

Extracting Mobility Statistics from Indexed Spatio-Temporal Datasets

Yoshiharu Ishikawa

Yuichi Tsukamoto

Hiroyuki Kitagawa

Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba
1-1-1 Tennoudai, Tsukuba, 305-8573, Japan
{ishikawa,kitagawa}@cs.tsukuba.ac.jp, yuichi@kde.is.tsukuba.ac.jp

Abstract

With the recent progress of spatial information technologies and mobile computing technologies, spatio-temporal databases that store information of moving objects have gained a lot of research interests. In this paper, we propose an algorithm to extract *mobility statistics* from indexed spatio-temporal datasets for interactive analysis of huge collections of moving object trajectories. We focus on mobility statistics called the *Markov transition probability*, which is based on a cell-based organization of a target space and the *Markov chain model*. The algorithm computes the specified Markov transition probabilities efficiently with the help of an R-tree spatial index. It reduces the statistics computation task to a kind of constraint satisfaction problem and uses internal structure of an R-tree in an efficient manner.

1 Introduction

The wide use of digitized geographic data has increased the demand for the spatial database technology to manage huge volume of spatial information. Moreover, effective data management for mobile users has become more important as the spreading use of mobile devices. Development of *spatio-temporal database* technologies to support moving objects is one of the important database research areas [5].

In the research field of moving object databases, development of efficient indexing techniques is one of the important issues and there exist many proposals of *spatio-temporal indexes* [5]. Additionally, there are some proposals for the extraction of statistical information from spatio-temporal databases [3, 8, 11, 12]. Statistics concerning spatio-temporal data is not only useful for the efficient query processing but also in *mobility analysis* [13] to analyze the movement patterns of objects from accumulated spatio-temporal trajectory data. Since accumulated trajectory data may have huge volume, we need an efficient method to calculate statistics.

In this paper, we propose an algorithm for extracting mobility statistics from moving object trajectories with the support of spatial indexes. Especially, we consider the mobil-

ity statistics based on the *Markov chain model*. The Markov chain model in spatio-temporal data analysis is used for analyzing movement tendency of moving objects such as how population moves from a certain region to other regions while a specified period [13]. Using such statistical information, we can estimate with high probability whether an object at some region will move to another region in the next period.

In this paper, we assume that trajectories of moving objects are accumulated in a spatial index such as an R-tree and aim to estimate Markov transition probabilities efficiently using the index. The problem of estimating a transition probability from an R-tree is formulated as a kind of *constraint satisfaction problem (CSP)*. This paper describes the framework, the algorithms, and the evaluation results.

This paper is organized as follows. Section 2 introduces the notion of mobility statistics based on the Markov chain model. Section 3 describes the related work. Section 4 shows a general trajectory indexing approach based on R-trees; it is used as the basic assumption to construct our algorithms. Section 5 presents the naïve algorithm for extracting mobility statistics from an R-tree, and Section 6 describes a more efficient algorithm that is based on the constraint satisfaction paradigm. Section 7 shows an illustrative example of query processing of the proposed algorithm. Section 8 presents the experimental setup and the results. Finally, Section 9 concludes the paper.

2 Markov chain-based mobility statistics

As shown in Fig. 1, we assume that the entire map is divided into cells. Each cell must be a rectangle but not necessarily in a uniform size. A cell number is assigned to each cell so that we can specify a cell using its number. The figure shows the situation that object A that was in cell c_0 at $t = \tau$ has moved to cell c_1 at $t = \tau + 1$, then moved to cell c_2 at $t = \tau + 2$.

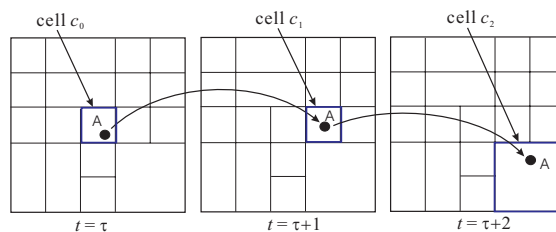


Figure 1: Notion of the Markov chain model

Suppose that we want to estimate the probability $\Pr(c_1|c_0)$ that an object existing in cell c_0 next moves to cell c_1 , like

Copyright held by the author(s).

Proceedings of the Second Workshop on Spatio-Temporal
Database Management (STDBM'04),
Toronto, Canada, August 30th, 2004.

object A, using the trajectory data stored in a spatio-temporal database. Assume that the database stores a huge volume of trajectories of moving objects and each trajectory record starts from $t = 0$ and ends at $t = T$. If we can determine which objects are located in a given cell at a specified time ($t = 0, 1, \dots, T$), we can compute the probability as follows:

$$\Pr(c_1|c_0) = \frac{\sum_{t=0}^{T-1} |\text{objs}(c_0, t) \cap \text{objs}(c_1, t+1)|}{\sum_{t=0}^{T-1} |\text{objs}(c_0, t)|}, \quad (1)$$

where $\text{objs}(c_i, t)$ is a function that returns the set of objects that were in cell c_i at time t . In this formula, the denominator is the sum of the number of objects that existed in c_0 at each time $t = 0, \dots, T-1$. Among these objects, the ones that have moved to c_1 in the next time period are included into the count of the numerator.

The probability $\Pr(c_1|c_0)$ corresponds to a transition probability of a first-order Markov chain because the next state (cell c_1) is predicted using the current state (cell c_0) only. We can generalize this formulation to multiple orders. The probability that an object which was in cells c_0, c_1, \dots, c_{n-1} with this order in each period of unit time interval moves to cell c_n in the next period is denoted as $\Pr(c_n|c_0, \dots, c_{n-1})$, and estimated by the following generalized form:

$$\Pr(c_n|c_0, \dots, c_{n-1}) = \frac{\sum_{t=0}^{T-n} |\bigcap_{i=0}^n \text{objs}(c_i, t+i)|}{\sum_{t=0}^{T-n} |\bigcap_{i=0}^{n-1} \text{objs}(c_i, t+i)|}. \quad (2)$$

Note that the set of cells $\{c_0, c_1, \dots, c_n\}$ may contain duplicates.

If we can estimate Markov transition probabilities efficiently, we would be able to forecast the cell where a moving object next moves to. Moreover, given the status of moving objects at $t = \tau$, we can simulate how the movement status changes as time passes ($t = \tau + 1, 2, \dots$). As described later, our algorithms allow us to specify a cell decomposition of the target space and to set a unit time interval in an interactive manner, based on the analysis requirement. Therefore, we can say that the proposed algorithms are suited for interactive exploratory mobility analyses.

In this paper, we assume that moving objects obey a stationary process and do not change their transition behaviors depending on time. Treatment of the non-stationary case will be considered in the future work.

3 Related work

3.1 Spatio-temporal data mining

Estimation of statistics from databases is important for the efficient evaluation of queries. Estimated *selectivity* value for a query is often used in query optimization. There are a few proposals of selectivity estimation methods for spatio-temporal databases. For example, [3] proposes a selectivity estimation method for a spatial range query (does not include a temporal dimension) on a spatio-temporal database which stores moving point objects. [11] generalizes this approach and provides a selectivity estimation method for a spatio-temporal range query which changes the shape of a query area depending on time.

A Markov transition probability can be seen as a special kind of an *association rule* [4], but the probability considers “sequences” of spatial object movements instead of “sets” of

items in association rule mining. Namely, the Markov transition probability represents a kind of *sequence association rule* in a spatio-temporal environment.

According to sequence mining from spatio-temporal databases, we can find some approaches. [8] proposes a method of user moving pattern mining for a mobile environment. [12] presents an efficient mining method of spatio-temporal patterns from environmental data. Both approaches aim to find frequent spatio-temporal patterns defined as sequences of locations. In contrast to our dynamic cell specification approach, their methods require that a space decomposition (i.e., the set of sequence items) is fixed beforehand.

Additionally, the use of a spatial index is another characteristics of our research compared with the related papers. Using the internal information of a spatial index, the proposed algorithm calculates mobility statistics efficiently.

3.2 Solving CSPs using spatial indexes

The purpose of this research, namely, to derive transition probabilities between cells from moving object trajectories indexed by a spatial index, can be reduced to a task to enumerate groups of objects each of which satisfies a kind of temporal constraint, as described later. A technique to enumerate all groups of objects, each of which fulfills a specified spatial relationships, using a spatial index R-tree is proposed in [7]. The method aims to solve a *constraint satisfaction problem (CSP)* defined by spatial constraints. An example of a query is “Find all the tuples (x, y, z) of spatial objects each of which satisfies the constraint overlaps(x, y) and north(y, z)”. The proposed algorithm descends a spatial index from the root toward the leaves by pruning non-necessary candidates and enumerates the groups of objects that satisfy the constraints. We extend this technique according to our context.

The process of enumerating object groups that satisfy the given constraints from a spatial index can be considered as a kind of *spatial joins* with spatial indexes [2]. In contrast to [7] that searches the object groups satisfying some spatial constraints, our approach aims to find object groups that satisfy temporal constraints derived from the definition of the Markov chain model. In our approach, spatial constraints on cells are used to restrict the search space of the solutions that satisfy temporal constraints. Although our approach is an extension of [7] in the sense that we reduce an enumeration problem to CSP, the constraints used are totally different from [7].

4 Indexing spatio-temporal objects

4.1 Indexing methods for trajectories of moving objects

As described in Section 2, the key point to estimate a transition probability is to find the set of objects $\text{objs}(c, t)$ which were in cell c at time t . Efficient use of available information from the underlying spatio-temporal database is indispensable for that purpose. In our approach, we assume that the trajectories of moving objects are indexed by a spatial index R-tree.

There are some existing approaches to trajectory data indexing based on R-trees. For example, [6] introduces two index structures; *3D R-trees* incorporate a temporal dimension and represent trajectory data with three dimensions, and *2+3*

R-trees represent the positions of moving objects using two-dimensional R-trees and represent movement histories using tree-dimensional R-trees. *STR-trees* [9] decompose a trajectory into multiple line segments to store them into its R-tree-based index structure.

Next we introduce an R-tree-based trajectory indexing method that is based on a simple and general approach. The algorithm presented later assumes the use of them.

4.2 Illustrative example of trajectory indexing

As an example, let us consider a moving object in one dimensional space. Figure 2 shows that objects A and B move on the x -axis from $t = 0$ to $t = T (= 8)$. A trajectory is expressed using a curve. Since a real trajectory is complex as shown, some approximation is necessary for the representation on a computer. Each point on the curves shown in Fig. 2 represents a sampled point at every time. Using this approximation, the trajectory of each object can be expressed by a sequence of (time, x -value) pairs. In an environment where the position of a moving object is detected by a GPS at every unit time, this representation would be a natural one.

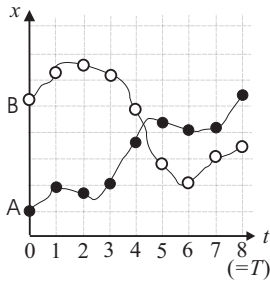


Figure 2: Representation of trajectories

If the position of a moving object o at time $t = \tau$ ($\tau = 0, 1, \dots, T$) is represented as a point $[x_1, \dots, x_d]$ in a d -dimensional space, we can construct a $(d+1)$ -dimensional R-tree and represent information about o at each time t as a $(d+1)$ -dimensional point $[x_1, \dots, x_d, t]$, and store it with the object id of o into the R-tree. This is a kind of generalization of 3D-R trees [6].

5 Naïve algorithm for probability estimation

Here we introduce some notions and present the naïve transition probability estimation algorithm which is directly based on the definition of the Markov transition probability.

5.1 Preliminaries

First we introduce some terms. The times $t = 0, 1, \dots, T$ for each of which we collect statistics value are called *sampling times*. The time interval length between two adjacent sampling times (e.g., one minute) is called a *base sampling period*.

Assume that T has the form $T = 2^m$ and a user can specify a *user-level sampling period* 2^k ($0 \leq k < m$). For example, if a user specify the user-level sampling period as $2^1 = 2$ ($k = 1$), the estimation of a probability is performed using the items for $t = 0, 2, 4, \dots, T$. This generalization allows us to perform coarser mobility analysis if we want. If the base sampling period is too small for an analysis (e.g., one second), we can use a longer sampling period (e.g., 64 seconds). Although we only consider the case of $k = 0$ in the

following discussion, we can easily extend the proposed algorithms to more general cases.

Next we describe the restrictions on cells. Each cell region must have a rectangular shape and any pair of cells should not overlap, but a cell partitioning can contain cells with different sizes (e.g., Fig. 1). A cell partitioning does not have to cover the entire space; it should only cover the target area on which the user has interests. Moreover, we do not have to determine a cell partitioning beforehand; we can specify a partitioning *dynamically* according to the analysis requirement. Therefore, a user can specify fine cell partitioning for the region on which the user has high interests and coarse cell partitioning to other regions. Also, we can set high resolution settings to high-density regions where the traffic is heavy.

In summary, our approach allows a user to specify a user-level sampling period and cell decomposition. The features enable exploratory mobility analyses.

5.2 Formulation of the Problem

Now consider to estimate an order- n Markov transition probability shown in Eq. (2) using a spatial index R-tree. We generalize the problem from the problem of a specific combination of cells c_0, \dots, c_n to the following more general one:

Definition 1 (Transition Probability Estimation Problem) Assume that we are given $n + 1$ sets of cells $C_0 = \{c_{0,1}, \dots, c_{0,|C_0|}\}, \dots, C_n = \{c_{n,1}, \dots, c_{n,|C_n|}\}$. For a combination of cells $(c_0, c_1, \dots, c_n) \in C_0 \times C_1 \times \dots \times C_n$, if $\Pr(c_n | c_0, \dots, c_{n-1})$ is not undefined, output the value. Note that C_i and C_j ($0 \leq i, j \leq n$) may have an overlap.

We say that a probability $\Pr(c_n | c_0, \dots, c_{n-1})$ is *undefined* when there are no moving objects which existed on c_0, \dots, c_{n-1} at each unit time. It corresponds to the situation when the denominator of Eq. (2) is 0.

5.3 Naïve algorithm

Consider the program estimating $\Pr(c_n | c_0, \dots, c_{n-1})$ for a specific cell combination c_0, \dots, c_n . For this purpose, we need to calculate the following two sets S and Q :

- S is a set of the objects which were in cell c_0 when $t = \tau$ ($\tau = 0, 1, \dots, T - n$), and in cell c_1 when $t = \tau + 1$, \dots and in cell c_{n-1} when $t = \tau + n - 1$:

$$S = \{o \mid \exists \tau \in \{0, 1, \dots, T - n\}, o \in \text{objs}(c_0, \tau) \wedge o \in \text{objs}(c_1, \tau + 1) \wedge \dots \wedge o \in \text{objs}(c_{n-1}, \tau + n - 1)\}, \quad (3)$$

- Q is a set of the objects which belong to S and were in cell c_n when $t = \tau + n$:

$$Q = \{o \mid o \in S \wedge o \in \text{objs}(c_n, \tau + n)\}. \quad (4)$$

The naïve algorithm is shown in Fig. 3. Assume that the underlying spatial index can support function range query(r, t), which receives a rectangle region r and time t and returns ids of the point objects contained in r at time t .

Lines 2 to 12 are the process to output $\Pr(c_n | c_0, \dots, c_{n-1})$ for a combination of cells c_0, \dots, c_{n-1} and each $c_n \in C_n$, and iterates $|C_0| \times \dots \times |C_{n-1}|$ times. Lines 3 to 10 are its body and executed $T - n + 1$ times.

Procedure `naive_estimation`

Output: A list of “defined” $\Pr(c_n | c_0, \dots, c_{n-1})$ values

1. **for each** $(c_0, \dots, c_{n-1}) \in C_0 \times \dots \times C_{n-1}$ **do**
2. $Scount := 0$; $Qcount[] := 0$
3. **for** $t := 0$ **to** $T - n$ **do**
4. **for** $i := 0$ **to** $n - 1$ **do** $oids_i := \text{range_query}(c_i, t + i)$;
5. $Scount += |\bigcap_{i=0}^{n-1} oids_i|$;
6. **for each** $c_n \in C_n$ **do**
7. $oids_n := \text{range_query}(c_n, t + n)$;
8. $Qcount[c_n] += |\bigcap_{i=0}^n oids_n|$;
9. **end**
10. **end**
11. **if** $Scount = 0$ **then break**;
12. **for each** $c_n \in C_n$ **output** $(c_0, \dots, c_n, Qcount[c_n]/Scount)$;
13. **end**

Figure 3: The naïve algorithm

Each of the iteration contains n times (line 4) and $|C_n|$ times (line 7) invocations of range queries. Therefore, the number of range query invocations of the algorithm is $|C_0| \times \dots \times |C_{n-1}| \times \{(T - n + 1) + |C_n|\}$. When $T \gg n$ and $T \gg |C_n|$ hold, the value can be approximated as $T \times |C_0| \times \dots \times |C_{n-1}|$. Since it is proportional to T , a huge number of range queries are issued for large T values.

In the following section, we describe a more efficient algorithm which utilizes the internal structure of an R-tree.

6 CSP-based algorithm

6.1 Constraints derived from Markov chain model

In this subsection, we derive constraints to compute Markov transition probabilities. The algorithm searches all the solutions which satisfy the derived constraints.

Let P_i ($i = 0, \dots, n$) be a set of time intervals in which the trajectory of a moving object o and the cell regions of cell set C_i overlap. Note that a trajectory may overlap with a cell region of $c \in C_i$ multiple times; in that case P_i contains multiple time intervals for c . Here we assume that the start time and the end time of a time interval take integer values. Next, given two time intervals p and q , we denote their overlap by $p \sqcap q$. For example, it holds that $[2, 6] \sqcap [3, 9] = [3, 6]$. And we denote the null time interval by \perp . For the overlap of a time interval set P_i and a time interval q , we can naturally extend this idea and define it as

$$P_i \sqcap q = \{p \sqcap q \mid p \in P_i, p \sqcap q \neq \perp\}. \quad (5)$$

For example, the equation $\{[1, 3], [4, 8], [10, 13]\} \sqcap [2, 6] = \{[2, 3], [4, 6]\}$ holds. Additionally, we say that the predicate $t \in P_i$ is *true* when t is contained in some of the time intervals in P_i . Last, we define $\text{shift}(P_i, j)$ as a set of time intervals which is obtained by shifting each time interval in P_i with j unit times. For example, $\text{shift}(\{[1, 3], [5, 8]\}, 1) = \{[2, 4], [6, 9]\}$.

Now consider formulas S (Eq. (3)) and Q (Eq. (4)) defined in Subsection 5.3. The following proposition holds.

Proposition 1 A moving object o with associated sets of time intervals P_i satisfies $o \in Q$ when

$$\forall i \in \{0, \dots, n\}, P_i \sqcap [i, T - n + i] \neq \emptyset \quad (6)$$

and there is an integer τ that satisfies

$$\forall i \in \{0, \dots, n\}, \tau + i \in P_i \sqcap [i, T - n + i]. \quad (7)$$

The condition above is equivalent to the following condition: there is an integer τ such that

$$\forall i \in \{0, \dots, n\}, P_i \sqcap [i, T - n + i] \neq \emptyset \wedge \tau \in \text{shift}(P_i, -i) \sqcap [0, T - n]. \quad (8)$$

The condition that o satisfies $o \in S$ is also given by replacing all n 's in Eq. (8) by $(n - 1)$'s.

We explain the meaning of Eq. (6). Let us consider the case of $i = 0$ as an example; the condition of Eq. (6) becomes $P_0 \sqcap [0, T - n] \neq \emptyset$. From its definition, P_0 is a set of time intervals in which the regions of cell set C_0 contain object o . The time interval $[0, T - n]$ is a constraint derived from the fact that object o corresponds to the 0-th state (cell set C_0) of the Markov chain. For the illustration, suppose that $P_0 = \{[T - n + 1, T - n + 1]\}$ (o is contained in C_0 only when $t = T - n + 1$). In this case, we cannot construct an $(n + 1)$ -length transition sequence (an order- n Markov chain) for o which begins at $t = T - n + 1$. The maximal t value for which an $(n + 1)$ -length transition sequence may exist is $t = T - n$. In this case, there is a possibility that an object o has moved to each of the specified cell sets C_0, C_1, \dots, C_n at $t = T - n, t = T - n + 1, \dots, t = T$, respectively. Other cases ($i = 1, \dots, n$) are treated similarly.

Eq. (7) says that we can select an integer τ such that $t = \tau + i$ is included in the time interval $P_i \sqcap [i, T - n + i]$ for each C_i ($i = 0, \dots, n$). The condition means that object o is contained in C_0, C_1, \dots, C_n at $t = \tau, t = \tau + 1, \dots, t = \tau + n$, respectively. Namely, it means that $o \in Q$.

6.2 Search algorithm for CSP solutions

6.2.1 Main routine

The main routine to search constraint solutions is shown in Fig. 4. At line 1, we assign the reference to the root node of an R-tree to each element of $(n + 1)$ -dimension array *nodes*. The role of the array is described later. Function `FC_count` called in lines 2 to 3 plays the main role in the algorithm. At line 2, the function receives the cell sets C_0, \dots, C_{n-1} and enumerates the objects that move according to the specified order- $(n - 1)$ Markov transition sequences. The result of `FC_count` is returned as an association array *Scount*. We can obtain the number of objects which moves c_0, \dots, c_{n-1} with this order as $Scount\{c_0\#\dots\#c_{n-1}\}$, where $c_0\#\dots\#c_{n-1}$ is the concatenated string of the cell numbers. At line 3, the occurrences of order- n transition sequences are enumerated into *Qcount* in a similar manner. Using these association arrays, the estimated probabilities are outputted in lines 4 to 7. The role of *max_dist*, given as the argument of `FC_count`, is described later.

Procedure `FC_estimation`($n, \text{root}, \text{level}, (C_0, \dots, C_n), \text{max_dist}$)

Input: n : the order of Markov transition
 root : the root node of the R-tree
 level : the number of levels of the R-tree
 (C_0, \dots, C_n) : a list of cell sets
 max_dist : the maximal distance that an object can move in a unit time
Output: An estimated value for each “defined” $\Pr(c_n | c_0, \dots, c_{n-1})$

1. **for** $j := 0$ **to** n **do** $\text{nodes}[j] := \text{root}$;
2. $Scount := \text{FC_count}(n - 1, \text{level}, \text{nodes}, (C_0, \dots, C_{n-1}), \text{max_dist})$;
3. $Qcount := \text{FC_count}(n, \text{level}, \text{nodes}, (C_0, \dots, C_n), \text{max_dist})$;
4. **foreach** $(c_0, \dots, c_n) \in C_0 \times \dots \times C_n$ **do**
5. **if** $Scount\{c_0\#\dots\#c_{n-1}\} > 0$ **then**
6. **output** $(c_0, \dots, c_n,$
7. $Qcount\{c_0\#\dots\#c_n\}/Scount\{c_0\#\dots\#c_{n-1}\})$;

Figure 4: The main routine

As described below, `FC_count` searches the solutions of a constraint satisfaction problem from the root toward the leaves of an R-tree. When the areas corresponding to the specified cell sets are sufficiently smaller than the entire space, we can estimate that the algorithm only accesses a

part of the R-tree. Since `FC_count` is called only twice, efficient processing can be achieved compared with the naïve algorithm.

6.2.2 Transition sequence enumeration algorithm

The algorithm `FC_count` is shown in Fig. 5. This function is an extended version of the algorithm proposed in [7] to solve a spatial constraint satisfaction problem using an R-tree. `FC_count` looks for the solutions of constraints from the root of an R-tree to the leaves using backtrack and pruning.

```

Function FC_count(n, level, nodes, (C0, ..., Cn), max_dist)
Output: count: a hash table that contains the enumerated results
1. for j := 0 to n do // set an initial solution set for each constraint
2.   child_set := ∅;
3.   foreach v ∈ nodes[j].children do
4.     if sp_overlap(Cj, v) then // v overlaps spatially with Cj
5.       child_set := child_set ∪ {v};
6.   dom[0][j] := child_set; // insert child node sets
7. end
8. i := 0; // specifies the current target constraint
9. while true do
10.  if dom[i][i].isempty then // no more next candidate
11.    if i = 0 then return count; // end of the procedure
12.    else i −; continue; // backtrack
13.  else
14.    new_val := get_next(dom[i][i]); // get next candidate
15.    inst[i].value := new_val; // assign the candidate for constraint Ci
16.    if level ≥ 1 then // set a valid time interval which inst[i] value can take
17.      inst[i].trange := t_overlap(new_val.trange, [i, T − n + i]);
18.    else inst[i].trange := new_val.trange;
19.  end
20.  if i = n then // the constraints are satisfied by the current candidates
21.    if level ≥ 1 then // the case of non-leaf nodes
22.      for k := 0 to n do refs[i] := inst[k].value;
23.      FC_count(n, level − 1, refs, {C0, ..., Cn});
24.    else // the case for leaf nodes: increment count for the solution
25.      count{cell(inst[0].value) # ... # cell(inst[n].value)}++;
26.    end
27.  else // while the intermediate of constraint satisfaction
28.    if check_forward(i, n, level, dom, inst, (Ci+1, ..., Cn), max_dist)
29.      i++; // examine the next constraint Ci+1
30. end

```

Figure 5: Transition sequence enumeration algorithm

At lines 1 to 7, an initial solution *candidate set* is set to *dom*[0][*j*] (*j* = 0, ..., *n*). If *nodes*[*j*] is a non-leaf node, a candidate set assigned consists of the child nodes of *nodes*[*j*]; otherwise a candidate set consists of the trajectory entries (*n*+1-dimensional point objects) contained in *nodes*[*j*]. Note that *dom* is a set-valued (*n*+1) × (*n*+1) array, and *dom*[*i*][*j*] holds the candidate set for *C_j* while we are examining the satisfaction of the constraints according to *C_i*.

The predicate `sp_overlap(Cj, v)` appeared in line 4 is used to judge whether *v* overlaps with any cells contained in *C_j*, and used to prune the candidates that do not satisfy the spatial constraints.

Next we explain the while loop. In the loop, an array *inst* maintains a partial solution while we are searching for a solution that satisfies the constraints. When we process the *i*-th constraint *C_i*, *inst*[0], ..., *inst*[*i* − 1] hold a partial solution.

In the loop, we first try to set *inst*[*i*] a candidate which may satisfy the *i*-th constraint *C_i*. In line 10, we examine whether *dom*[*i*][*i*] is empty or not; if it is empty, we can say that there are no candidates remained. When *i* = 0, the function terminates because there are no candidates that satisfy the entire *n* + 1 constraints (since no candidate remain for constraint *C₀*, it is impossible to satisfy the entire constraints). If *i* > 0, *i* is decremented and the while loop is

continued. This means that a backtrack occurs and we search again using other candidates to check constraint *C_{i−1}*.

If *dom*[*i*][*i*] is not empty, a new candidate is assigned to *inst*[*i*].value at line 15. When *level* ≥ 1, *new_value* is an R-tree node; otherwise it is a trajectory entry. At lines 16 to 18, we set *inst*[*i*].trange the time interval which an element assigned to *inst*[*i*] can take to satisfy the *i*-th constraint. When *level* ≥ 1, *new_value*.trange is the interval which the minimum bounding box (MBR) of an R-tree node *new_node* takes on the temporal dimension. At line 17, we take the intersection of this interval and the time interval [*i*, *T* − *n* + *i*] shown in Eq. (8) and assign it to *inst*[*i*].trange. The obtained time interval represents that the trajectory entries stored in the descendant of the R-tree node *new_value* must satisfy this temporal constraint to become solutions. When *level* = 0, we simply assign the value of *new_value* on the temporal dimension to *inst*[*i*].trange. In this case, the time interval *inst*[*i*].trange takes a 0-length period such as [5, 5].

At line 20, it is checked whether the current target is the *n*-th condition or not. When *level* ≥ 1, `FC_count` is called recursively taking the nodes bounded to *inst*[0].value, ..., *inst*[*n*].value as the arguments and decrementing *level*. When *level* = 0, the entry of association array *count* is incremented because a new solution is found. Function cell returns the id of the cell to which the given point object belongs.

If *i* < *n*, function `check_forward` called at line 28 is used to check the current partial solution *inst*[0], ..., *inst*[*i*] has a possibility to generate a solution which satisfies all the following constraints. Such an examination is called *forward checking* [7], a popular strategy for the efficient search of CSP solutions. If `check_forward` returns true, we increment *i* and go to check the next (*i* + 1)-th constraint. Otherwise, we can safely say that the current partial solution *inst*[0], ..., *inst*[*i*] does not produce a constraint satisfaction solution. In this case, we back to line 9, then perform similar process for the next candidate of *inst*[*i*].

6.2.3 Forward checking processing

Figure 6 shows function `check_forward`. Based on the partial solution *inst*[0], ..., *inst*[*i*], the function checks whether there is any solution candidates for the (*i* + 1)-th to the *n*-th constraints. If the candidates exist, the function assigns the candidate set to *dom*[*i* + 1][*j*] (*j* = *i* + 1, ..., *n*) then returns true. Otherwise it returns false.

In the loop from line 1 to 20, a candidate set *dom*[*i* + 1][*j*] is computed according to each of the *j*-th constraint (*j* = *i* + 1, ..., *n*). In this process, if *dom*[*i* + 1][*j*] = ∅ holds for some *j*, we can say that there is no solution for the current target *inst*[0], ..., *inst*[*i*], therefore the function returns false at line 19. We first initialize the candidate set *dom*[*i* + 1][*j*] then iterates on the loop from line 3 to 18 by deleting a candidate which do not satisfy the constraints.

At line 4, we consider the leaf node level. In this case, *inst*[0].value, ..., *inst*[*i*].value holds an (*i* + 1)-length trajectory of a moving object. If the object id (*inst*[0].value.id) corresponding to the trajectory which we are checking (*inst*[0].value, ..., *inst*[*i*].value) does not equal to the object id (*v*.id) of the current candidate point data (*v*), we can delete *v* from the candidate list because it does not constitute a trajectory of a moving object.

At line 8, we calculate the valid time interval for *v*; note that if *level* ≥ 1 then *v* is an R-tree node, otherwise *v* is a

Function `check_forward(i, n, level, dom, inst, (Ci+1, ..., Cn), max_dist)`

Output: if there are candidates $inst[i+1], \dots, inst[n]$ for the given $inst[0], \dots, inst[i]$ return *true*, otherwise return *false*

Note: modifies *dom* as a side effect

```

1. for j := i + 1 to n do // for each unchecked constraint
2.   dom[i + 1][j] := dom[i][j]; // initialize the candidate set
3.   foreach v ∈ dom[i + 1][j] do // for each candidate
4.     if (level = 0) and (inst[0].value.id ≠ v.id)
5.       // v is a trajectory for other object
6.       then goto line 17;
7.       // calculate the valid time interval
8.       vrange := t_overlap(v.trange, [j, T - n + j]);
9.       if vrange = ⊥ then goto 17; // vrange is empty
10.      if t_overlap(shift(vrange, -j), inst[0].trange) = ⊥
11.        then goto 17; // Does not satisfy temporal constraints
12.      if not sp_overlap(Cj, v) then goto 17;
13.        // v does not overlap with the area of cell set Cj
14.      if sp_dist(inst[i].value, v) > max_dist × (j - i) then goto 17;
15.        // we cannot move from inst[i].value to v within j - i unit times
16.      continue; // since v satisfies the condition, go to the check of next v
17.      dom[i + 1][j] := dom[i + 1][j] - {v}; // delete v from the candidates
18.    end
19.  if dom[i + 1][j] = ∅ then return false; // no remaining candidates of solution
20. end
21. return true;

```

Figure 6: Forward checking function

$(d+1)$ -dimensional point constituting a trajectory. At line 10, we examine an integer value τ defined in Eq. (8) can actually exist or not. At line 12, it is checked whether cell set C_j and v overlap or not. Finally, at line 14, the spatial distance between the candidate for the i -th constraint $inst[i].value$ (if $level \geq 1$ then it is an R-tree node and if $level = 0$ then a point object) and v using function `sp_dist`. Note that when we compute a spatial distance between R-tree nodes, we use the minimum distance between their MBRs. If the computed distance is larger than $max_dist \times (j - i)$, it is clear that an object cannot move from the area (or point) of $inst[i].value$ to the area (or point) of v within $j - i$ unit times so that we can delete v from the candidate list.

7 Query processing example

Figure 7 shows an example of an R-tree structure constructed for the data shown in Fig. 2. MBRs 1 to 6 are on the leaf-level, and a, b, and c are the parent MBRs of nodes 1 and 2, 3 and 4, and 5 and 6, respectively. The parent node of nodes a, b, and c is the root node. As shown in the figure, the regions of cell c_1 and c_2 are $[1, 3)$ and $[3, 6)$, respectively.

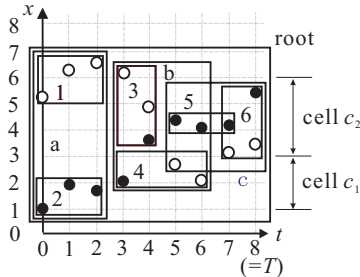


Figure 7: Example of an R-tree

Suppose that $C_0 = \{c_1\}$, $C_1 = \{c_1, c_2\}$, and $C_2 = \{c_2\}$ and we execute `FC_estimation(2, root, 2, (C0, C1, C2), 2.5)`. Namely, we let the order of Markov chains to be estimated be $n = 2$, the number of levels be $level = 2$, and the maximal distance that a moving object can move within a unit time be $max_dist = 2.5$.

Consider that `FC_count` is called in `FC_estimation` (Fig. 4). At lines 1 to 7 in `FC_count` (Fig. 5), we get $dom[0][0] = dom[0][1] = dom[0][2] = \{a, b, c\}$. After entering the while

loop with $i = 0$, we set $inst[0].value = a$, $inst[0].trange = [0, 2]$, $dom[0][0] = \{b, c\}$ at lines 14 to 18. Then we call `check_forward` at line 28. In `check_forward` (Fig. 6), we first process the case of $j = i + 1 = 1$. At line 2, the candidates are initialized as $dom[1][1] = dom[0][1] = \{a, b, c\}$. Since $v = a$ and $v = b$ satisfy the following all conditions, they are not removed from $dom[1][1]$ in the process. However, when $v = c$, we get $vrange = [5, 8]$ at line 8 and since $shift([5, 8], -1) \cap [0, 2] = \perp$, the candidate $v = c$ is removed at line 10. Therefore, we get $dom[1][1] = \{a, b\}$. For $j = 2$, we get $dom[1][2] = \{a, b\}$ in a similar manner. Namely, even if a trajectory point object which satisfies the 0-th constraint is inside of $inst[0].value = a$, its corresponding trajectory point objects which satisfy the first and the second constraints are not contained under node c . Finally, `check_forward` returns true.

After returning to `FC_count` (Fig. 5), we increment i at line 29 since `check_forward` was true then continue the while loop. After lines 14 to 18, we next get $inst[1].value = a$, $inst[1].trange = [1, 2]$, $dom[1][1] = \{b\}$. Since `check_forward` returns true, we get $dom[2][2] = \{a, b\}$. Then we return to `FC_count` again and increment i . In the next while loop, `FC_count(2, 1, refs, {C0, C1, C2})` is called recursively at line 23, where $refs[0] = a$, $refs[1] = a$, $refs[2] = a$.

When `FC_count` is called recursively, we set $dom[0][0] = \{2\}$, $dom[0][1] = \{1, 2\}$, $dom[0][2] = \{1\}$ by considering the constraints C_0, C_1, C_2 then perform similar process. First, `check_forward` is called by setting $inst[0].value = 2$ then it returns true, and we get $dom[1][1] = \{2\}$, $dom[1][2] = \{1\}$. The reason of deletion of node 1 from $dom[1][1]$ is that a moving object cannot move from node 2 ($inst[0].value$) to node 1 within a unit time ($sp_dist(2, 1) > max_dist$). Therefore, we next call `FC_count(2, 0, refs, {C0, C1, C2})` recursively, where $refs[0] = 2$, $refs[1] = 2$, $refs[2] = 1$. Although the detail is omitted, we cannot obtain a constraint satisfaction solution in this case (there is no object which was in cell 2 at $t = \tau$, in cell 2 at $t = \tau + 1$, and in cell 1 at $t = \tau + 2$). Therefore, a backtrack occurs finally and the process returns to level 1.

Next the algorithm searches for the case of $inst[0].value = a$, $inst[1].value = a$, $inst[2].value = b$ and fails again then performs a backtrack. Next another fail occurs for $inst[0].value = a$, $inst[1].value = b$, $inst[2].value = a$ then we try the case of $inst[0].value = a$, $inst[1].value = b$, $inst[2].value = b$. In this case, a constraint solution $inst[0].value = (id = A, t = 2, x = 1.8)$, $inst[1].value = (id = A, t = 3, x = 2.2)$, $inst[2].value = (id = A, t = 4, x = 3.6)$ is obtained. As a result, $count(c_1 \# c_1 \# c_2)$ is incremented. By repeating the above process, we enumerate all the constraint satisfaction solutions.

Finally note that the result of `FC_estimation` in this example produces $\Pr(c_2|c_1, c_1) = 2/4 = 0.5$ and $\Pr(c_2|c_1, c_2) = 2/2 = 1.0$.

8 Experimental results

In this section, we describe the experiments using a dataset generated by a moving object simulator and the implemented algorithms on an R-tree package. We compare the performance of the naive algorithm and the proposed CSP-based

algorithm.

8.1 Dataset generation

In the experiments, we use a moving object simulator developed by Brinkoff [1]. The system simulates the situation when moving objects (i.e., cars) move on an actual city road network. The dataset used in the experiments is generated from the road network of the center part (2.5 km \times 2.8 km) of German Oldenburg city which is offered by the system. Figure 8 shows the simulated area where objects move.



Figure 8: The simulation area

In the moving object trajectory generation, the number of initial moving objects is set to five and five moving objects are generated on every minute randomly on the map. Each object has its destination and when an object arrives at the destination, the object disappears from the map. Also, when a moving object goes outside of the map, it is deleted from the consideration. As a result, nearly 100 moving objects are on the map on average. For the experiments, the trajectory data is generated for the period of $T = 1,000$ minutes setting the unit time interval as one minute. Finally, 124,752 tuples with the form of (object id, time, x -axis value, y -axis value) are generated. These tuples are registered in an R-tree of three dimensions.

8.2 Performance comparison and analysis

In this subsection, we show the experimental results performed using the dataset described above. In the experiments, we utilize a PC with Pentium III (500MHz) with 128 MB main memory and the Linux operating system. Each experiment starts from a “cold” buffer state.

In the first experiment, the map shown in Fig. 8 is decomposed into 30×30 cells. Then we select a 3×3 cell region C which consists of 9 cells then compute first-order Markov transition probabilities $\{\Pr(c_1|c_0) \mid c_0 \in C, c_1 \in C\}$. Namely, we compute probabilities for all the combination ($9^2 = 81$) of cells in C . As described in Subsection 8.1, the dataset was generated by simulating while $T = 1,000$ unit times, but we have also constructed its subsets ($T = 100, 200, \dots, 900$) to examine the scalability of the algorithms. As shown in Fig. 9, the CSP-based algorithm (CSP) performs well compared to the naive algorithm (naive) in this case. Both algorithms have almost linear behaviors according to T , the size of the dataset.

Figure 10 shows the result of an experiment which obeys the setting of Fig. 9, except for the order of the Markov chain is two; namely, we estimate all the $9^3 = 729$ transition probabilities $\{\Pr(c_2|c_0, c_1) \mid c_0 \in C, c_1 \in C, c_2 \in C\}$. Since the number of transition probabilities to be estimated have

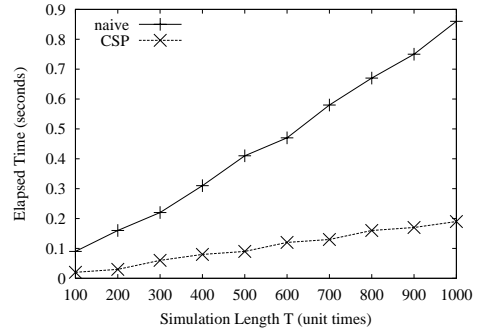


Figure 9: Estimation of probabilities ($n = 1, 3 \times 3/30 \times 30$)

increased nine times, we can see that the total computation times also have increased almost nine times, but the overall behaviors shown in Fig. 10 is quite similar with Fig. 9.

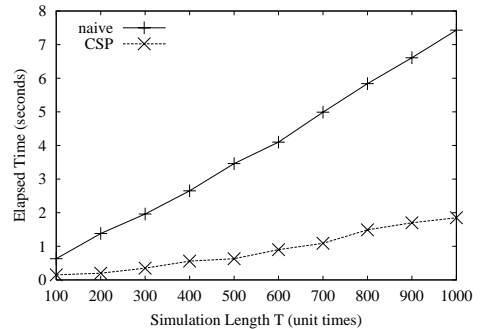


Figure 10: Estimation of probabilities ($n = 2, 3 \times 3/30 \times 30$)

Fig. 11 also shows the case of the third-order Markov transition probabilities $\{\Pr(c_3|c_0, c_1, c_2) \mid c_0 \in C, c_1 \in C, c_2 \in C, c_3 \in C\}$. It shows quite similar tendencies with Figs. 9 and 10.

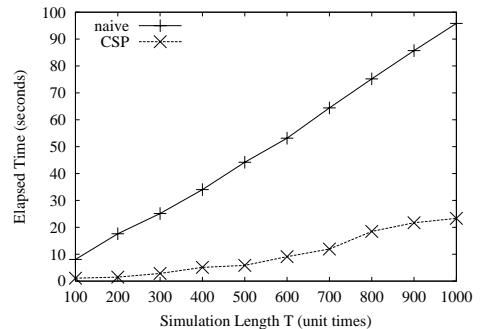


Figure 11: Estimation of probabilities ($n = 3, 3 \times 3/30 \times 30$)

In the experiments shown above, the CSP-based algorithm highly well performed than the naive algorithm. As shown below, however, this tendency does not always hold. Figure 12 shows the result of an experiment which has the same setting with Fig. 10 except for the cell decomposition; in this case, we utilize the setting of a coarser 20×20 decomposition. The CSP-based algorithm is still better than the naive algorithm, but their difference is relatively small. The main reason is that the areas of the target cells are increased due to the coarse 20×20 decomposition. Although the region used for the estimation consists of only 3×3 cells, since the MBRs of the R-tree non-leaf/leaf nodes overlap each other, the actual search region almost includes the entire map region. Therefore, the pruning used in the CSP-based algorithm cannot re-

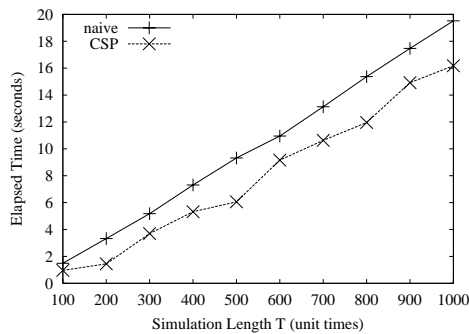


Figure 12: Estimation of probabilities ($n = 2, 3 \times 3/20 \times 20$)

duce the number of intermediate solutions effectively until we reach the leaf level of the R-tree.

We can consider two additional factors concerning the R-tree implementation. First, conventional R-tree implementation aims to optimize the processing of a single point/range query; the internal implementation (e.g., a buffering module) is tuned according to this type of queries. The buffering module of the R-tree program used in the experiments did not well perform for the CSP-based algorithm which has a different page access pattern. If we use a more sophisticated page buffering scheme, we would be able to improve the performance of the CSP-based algorithm. Second, the R-tree implementation that we used in the experiments did not use an optimized R-tree construction method. If we use a more optimized R-tree construction method such as packed R-trees [10] which has less MBR overlaps than the conventional R-trees, we would be able to reduce the pruning cost.

Based on the above experiments and other experiments omitted here, we can observe as follows:

- As the increase of the dataset size T , the costs of two algorithms increase almost linearly.
- In both algorithms, even if we use different settings of orders of Markov chains, the computation time of one transition probability is almost constant when other parameters are fixed.
- When we use small cell decompositions (e.g., 40×40), the CSP-based algorithm performs quite well than the naïve one. On the other hand, the relative performance of the naïve algorithm is improved when we use coarse cell decompositions.

Based on the above observations, we can say that the CSP-based algorithm is best suited to the estimation of transition probabilities for relatively small “focused” regions. Such an analysis often occurs in an interactive mobility analysis that requires the system to focus on a specific region and demands a quick response. Moreover, the CSP-based approach would also perform better when the entire map region is relatively large. In this case, an actual mobility analysis should often focus on some specific regions so that the CSP-based algorithm would work well.

9 Conclusions and future work

In this paper, we have proposed an approach to extract the mobility statistics from an indexed spatio-temporal database. The mobility statistics is formulated based on the Markov chain model. We have proposed two algorithms. The naïve

algorithm is derived straightforwardly from the definition of the Markov chain model. The CSP-based algorithm uses the internal structure of a spatial index R-tree and enumerates the target items in an efficient manner. For this purpose, we have extended an algorithm to solve a spatial constraint satisfaction problem with an R-tree. We have compared two algorithms using a trajectory dataset generated from a moving object simulator and made the performance comparisons of two algorithms. Based on the experimental results, we can say that the CSP-algorithm is well suited to the interactive mobility analyses which focus on specific regions and issue pin-point estimation queries.

The work presented here is currently ongoing in our research group. The future work includes 1) improvement of the buffer maintenance algorithms, 2) an adaptive decomposition of spatial cells based on the density of moving objects, and 3) an extension of the work to the non-stationary Markov chain model.

Acknowledgments

This research is partly supported by the Grant-in-Aid for Scientific Research (16500048) from Japan Society for the Promotion of Science (JSPS), Japan and the 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 Asahi Glass Foundation and the Inamori Foundation.

References

- [1] T. Brinkhoff, A Framework for Generating Network Based Moving Objects, *Geoinformatica*, 6(2), pp. 153-180, 2002.
- [2] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, Efficient Processing of Spatial Joins Using R-trees, *Proc. ACM SIGMOD*, 1993.
- [3] Y. Choi and C. Chung, Selectivity Estimation for Spatio-Temporal Queries to Moving Objects, *Proc. of ACM SIGMOD*, pp. 440-451, 2002.
- [4] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, 2001.
- [5] C.S. Jensen (ed.), Special Issue: Indexing of Moving Objects, *IEEE Data Engineering Bulletin*, 25(2), Jun. 2002.
- [6] M.A. Nascimento, J.R.O. Silva, and Y. Theodoridis, Evaluation of Access Structures for Discretely Moving Points, *Proc. STDBM*, pp. 171-188, 1999.
- [7] D. Papadias, N. Mamoulis, and Vasilis Delis, Algorithms for Querying by Spatial Structure, *Proc. of VLDB*, 1998.
- [8] W.-C. Peng and M.-S. Chen, Developing Data Allocation Schemes by Incremental Mining of User Movement Patterns in a Mobile Computing System, *IEEE TKDE*, 15(1), pp. 70-85, 2003.
- [9] D. Pfoser, C.S. Jensen, and Y. Theodoridis, Novel Approaches to the Indexing of Moving Object Trajectories, *Proc. of VLDB*, 2000.
- [10] N. Roussopoulos and D. Leifker, Direct Spatial Search on Pictorial Databases using Packed R-trees, *Proc. ACM SIGMOD*, 1985.
- [11] Y. Tao, J. Sun, and D. Papadias, Selectivity Estimation for Predictive Spatio-Temporal Queries, *Proc. of ICDE*, 2003.
- [12] I. Tsoukatos and D. Gunopulos, Efficient Mining of Spatiotemporal Patterns, *Proc. of SSTD*, pp. 425-442, 2001.
- [13] G.J.G. Upton and B. Fingleton, *Spatial Data Analysis by Example, Volume II: Categorical and Directional Data*, John Wiley & Sons, 1989.