in terms of storage cost. Note that if we employ $P = 4$, the size of the bitmap becomes 256 KB and its effect on the total histogram size is non-negligible. Bitmap compression techniques may be effective in solving this problem.

6.2.2 Histogram construction time

Figure 7 shows the construction time of each histogram for different data sizes: 1K, 10K, and 50K. The data is an averaged value of ten measurements. Figure 8 shows the same data, but averaged per transition sequence insertion. We have ommited APR-BM in these figures since its construction cost is almost same as APR. These figures also includes the experimental results for the input trajectory data with $M = 5$ to evaluate the difference in the construction cost when the base resolution of the space is coarse.

As shown in the above figures, the construction cost of APR is higher than BASE. This is due to the non-uniformity checking of APR: it has to check whether the transition sequences in the leaf nodes were skewed in the occurrence patterns. As shown in the appendix, we have developed an efficient method for statistical testing, but it has a small overhead. The reason for the increase in with the data size is that the number of leaf nodes becomes large as the tree grows and we have to check more leaf nodes.

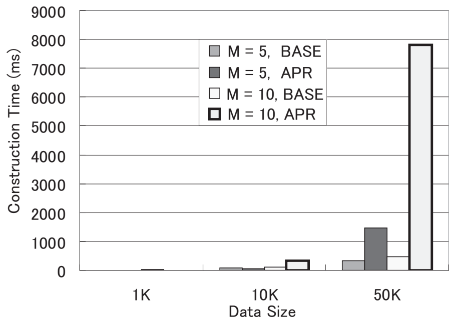

Figure 7. Construction time

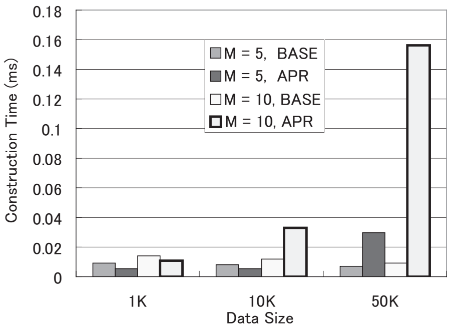

Figure 8. Construction time (per sequence)

Figure 8 shows that the averaged construction time per transition sequence is only 0.16 ms even for 50K data and $M = 10$. Although APR (also APR-BM) is slower than BASE, the construction overhead is quite small. For example, consider an application for car traffic monitoring. The speed of a vehicle is relatively slow compared to the histogram construction speed so that a 10 second or more interval for the monitoring would suffice. Of course, monitoring a large number of vehicles would be a problem, but we would be able to employ sampling and/or load shedding [1] for the problem.

6.3. Query processing time

We present the results on query processing time. The target histograms are based on the same parameters in the previous subsection and $M = 10$. We randomly constructed 100 transition sequences as queries, and measured the average query processing time. Two types of queries are considered:

1. *fine-level query*: The partition level of each region of a transition sequence is equal to the maximal partition level $M = 10$ (e.g., $909876^{(10)} \rightarrow 397555^{(10)} \rightarrow 399468^{(10)}$).

2. *mixed-level query*: The target level of a query is selected randomly between 1 and 9 (e.g., $2^{(2)} \rightarrow 2^{(2)} \rightarrow 2^{(2)}$, $53662^{(9)} \rightarrow 66816^{(9)} \rightarrow 109748^{(9)}$).

Figure 9 shows the query processing time for each query type per query. The result is an average of the same 10 experiments. 1K, 10K, and 50K are the sizes of the input data. We omit the data for APR-BM because the cost is almost the same as APR.

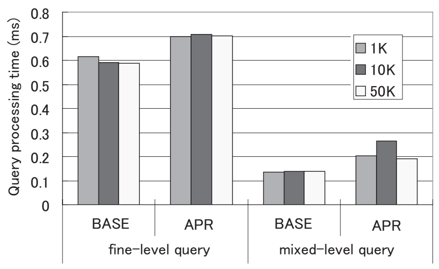

Figure 9. Query processing time

As shown in the figure, the query processing cost does not depend on the target data size and the type of a histogram. One dominant factor is the resolution level of a query, a fine-level query takes more time than a mixed-level query. However, the execution of a query is quite fast and less than 1 ms in any case. Therefore, we can say that any types of histogram can achieve fast query processing.

6.4. Accuracy of histograms

6.4.1 Intuitive example

In order to obtain an intuition for the accuracy of the constructed histograms, we present some plots of histograms. Figure 10 shows a constructed histogram for 10K transition sequences using the BASE method. The order of Markov chains is $n = 1$ and the space partition level is $P = 2$. Figure 11 is an APR histgoram for the same dataset. As shown in the figures, their trends are quite similar. The diagonal cells take high scores; the reason is that a moving object in a region will either stay at the region or move to neighboring regions—it cannot move to distant regions.

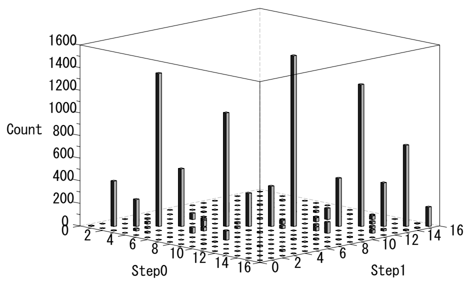

Figure 10. BASE **histogram ($n = 1$, $P = 2$)**

Figure 12 shows their absolute difference. As we can observe, the difference is quite small compared to the counts in the original histograms.

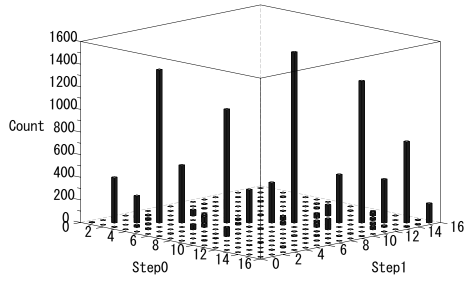

Figure 11. APR **histogram ($n = 1$, $P = 2$)**

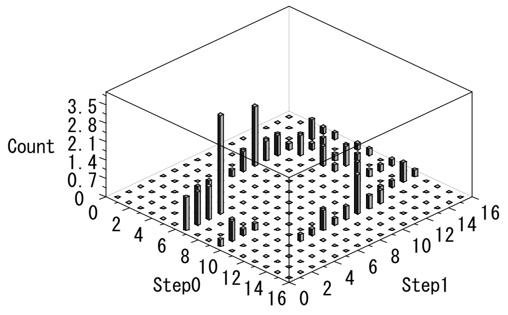

Figure 12. Absolute histogram difference

6.4.2 Evaluation measures

We evaluate the accuracy of approximated histograms by the following two evaluation measures:

$$Dist = \sqrt{\sum_{i=1}^{R^{n+1}} (ACT_i - EST_i)^2} \tag{4}$$

$$RelErr = \sqrt{\frac{1}{R^{n+1}} \sum_{i=1}^{R^{n+1}} \left(\frac{ACT_i - EST_i}{ACT_i}\right)^2} \tag{5}$$

where $R = 2^{2P}$ and R^{n+1} is the total number of cube cells. Cube cells are numbered from 1 to R^{n+1}. ACT_i represents the actual value of cube cell i, which is taken from a cube of the BASE method. EST_i is the estimated value of cell i in an approximated histogram. The measure $Dist$ is the conventional Euclidean distance and $RelErr$ is the relative error.

The computation of a relative error value has a problem: EST_i may take zero so that "division by zero" occurs in Eq. (5). To solve this problem, we employ the *Laplace estimator* to compute Eq. (5):

$$\hat{C}_i = \frac{C_i + 1}{S + B} \times S, \tag{6}$$

where C_i is the original value (ACT_i or EST_i) and $\hat{C}_i$ is its corrected value. B is the number of *bins* and S is the total number of *instances*; they are defined as follows:

$$B = R^{n+1} \tag{7}$$

$$S = \sum_{i=1}^{B} C_i \tag{8}$$

See [18] for details of the Laplace estimator.

6.4.3 Evaluation result

We present the evaluation result for a typical parameter setting. Figure 13 shows the comparison of `APR` and `APR-BM` based on the Euclidean distance $Dist$ in Eq. (4). We used order-2 histograms for 50K transition sequences. The bitmap of `APR-BM` is constructed for the partition level $P = 3$. For the calculation using Eq. (4), we set the partition level $P = 3$; that is, we enumerate all the transition sequences at the level $P = 3$ (e.g., $1^{(3)} \rightarrow 2^{(3)} \rightarrow 3^{(3)}$) and then estimate the corresponding cube cell values and use them to compute a distance. 10K, 20K, 30K, and 34K in the x-axis are the number of allocated histogram nodes. We notice that the allocation of the nodes improves the accuracy, but its effect decreases gradually. In this figure, we cannot observed clear difference between `APR` and `APR-BM`.

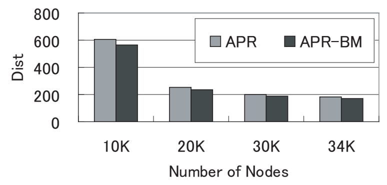

Figure 13. Euclidean distance

Figure 14 shows the result based on the relative error $RelErr$ (Eq. (5)) for the same histograms. It is clear that `APR-BM` has a good accuracy. The reason is that `APR-BM` can provide accurate estimates for cube cells that actually take zero values. In the case of the Euclidean distance, the cells that take large values dominate the distance score and the cells with small values have little effect. On the other hand, it is critical that the relative error estimates the values of the cells that take small values.

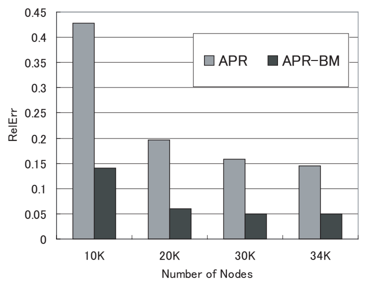

Figure 14. Relative error

Therefore, the use of a bitmap is quite effective with a little storage overhead when the relative error is important.

7. Conclusions and future work

In this paper, we proposed a method to construct a histogram to represent mobility statistics based on the Markov chain model to represent movement patterns of a large number of moving objects. We described an OLAP-like mobility analysis approach and showed the logical data cube representation for the analysis. Then, we introduced the physical histogram with a tree structure.

To reduce the storage overhead of the histogram, we introduced the approximated histogram representation and presented a histogram construction algorithm that gradually expands a tree considering the statistical property of the leaf nodes in the tree. We also proposed the additional use of a bitmap to enhance the accuracy of the approximated histogram.

As shown in the experiments, our approximated histogram achieved low storage cost compared to the naive tree histogram. Although its construction cost is larger than that of the naive method, the cost is still quite low and enables us to use the method for monitoring the movement of a large number of moving objects. The accuracy of the proposed histograms were evaluated using some measures and the result shows that the approximated histogram enhanced with a bitmap can achieve high precision with a little storage overhead.

The work presented in this paper is ongoing in our laboratory. The future work includes more complete evaluation based on real-time data. Further, we aim to develop a more sophisticated histogram construction scheme that can achieve high precision without the increase of the processing costs. Finally, we want to construct an application framework that utilizes the proposed histogram technique to support the exploratory mobility analyses performed by users.

Acknowledgments

This research is partly supported by a Grant-in-Aid for Scientific Research (16500048, 15300027) from the Japan Society for the Promotion of Science (JSPS), Japan, and a Grant-in-Aid for Scientific Research on Priority Areas (16016205) from the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan. In addition, this work is supported by the grants from the Inamori Foundation.

References

[1] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *VLDB Journal*, 12(2):120–139, 2003.

[2] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity estimation in spatial databases. In *Proc. ACM SIGMOD*, pages 13–24, 1999.

[3] T. Brinkhoff. A framework for generating network based moving objects. *GeoInfomatica*, 6(2):153–180, 2002.

[4] N. Bruno, L. Gravano, and S. Chaudhuri. STHoles: A workload aware multidimensional histogram. In *Proc. SIGMOD*, pages 211–222, 2001.

[5] W. Choi, B. Moon, and S. Lee. Adaptive cell-based index for moving objects. *Data & Knowledge Engineering*, 48(1):75–101, 2004.

[6] Y.-J. Choi and C.-W. Chung. Selectivity estimation for spatio-temporal queries to moving objects. In *Proc. ACM SIGMOD*, pages 440–451, 2002.

[7] Y.-J. Choi, H.-H. Park, and C.-W. Chung. Estimating the result size of a query to velocity skewed moving objects. *Information Processing Letters*, 88(6):279–285, 2003.

[8] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proc. KDD*, pages 71–80, 2000.

[9] P. Gibbons, Y. Mattias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proc. VLDB*, pages 466–475, 1997.

[10] R. H. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann, 2005.

[11] M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. J. Tsotras. On-line discovery of dense areas in spatio-temporal databases. In *Proc. SSTD*, volume 2750 of *LNCS*, pages 306–325, 2003.

[12] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[13] Y. Ioannidis. The history of histograms (abridged). In *Proc. VLDB*, pages 19–30, 2003.

[14] Y. Ishikawa, Y. Tsukamoto, and H. Kitagawa. Extracting mobility statistics from indexed spatio-temporal datasets. In *Proc. 2nd Workshop on Spatio-Temporal Database Management (STDBM'04)*, Toronto, Canada, 2004.

[15] C. S. Jensen (ed.). Special issue: Indexing of moving objects. *IEEE Data Engineering Bulletin*, 25(2), 2002.

[16] C. S. Jensen (ed.). Special issue: Infrastructure for research in spatio-temporal query processing. *IEEE Data Engineering Bulletin*, 26(2), 2003.

[17] I. F. V. López, R. T. Snodgrass, and B. Moon. Spatiotemporal aggregate computation: A survey. *IEEE TKDE*, 17(2):271–286, 2005.

[18] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[19] Y. Mattias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *Proc. VLDB*, pages 101–111, 2000.

[20] S. Muthukrishnan, V. Poosala, and T. Suel. On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. In *Proc. ICDT*, pages 236–256, 1999.

[21] J. A. Orenstein. Spatial query processing in an object-oriented database system. In *Proc. SIGMOD*, pages 326–336, 1986.

[22] V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proc. VLDB*, pages 486–495, 1997.

[23] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1989.

[24] J. Sun, D. Papadias, Y. Tao, and B. Liu. Querying about the past, the present, and the future in spatio-temporal databases. In *Proc. ICDE*, pages 202–213, 2004.

[25] Y. Tao, J. Sun, and D. Papadias. Selectivity estimation for predictive spatio-temporal queries. In *Proc. ICDE*, pages 417–428, 2003.

[26] N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic multidimensional histograms. In *Proc. SIGMOD*, pages 428–439, 2002.

[27] G. J. G. Upton and B. Fingleton. *Spatial Data Analysis by Example, Volume II: Categorical and Directional Data*. John Wiley & Sons, 1989.

[28] J. H. Zar. *Biostatistical Analysis*. Prentice Hall, 4th edition, 1998.

Appendix: Non-uniformity checking using nonparametric statistics

The *binomial test* [28] is one of the nonparametric statistical tests. A typical null hypothesis H_0 is that the head and the tail probabilities of a coin are $1/2$. Consider performing t trials and let the smaller number of occurrence of head or tail be k. For example, when $t = 10$, the probability for $k = 0$ is $\Pr(k = 0) = 2 \cdot \binom{10}{0} \cdot (\frac{1}{2})^{10} = 0.002$, and $\Pr(k = 1) = 2 \cdot \binom{10}{1} \cdot (\frac{1}{2})^{10} = 0.020$. Similarly, we get $\Pr(k = 2) = 0.088, \Pr(k = 3) = 0.234$, etc. Therefore, the *rejection regions* for the confidence level 5% are [0, 1] since $\sum_{i=0}^{1} \Pr(10, i) = 0.021 < 0.05$ and $\sum_{i=0}^{2} \Pr(10, i) = 0.109 > 0.05$. That is, we can reject H_0 when $k = 0$ or 1.

We generalize the idea to multinomial cases. Suppose that we have a four-face dice. The null hypothesis H_0 is that the occurrence probability of each face is $1/4$. Consider, for example, that we have the occurrence pattern (6, 1, 1, 0) after $t = 8$ trials, where an occurrence pattern is a sequence of four occurrence numbers with the descending order. The probability that we get the pattern is $\Pr(6, 1, 1, 0) = 4 \cdot 3 \cdot \frac{8!}{6!1!1!0!}(\frac{1}{4})^8 = 0.0103$. Note that patterns rarer than this one are (6, 2, 0, 0), (7, 1, 0, 0), and (8, 0, 0, 0), and their probabilities are $\Pr(6, 2, 0, 0) = 0.00513$, $\Pr(7, 1, 0, 0) = 0.00146$, and $\Pr(8, 0, 0, 0) = 0.0000610$ respectively. Since the sum of these probabilities is $\Pr(6, 1, 1, 0) + \Pr(6, 2, 0, 0) + \Pr(7, 1, 0, 0) + \Pr(8, 0, 0, 0) = 0.0169 < 0.05$, the pattern (6, 1, 1, 0) is in the rejection region for the confidence level 5%.

Based on similar analyses for the patterns of $t = 1, 2, \ldots$, the patterns that can be rejected are derived as follows:

- $t = 1, \ldots, 3$: no patterns
- $t = 4$: (4, 0, 0, 0)
- $t = 5$: (5, 0, 0, 0)
- $t = 6$: (6, 0, 0, 0), (5, 1, 0, 0)
- $t = 7$: (7, 0, 0, 0), (6, 1, 0, 0), (5, 2, 0, 0)
- …

In our implementation, we have derived these patterns until $t = 52$. After the t-value, the χ^2 test for goodness of fit can be safely used.

In the histogram construction module, precomputed rejection patterns are stored in a table and then used for the non-uniformity check as described in Fig. 5. Since the non-uniformity check is performed every time when a new transition sequence is inserted into a buffer, we can reduce the size of the table. For instance, the rejection pattern (5, 0, 0, 0) for $t = 5$ does not happen in reality because it requires (4, 0, 0, 0) as the preceding pattern, but this pattern (4, 0, 0, 0) triggers tree expansion so that we cannot reach (5, 0, 0, 0). Based on the reduction, the table in our experiments only contains 465 entries of the rejection patterns.