Proceedings

2ND INTERNATIONAL WORKSHOP ON SPATIO-TEMPORAL DATA MANAGEMENT

Toronto, Canada, August 30, 2004

Workshop Co-Chairs:

Jörg Sander
Mario A. Nascimento

Advisory Committee:

Ralf Hartmut Güting
Christian S. Jensen
Yannis Manolopoulos

Co-located with the 30th International Conference on Very Large Data Bases

Organization

Organizers and PC Co-Chairs

Jörg Sander, University of Alberta, Canada
Mario A. Nascimento, University of Alberta, Canada

Advisory Committee

Ralf Hartmut Güting, Fernuniversität Hagen, Germany
Christian S. Jensen, Aalborg University, Denmark
Yannis Manolopoulos, Aristotle University, Greece

Program Committee

Dave Abel, CSIRO, Australia
Divyakant Agrawal, UC Santa Barbara, USA
Walid Aref, Purdue University, USA
Lars Arge, Duke University, USA
Elisa Bertino, Purdue University, USA
Edward P.F. Chan, University of Waterloo, Canada
Jan Chomicki, University at Buffalo, USA
Max Egenhofer, University of Maine, USA
Andrew U. Frank, Technical Univ. Vienna, Austria
Fabio Grandi, University of Bologna, Italy
Oliver Günther, Humboldt Univ. Berlin, Germany
Ralf Hartmut Güting, Fernuniversität Hagen, Germany
Erik Hoel, ESRI, USA
Kathleen Hornsby, University of Maine, USA
Christian S. Jensen, Aalborg Univ., Denmark
Christopher B. Jones, Cardiff University, UK
Ravikanth V. Kothuri, Oracle Corporation, USA
Manolis Koubarakis, Technical University of Crete, Greece
Yannis Manolopoulos, Aristotle University, Greece
Enrico Nardelli, Universita' di Roma "Tor Vergata", Italy
Dimitris Papadias, UST Hong Kong
Jignesh M. Patel , University of Michigan, USA
Dieter Pfoser, CTI, Greece
Philippe Rigaux, Univ. Paris Sud, France
John Roddick, Flinders University, Australia
Bernhard Seeger, University of Marburg, Germany
Timos Sellis, NTUA, Greece
Richard Snodgrass, University of Arizona, USA
Stefano Spaccapietra, EPFL, Switzerland
Jianwen Su, UC Santa Barbara, USA
Yannis Theodoridis, University of Piraeus, Greece

Tzouramanis Theodoros, University of the Aegean, Greece
Babis Theodoulidis, University of Manchesters, UK
Nectaria Tryfona, CTI, Greece
Vassilis Tsotras, UC Riverside, USA
Can Türker, ETH Zurich, Switzerland
Jeffrey S. Vitter, Purdue University, USA
Agnès Voisard, Fraunhofer ISST and FU Berlin, Germany
Peter Widmayer, ETH Zurich, Switzerland
Jef Wijsen, University of Mons-Hainaut, Belgium
Michael Worboys, University of Maine, USA

External Reviewers

Marios Hadjieleftheriou, UC Riverside, USA
Xuegang Huang, Aalborg University, Denmark
Bin Lin, Univ. California Santa Barbara, USA
Michael Mlivoncic, ETH Zurich, Switzerland
Hoda Mohktar, Univ. California Santa Barbara, USA
Mohamed Mokbel, Purdue University, USA
Rahul Shah, Purdue University, USA
Spiros Skiadopoulos, NTUA, Greece
Xiapeng Xiong, Purdue University, USA

Preface

This volume contains papers presented at the one-day Second Workshop on Spatio-Temporal Database Management, STDBM'04, co-located with the 30th International Conference on Very Large Data Bases, VLDB 2004.

Managing spatially and temporally referenced data is becoming increasingly important, given the continuing advances in wireless communications, ubiquitous computing technologies and the availability of real datasets to be managed. The goal of this workshop is to bring together leading researchers and developers in the area of spatio-temporal databases in order to discuss and exchange state novel research ideas and experiences with real world spatio-temporal databases.

The workshop received 18 submissions, from 9 different countries in Europe, Asia, South and North America. All papers were evaluated by at least three of the forty-one members of the Program Committee, and, at the end, 10 papers were accepted and are included in these Proceedings.

We wish to thank Philippe Rigaux for allowing us to use the MyReview conference management system and Alex Coman for helping us to manage it locally. We also express our appreciation to the members of the Workshop's steering committee for their guidance and advice throughout the whole process, from proposing the workshop to the final technical program. As well, we acknowledge the support provided by the VLDB organization.

Jörg Sander and Mario Nascimento
STDBM'04 Organizers and PC co-chairs
Edmonton, Canada, July 2004

Contents

Mobility Patterns

Cédric du Mouza
lab. CEDRIC
CNAM - Paris
FRANCE
dumouza@cnam.fr

Philippe Rigaux
LRI
Orsay - Paris Sud
FRANCE
rigaux@lri.fr

Abstract

We present a data model for tracking mobile objects and reporting the result of continuous queries. The model relies on a *discrete* view of the spatio-temporal space, where the 2D space and the time axis are respectively partitioned in a finite set of user-defined areas and in constant-size intervals. We define a query language to retrieve objects that match *mobility patterns* describing a sequence of moves and discuss evaluation techniques to maintain incrementally the result of queries.

1 Introduction

In the database community, several data models have been proposed to enable novel querying facilities over collections of moving objects. A common feature of most of these models is the strong focus on the *geometric* properties of trajectories. Indeed, in most cases, the data representation and the query language are considered as extensions of some existing data model previously designed for (and limited to) 2D geometric data handling. The modeling of moving objects has been therefore strongly influenced by the existing spatial models, and relies usually on a set of data structures providing support for geometric operations (e.g., geometric intersection) [21, 10, 8, 9].

An assumption commonly adopted by all the above mentioned models is to consider a *dense* embedding space and to model trajectories as *continuous* functions in this space. While this property allows several suitable computations (for instance the position of an object can be obtained at any instant), it is not well adapted to some common requests. Let us consider the tracking of objects with *continuous queries*, i.e., queries whose result must be maintained during a given (and possibly unbounded) period of time. When asking, for instance, for all the objects that belong to a given rectangle R during the next 3 days, the initial result is subject to vary by considering the objects that leave of enter R. Managing *incrementally* the evolutions of the result (i.e., without recomputing periodically the entire result) is a hard task with a geometric-based query language because the dense-space assumption of the data model often contradicts with the discrete nature of the observation.

In the present work we investigate an alternative approach, namely the management of continuous queries as a discrete process relying on *events* related to the moves of objects over the underlying space. Intuitive examples of events are, for instance, an object *enters* a zone, an object *stays* in a zone, and an object *leaves* a zone. A query in such a setting is a sequence of primitive events which can be specified either by explicitely referring to the zones of interest ("Give all the objects currently in a which arrived 5 minutes ago, coming from b"), or by more generic *patterns* of mobility such as, for instance, "Give these objects that moved from a to another zone and came back to a".

We propose in the current paper a data model for representing trajectories as sequences of moves in a discrete spatio-temporal space, and study the languages to query such sequences of events. Essentially, the languages that we consider allow to construct expressions, or *mobility patterns*, to express search operations. We focus specifically on the family of patterns that satisfy the following properties (i) we do not need the past moves of an object o to determine whether o matches or not a given pattern and (ii) the amount of memory required to maintain a query result is small. These properties are essential in the context of continuous queries since they guarantee that a large amount of queries can be evaluated efficiently with limited resources by just considering the last event associated to an object. We define a class of queries which provides an appropriate balance between expressiveness and the fulfillment of these requirements.

Related work

Expressing sequences of moves as proposed in the present paper is close in spirit to the area of sequence databases [20, 15, 18, 22]. The SQL-TS language of [18] and [19] allows to express sequences of conditions and describes an effi-

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

cient algorithm for query evaluation. The idea of representing temporal sequences as strings and to rely on pattern-matching algorithms is also present in [6] and [5]. In [17] sequences are considered as sorted relations, and each tuple gets a number that represents its position in the sequence. All these approaches are significantly different from ours. In particular there is nothing similar to the concept of mobility pattern, featuring variables, proposed in our data model.

The notion of continuous queries, described as queries that are issued once and run continuously, is first proposed in [24]. The approach considers append-only databases and relies on an incremental evaluation on delta relations. Availability of massive amounts of data on the Internet has considerably increased the interest in systems providing event notification across the network. Some representative works are the *Active Views* system [1], the NiagaraCQ system [4], and the prototypes described in [14, 7]. In the area of spatio-temporal databases, the problem is explicitely addressed in several works [16, 3, 13, 23, 11, 25]. [3] for instance describes a web-based architecture for reducing the volume and frequency of data transmissions between the client and the server. [13] presents a system that indexes queries in order to recompute periodically the whole result of each query. This is in contrast with the incremental computation advocated in the current paper.

In the rest of this paper we first develop an informal presentation of our work (Section 2) with examples of *mobility patterns* that illustrate the intuition behind the model and its practical interest from the user's point of view. The data model is presented in Section 3. Finally Section 4 concludes the paper and discusses future work. A long version is available at *http://www.lri.fr/~rigaux/DOC/MR04b.pdf*.

2 Introduction to mobility patterns

Figure 1 shows a map partitioned in several zones identified with simple labels (`a, b, c, ...`). Over this map we consider a set of mobile objects, each of them coupled with a localization device which periodically provides their position. The minimal period between two events related to the same object defines the *time unit*. For the sake of concreteness we shall assume in the following that objects are tracked by a GPS system giving the location of an object, and that the time unit is 1 (one) minute.

Consider now a traffic monitoring application supporting tracking of the mobile objects, and the following queries:

1. Give all the objects that traveled from `a` to `f`, stayed more than 10 minutes in `f` and then traveled from `f` to `c`.

2. Give all the objects traveling from `f` to `d` or `c` through another, third, zone of the map.

3. Give all the objects that left a given zone, went to `c` and came back to the first zone.

The common feature of these examples is a specification of the successive zones an object belongs to during its travel, along with temporal constraints. We call *mobility pattern* this specification. The geometric-based approach used in most of the spatio-temporal data models so far is not really adapted for expressing queries based on mobility patterns. Actually we do no longer need an interpolation or extrapolation mechanism to infer the position of an object at each instant since the discrete succession of events provided by the GPS server is naturally suitable to serve as a support for evaluating these patterns.

Each GPS event provides the position of an object, and this suffices to compute the zone where the object resides when the event is received. It is therefore quite easy to construct a discrete representation of the trajectory of an object as a sequence of the form $l_1\{T_1\}.l_2\{T_2\}.\cdots.l_n\{T_n\}$ featuring the list $l_1, l_2, \ldots, l_n$ of successive zone labels as well as the time spent in each zone. For instance the trajectory of o_1 in Figure 1, assuming that o_1 spent 2 minutes in `f`, 4 minutes in `a`, 3 minutes in `d` and 6 minutes in `c`, will be represented in our model as a sequence `[f{2}.a{4}.d{3}.c{6}]`. Note that each new event either increments the time component of the last label if the object remains in the same zone, or appends a new label to the trajectory's representation.

Let us now turn to mobility patterns. Basically, they constitute a specific kind of regular expression, featuring variables which can be instantiated to any of the labels of the map. As a first example, assume that we want to retrieve all the objects that started from `a` or `b`, moved to `e`, crossing one of the zones `c`, `d`, or `f` (see Figure 1), and finally went back from `e` to `a` via the *same* zone. This class of trajectories is represented by the following mobility pattern:

$$(a|b).@x^{+}.e^{+}.@x^{+}.a$$

In a pattern, a zone is represented either by its label (here `a`, `b`, `e`) or by a variable (here `@x`) if it is left undetermined by the user. A variable is here necessary to represent the zone where an object moved after leaving `a` or `b`, and expressing that the object must come back to `a` via the *same* zone. Each occurrence of a variable in a pattern must be instantiated to the same value. Labels or variables can be concatenated (for instance `@x.a` in our example) to describe a path, or grouped in sets (for instance `(a|b)`) to describe a union of zones. The "+" operator expresses the fact that the object can stay an undetermined time in a given zone, but at least one time unit. Alternatively, one can associate to each label simple temporal constraints of the form {`min, max`} where `min` and `max` denote respectively the minimal and maximal number of time units spent in the zone.

Intuitively, an object o *matches* a pattern P if the following conditions hold:

1. one can find a word in the language $\mathcal{L}(P)$ which is equal to a suffix of the trajectory of o, *modulo* an assignment of the variables in P.

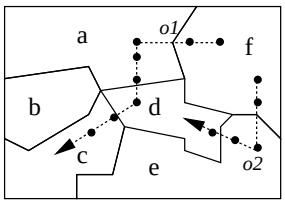


Figure 1: Objects moving over a partitioned map

2. the time spent in each zone complies with the temporal constraint expressed in the pattern.

For instance an object whose trajectory is represented by the sequence [f.d.c.b.**a.d.e.d.a**] (we omit the temporal information for simplicity) matches the pattern above where the value of the variable @x is set to the label d. The suffix in boldface is then a word in the language denoted by the pattern.

The suffix represents here the most recent part of the trajectory received from the continuous stream of GPS events. It determines whether an object belongs or not to the result set of a query. Note also that, since the trajectory representation evolves as new events are received, the matching must be evaluated periodically – almost continuously. Our goal is to perform this evaluation with minimal space and time consumption.

Patterns can easily be introduced in a SQL-like query language, as illustrated by the following examples which will be used throughout the rest of the paper. The syntax of regular expression is that of the Perl language [26].

- **Q1.** Give all the objects that traveled from a to f, stayed at least 2 minutes in f and then traveled from f to c.

```
SELECT *
FROM Mob
WHERE matches(traj,'a.f{2,}.c')
```

  The *matches* function checks whether a suffix of the spatio-temporal attribute *traj* matches the mobility pattern a.f.c. An additional temporal constraint states that the object must spend at least 2 time units (e.g., 2 minutes) in f.

- **Q2.** Give all the objects that stay in a or b all the time except for one minute when they were in another, third, zone.

```
SELECT *
FROM Mob
WHERE matches (traj,'(a|b)+.@x.(a|b)+')
AND    @x != 'a' AND @x != 'b'
```

  This example requires a variable @x which expresses a move not assigned to a specific label but instantiated to the choice of a moving object when it leaves a or b. It is possible to express additional constraints on the instantiations allowed for a variable, using equalities or inequalities. The user requires in this example the object to leave a or b for a third, distinct, area.

- **Q3.** Give all the objects that went through f to another zone then went to d or c, and came back to f using the *same* zone.

```
SELECT *
FROM   Mob
WHERE  matches (traj,'f.@x+.(d|c)+.@x+.f')
AND    @x != 'f'
```

Let us turn now to the query evaluation process, and in particular to the *continuous* evaluation which maintains a result by adding or removing objects. We consider two essential criteria for measuring the easiness and efficiency of this evaluation:

1. Do we need to consider the past moves of an object to evaluate a query?
2. What is the amount of memory required to maintain a query result?

Consider first the case of patterns without variable. Evaluating a pattern P is then a standard operation which simply requires to build the Finite State Automata (FA) that recognizes the language $\Sigma^*.L_P$, where L_P is the regular language denoted by P and Σ is the set of labels of the map.

In the general case, the FA associated to a regular expression is non-deterministic. Then an object o might be associated to several states at a given time instant, and we must record the list of current states for o. This list can be represented as a mask of bits, one bit for each state of the FA. The value 1 (resp. 0) for a bit means that o is (resp. is not) in the associated state. This gives a rather compact structure: for a pattern with 8 symbols, a mask of 8 bits (one byte) must be recorded for each object. One can track a database of one million objects with only one megabyte in main memory.

The pseudo-code of the procedure *HandleEvent(q, id, x, y)* summarizes how to actualize the result of a query q when a GPS event is received, giving a new location (x, y) for the object o. The reference map is a set of zones denoted by M.

HandleEvent (q, o, x, y)
begin
  // Compute the current zone, z
  z = *PointInPolygon(M, x, y)*
  // Get the label of z
  l = *label(z)*
  // For each bit set to 1 in the status of o,
  // compute the transition l

for each bit i with value 1 in $status_o$
Compute $s_j = \delta(FA_q, s_i, l)$
Set the bit j to 1 and the bit i to 0 in $status_o$
end for
end

The result set of q can then be updated according to the new status of object o. Essentially, if at least one of the new states is an accepting one, o will be *in* the result set, else it will be *out* of this result set. In this simple case we obtain a direct answer to the two questions above:

1. It is *not* required to maintain historical information on a trajectory, since, it suffices to know the current state(s) of the FA, reached by taking account of the events received so far.

2. The space required to maintain a query result is, in the worst case, the set of all states in the FA (which might be non-deterministic) and is therefore proportional to the size of the query[1].

If we consider now patterns with variables, the language is much more expressive, but some care is required for executing queries. Take for instance the example Q3 above. Each time an object leaves the zone `f` for another one, a new label is bound to the variable `@x`. One must then store this value in order to check for the consistency of any further occurrence of `@x`.

The next section is devoted to the data model, and focuses on the evaluation of queries with variables. We show that we can still avoid to rely on historical information on trajectories, and study more specifically the memory requirements for several classes of queries.

3 The model

We consider an embedding space partitioned in a finite set of zones, each zone being uniquely labeled with a symbol from a finite alphabet Σ. The time axis is divided in constant size units. For concreteness we still assume in the following that the time unit is 1 minute. We also assume a set $\mathcal{V}$ of variables with $\Sigma \cap \mathcal{V} = \emptyset$ and denote as Γ the union $\Sigma \cup \mathcal{V}$. In the following, letters `a`, `b`, `c`, ... will denote symbols from Σ, and `@x`, `@y`, `@z`, ... variables. We assume the reader familiar with the basic notions of regular expressions and regular languages, as found in [12].

3.1 Data representation and query language

We adopt a standard extended relational framework for the database, with $\mathcal{O}$ denoting the relation of moving objects, and $o.traj$ the trajectory of an object o. The representation of trajectories is then defined as follows:

Definition 1 (Representation of trajectories) *A* trajectory *is represented by an expression of the form*

$$s_1\{T_1\}.s_2\{T_2\}.\cdots.s_n\{T_n\}$$

where $s_i, i = 1, \ldots, n$ are symbols from Σ and T_i represents the number of time units spent in the zone s_i.

Hereafter, we shall use the term "trajectory" to mean its representation. For convenience, we shall often omit the temporal components and use a simplified representation of a trajectory as a word $[s_1.s_2.\cdots.s_n]$ in Σ^*.

A natural choice is to build mobility patterns as regular expressions on $\Gamma = \Sigma \cup \mathcal{V}$, and to search for the suffix of trajectories that match the expression for some value of the variables. Consider for example the regular expression $E =$`a.@x+.b+.@x`. The trajectory $t =$`f.d.a.c.b.c` matches E because we can find a word $w =$ `a.@x.b.@x` in the language denoted by E (w is called a *witness* in the following) and an instantiation $\nu : \{$`@x` $:= c\}$ such that $\nu(w)$ is a suffix of t. However this approach raises some ambiguities regarding the role of variables. Consider the following examples:

1. Let E be the regular expression `b.(a|@x)+.c`. Then the trajectory `b.a.c` has two witnesses in the regular language denoted by E: `b.@x.c` and `b.a.c`. In the first case `@x` must be instantiated to `a`, but in the second case any value of `@x` is acceptable.

2. Let E be the regular expression `a.(@x|@y).b.(@x|@y)`. The variables `@x` and `@y` can be used interchangeably, which makes the role of variables undetermined.

As shown by the previous examples, if we build mobility patterns with unrestricted regular expressions over Γ, the assignment of variables is non deterministic, and sometimes meaningless. For safety reasons, when reading a word w and checking whether w matches a mobility pattern P, we require each variable in P to be explicitely bound to one of the symbols in w. We thus adopt a more rigorous definition of the language by considering only *unambiguous* regular expressions on Γ such that each variable always plays a role in the evaluation of the query. We need first to introduce *marked* regular expressions.

Definition 2 (Marked expressions [2]) *Let E be a regular expression over the alphabet Γ. We define the* marking *of E as the regular expression E' where each symbol of Γ is marked by a subscript over $\mathbb{N}$, representing the position of the symbol in the expression.*

For instance the marking of the regular expression $\mathtt{a}^*\mathtt{.@x.((b.a)|(c.b)).c.@x}^*\mathtt{.a}$ is the expression $\mathtt{a}_1^*\mathtt{.x}_2\mathtt{.((b}_3\mathtt{.a}_4\mathtt{)|(c}_5\mathtt{.b}_6\mathtt{)).c}_7\mathtt{.@x}_8\mathtt{.a}_9$. We can now define *mobility patterns* as the class of regular expressions that satisfy the following property:

Definition 3 (Mobility patterns) *A* mobility pattern *is a regular expression P over Γ such that each variable of P' appears in each word of the language $\mathcal{L}(P')$.*

[1] It is possible, for any regular expression E, to construct a FA whose number of states is equal to the number of symbols in E.

This property ensures that each variable in any pattern is always assigned to a relevant label during query evaluation. The expression P = `(a|b)+.@x.(a|b)+` is for instance a mobility pattern because `@x` appears in all the words of the language $\mathcal{L}(P)$. Any successful matching of P with a trajectory `t` results therefore in an assignment of `@x` to one of the symbols of `t`. It can be tested whether a regular expression matches the required condition, and thus can be used as a mobility pattern.

Proposition 1 *There exists an algorithm to check whether a regular expression is a mobility pattern.*

In the following we shall denote as $var(P)$ the set of variables in a pattern P. The query language and its semantics are now defined as follows.

Definition 4 (Syntax of queries) *A* query *is a pair* $(P, \mathcal{C})$ *where* P *is a mobility pattern and* $\mathcal{C}$ *is a set of constraints of the form* $s_1 \neq s_2$*, for* $s_1, s_2 \in \Sigma \cup var(P)$

Let $q = (L, \{C_1, \ldots C_l\})$ be a query. The answer to of q over $\mathcal{O}$, denoted $ans(q)$, is a subset of $\mathcal{O}$ defined as follows:

Definition 5 (Semantics of queries) *An object* $o \in \mathcal{O}$ *belongs to* $ans(q)$ *if there exists a mapping* $\nu : \mathcal{V} \rightarrow \Sigma$*, called a* valuation*, with the following properties:*

1. ν *satisfies all the constraints* $C_i, i = 1, \ldots, l$
2. $o.traj$ *belongs to* $\Sigma^*.\mathcal{L}(\nu(P))$.

The constraints in a query can be used to forbid explicitely a variable to take a value (e.g., `@x` $\neq$ `a`). The *domain* of a variable `@x` for a given query q, denoted dom_q(`@x`), represents the set of possible values for `@x` given the constraints of q.

Example 1 *The following queries correspond to the 3 examples given in Section 2.*

1. $q_1 = (a.f\{2,\}.c, \emptyset)$
2. $q_2 = ((a|b)^+.@x.(a|b)^+, \{@x \neq a, @x \neq b\})$
3. $q_3 = (f.@x^+.(c|d)^+.@x^+.f, \{@x \neq f\})$

3.2 Query evaluation

We describe now an algorithm for evaluating a query q. First we show how to obtain an automaton which, given a mobility pattern P, accepts the trajectories that match P. This automaton also provides the valuation of variables in P. In a second step we explain how the automaton can be used at run time, and discuss the size of the memory used to store the relevant information. For simplicity, we consider the automata that accept the language $\mathcal{L}(P)$: their extension to automata that accept $\Sigma^*.\mathcal{L}(P)$ is trivial and can be found in any specialized textbook.

Since a mobility pattern P is a regular expression over the alphabet Γ, we can build a non-deterministic finite state automaton (NFA) N_Γ that accepts the language of Γ^* denoted by P. Starting from N_Γ we can build a new automaton, N_Σ, which checks whether a trajectory t of Σ^* belongs to $\nu(\mathcal{L}(P))$, and delivers the valuation ν.

Essentially, N_Σ is N_Γ with a management of variable bindings based on the following extensions: (i) a transition labeled with a variable `@x` on a symbol α sets the value of `@x` to α if `@x` was not yet bound and (ii) with each state one maintains the bindings of the variables met so far. The definition of N_Σ is as follows.

- The set of states of N_Σ, $states(N_\Sigma)$, is $states(N_\Gamma) \times \Sigma^{|var(P)|}$, i.e., all the possible associations of a state of N_Γ with a valuation ν of the variables in P. A state of N_Σ is denoted $< S, \nu >$.
- The set of accepting states of N_Σ, $accept(N_\Sigma)$ is $accept(N_\Gamma) \times \Sigma^{|var(P)|}$.
- The transition function of N_Σ, δ_Σ, is drawn from the transition function of N_Γ, δ_Γ, as follows:
  - if $\delta_\Gamma(S_i, \alpha) = S_j$ is a transition of N_Γ with $\alpha \in \Sigma$, then $\delta_\Sigma(< S_i, \nu >, \alpha) =< S_j, \nu >$. In other words the transition has no effect on variable bindings.
  - if $\delta_\Gamma(S_i, @x) = S_j$ is a transition of N_Σ with $@x \in \mathcal{V}$, then $\delta_\Sigma(< S_i, \nu >, \alpha) =$
    $\begin{cases} < S_j, \nu + @x := \alpha > \\ \quad \text{if } \nu(@x) \text{ is undetermined and the binding} \\ \quad \text{of } @x \text{ with } \alpha \text{ is allowed by the constraints.} \\ < S_j, \nu > \quad \text{if } \nu(@x) = \alpha. \\ \text{is undefined otherwise.} \end{cases}$

Whenever an accepting state $< S, \nu >$ of N_Σ is reached, the input trajectory is accepted and the valuation ν defines the instantiations of all the variables (recall that, by definition, any word in a language defined by a mobility pattern contains all the variables).

In order to check at run time whether an object o matches a mobility pattern, we do not need to fully construct the automaton described above. Instead, we start with a minimal representation, and build in a progressive way, according to the symbols appended to the trajectory of o, the instantiation of the variables which potentially leads to an accepting state. Here is an example that illustrates the process (more details can be found in the long version).

Example 2 *Consider the mobility pattern* $P = (a|b)^+.@x.(a|b)^+$. *Figure 2 shows an NFA automaton* N_Γ *which recognizes the words of* $\mathcal{L}(P)$*,* S_0 *being the initial state and* S_4, S_5 *the final states.*

Assume that one receives successively the following events for an object o: `a`*,* `a`*,* `b`*,* `b`*,* `c` *and* `a`*. Each row in the table of the figure 3 shows the states of the NFA* N_Σ *after reading a symbol, as well as the possible valuations of variable* `@x`*. The accepting states are in bold font and mean that the trajectory belongs to the query result set.*

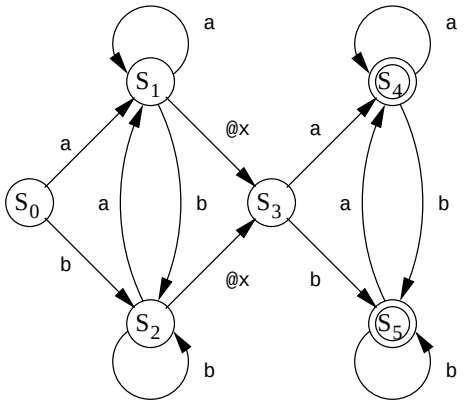


Figure 2: An automaton for the mobility pattern `(a|b)`$^+$`.@x.(a|b)`$^+$

Input	Reached states in N_Σ
a	$< S_1$,@x=$\perp >$
a[2]	$< S_1$,@x=$\perp >, < S_3$,@x=a>
a[2].**b**	$< S_2$,@x=$\perp >, < S_3$,@x=b>, $< \mathbf{S_5}$,**@x=a**>
a[2].**b**[2]	$< S_2$,@x=$\perp >, < S_3$,@x= $b >, < \mathbf{S_5}$,**@x=a**$>, < \mathbf{S_5}$,**@x=b**>
a[2].b[2].**c**	$< S_3$,@x= $c >$
a[2].b[2].c.**a**	$< \mathbf{S_4}$,**@x=c**>

Figure 3: Evaluation of a undeterministic query

Example 2 shows that we might have to maintain, during the analysis of an input trajectory, several valuations associated to a same state. In the worst case one might have $|states(N_\Gamma)| \times |\Sigma^k|$ simultaneous states to maintain, representing all the possible instantiations of variables that lead to an accepting state.

Depending on the application, the size of the database and the number of queries, maintaining a large amount of informations to continuously evaluate a query might become costly. In some cases we might therefore want to restrict the expressive power of the language to obtain very low memory needs. Consider for instance a web server providing a subscribe/publish mechanism over a (possibly large) set of moving objects. In such a system, web users can register queries, waiting for notification of the results. The performance of the system, and in particular its ability to serve a lot of queries under an intensive incoming of events, depends on the efficiency of the query result maintenance, and therefore on the size of the data required to perform this maintenance. We define below a fragment of the query language which meets the requirement of this kind of application.

3.3 Deterministic queries

The class of *deterministic* queries is such that, at any instant, there is only one possible instantiation for each variable of the mobility patterns. Deterministic queries are defined by the following property:

Definition 6 (Deterministic queries) *A query $q(P,\mathcal{C}$ is* deterministic *iff* $\forall u,v \in (\Sigma \cup \mathcal{V})^*, \forall$`@x`$\in \mathcal{V}, u.$`@x`$.v \in \mathcal{L}(P) \Rightarrow \nexists \alpha \in dom_q($`@x`$), \nexists w \in (\Sigma \cup \mathcal{V})^*, u\alpha w \in \mathcal{L}(P)$.

The intuition is that when it becomes possible to instantiate a variable during the analysis of a trajectory, then this transition is the only possible choice. This makes the binding of variables deterministic, and ensures that, for a given word, there is only one (if any) possibility to instantiate a variable.

Example 3 *The following examples illustrate deterministic queries.*

- *The query $q($`f.@x.(c|d).@x.f`$,\emptyset)$ is deterministic. Whenever a* `f` *symbol has been read, the only possible choice is to bind* `@x` *to the symbol that follows immediately* `f`.
- *The query $q((a|b)^+.@x.(a|b)^+,\emptyset)$ is non-deterministic since the words* `a.@x.a.b` *and* `a.b.@x.b` *both belong to $\mathcal{L}(P)$. However $q'((a|b)^+.@x.(a|b)^+,\{@x \neq a, @x \neq b\})$ is deterministic.*

We state the following properties of deterministic queries without the proofs which can be found in the long version.

Proposition 2 *Let $q(P,\mathcal{C})$ be a deterministic query. Then, for each word w of Σ^*, there is at most one witness of w in $\mathcal{L}(P)$.*

Consider again the queries of Example 3. In the first example an accepted word can only have one single witness, either `f.@x.d.@x.f` or `f.@x.c.@x.f`. In the second example, with constraints {`@x`$\neq$`a`,`@x`$\neq$`b`}, any witness consists of two words of {`a`,`b`}$^+$, separated by a symbol distinct from `a` or `b`. It follows that if $q(P,\mathcal{C})$ is a

deterministic query, the memory space required to check whether a word matches q is $|P|+|var(P)|$, where $|P|$ represents the number of symbols in P. Essentially, we need one FA for q, plus a storage for each variable, and we can build an FA with a number of states equal to the number of symbols in the expression.

When evaluating a continuous query, we need to maintain for each object o the set of its current states, as well as the binding of variables and this suffices to determine, at each GPS event, whether o enters, stays or quits the query result.

Example 4 *Let us consider again the query $q(P, \mathcal{C})$, with $P = (a|b)^+.@x.(a|b)^+$ and $\mathcal{C} = \{@x \neq a, @y \neq b\}$. The automaton remains identical (see Figure 2) but the evaluation on input* `a[2].b[2].c.a` *is now as presented in the table of the figure 4.*

The properties of deterministic queries ensure that the required amount of memory is independent from the size of Σ, and thus of the underlying partition of space used to describe the trajectories of moving objects. This property might be quite convenient if the space of interest is very large, or if the number of queries to maintain is such that the memory usage becomes a problem.

4 Conclusion and further work

We described in this paper a new approach for querying a moving object database by means of *mobility patterns*. Our proposal is based on a data model which allows to retrieve objects whose trajectory matches a parameterized sequence of moves expressed with respect to a set of labeled zones. We investigated the applicability of the model to continuous query evaluation, showed how to maintain incrementally the result of a query, and identify a fragment of the query language such that the amount of space required to maintain this result is very low.

A version of the language can easily be introduced as complement of a geometric-based extension of SQL, as shown by the query samples proposed in Section 2. The properties of the language make it a convenient candidate for mobile object tracking based on sequences patterns, and its simplicity leads to an easy implementation.

We are currently developing a prototype to assess the relevancy of this approach in a web-based context where a lot of clients can register queries, receive an initial result set, and wait for notification of updates to this result set.

References

[1] S. Abiteboul, B. Amann, S. Cluet, A. Eyal, L. Mignet, and T. Milo. Active Views for Electronic Commerce. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 1999.

[2] R. Book, S. Even, S. Greibach, and G. Ott. Ambiguity in graphs and expressions. *IEEE Transactions on Computers*, 20(2):149–153, 1971.

[3] T. Brinkhoff and J. Weitkämper. Continuous Queries within an Architecture for Querying XML-Represented Moving Objects. In *Proc. Intl. Conf. on Large Spatial Databases (SSD)*, 2001.

[4] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 2000.

[5] N. D., A. Fernandes, N. Paton, and T. Griffiths. Spatio-Temporal Evolution: Querying Patterns of Change in Spatio-Temporal Databases. In *Proc. Intl. Symp. on Geographic Information Systems*, pages 35–41, 2002.

[6] M. Dumas, M.-C. Fauvet, and P.-C. Scholl. Handling Temporal Grouping and Pattern-Matching Queries in a Temporal Object Model. In *Proc. Intl. Conf. on Information and Knowledge Management*, pages 424–431, 1998.

[7] F. Fabret, H. Jacobsen, F. Llirba, K. Ross, and D. Shasha. Filtering Algorithms and Implementations for Very Fast Publish/Subscrib Systems. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 2001.

[8] L. Forlizzi, R. Güting, E. Nardelli, and M. Schneider. A Data Model and Data Structures for Moving Objects Databases. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 2000.

[9] S. Grumbach, P. Rigaux, and L. Segoufin. Manipulating Interpolated Data is Easier than you Thought. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 2000.

[10] R. H. Gting, M. H. Bhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A Foundation for Representing and Quering Moving Objects. *ACM Trans. on Database Systems*, 25(1):1–42, 2000.

[11] M. Hammad, W. Aref, and A. Elmagarmid. Stream Window Join: Tracking Moving Objects in Sensor-Network Databases. In *Proc. Intl. Conf. on Scientific and Statistical Databases (SSDBM)*, 2003.

[12] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[13] D. Kalashnikov, S. Prabhakar, W. Aref, and S. Hambrusch. Efficient evaluation of continuous range queries on moving objects. In *Proc. Intl. Conf. on Databases and Expert System Applications (DEXA)*, pages 731–740, 2002.

Input	Reached states in N_Σ	Transitions not allowed
a	$< S_1$,@x=$\perp >$	
a[2]	$< S_1$,@x=$\perp >$	$< S_3$,@x=a> since a $\notin dom(@x)$
a[2].**b**	$< S_2$,@x=$\perp >$	$< S_3$,@x=b> since b $\notin dom(@x)$
a[2].**b**[2]	$< S_2$,@x=$\perp >$	$< S_3$,@x=b> since b $\notin dom(@x)$
a[2].b[2].**c**	$< S_3$,@x=c>	
a[2].b[2].c.**a**	$< \mathbf{S_4}$,**@x=c**>	

Figure 4: Evaluation of a deterministic query

[14] L. Liu, C. Pu, and W. Tang. Continual Queries for Internet Scale Event-Driven Information Delivery. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):610–628, 1999.

[15] G. Mecca and A. J. Bonner. Finite Query Languages for Sequence Databases. In *Proc. Intl. Workshop on Database Programming Languages*, 1995.

[16] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel Approaches in Query Processing for Moving Objects. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 2000.

[17] R. Ramakrishnan, D. Donjerkovic, A. Ranganathan, K. S. Beyer, and M. Krishnaprasad. Srql: Sorted relational query language. In *Proc. Intl. Conf. on Scientific and Statistical Databases (SSDBM)*, pages 84–95. IEEE Computer Society, 1998.

[18] R. Sadri, C. Zaniolo, A. M. Zarkesh, and J. Adibi. Optimization of sequence queries in database systems. In *Proc. ACM Symp. on Principles of Database Systems*, 2001.

[19] R. Sadri, C. Zaniolo, A. M. Zarkesh, and J. Adibi. A sequential pattern query language for supporting instant data mining for e-services. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 2001.

[20] P. Seshadri, M. Livny, and R. Ramakrishnan. SEQ: A Model for Sequence Databases. In *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*, pages 232–239, 1995.

[21] A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*, pages 422–433, 1997.

[22] A. P. Sistla, T. Hu, and V. Chowdhry. Similarity based retrieval from sequence databases using automata as queries. In *Proc. Intl. Conf. on Information and Knowledge Management*, pages 237–244, 2002.

[23] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, pages 287–298, 2002.

[24] D. Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous Queries over Append-Only Databases. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 1992.

[25] G. Trajcevski, P. Scheuermann, O. Wolfson, and N. Nedungadi. Cat: Correct answers of continuous queries using triggers. pages 837–840, 2004.

[26] L. Wall, T. Christiansen, and J. Orwant. *Programming Perl (3rd Edition)*. O'Reilly, 2000.

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 mobility 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 describe 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 $t = \tau + 2$.

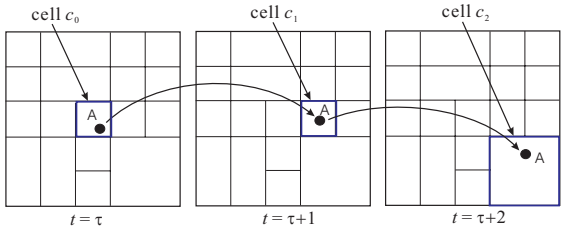


Figure 1: Notion of the Markov chain model

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

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 ($t = 0, 1, \ldots, T$), we can compute the probability as follows:

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

where $\mathrm{objs}(c_i, t)$ 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 $t = 0, \ldots, T-1$. 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 $c_0, c_1, \ldots, c_{n-1}$ with this order in each period of unit time interval moves to cell c_n in the next period is denoted as $\Pr(c_n|c_0, \ldots, c_{n-1})$, and estimated by the following generalized form:

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

Note that the set of cells $\{c_0, c_1, \ldots, c_n\}$ 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 ($t = \tau+1, 2, \ldots$). 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 $\mathrm{overlaps}(x, y)$ and $\mathrm{north}(y, z)$". 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 $\mathrm{objs}(c, t)$ 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.

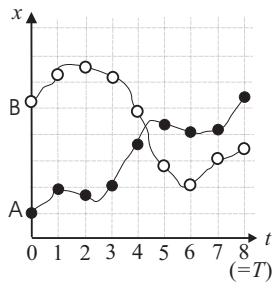


Figure 2: Representation of trajectories

If the position of a moving object o at time $t = \tau$ ($\tau = 0, 1, \ldots, T$) is represented as a point $[x_1, \ldots, x_d]$ 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 $[x_1, \ldots, x_d, t]$, 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 $t = 0, 1, \ldots, T$ 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 ($0 \le k < m$). 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 $t = 0, 2, 4, \ldots, T$. 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 $c_0, \ldots, c_n$ to the following more general one:

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

We say that a probability $\Pr(c_n|c_0, \ldots, c_{n-1})$ is *undefined* when there are no moving objects which existed on $c_0, \ldots, c_{n-1}$ 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 $\Pr(c_n|c_0, \ldots, c_{n-1})$ for a specific cell combination $c_0, \ldots, c_n$. 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 $t = \tau$ ($\tau = 0, 1, \ldots, T - n$), and in cell c_1 when $t = \tau + 1$, ... and in cell c_{n-1} when $t = \tau + n - 1$:

$$\begin{aligned} S \;=\; \{o \mid \exists \tau \in \{0, 1, \ldots T - n\},\, o \in \mathrm{objs}(c_0, \tau) \\ \wedge\, o \in \mathrm{objs}(c_1, \tau + 1) \wedge \cdots \\ \wedge\, o \in \mathrm{objs}(c_{n-1}, \tau + n - 1)\}, \end{aligned} \tag{3}$$

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

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

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

Procedure naïve_estimation
Output: A list of "defined" $\Pr(c_n|c_0, \ldots, c_{n-1})$ values
1. **for each** $(c_0, \ldots, c_{n-1}) \in C_0 \times \cdots \times C_{n-1}$ **do**
2. $Scount := 0$; $Qcount[\,] := 0$
3. **for** $t := 0$ **to** $T - n$ **do**
4. **for** $i := 0$ **to** $n-1$ **do** $oids_i := \mathrm{range\_query}(c_i,\ t+i)$;
5. $Scount$ += $|\bigcap_{i=0}^{n-1} oids_i|$;
6. **for each** $c_n \in C_n$ **do**
7. $oids_n := \mathrm{range\_query}(c_n,\ t+n)$;
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**$(c_0, \ldots, c_n,\ Qcount[c_n]/Scount)$;
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 $|C_0| \times \cdots |C_{n-1}| \times \{(T-n+1) + |C_n|\}$. When $T \gg n$ and $T \gg |C_n|$ hold, the value can be approximated as $T \times |C_0| \times \cdots \times |C_{n-1}|$. 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 $(i = 0, \ldots, n)$ 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

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

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 $\mathrm{shift}(P_i,\ j)$ as a set of time intervals which is obtained by shifting each time interval in P_i with j unit times. For example, $\mathrm{shift}(\{[1,\ 3],\ [5,\ 8]\},\ 1) = \{[2,\ 4],\ [6,\ 9]\}$.

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

$$\forall i \in \{0, \ldots, n\},\ P_i \sqcap [i,\ T-n+i] \neq \emptyset \tag{6}$$

and there is an integer τ that satisfies

$$\forall i \in \{0, \ldots, n\},\ \tau + i \in P_i \sqcap [i,\ T-n+i]. \tag{7}$$

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

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

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 $C_0,\ C_1,\ \ldots,\ C_n$ at $t = T-n,\ t = T-n+1, \ldots, t = T$, respectively. Other cases $(i = 1, \ldots, n)$ 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 $(i = 0, \ldots, n)$. The condition means that object o is contained in $C_0,\ C_1, \ldots, C_n$ at $t = \tau,\ t = \tau+1, \ldots, t = \tau+n$, 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 $C_0, \ldots, C_{n-1}$ 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 $c_0, \ldots, c_{n-1}$ with this order as $Scount\{c_0\# \cdots \#c_{n-1}\}$, where $c_0\# \cdots \#c_{n-1}$ 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$(n,\ root,\ level,\ (C_0, \ldots, C_n),\ max_dist)$
Input: n: the order of Markov transition
$root$: the root node of the R-tree
$level$: the number of levels of the R-tree
$(C_0, \ldots, C_n)$: 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" $\Pr(c_n|c_0, \ldots, c_{n-1})$
1. **for** $j := 0$ **to** n **do** $nodes[j] := root$;
2. $Scount := \mathrm{FC\_count}(n-1,\ level,\ nodes,\ (C_0, \ldots, C_{n-1}),\ max\_dist)$;
3. $Qcount := \mathrm{FC\_count}(n,\ level,\ nodes,\ (C_0, \ldots, C_n),\ max\_dist)$;
4. **foreach** $(c_0, \ldots, c_n) \in C_0 \times \cdots \times C_n$ **do**
5. **if** $Scount\{c_0\# \cdots \#c_{n-1}\} > 0$ **then**
6. **output**$(c_0, \ldots, c_n,$
7. $Qcount\{c_0\# \cdots \#c_n\}/Scount\{c_0\# \cdots \#c_{n-1}\})$;

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($n, level, nodes, (C_0, \ldots, C_n), max\_dist$)
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. $child\_set := \emptyset$;
3. **foreach** $v \in nodes[j]$.children **do**
4. **if** sp_overlap(C_j, v) **then** // v overlaps spatially with C_j
5. $child\_set := child\_set \cup \{v\}$;
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 C_i
16. **if** $level \geq 1$ **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** $level \geq 1$ **then** // the case of non-leaf nodes
22. **for** $k := 0$ **to** n **do** $refs[i] := inst[k]$.value;
23. FC_count(n, $level - 1$, $refs$, $\{C_0, \ldots, C_n\}$);
24. **else** // the case for leaf nodes: increment count for the solution
25. $count\{$cell($inst$.[0].value)#$\cdots$#cell($inst[n]$.value)$\}$++;
26. **end**
27. **else** // while the intermediate of constraint satisfaction
28. **if** check_forward(i, n, $level$, dom, $inst$, $(C_{i+1}, \ldots, C_n)$, max_dist)
29. i++; // examine the next constraint C_{i+1}
30. **end**

Figure 5: Transition sequence enumeration algorithm

At lines 1 to 7, an initial solution *candidate set* is set to $dom[0][j]$ ($j = 0, \ldots, n$). 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 $(n+1) \times (n+1)$ 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(C_j, 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, $inst[0], \ldots, inst[i-1]$ 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 C_0, 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 $level \geq 1$, 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 $level \geq 1$, 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 $level \geq 1$, FC_count is called recursively taking the nodes bounded to $inst[0]$.value, $\ldots$, $inst[n]$.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 $inst[0], \ldots, inst[i]$ 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 $inst[0], \ldots, inst[i]$ 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 $inst[0], \ldots, inst[i]$, 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 $dom[i+1][j]$ ($j = i+1, \ldots, n$) 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 ($j = i + 1, \ldots, n$). In this process, if $dom[i + 1][j] = \emptyset$ holds for some j, we can say that there is no solution for the current target $inst[0], \ldots, inst[i]$, 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, $\ldots$, $inst[i]$.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, $\ldots$, $inst[i]$.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 $level \geq 1$ then v is an R-tree node, otherwise v is a

Function check_forward(i, n, $level$, dom, $inst$, $(C_{i+1}, \ldots, C_n)$, max_dist)
Output: if there are candidates $inst[i+1], \ldots, inst[n]$ for the given $inst[0], \ldots, inst[i]$ 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** $v \in dom[i+1][j]$ **do** // for each candidate
4. **if** $(level = 0)$ **and** $(inst[0].\text{value.id} \neq v.\text{id})$
5. // v is a trajectory for other object
6. **then goto** line 17;
7. // calculate the valid time interval
8. $vrange := \text{t\_overlap}(v.\text{trange}, [j, T - n + j])$;
9. **if** $vrange = \perp$ **then goto** 17; // $vrange$ is empty
10. **if** $\text{t\_overlap}(\text{shift}(vrange, -j), inst[0].\text{trange}) = \perp$
11. **then goto** 17; // does not satisfy temporal constraints
12. **if not** $\text{sp\_overlap}(C_j, v)$ **then goto** 17;
13. // v does not overlap with the area of cell set C_j
14. **if** $\text{sp\_dist}(inst[i].\text{value}, v) > max\_dist \times (j - i)$ **then goto** 17;
15. // we cannot move from $inst[i].\text{value}$ 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** $dom[i+1][j] = \emptyset$ **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 $inst[i].\text{value}$ (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 $inst[i].\text{value}$ 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.

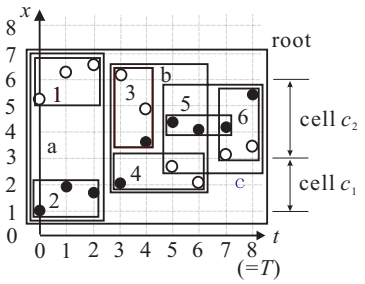


Figure 7: Example of an R-tree

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, (C_0, C_1, C_2), 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 $dom[0][0] = dom[0][1] = dom[0][2] = \{\text{a, b, c}\}$. After entering the while loop with $i = 0$, we set $inst[0].\text{value} = \text{a}$, $inst[0].\text{trange} = [0, 2]$, $dom[0][0] = \{\text{b, c}\}$ 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 $dom[1][1] = dom[0][1] = \{\text{a, b, c}\}$. Since $v = \text{a}$ and $v = \text{b}$ satisfy the following all conditions, they are not removed from $dom[1][1]$ in the process. However, when $v = \text{c}$, we get $vrange = [5, 8]$ at line 8 and since $\text{shift}([5, 8], -1) \sqcap [0, 2] = \perp$, the candidate $v = \text{c}$ is removed at line 10. Therefore, we get $dom[1][1] = \{\text{a, b}\}$. For $j = 2$, we get $dom[1][2] = \{\text{a, b}\}$ in a similar manner. Namely, even if a trajectory point object which satisfies the 0-th constraint is inside of $inst[0].\text{value} = \text{a}$, 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 $inst[1].\text{value} = \text{a}$, $inst[1].\text{trange} = [1, 2]$, $dom[1][1] = \{\text{b}\}$. Since check_forward returns true, we get $dom[2][2] = \{\text{a, b}\}$. Then we return to FC_count again and increment i. In the next while loop, $\text{FC\_count}(2, 1, refs, \{C_0, C_1, C_2\})$ is called recursively at line 23, where $refs[0] = \text{a}$, $refs[1] = \text{a}$, $refs[2] = \text{a}$.

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 $inst[0].\text{value} = 2$ 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 ($inst[0].\text{value}$) to node 1 within a unit time ($sp\_dist(2, 1) > max\_dist$). Therefore, we next call $\text{FC\_count}(2, 0, refs, \{C_0, C_1, C_2\})$ 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 $inst[0].\text{value} = \text{a}$, $inst[1].\text{value} = \text{a}$, $inst[2].\text{value} = \text{b}$ and fails again then performs a backtrack. Next another fail occurs for $inst[0].\text{value} = \text{a}$, $inst[1].\text{value} = \text{b}$, $inst[2].\text{value} = \text{a}$ then we try the case of $inst[0].\text{value} = \text{a}$, $inst[1].\text{value} = \text{b}$, $inst[2].\text{value} = \text{b}$. In this case, a constraint solution $inst[0].\text{value} = (id = \text{A}, t = 2, x = 1.8)$, $inst[1].\text{value} = (id = \text{A}, t = 3, x = 2.2)$, $inst[2].\text{value} = (id = \text{A}, t = 4, x = 3.6)$ is obtained. As a result, $\text{count}(c_1 \# c_1 \# c_2)$ 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 naïve 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 Oldenbuerg 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 $\times$ 30 cells. Then we select a 3 $\times$ 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 ($T = 100, 200, \ldots, 900$) to examine the scalability of the algorithms. As shown in Fig. 9, the CSP-based algorithm (CSP) performs well compared to the naïve 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

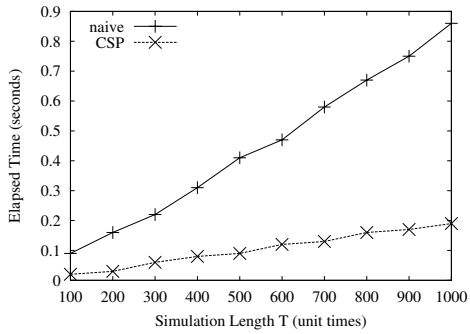


Figure 9: Estimation of probabilities ($n = 1$, $3 \times 3/30 \times 30$)

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.

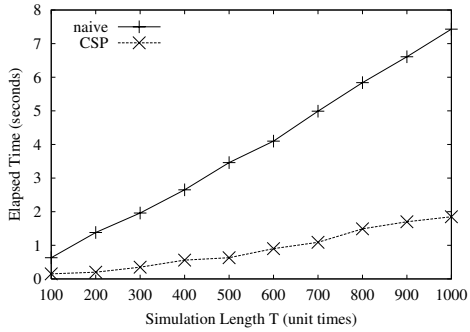


Figure 10: Estimation of probabilities ($n = 2$, $3 \times 3/30 \times 30$)

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.

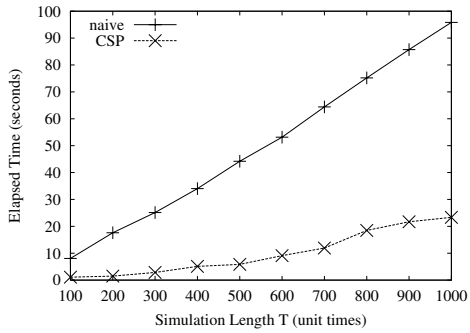


Figure 11: Estimation of probabilities ($n = 3$, $3 \times 3/30 \times 30$)

In the experiments shown above, the CSP-based algorithm highly well performed than the naïve 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 naïve 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-

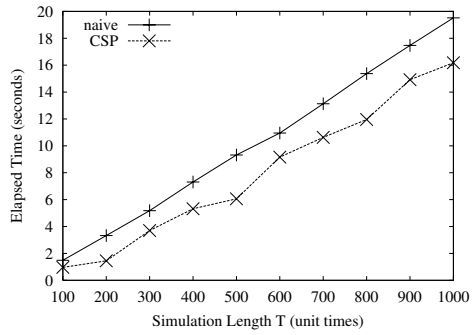


Figure 12: Estimation of probabilities ($n = 2,\ 3 \times 3/20 \times 20$)

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, *GeoInfomatica*, 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 Kanfmann, 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, Developping 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.

Utilizing Road Network Data for Automatic Identification of Road Intersections from High Resolution Color Orthoimagery

Ching-Chien Chen, Cyrus Shahabi, Craig A. Knoblock
University of Southern California
Department of Computer Science & Information Sciences Institute
Los Angeles, CA 90089-0781
[chingchc, shahabi, knoblock]@usc.edu

Abstract

Recent growth of the geo-spatial information on the web has made it possible to easily access various and high quality geo-spatial datasets, such as road networks and high resolution imagery. Although there exist efficient methods to locate road intersections from road networks for route planning, there are few research activities on detecting road intersections from orthoimagery. Detected road intersections on imagery can be utilized for conflation, city-planning and other GIS-related applications. In this paper, we describe an approach to automatically and accurately identifying road intersections from high resolution color orthoimagery. We exploit image metadata as well as the color of imagery to classify the image pixels as on-road/off-road. Using these chromatically classified image pixels as input, we locate intersections on the images by utilizing the knowledge inferred from the road network. Experimental results show that the proposed method can automatically identify the road intersections with 76.3% precision and 61.5% recall in the imagery for a partial area of St. Louis, MO.

1. Introduction

Recent advances in remote sensing technology are making it possible to capture geospatial orthoimagery (i.e., imagery that created so that it has the geometric properties of a map) with a resolution of 0.3 meter or better. These images are available online and have been utilized to enhance real estate listings, military targeting applications, and other GIS applications. One of the key issues with these applications is to automatically identify man-made spatial features, such as buildings, roads and road intersections in raster imagery. Computer vision researchers have long been trying to identify features in the imagery [1]. While the computer vision research has produced algorithms to identify the features in the imagery, the accuracy and run time of those algorithms are not suited for these real-time applications.

Integrating existing vector data, such as road network data, as part of the spatial feature recognition scheme alleviates these problems. For example, the spatial information of road network data represents the existing knowledge about the approximate location of the roads and intersection on imagery. In addition, the attribution information of road network data, such as road names and address ranges, can be utilized to locate and annotate buildings on imagery [2]. However, accurately and automatically integrating geo-spatial datasets is a difficult task, since there are often certain spatial inconsistencies between geo-spatial datasets. There are multiple reasons why different data products may not align: they may have been collected at different resolutions, they may use different spheroids, datums, projections or coordinate systems, they may have been collected in different ways or collected with different precision or accuracy, etc. Conflation is often a term used to describe the integration or alignment of different geospatial data sets.

The conflation [3] process can be divided into following subtasks: (1) Feature matching: Find a set of conjugate point pairs, termed control point pairs, in two datasets, (2) Match checking: Detect inaccurate control point pairs from the set of control point pairs for quality control, and (3) Alignment: Use the accurate control points to align the rest of the points and lines in both datasets using the triangulation [4] and rubber-sheeting

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

Publisher	***PubDate***	***OrdinateRes***	***WestCoord***	***EastCoord***	***NorthCoord***	***SouthCoord***
USGS	2003	0.3	-90.63578	-90.61862	38.38338	38.36987
USGS	2003	0.3	-90.63534	-90.61818	38.39689	38.38338
USGS	2003	0.3	-90.63490	-90.61773	38.41039	38.39689
:	:	:	:	:	:	:

(a) Sample (partial) metadata of USGS high resolution color orthoimagery

Source	***AreaCovered***	***Projection***	***Pub-Date***	***CFCC classification***	***WestCoord***	***EastCoord***	***NorthCoord***
USGS 1:100k DLG	El Segundo, CA	Lambert Conformal Conic	1998	Secondary roads	-90.44	-90.423	38.582
USGS 1:100k DLG	St. Louis, MO	Lambert Conformal Conic	1998	Primary roads	-118.4351	-118.3702	33.9164
USGS 1:100k DLG	St. Louis, MO	Lambert Conformal Conic	1998	City streets	-118.4351	-118.3702	33.9164
:	:	:	:	:	:	:	:

(b) Sample (partial) metadata of U.S. Census Bureau TIGER/Lines

Table 1: Sample metadata of orthoimagery and road networks

techniques. One major difficulty with current conflation techniques is that they require the manual intervention often including identification of a set of control points for feature matching to properly conflate two data sets.

Various GIS researchers and computer vision researchers have shown that the intersection points on the road networks provide an accurate set of control point pairs [5, 6, 10]. One cannot rely on a manual approach to locate road intersections to perform conflation, as the area of interest may be anywhere in the world and manually finding and filtering road intersections for a large region, such as, the continental United States, is very time consuming and error-prone. Moreover, performing conflation offline on two geo-spatial datasets is also not a viable option in online applications as both datasets may be obtained by querying different information sources at run-time. Therefore, an automatic approach to identifying road intersections in diverse geo-spatial datasets, especially in orthoimagery, is needed. Moreover, road intersections can not only be used for geo-spatial data fusion, but can also be utilized for transportation-related GIS [7], city-planning and GIS data updating, etc.

In this paper, we propose an approach to automatically identify and annotate road intersections on high resolution color imagery. In particular, we utilize road network data and imagery metadata to both improve the accuracy and reduce the running time of image analysis techniques. Consequently, the entire road detection process can be done without any manual intervention in real time. Experimental results show that our proposed method can automatically identify the road intersections with 76.3% precision and 61.5% recall in the imagery for a partial area of the county of St. Louis, MO. To the best of our knowledge, automatically exploiting these auxiliary structured data to improve the image recognition techniques has not been studied before.

The remainder of this paper is organized as follows. Section 2 describes our overall approach. Section 3 illustrates the techniques to label image pixels based on imagery metadata and Bayes classifier. Section 4 presents an image processing technique utilizing knowledge inferred from road network data to detect road intersections on imagery. Section 5 provides experimental results. Section 6 discusses the related work and Section 7 concludes the paper by discussing our future plans.

2. Overview

Recently, metadata (i.e., information about data) is used increasingly in geographic information systems to improve both availability and the quality of the spatial information delivered. Table 1 shows sample metadata about USGS high resolution color orthoimagery[1] and the road network U.S. Census Bureau TIGER/lines[2]. We exploit metadata from both imagery and road network data to perform the automatic road intersection detection procedure.

For the imagery, we can exploit the ground resolution and geo-coordinates to measure real world distance between any two spatial objects or perform image

[1] http://seamless.usgs.gov

[2] http://tiger.census.gov/cgi-bin/mapsurfer

processing techniques (such as edge-detection, region-segmentation and histogram-based classification) at the pixel level to extract primitive entities, such as corners, edges and homogeneous regions, etc. For vector data of roads, we can exploit metadata about the vectors, such as address ranges, road names, or even the number of lanes and type of road surface. In addition, we can analyze the road network to determine the location of intersections, the road orientations and road shapes around the intersections. This inferred knowledge from road network data can then be augmented with information retrieved from imagery. For instance, we can find the approximate location of intersections on the images from the metadata of vector data, while the information (such as edges or pixel intensity) on imagery can be utilized to locate the exact location of intersections nearby the approximate locations. In sum, these automatically exploited information are dynamically exchanged and matched across these geospatial datasets to accurately identify road intersections.

Figure 1 shows our overall approach. Using chromatically classified image pixels as input, we locate intersections on the images by utilizing the image metadata and the information inferred from vector, such as approximate location of intersections, road-directions, road-widths, and road-shapes. In addition, identified intersections could be annotated with the vector information, such as road names and zip code.

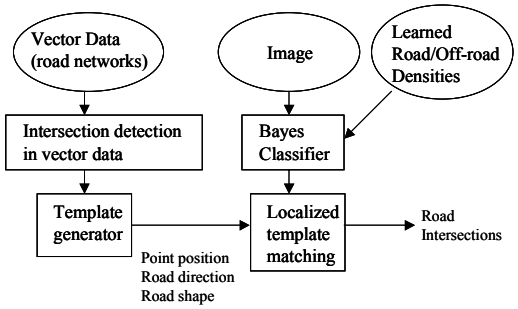


Figure 1: Overall approach

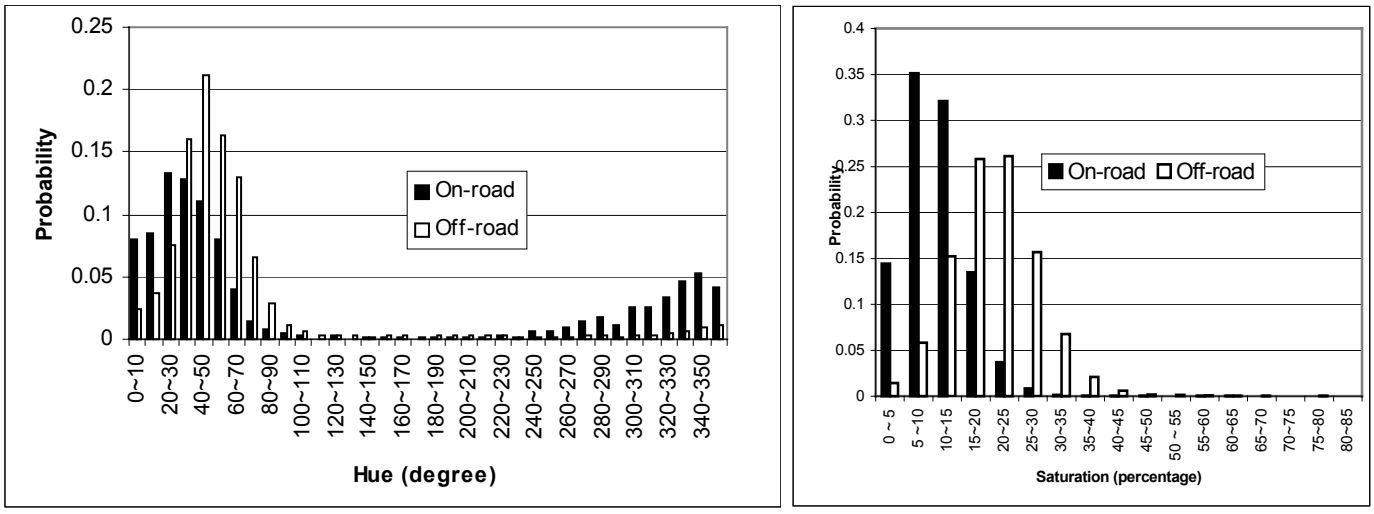
3. Labeling imagery using Bayes classifier

Towards the objective of identifying road intersections, the first vital step is to understand the characteristics of roads on imagery. On high resolution imagery, roads are exposed as elongated homogeneous regions with almost constant width and similar color along a road. In addition, roads contain quite well defined geometrical properties. For example, the road direction changes tend to be smooth, and the connectivity of roads follow some topological regularities. Road intersection can be viewed as the intersection of multiple road axes and it is located at the overlapping area of these elongated road regions. These elongated road regions form a particular shape around the intersection. Therefore, we can match this shape against a template derived from road network data (discussed next) to locate the intersection. Based on the characteristics of roads, the formation of this shape is either from detected road-edges or homogeneous regions. However, on high resolution imagery, more detailed outlines of spatial objects, such as edges of cars and buildings, are considered as noisy edges. This makes perceptual-grouping based method used for road-edges linking a difficult task.

In contrary to edge-detection, we propose a more effective way to identify intersection point on color imagery by using Bayes classifier, a histogram based classifier [8, 9], to classify images' pixels around road network data as on-road or off-road pixels. The classification is based on the assumption of consistency of image color on road pixels. That is, road pixels can be dark, or white, or have color spectrum in a specific range, but still we expect to find the same representative color on close by road pixels. We construct the statistical color distribution (called class-conditional density) of on-road/off-road pixels by utilizing histogram learning technique as follows. We first randomly select a small partial area from the imagery where we intend to identify road intersections. Then, we interactively specify on-road regions and off-road regions respectively. From these large amount of manually labeled training pixels, we learn the color distribution (histograms) for on-road and off-road pixels. Hence, we can construct the on-road and off-road densities.

Figure 2 shows the hue probability density and saturation probability density[3], after conducting the learning procedure on nearly 50,000 manually picked pixels of 2% of a set of USGS 30cm/pixel imagery (covering St. Louis County in Missouri of the United States). To illustrate, consider the hue density function on Figure2(a). It shows the conditional probabilities *Prob(Hue/On-road)* and *Prob(Hue/Off-road)*, respectively. The X-axis of this figure depicts the hue value grouped every 10 degrees. The Y-axis shows the probability of on-road (and off-road) pixels that are within the hue range represented by X-axis. For a particular image pixel, we can compute its hue value h. Given the hue value h, if the probability for off-road is higher than on-road, our system would predict that the pixel is an off-road pixel. As shown in Figure 2, these density functions depict the different distribution of on-road and off-road image pixels on hue and saturation dimensions, respectively. Hence, we may use either of them to classify the image pixels as on-road or off-road. In our experiments, we utilized hue density function for

[3] Due to lack of space, we eliminated the intensity (i.e., brightness of HSV model) density function. In fact, there is no obvious difference between the brightness distribution of on-road and off-road pixels, since these images were taken at the same time (i.e., under similar illumination conditions).



(a) Hue density function

(b) Saturation density function

Figure 2: Learned density function on HSV color space for On-road/Off-road pixels

(a) Layout (left) of the original road network data around an intersection point and template (right) inferred by using the road network data

(b) Original image

(c) Road-labeled image (White pixels: labeled road pixels; Black lines: existing road network data; Black circles: intersections on vector data, implying approximate location of intersections on imagery)

Figure 3: An example of the localized template matching

classification. In general, we can utilize the two chromatic components, hue and saturation, together.

Based on the learned hue density functions, an automated road-labeling is conducted as follows. A particular image pixel whose hue value is *h* is classified as road if

$$\frac{p(h/road)}{p(h/non-road)} \geq \theta$$, where θ is a threshold. θ depends on the application-specific costs of classification errors and it can be selected using ROC technique discussed in [9].

Since we know the approximate intersection locations on the images from the road network data (discussed next), the road-labeling procedure is applied only to image pixels within a radius of potential intersections. Therefore, we do not need to exhaustively label each pixel on the entire image.

4. Analyzing imagery using road network data

Using the classfied image (an example is shown in Figure 3(b)(c)) as input, we can now match it with a template inferered from the road network data to identify intersections. We term this procedure as localized template matching (LTM). First, our LTM technique finds the geo-coordinates of all the intersection points on the road network data. Since we also know the geo-coordinates of the images (from image metadata), we can obtain the approximate location of intersections on the imagery (as in Figure 3(c)). For each intersection point on

the road network data, LTM determines the area in the image where the corresponding intersection point should be located. The area size can be determined based on the accuracy and resolution (such as ground resolution from image metadata) of the two datasets. One option is conducting experiments using various sizes and selecting the size that has better performance (discussed in Section 5).

For each intersection point detected from the road network data, LTM picks a rectangular area in the image centered at the location of the intersection point from the road network data. Meanwhile, as an example shown in Figure 3(a), a template around an intersection on road network data is generated by the presence of regions inferred from the road network data using information, such as the road directions and road widths. LTM will then locate regions in the road-labeled image (see Figure 3(c)) that are similar to the generated template (as in Figure 3(a)) as follows. Given a template *T* with w x h pixels and road-labeled image *I* with W x H pixels, we move the template around each pixel at the image and compare the template against the overlapped image regions. Our adapted similarity measure is a normalized cross correlation defined as:

$$C(x,y) = \frac{\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1} T(x',y')I(x+x',y+y')}{\sqrt{\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1} T(x',y')^2 \sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1} I(x+x',y+y')^2}}$$

where *T(x,y)* equals one, if *(x,y)* belongs to a road region, otherwise; *T(x,y)* equals zero. *I(x,y)* equals one, if *(x,y)* is pre-classified as a road pixel, otherwise; *I(x,y)* equals zero. C(x,y) is the correlation on the pixel (x,y).

The highest computed correlation C(x,y) implies the location of the best match between the road-labeled image and template. In addition, an intersection will be identified, if C(x,y) is greater than a threshold *t (0 <= t <=1.0)*. We set the threshold *t* to 0.5. Hence, the best-matched point will be characterized as an intersection only if it is at least quasi-similar to the vector template.

The histogram-based classifier as illustrated in previous section may generate fragmented results, due to some noisy objects, such as cars, tree-clusters and building shadings on the roads. Furthermore, some non-road objects whose color is similar to road pixels might be misclassified as roads. However, LTM can alleviate these problems by avoiding exhaustive search of all the intersection points on the entire image and often locates the intersection point on the image that is the closest intersection point to the intersection point detected from the road network data. Moreover, this technique does not require a classifier to label every pixel for the entire region. Only the areas near the intersections on the image need to be pre-classified.

Figure 4 shows an image indicating the intersection points on road network data and the corresponding intersection points identified on imagery. One of the accurately identified intersections is annotated with the road network information, such as road names and zip code.

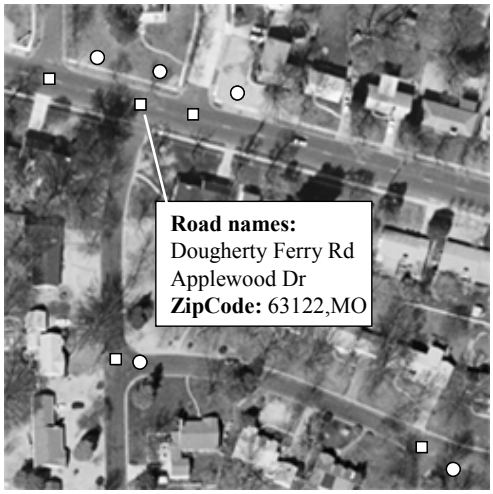


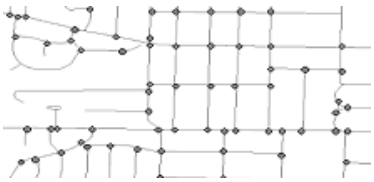
Figure 4: The intersections (circles) on road network data and the corresponding intersections (rectangles) on imagery.

5. Performance Evaluation

We conducted several experiments to evaluate our approach by measuring the precision and recall of the identified road intersections against real road intersections. For our experiments, we used a set of 0.3m/pixel resolution color orthoimagery (covering St. Louis County in Missouri of the United States) from USGS and road network data from NAVTEQ[4] and U.S. Census TIGER/Lines. In general, both road network data have rich attribution but TIGER/Lines has poor positional accuracy and poor road shapes. We learned the histogram (as shown in Figure 2) from nearly 50,000 manually picked pixels of 2% of these images. Then, we applied our approach to identify intersection points on randomly selected areas of these images (about 9% of this imagery). There are about 1200 intersections in total on these tested areas. Figure 5 shows 8% of the NAVTEQ road network data and 0.48% of the image area used in our experiments. The off-line learning process requires manual intervention to obtain conditional density functions, but it is performed only when new imagery dataset is introduced to the system. In addition, we can apply the learned results to automatically identify intersections of the area that is much larger than the area we learn from.

We applied a "buffer method" to evaluate recall and precision. When multiple elongated road regions merge at

[4] http://www.navteq.com/



(a) Road network (NAVSTREET)

(b) USGS Orthoimagery

Figure 5: A partial area of tested data

an intersection, their overlapping area at the intersection is a polygon (called buffer). Identified road intersections that fall within the buffer are considered as "accurately identified intersections". Using this term, we define:

$$\text{Recall} = \frac{\text{Number of accurately identified intersections}}{\text{Number of intersections in the image}}$$

$$\text{Precision} = \frac{\text{Number of accurately identified intersections}}{\text{Number of identified intersections}}$$

For each type of road network data, the area radius, a fixed constant used in LTM, was determined by conducting experiments using various sizes. An experimental result for NAVTEQ data (on an area with 106 intersections) is shown in Figure 6. In Figure 6, we also demonstrate the normalized intersection detection running time (with respect to the running time of using 180m as radius) to show that the detection time

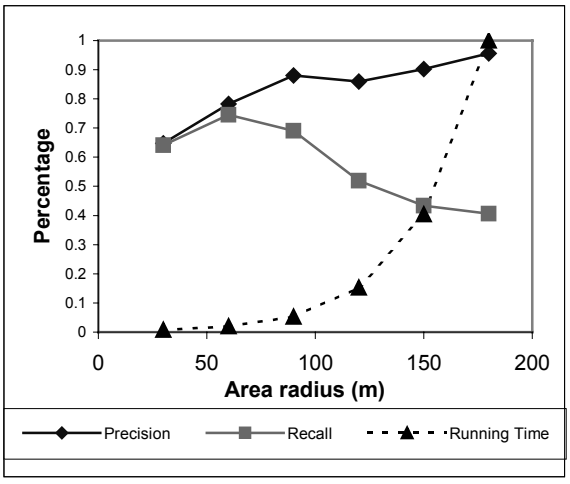


Figure 6: The impact of area radius
(Radius is increased by 30 meters, i.e. 100 pixels on imagery)

dramatically increases as area size increases. We selected 90m as our area radius for the rest of experiments. When setting the radius to 90m, we achieved 88% precision. Although it is less than the precision obtained using 180m as radius, we have much better recall (70% v.s. 40%) and 20 times better running time.

For the overall tested area, on the average, we obtained 76.3% precision and 61.5% recall with NAVTEQ data and 62% precision and 39.5% recall with TIGER/Lines. If we exclude the intersections detected on highways where the road widths vary and difficult to predict, we achieved 83% precision and 65% recall with NAVTEQ data. In order to explain our experiments, we show the performance of a sub-area (with 106 intersections) of the larger tested area in Table 2. As shown in Table 2(a), there are originally more than 30 intersection points on the NAVTEQ vector that match with the corresponding intersections on images, while there are only about four of these intersections on the TIGER/Lines. This is because the NAVTEQ vector data has a very high accuracy. Nevertheless, we significantly improved the precision and recall of both vector data.

	Precision	Recall
Original NAVTEQ	31.2%	31.5%
Localized Template Matching (LTM)	**88%**	**69%**
LTM with VMF filter	98%	53%

(a) Identified Intersections using NAVTEQ road network

	Precision	Recall
Original TIGER/Lines	4.1%	3.5%
Localized Template Matching (LTM)	**71%**	**41%**
LTM with VMF filter	91%	27.4%

(b) Identified Intersection Points using TIGER/Lines

Table 2: Road Intersection Identification Evaluation

Now, suppose we want to use our detected intersections as control points for a matching problem, such as the vector to imagery conflation problem described in Section 1. The conflation process does not require a large number of control point pairs to perform accurate alignment. In fact, a smaller set of control points with higher accuracy would serve better for the conflation process [10]. Therefore, for the conflation process higher precision is more important than higher recall. Towards this end, we can use a filter to eliminate misidentified intersections and only keep the accurately identified intersections, hence improving the precision with the cost of reducing the recall. VMF [10] is an example of such filter. As shown in Table 2, the VMF filter improves the precision, although it reduces the recall.

The VMF filter works based on the fact that there is a significant amount of regularity in terms of the relative positions of the intersections on the vector and the detected (corresponding) intersections on the imagery across data sets. More precisely, first the geographic coordinate displacement between the intersections on the road network and detected (corresponding) imagery intersections is modeled as 2D vectors. Next, for a small

area, the vectors whose directions and magnitudes are significantly different from the median vector are characterized as inaccurate vectors. By eliminating these vectors, we can filter out their corresponding misidentified intersections. Before applying VMF, there are three misidentified intersections (marked as 1, 2 and 3) of 21 intersections in Figure 7(a). The displacement between these 21 road network intersections and detected (corresponding) imagery intersections is shown with the arrows (Figure 7(b)). The thickest arrow is the vector median among these displacement vectors. After applying VMF, the eleven (half of the identified intersections) closest vectors to the vector median were kept. As shown in Figure 7(c), the three misidentified intersections are filtered out.

6. Related work

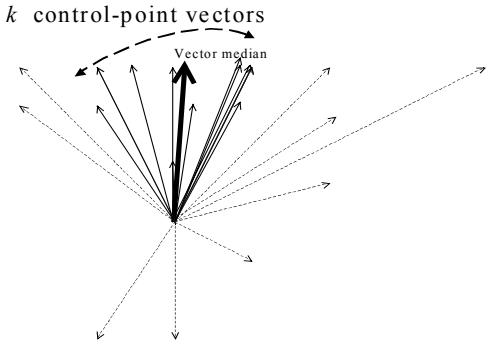
Automatic identification of road intersections is a complex procedure that utilizes work from a wide rage of subjects, such as knowledge discovery from metadata of vector and raster data, spatial (geometric) pattern recognition and digital image processing.

Many studies have been focussed on road or man-made object extractions from images [5, 11]. In particular, an on-road/off-road histogram is learned for black-white images in [12], while we deal with color images. Flavie et al. proposed techniques in [5] to find the junctions of all detected lines on images, then matched the extremities of the road vector with detected image junctions. Their method suffers from the high computation cost of finding all possible junctions. Road intersections are salient and useful features, particularly in solving matching problems, such as conflation [10, 13], GIS data correction [14] and imagery registration [15]. Most of the research activities in spatial data and GIS are centered around issues such as data representation, storage, indexing and retrieval. However, recent growth of the geospatial information on the web has made geospatial data conflation one of central issues in GIS [16]. In addition to efficiently storing and retrieving diverse spatial data, the users of these geo-spatial data products often want these different data sources displayed in some integrated fashion. Figure 8 shows the conflation results utilizing LTM detected intersections as control points and the conflation techniques proposed in [10]. Moreover, the intersections detected on images can match with the intersections detected on maps [6, 17] for the measure of the similarity of different spatial scenes. In addition, road intersections have been utilized for many transportation-related systems such as [7]. To the best of our knowledge, automatically exploiting these auxiliary structured data to improve the image recognition techniques has not been studied before.

7. Conclusion and future work

In this paper, we focus on the two commonly used spatial data storage and display structures: vector and raster.

(a). The intersection points (rectangles) on vector data and the corresponding intersection points (circles) on imagery.



(b). The distributions of twenty-one displacement vectors

(c). The intersections left after applying vector median filter on Figure 7(a).

Figure 7: VMF filter

Figure 8: Vector-Imagery conflation (White lines: original road network; Black lines: after applying conflation using identified intersections)

There have been a number of efforts to efficiently determine all intersection points from large number of line segments [18] of vector data (e.g., road networks), while there is little work on efficiently and accurately identifying road intersections from raster data (e.g., satellite imagery). The main contribution of this paper is the design and implementation of a novel approach to automatically identify and annotate road intersections on high resolution color orthoimagery. Our approach utilizes Bayes classifier, road network data and image metadata to detect road intersections. Although our histogram-based classifier requires extra operations to learn the conditional density functions, we can apply the learned results to automatically identify intersections of the area that is much larger than the area we learn from. Moreover, our approach is the first that exploits the metadata of imagery and vector data to take full advantage of all the available information from both datasets to achieve the automatic road intersection detection. We have also demonstrated the utility of our approach through several empirical experiments.

The accurately identified and annotated intersections can not only be utilized for geo-spatial data fusion, but can also be used for transportation, city planning and spatial data mining, etc. For example, the identified intersections on image are annotated with vector attributes, such as road names, road directions and zip codes. We can then build an approximate zip code map on the image, using these intersections and the technique proposed in [19]. In future, we plan to utilize the similar techniques to identify road intersections on maps.

8. Acknowledgement

This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), IIS-0238560 (CAREER), and IIS-0324955 (ITR), in part by the Air Force Office of Scientific Research under grant numbers F49620-01-1-0053 and FA9550-04-1-0105, in part by a gift from the Microsoft Corporation, and in part by a grant from the US Geological Survey (USGS). The U.S. Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

9. Reference

[1] Nevatia, R. and K. Price, *Automatic and Interactive Modeling of Buildings in Urban Environments from Aerial Images* . IEEE ICIP 2002, 2002. **III**: p. 525-528.

[2] Chen, C.-C., C.A. Knoblock, C. Shahabi, and S. Thakkar. *Building Finder: A System to Automatically Annotate Buildings in Satellite Imagery*. In *International Workshop on Next Generation Geospatial Information* 2003. Cambridge (Boston), Massachusetts, USA.

[3] Saalfeld, A., *Conflation: Automated Map Compilation*, in *Computer Vision Laboratory, Center for Automation Research*. 1993, University of Maryland.

[4] Hwang, J.-R., J.-H. Oh, and K.-J. Li. *Query Transformation Method by Delaunay Triangulation for Multi-Source Distributed Spatial Database Systems*. In *the Proceedings of the 9th ACM Symposium on Advances in Geographic Information Systems*. 2001.

[5] Flavie, M., A. Fortier, D. Ziou, C. Armenakis, and S. Wang. *Automated Updating of Road Information from Aerial Images*. In *American Society Photogrammetry and Remote Sensing Conference*. 2000.

[6] Habib, A., Uebbing, R., Asmamaw, A., *Automatic Extraction of Road Intersections from Raster Maps*. 1999, Center for Mapping, The Ohio State University.

[7] Yue, Y. and A.G.O. Yeh. *Determining Optimal Critical Junctions for Real-time Traffic Monitoring for Transport GIS*. In *The 11th International Symposium on Spatial Data Handling*. 2004. Leicester, United Kingdom.

[8] Forsyth, D.a.J.P., *Computer Vision : A Mordern Approach*. 2001: Prentice-Hall.

[9] Jones, M. and J. Rehg. *Statistical color models with application to skin detection*. In *IEEE Conf. on Computer Vision and Pattern Recognition*. 1999.

[10] Chen, C.-C., S. Thakkar, C.A. Knoblock, and C. Shahabi. *Automatically Annotating and Integrating Spatial Datasets*. In *the Proceedings of International Symposium on Spatial and Temporal Databases*. 2003. Santorini Island, Greece.

[11] Fortier, A., D. Ziou, C. Armenakis, and S. Wang, *Survey of Work on Road Extraction in Aerial and Satellite Images, Technical Report*. 1999.

[12] Xiong, D., *Automated Road Extraction from High Resolution Images*. 2001, U.S. Department of Transportation, University Consortium on Remote Sensing in Transportation.

[13] Cobb, M., M.J. Chung, V. Miller, H.I. Foley, F.E. Petry, and K.B. Shaw, *A Rule-Based Approach for the Conflation of Attributed Vector Data.* GeoInformatica, 1998. **2**(1): p. 7-35.

[14] Ubeda, T. and M.J. Egenhofer. *Topological Error Correcting in GIS*. In *the Proceedings of International Symposium on Spatial Databases*. 1997.

[15] DARE, P. and I. DOWMAN, *A new approach to automatic feature based registration of SAR and SPOT images.* IAPRS, 2000. **33**.

[16] Usery, E.L., M.P. Finn, and M. Starbuck. *Data Integration of Layers and Features for The National Map*. In *American Congress on Surveying and Mapping*. 2003. Phoenix, AZ.

[17] Chen, C.-C., C.A. Knoblock, C. Shahabi, and S. Thakkar. *Automatically and Accurately Conflating Satellite Imagery and Maps*. In *In the Proceedings of International Workshop on Next Generation Geospatial Information*. 2003. Cambridge (Boston), Massachusetts, USA.

[18] Chan, E.P.F. and J.N.H. Ng. *A General and Efficient Implementation of Geometric Operators and Predicates*. In *the Proceedings of International Symposium on Spatial Databases*. 1997.

[19] Sharifzadeh, M., C. Shahabi, and C.A. Knoblock. *Learning Approximate Thematic Maps from Labeled Geospatial Data*. In *In the Proceedings of International Workshop on Next Generation Geospatial Information*. 2003. Cambridge (Boston), Massachusetts, USA.

Distributed Spatial Data Warehouse Indexed with Virtual Memory Aggregation Tree

Marcin Gorawski Rafał Malczok

Institute of Computer Science
Silesian University of Technology
Akademicka 16
44-100 Gliwice
Poland
{M.Gorawski, R.Malczok}@polsl.pl

Abstract

In this paper we present a distributed spatial data warehouse system designed for storing and analyzing a wide range of spatial data. The data is generated by media meters working in a radio based measurement system. The distributed system is based on a new model called the cascaded star model. In order to provide satisfactory interactivity for our system, we used distributed computing supported by a special indexing structure called an aggregation tree. We provide a detailed description of distributed system components and their co-operation. The indexing structure operation is tightly integrated with the spatial character of the data and the data model. Thanks to a memory managing mechanism the system is very flexible in the field of aggregates accuracy. A final selective materialization of indexing structure fragments strongly increases the system's efficiency. Basing on the tests results, we compare the efficiencies of the distributed and a single-machine systems, and demonstrate the importance of indexing structure materialization.

1. Introduction

During the last 3 years deregulation of energy sectors in the USA and the EU has laid the foundations for a new energy market as well as created new customers for electrical energy, natural gas, home heating and water. The most crucial issue in this regard is the automated metering of utilities customers' usage and the fast analysis of terabytes of relevant data gathered thusly. In case of electrical energy providers, the reading, analysis, and decision-making is highly time sensitive. For example, in order to take stock of energy consumption all meters should be read and the data analyzed every thirty minutes [7]. This can be achieved by use of the technology called Automatic (Integrated) Meter Reading (AMR) in tandem with data warehouse-based Decision Support Systems (DSS) on the other[9]. Our solution consists of two layers (fig 1): the first is a telemetric system of integrated meter readings called AIUT_GSM and the second is Distributed Spatial Telemetric Data Warehouse (DSTDW).

The telemetric system of integrated meter readings is based on AMR and GSM/GPRS technology. The system sends the data from meters located in a huge geographical region to the telemetric server using the GSM mobile phone net utilizing GPRS technology. AIUT_GSM possesses all those features.

The Distributed Spatial Telemetric Data Warehouse System is a decision-making support system. The decisions concerning a given medium supply are made based on the short-term prognoses of a medium consumption. The prognoses are calculated using the analysis of the data gathered in DSTDW which, in turn, is supplied with data by the telemetric server. In order to meet the need for improved performance created by growing data sizes in Spatial Telemetric Data Warehouse (STDW), parallel processing and efficient indexing become increasingly important.

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

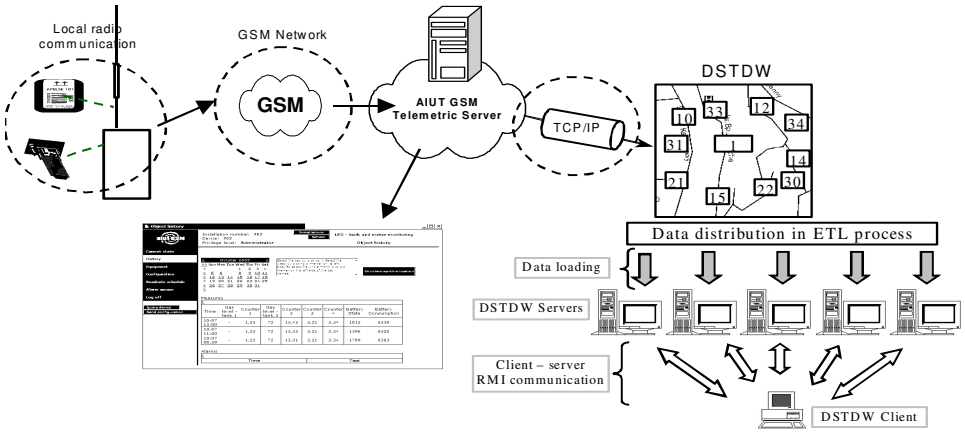


Figure 1. Two layers of a telemetric infrastructure.

2. System presentation

The telemetric system of integrated meter readings consists of meters located in a huge geographical region, collection nodes and headquarters housing the data warehouse and distributed processing structure. The collective nodes are the intermediate point of communication between the meters and the data warehouse. The nodes and meters are located in the same region. A single node is a collection point for a specified set of meters. The meters and nodes communicate via radio wave. Thus it features some restrictions – a single collective node can serve but a limited number of meters, and the distance between a node and a meter is limited. Those limits determine that the radio wave communication structure is akin to a set of circles, whose centers are collection nodes surrounded by meters. The system presented in this paper is real; it is a working system used for reading and storing data from media meters.

We designed our system of gathering, storing and analyzing telemetric data in two versions – centralized (single-machine) and distributed. Because the system is implemented in Java, we were able to separate packages common to both system versions. One of the packages is a Virtual Memory Aggregation Tree package, hence, the functionality of this solution is identical in both system versions.

The interactive system task is to provide a user information about utilities consumption in regions encompassed by the telemetric system. The interface used in both versions of our system relies on maps of the regions where the meters are located and is very similar to that presented in [8]. A user, using a "rubber rectangle", defines a piece of a region from which the aggregates are to be collected. That piece is called an aggregation window (fig 2). After setting the lists of aggregation windows and selecting those windows which are to be evaluated, a user starts a process of evaluating aggregation windows list value. A special algorithm splits the overlapping windows and transfers them to a function which evaluates the aggregates. The function is performed separately for every aggregation window. The obtained results are presented in form of both tables and graphs, the latter are generated by means of *JFreeChart* package.

Both centralized and distributed systems provide the same functionality, and for a user the system version is entirely transparent. However, according to the tests results, the distributed system runs much faster.

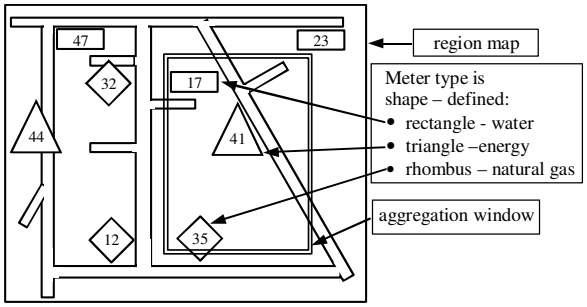


Figure 2. Map of region with aggregation window and meters.

2.1 Details of the distributed system

The current parallel ROLAP approaches can be grouped into two categories: a) work partitioning [13, 4, 5] and b) data partitioning [8, 2, 12, 6, 16, 3]. In this paper, we study data partitioning methods for the ROLAP case of STDW and implement the data distribution in a highly scalable DSTDW structure. In [8] the project creates and uses aggregation trees, requiring change to and develop of the RAID-3 algorithm-based concept of data warehouse stripping used in [2] and introduce two groups of distributing data algorithms applied in the new concept of distributed data. Using Java, we implemented both the distributed system using RMI [17] as well as the system running on a single computer.

The structure of the distributed system is based on the well known client-server standard. The distributed system client module manages the servers, integrates the partial results and provides the user interface.

The server module contains implementation of database communication and analytical functionality. During system operation the servers store data, generate the meter reading lists and build aggregation trees.

The communication between client and server is based on Remote Method Invocation (RMI) mechanism. The RMI allows Java language remote objects method invocation. RMI is making available of services through the registry. Each DSTDW server provides the following services:

- server management (checking, connecting, shutting down),
- extraction service (specially designed program using this service performs data distribution over the servers),
- access to the server's data base (connecting, disconnecting, query and DDL statements execution),
- access to server's aggregation trees (generating a tree of specified parameters, evaluating aggregates values for a given windows list).

The distributed system client is a multithread application; one thread is assigned to each server. The threads operation is coordinated by a main thread. A single server thread operation can be divided into the following phases:

- server localization. The server is described with name and port number. When the server is localized, the thread checks the connection using function similar to *ping* command broadly known from many operating systems,
- aggregation tree creation. The thread provides all necessary construction parameters and waits for a positive confirmation,
- connection checking. In case there are no aggregation tasks, the thread checks the connection, periodically invoking *ping* function,
- aggregates evaluating. After a user defined a set of aggregation windows, the overlapping windows are split and a copy of aggregation windows list is delivered to each distributed system server. A single server evaluates aggregates from its data and returns a partial result. Server thread transfers the result to the main thread. The result is integrated with other partial results and the final outcome is presented to a user in form of tables or graphs.

More detailed information about hardware system configuration can be found in a section describing tests.

3. Cascaded star

The data warehouse model consists of facts oriented in a set of dimensions, which in turn are organized in hierarchical aggregates levels. The models are adapted to relational data base environment. The most popular data models are star model and snowflake model.

As mentioned above, both versions of STDW use the cascaded star schema. The cascaded star was used for the first time by the authors of [1] for storing spatial data. It turned out to be useful in the Spatial Data Warehouse described in [8]. The necessity of cascaded star usage results from the fact that traditional schemas are not appropriate for storing spatial data [1].

A central point of the cascaded star is a main fact table. The main dimensions form smaller star schemas in which some dimension tables may become a fact table for other, nested star schemas. The cascaded star allows convenient modeling of a wide range of spatial data.

A cascaded star schema used in the presented system is illustrated in figure 3. The central fact table describes installation and interconnects three main dimensions storing information about maps, readings and meters, and collecting nodes.

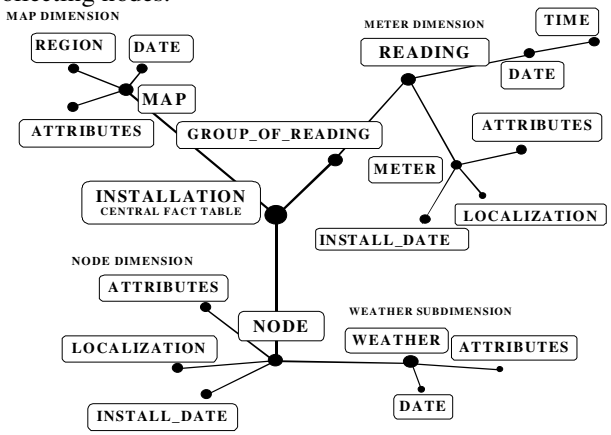


Figure 3. A schema of used data model – cascaded star

The first dimension, concerning maps, stores the information about maps of the region where the meters are located. Map files are described with the following information: type of map (vector, bitmap), encompassed region characteristics (geographical location, dimensions) and date of map creation, which permits exchanging obsolete maps for new ones.

The dimension storing node information contains a precise, three-dimensional location of each node, and parameters describing a node. As shown in figure 3, there is a subdimension with weather information such as humidity, temperature, and clouds. We use the data to analyze the influence of weather conditions on the utilities consumption.

The most loaded part of the schema stores the meter readings. A single reading sent from a meter to a collection node contains the following information: a precise timestamp, a meter identifier, and the reading values. The reading values vary depending on the type of meter. There are values of two zones for energy and water counters, and gas counters have a value of one zone.

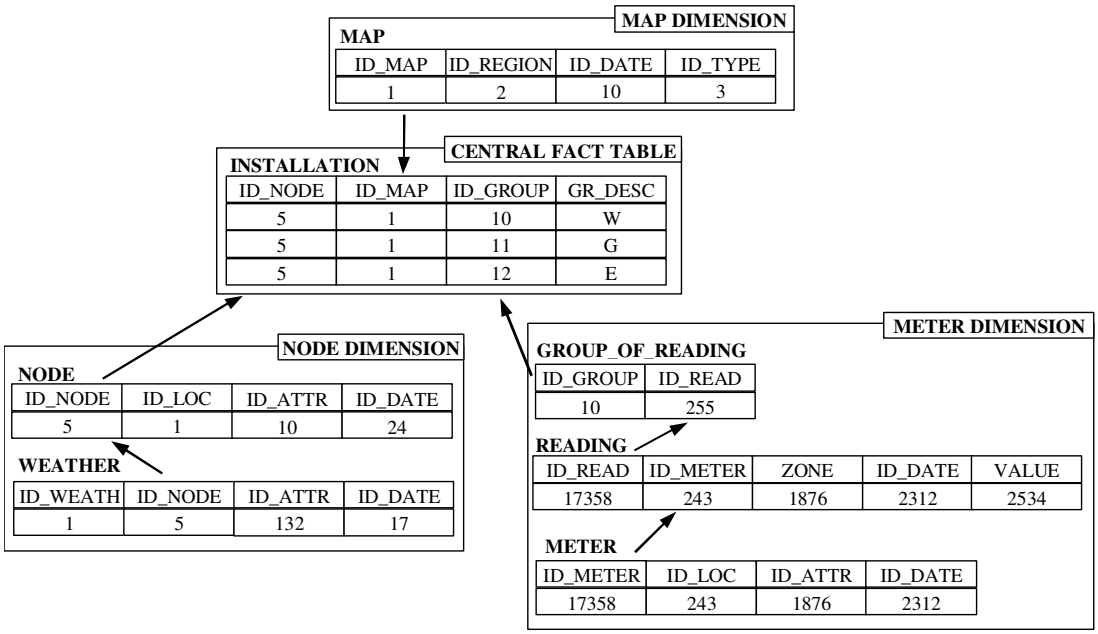


Figure 4. A hypothetical contents of the schema's tables located on a server gathering data of node

In order to connect the dimension with the central fact table we use the fact that each collective node serves a few types of meters. These types are described by the table GROUP_OF_READING that permits identifying a group of readings to which a given reading belongs, in other words, from which collective node it comes and which medium it concerns. The data concerning meters is gathered in the form of a subdimension of the meter readings dimension. As with collection nodes, a meter is described with three-dimensional location and a set of attributes like type or radio scope.

In order to facilitate an understanding of data connections in the applied cascaded star, we present the hypothetical contents of the schema tables (fig 4). That hypothetical schema exists on a server gathering data concerning node number 5.

3.1 Data distribution

Very important and requiring special attention is the problem of data distribution in the distributed system. In our system the data is distributed according to the following rules:

- each server uses the same identical cascaded star schema (presented in figure 3),
- we assign a set of collective nodes to a given server. The sets of nodes should contain an approximately equal number of meters in order to evenly balance the data storing and processing load,
- the data are distributed over the servers according to the node they concern. That is true for METER dimension and WEATHER subdimension. Data from MAP dimension are replicated on every server. This dimension does not contain much information; such approach brings almost no extra costs.

Our approach to data distribution is very simplified, and we are aware that advanced studies on the problem of data and load balancing are a pressing need. We are now investigating an approach to data balancing during the extraction process. We want to estimate the distributed system units efficiency and basing on this we want to distribute the incoming measure data. Although the results are very interesting, they are too tentative to be published.

4. Aggregation tree

The key aspect of data warehouse designing is to provide it with sufficient efficiency. One of the best ways to improve a system's efficiency is to use indexing and data pre-aggregation. STDW in both versions uses, next to the standard data base indices, a spatial index called an aggregation tree. The aggregation tree is a dynamic structure containing aggregates of spatial data. Its structure is based on the most popular aggregates index, the aR-tree [14]. As concerns efficiency, the aR-Tree significantly surpasses the R-Tree during the process of evaluating the value of range queries [10, 11, 15].

The structure of the aggregation tree is integrated with the cascaded star data model. We designed the index in such a way that it contains the aggregates of meter readings lightening the load of the most loaded part of the data model. Thanks to said approach, the index speeds up the execution of the most commonly evaluated queries concerning utility usage in a given region.

In order to save the time spent on the raw data processing we decided to introduce an innovative idea of partial aggregates materialization. We called the index

Virtual Memory Aggregation Tree (VMAT), because the aggregation tree uses the database as a place where the aggregates are stored. Thanks to this solution we are able to overcome the RAM memory limitations when indexing huge amounts of data. The tests proved that VMAT works effectively even with small amounts of memory.

The data is distributed in our system. Each server manages its own aggregation tree that indexes the data located on the server. Below we present how the aggregation tree is built and materialized and how the queries are evaluated in our system.

4.1 Aggregation tree structure

As in the case of aR-tree, the aggregation tree consists of connected nodes. The whole region encompassed by the data warehouse is divided into units called *Minimal Bounding Rectangle* (MBR). In the proposed approach MBR dimensions are defined by a system user when the system starts. Thanks to this solution our system is very flexible as far as the aggregates precision is concerned (a user may define MBRs as small as desired). The tree leaves (the lowest level) are nodes encompassing regions that are in MBR size. Upper tree levels are created through integration of a few elements from lower level into one node in the upper level. Integration operation includes merging regions of the lower level nodes and aggregation of their values, aggregates lists and quantity of meters both. In figure 5 we presented a very simplified scheme of the aggregation tree indexing data from a region that was divided as shown also in the figure. Also shown is the hypothetical contents of three tree nodes, illustrating the details of the integration operation.

4.2 Aggregates collecting

In our system the queries are defined in the form of aggregation windows. Using the client module, a user draws an aggregation window selecting the region from which the aggregates are to be collected. After setting a list of aggregation windows, the overlapping windows are split and a sent to the distributed system servers.

Aggregation collecting operations are performed for each aggregation window separately. The aggregates collection algorithm, operating identically on each server, starts from the tree root and proceed towards the tree leaves. For each node the algorithm checks whether the node's region has a common part with the aggregation window's region and, based on the result, the algorithm determines the proper action. If the regions share no part, the node is skipped. In case the regions overlap a bit, the algorithm proceeds recursively to the lower tree level (but only if the level considered is not the lowest level) as a window region assuming the overlapping part. In case the regions entirely overlap, then the algorithm aggregates the given node's aggregates lists. The aggregation operation consists of retrieving the node's aggregates lists and aggregates them to the lists of a global collecting element.

When all the aggregation windows are evaluated, the partial results are sent to the client module, which merges them and presents to a user.

4.3 Materialization

As mentioned above, in our solution we introduced the idea of materialization of data stored in the tree nodes. The materialization is performed independently on each server and is managed by the aggregation tree.

To store materialized data the aggregation tree uses a table with two columns: the first identifies node and the second stores the data. During the materialization a given tree node retrieves a handle to an input stream of a BLOB column from a row identified by its identification number. The handle is then transferred to all aggregates objects, which stores their values to the stream (fig 6).What nodes should be materialized and when is determined by a specially designed algorithm described below.

We materialize only the data and not the tree frame because creation of an empty tree structure (i.e., nodes not containing aggregates lists) is an operation of short duration and is performed each time the tree object is created with almost no extra time cost. When a query is evaluated, the aggregation tree refers to the table containing materialized nodes. In the event that a query concerning the same region was already evaluated, the data was materialized and then can be easily restored.

The operation of materialization aggregates lists is initialized by the aggregation tree object. There are two situations when tree nodes are materialized. The first is when the application is being shut down – all the not-materialized nodes are stored in the database.

The second situation results from the operation of memory managing algorithm during the action of aggregates lists retrieval.

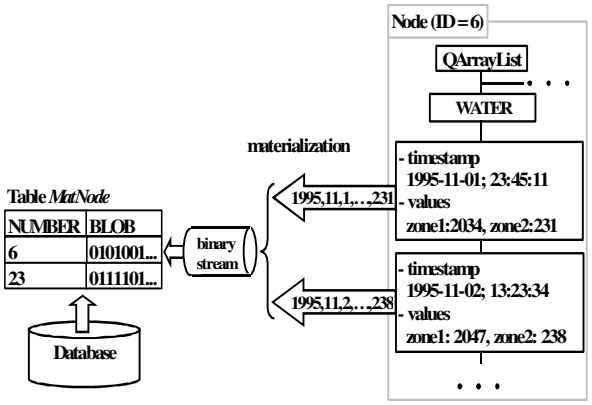


Figure 6. A scheme of aggregates lists materialization operation

Every node has a reading counter. When the node's aggregates lists are being retrieved, the reading counter increases. If a memory managing algorithm recognizes there is not enough memory to store the aggregation tree,

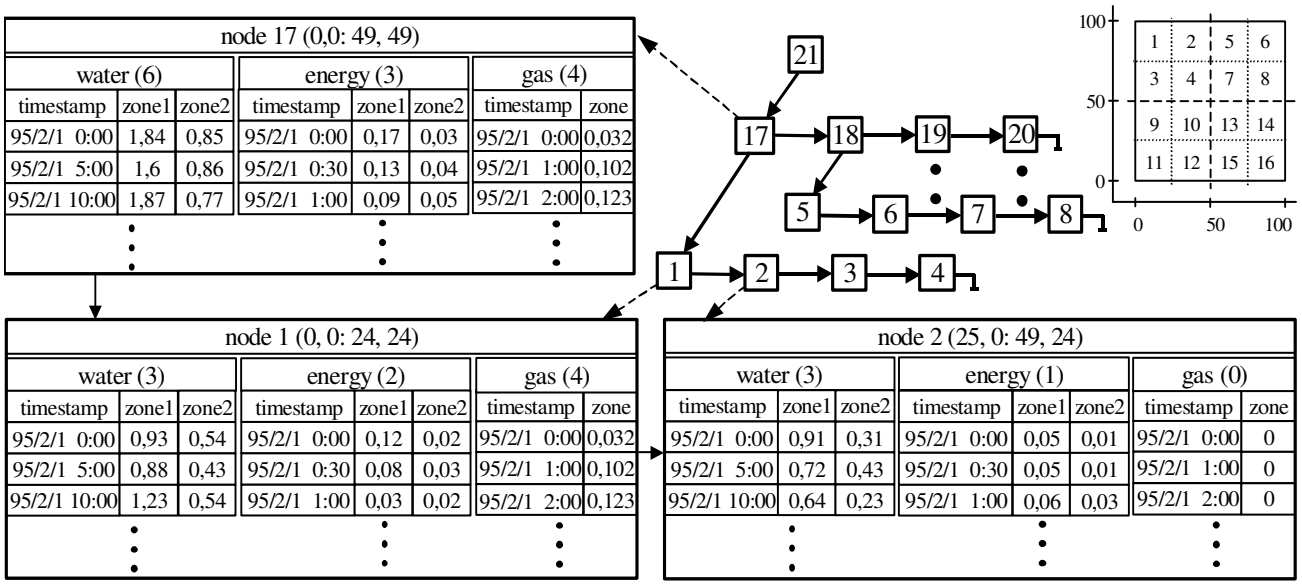


Figure 5. A simplified scheme of aggregation tree and the related nodes with their hypothetical contents

it searches for a node whose aggregates lists were least frequently read (the reading counter has the least value). If the lists have not been stored in the database, the node's aggregates lists are materialized and removed from the memory. At the beginning of the operation of aggregates collection, the nodes, from which the aggregates lists will be retrieved, are marked, to preclude the removal from the memory of those nodes' aggregates lists due to be collected in the next step.

5. Tests

We performed the tests of the DSTDW on a set of six computers comprising the distributed structure. Five computers run the server module. The DSTDW client module runs separately on the additional machine. System configuration was as follows: servers Pentium 4 2.8 GHz, 512 MB RAM. The client machine was AMD Athlon 2GHz with 1024 MB RAM. Tests of the STDW were run on the latter machine. The computers were connected with a 100 Mb/s LAN.

The data base was Oracle 9i; the runtime environment was Java Sun 1.4.0. The data gathered in the data base concerned 20 collection nodes (number of meters exceeded 1000). We were collecting the weather information and meters readings for over four months (from the 1 January to the 1 May of 1995). Each meter communicates with its collective node in an appropriate frequency. This frequency differs according to type of meter. Electricity meters are read every thirty minutes, gas meters every two hours, and water meters every five hours. Size of the not-distributed data gathered in the data base exceeded 0.5 GB; in the MEASURES table, the number of rows was about 10 millions.

The data was evenly distributed. We assigned a set of collection nodes to every server (4 nodes per server). Then the data (weather information, meters and their measurements) were distributed according to the node they were connected to. Aggregation windows determining the region from which the aggregates were to be evaluated were placed on the whole region encompassed by the data warehouse, hence the servers were evenly loaded. In the tested system a single collection node served only about 50 meters of various kinds. We performed the tests for the following sets of meters: 40, 80 and 160 in the aggregation periods one, two, three and four months.

The tests were performed for various percentages of materialized aggregates. First, we were executing queries when the table storing the materialized information was empty. We then executed the same queries with a full set of materialized data. In the next steps, using a simple program we removed 75%, 50% and 25% of materialized aggregates lists. By this we were imitating the normal system operation, when the number of materialized aggregates lists increases with the number of executed queries.

The measured values were: query execution time (time of evaluating the value of a list of aggregation windows) and the number of lists materialized during query execution.

We present the tests results showing trends in the distributed system operation. Also, the distributed and centralized system versions efficiency is compared.

In figure 7 we present a graph showing distributed system operation while aggregating measures data for over 80 meters. The graph shows the relation between aggregation time and amount of materialized aggregates available in the data base. We can observe a strong linear decrease of aggregation time. More detailed analysis reveal that the decrease is a bit slighter between 50% and 75% of available materialized information. Of special

notice, the aggregation times for various aggregation periods are much more congenial for 100% of materialized aggregates than when the system used no materialized information. Hence we hypothesize that indexing structure materialization strongly increases system's efficiency and improves scalability.

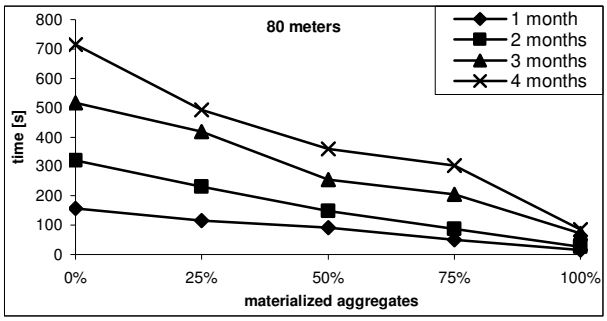


Figure 7 . Distributed system operation while aggregating measures data from over 80 meters.

In figure 8 we present a graph showing the materialized aggregates lists number while evaluating aggregates for over 160 meters. The graph shows that the number of materialized aggregates grows quickly with extension of the aggregation period. We have to stress that the aggregates list materialization and its further restoring is a very short-lasting operation in comparison with the raw data processing and imposes relatively small overhead.

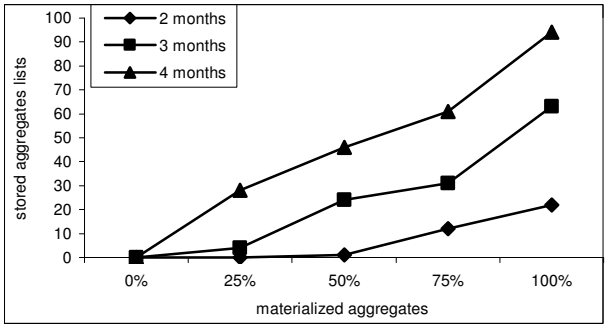


Figure 8. Number of materialized lists while evaluating aggregates for over 160 meters

Figure 9 presents a graph comparing the distributed and centralized system measures aggregation time. The aggregations were performed for more than 160 meters; the aggregation periods extend from one to four month. The graph shows that the distributed system significantly surpasses the centralized version in terms of efficiency.

The system distribution results in significant speedup of aggregating operations. The speedup is very close to linear (a linear coefficient *N* is number of servers in the distributed system) where there are no materialized aggregates in the data base and the system must process the raw data.

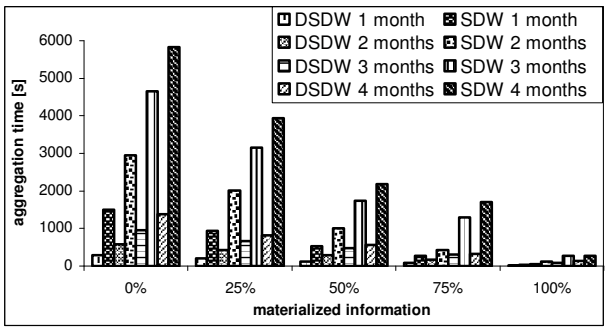


Figure 9. Aggregation times comparison for distributed and centralized system (160 meters)

The more aggregates are materialized the less is the benefit of distribution. This result shows the importance of indexing structure materialization.

6. Conclusions and future work plans

In this paper we give a description of a telemetric system of integrated meter readings and present in detail a distributed spatial telemetric data warehouse system designed for storing and analyzing a wide range of spatial data. We presented both the centralized and distributed structures of our system. The data is generated by media meters working in a radio wave-based measurement system. The STDW works with the new data model called cascaded star model. The cascaded star model allows efficient storing and analyzing huge amounts of spatial data whose range is very wide and extends from meter measurement values to weather information.

The paper contains detailed description of data distribution process as well as the whole distributed system operation. In order to provide satisfactory interactivity for our system, in addition to distributed structure we used a special indexing structure called an aggregation tree. Its structure and operation is tightly integrated with the spatial character of the data. Thanks to a memory managing mechanism the system is very flexible in the field of aggregates accuracy. We called the index Virtual Memory Aggregation Tree (VMAT), the reason being the aggregation tree intensively uses a disk to store the aggregates. The approach using VMAT allows the overcoming of available RAM memory limitations when indexing huge amounts of data. The tests proved that VMAT works effectively even with small amounts of memory. A final selective materialization of indexing structure fragments performed by combining the Java streams and the Oracle BLOB table column strongly increases the system's efficiency.

Tests results show that by means of aggregation tree materialization the system's efficiency can be greatly increased. Another positive outcome of materialization is, the greater the number of aggregation actions performed, the faster the system works, because more nodes were

materialized. Data and workload distribution is another way to improve the system efficiency.

Our future plans call for implementation of updating the aggregation tree contents algorithm and research in the field of distributed data warehouse solutions involving dynamic load balancing.

References

[1] Adam, N., Atluri, V., Yesha, Y., Yu, S. Efficient Storage and Management of Environmental Information, *IEEE Symposium on Mass Storage Systems*, 2002.

[2] Bernardino, J., Madera, H. Data Warehousing and OLAP: Improving Query Performance Using Distributed Computing. *CAiSE*00 Conference on Advanced Information Systems Engineering*, Sweden, 2000

[3] Chen, Y., Dehne, F., Eavis, T., Rau-Chaplin, A. Parallel ROLAP Data Cube Construction On Shared-Nothing Multiprocessor.Proc. 2003 *International Parallel and Distributed Processing Symposium* (IPDPS 2003),France, 2003.

[4] Datta, A., VanderMeer, D., Ramamritham, K. Parallel Star Join + DataIndexes: Efficient Query Processing in Data Warehouses and OLAP. *IEEE Trans. Knowl. Data Eng.* 2002

[5] Dehne, F., Eavis, T., Hambrusch, S., Rau-Chaplin, A. Parallelizing the data cube. *Distributed and Parallel Databases*, 11(2), 2002.

[6] Dehne, F., Eavis, T., Rau-Chaplin, A. Parallel Multi-Dimensional ROLAP Indexing. Proc. 3rd IEEE/ACM *International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, Japan, 2003.

[7] Gorawski, M. Modeling the intelligent systems for strategic management. *4th Conf. MSK*, Kraków 2003

[8] Gorawski, M., Malczok, R. Distributed Spatial Data Warehouse. *5th Int. Conf. on Parallel Processing and Applied Mathematics, PPAM2003*, Częstochowa, SpingerVerlag, LNCS3019.

[9] Han, J., Stefanovic, N., Koperski, K. Selective Materialization: An Efficient Method for Spatial Data Cube Construction. In Research and Development in Knowledge Discovery and Data Mining, *Second Pacific-Asia Conference, PAKDD'98*, 1998.

[10] Jurgens, M., Lenz, H. The Ra*-tree: An Improved R-tree with Materialized Data for Supporting Range Queries on OLAP-Data. *DEXA Workshop*, 1998

[11] Lazaridis, I., Mehrotra, S., Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure. *SIGMOD*, 2001.

[12] Märtens, H., Rahm, E., Stöhr, T. Dynamic Query Scheduling in Parallel Data Warehouses *Euro-Par,* Germany 2002.

[13] Muto, S., Kitsuregawa, M. A DynamicLoad Balancing Strategy for Parallel Datacube Computation. *DOLAP* 1999.

[14] Papadias, D., Kalnis, P., Zhang, J., Tao, Y. Efficient OLAP Operations in Spatial Data Warehouses. Proceedings of the *7th International Symposium on Spatial and Temporal Databases, (SSTD)*, Spinger Verlag, LNCS 2001.

[15] Rao, F., Zhang, L., Yu, X., Li, Y., Chen, Y., Spatial Hierarchy and OLAP - Favored Search in Spatial Data Warehouse. *DOLAP*, Louisiana,2003

[16] Rohm, U., Bohm, K., Schek, H.J., Schuldt, H. FAS- a Freshness-Sensitive Coordination Middleware for a Cluster of OLAP Component. Proceedings of the Conference on *Very Large Databases (VLDB),* Hong Kong, 2002.

[17] Java™ 2 SDK, Standard Edition, Documentation, Sun Microsystems, 2002.

Continuous K Nearest Neighbor Queries in Spatial Network Databases

Mohammad R. Kolahdouzan and Cyrus Shahabi

Department of Computer Science
University of Southern California
Los Angeles, CA, 90089
kolahdoz,shahabi@usc.edu

Abstract

Continuous K nearest neighbor queries (C-KNN) are defined as the nearest points of interest to all the points on a path (e.g., continuously finding the three nearest gas stations to a moving car). The result of this type of query is a set of intervals (or split points) and their corresponding KNNs, such that the KNNs of all objects within each interval are the same. The current studies on C-KNN focus on Euclidean spaces. These studies are not applicable to spatial network databases (SNDB) where the distance between two objects is defined as the length of the shortest path between them. In this paper, we propose two techniques to address C-KNN queries in SNDB: Intersection Examination (IE) and Upper Bound Algorithm (UBA). In the IE, we first find the KNNs of all nodes on a path and then, for those adjacent nodes whose nearest neighbors are different, we find *split points* between them and compute the KNNs of the split points using the KNNs of the nodes. The intuition behind UBA is that the performance of IE can be improved by determining the adjacent nodes that cannot have any split points in between, and consequently eliminating the computation of KNN queries for those nodes. Our empirical experiments show that the UBA approach outperforms IE, specially when the points of interest are sparsely distributed in the network.

1 Introduction

The problem of K nearest neighbor (KNN) queries in spatial databases have been studied by many researchers. This type of query is frequently used in Geographical Information Systems (GIS) and is defined as: given a set of spatial objects and a query point, find the K closest objects to the query. An example of KNN query is a query initiated by a GPS device in a vehicle to find the five closest restaurants to the vehicle. Different variations of KNN queries are also introduced. One variation is the *continuous KNN* query which is defined as the KNNs of *any* point on a given path. An example of continuous KNN is when the GPS device of the vehicle initiates a query to continuously find the five closest restaurants to the vehicle at any point of a given path from a source to a destination. The result of this type of query is a set of intervals, or split points, and their associated KNNs. The split points specify where on the path the KNNs of a moving object will change, and the intervals (bounded by the split points) specify the locations that the KNNs of a moving object remains the same. The challenge in this type of query is to efficiently specify the location and the KNNs of the split points.

The majority of the existing work on KNN queries and its variations are aimed at Euclidean spaces, where the path between two objects is the straight line connecting them. These approaches are usually based on utilizing index structures. However, in spatial network databases (SNDB), objects are restricted to move on pre-defined paths (e.g., roads) that are specified by an underlying network. This means that the shortest network path/distance between the objects (e.g., the vehicle and the restaurants) depend on the connectivity of the network rather than the objects' locations. Index structures that are designed for spaces where the distance between objects is only a function of their spatial attributes (e.g., Euclidean distance), cannot properly approximate the distances in SNDB and hence the solutions that are based on index structures cannot be extended to SNDB.

We proposed [4] a Voronoi based approach, VN^3, to efficiently address regular KNN queries in SNDB. The VN^3 has two major components, network Voronoi

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

polygons (NVP) for each point of interest, and the pre-computed distances between the border points of each polygon to the points inside the polygon. The VN^3 approach provides the result set in an incremental manner and it works in two steps: the filter step uses the first component to generate a candidate set, and the pre-computed component is used in the refinement step to find the distances between the query objects and the candidates, and hence refine the candidates.

In this paper, we propose several approaches to address continuous KNN queries in SNDB. Depending on the number of neighbors requested by a CNN query, we divide the problem into two cases. When only the first nearest neighbor is requested (e.g., finding only the closest restaurant to a vehicle on a given path), our solution relies entirely on the properties of VN^3. We show that the split points on the path are simply the intersections of the path with the NVPs of the network, which are a subset of the border points of the NVPs.

We propose two solutions for the cases when more than one neighbor is requested by the CNN query (i.e., C-KNN). The main idea behind our first approach is that the KNNs of any object on a path between two adjacent nodes (e.g., intersections in road system) can only be a subset of any point(s) of interest (e.g., restaurants) on the path, plus the KNNs of the end nodes. Hence, the first solution is based on breaking the entire path to smaller segments, where each segment is surrounded by two adjacent nodes, and finding the KNNs of all nodes. We then show that for two adjacent nodes with different KNNs, by specifying whether the distances from a query object to the KNNs of the nodes will be increasing or decreasing as the object moves, we can find the location of the split points between the two nodes. The intuition behind our second solution is that if an object moves slightly, its KNNs will probably remain the same. Our second approach is then based on finding the minimum distance between two subsequent nearest neighbors of an object, only when the two neighbors can have a split point between them. This distance specifies the minimum distance that the object can move without requiring a new KNN query to be issued. Our empirical experiments show that the second approach always outperforms the first solution. To the best of our knowledge, the problem of continuous K nearest neighbors is spatial network databases has not been studied.

The remainder of this paper is organized as follows. We review the related work on regular and continuous nearest neighbor queries in Section 2. We then provide a review of our VN^3 approach that can efficiently address KNN queries in SNDB in Section 3. In Section 4, we discuss our approaches to address continuous KNN queries. Finally, we discuss our experimental results and conclusions in Sections 5 and 6, respectively.

2 Related Work

The regular K nearest neighbor queries have been extensively studied and for which numerous algorithms have been proposed. A majority of the algorithms are aimed at m-dimensional objects in Euclidean spaces, and are based on utilizing one of the variations of multidimensional index structures. There are also other algorithms that are based on computation of the distance from a query object to its nearest neighbors online and per query. The regular KNN queries are the basis for several variations of KNNs, e.g., continuous KNN queries. The solutions proposed for regular KNN queries are either directly used, or have been adapted to address the variations of KNN queries. In this section, we review the previous proposed solutions for regular and continuous KNN queries.

The regular KNN algorithms that are based on index structures usually perform in two filter and refinement steps and their performance depend on their selectivity in the filter step. Roussopoulos et al. in [8] present a branch-and-bound R-tree traversal algorithm to find nearest neighbors of a query point. The main disadvantage of this approach is the depth-first traversal of the index that incurs unnecessary disk accesses. Korn et al. in [5] present a multi-step k-nearest neighbor search algorithm. The disadvantage of this approach is that the number of candidates obtained in the filter step is usually much more than necessary, making the refinement step very expensive. Seidl et al. in [9] propose an optimal version of this multi-step algorithm by incrementally ranking queries on the index structure. Hjaltason et al. in [2] propose an incremental nearest neighbor algorithm that is based on utilizing an index structure and a priority queue. Their approach is optimal with respect to the structure of the spatial index but not with respect to the nearest neighbor problem. The major shortage with all these approaches that render them impractical for networks is that the filter step of these approaches performs based on Minkowski distance metrics (e.g., Euclidean distance) while the networks are metric space, i.e. the distance between two objects depends on the connectivity of the objects and not their spatial attributes. Hence, the filter step of these approaches cannot be used for, or properly approximate exact distances in networks. Papadias et al. in [7] propose a solution for SNDB which is based on generating a search region for the query point that expands from the query, which performs similar to Dijkstra's algorithm. Shekar et al. in [10] and Jensen et al. in [3] also propose solutions for the KNN queries in SNDB. These solutions are based on computing the distance between a query object and its candidate neighbors on-line and per query. Finally, in [4], we propose a novel approach to efficiently address KNN queries in SNDB. The solution is based on the first order network Voronoi diagrams and the result set is generated incrementally.

Sistla et al. in [11] first identify the importance of the continuous nearest neighbors and describe modeling methods and query languages for the expression of these queries, but did not discuss the processing methods. Song et al. in [12] propose the first algorithms for CNN queries. They propose fixed upper bound algorithm that specifies the minimum distance that an object can move without requiring a new KNN to be issued. They also propose a dual buffer search method that can be used when the position of the query can be predicted. Tao et. al in [13] present a solution that is based on the concept of time parameterized queries. The output of this approach specifies the current result of the CNN query, the expiration period of the result, and the set of objects that will effect the results after the expiration period. This approach provides the complete result set in an incremental manner. Tao et al. in [14] propose a solution for CNN queries based on performing one single query for the entire path. They also extend the approach to address C-KNN queries. The main shortcoming of all of these approaches is that they are designed for Euclidean spaces and utilize a spatial index structure, hence they are not appropriate for SNDB. Finally, Feng et al. in [1] provide a solution for C-NN queries in road networks. Their solution is based on finding the locations on a path that a NN query must be performed at. The main shortcoming of this approach is that it only addresses the problem when the first nearest neighbor is requested (i.e., continuous 1-NN) and does not address the problem for continuous K-NN queries. To the best of our knowledge, the problem of continuous K nearest neighbor queries in spatial network database has not been studied.

3 Background: VN^3

Our proposed solutions to address continuous KNN queries utilize the VN^3 approach to efficiently find the KNNs of an object. The VN^3 approach is based on the concept of the *Voronoi diagrams*. In this section, we start with an overview of the principles of the network Voronoi diagrams. We then discuss our VN^3 approach ([4]) to address KNN queries in spatial network databases. A thorough discussion on Voronoi diagrams and VN^3 are presented in [6] and [4], respectively.

3.1 Network Voronoi Diagram

A Voronoi diagram divides a space into disjoint polygons where the nearest neighbor of any point inside a polygon is the generator of the polygon. Consider a set of limited number of points, called *generator points*, in the Euclidean plane. We associate all locations in the plane to their closest generator(s). The set of locations assigned to each generator forms a region called *Voronoi polygon* (VP) of that generator. The set of Voronoi polygons associated with all the generators is called the Voronoi diagram with respect to the generators set. The Voronoi polygons that share the same edges are called *adjacent polygons*. A network Voronoi diagram, termed *NVD*, is defined for (directed or undirected) graphs and is a specialization of Voronoi diagrams where the location of objects is restricted to the links that connect the nodes of the graph and distance between objects is defined as their shortest path in the network rather than their Euclidean distance. A network Voronoi polygon $NVP(p_i)$ specifies the links (of the graph), or portions of the links, that p_i is closest point of interest to any point on those links. A *border point* of two NVPs is defined as the location where a link crosses one NVP in to the other NVP.

3.2 Voronoi-Based Network Nearest Neighbor: VN^3

Our proposed approach to find the K nearest neighbor queries in spatial networks [4], termed VN^3, is based on the properties of the Network Voronoi diagrams and also *localized* pre-computation of the network distances for a very small percentage of neighboring nodes in the network. The intuition is that the NVPs of an NVD can directly be used to find the first nearest neighbor of a query object q. Subsequently, NVPs' adjacency information can be utilized to provide a candidate set for other nearest neighbors of q. Finally, the pre-computed distances can be used to compute the actual network distances from q to the generators in the candidate set and consequently refine the set. The filter/refinement process in VN^3 is iterative: at each step, first a new set of candidates is generated from the NVPs of the generators that are already selected as the nearest neighbors of q, then the pre-computed distances are used to select "only the next" nearest neighbor of q. VN^3 consists of the following major components:

1. Pre-calculation of the solution space: As a major component of the VN^3 filter step, the NVD for the points of interest (e.g., hotels, restaurants,...) in a network must be computed and its corresponding NVPs must be stored in a table.
2. Utilization of an index structure: In the first stage of the filter step, the first nearest neighbor of q is found by locating the NVP that contains q. This stage can be expedited by using a spatial index structure generated on the NVPs.
3. Pre-computation of the exact distances for a very small portion of data: The refinement step of VN^3 requires that for each NVP, the network distances between its border points be pre-computed and stored. These pre-computed distances are used to find the network distances across NVPs, and from the query object to the candidate set generated by the filter step.

Our empirical experiments shows that VN^3 outperforms the only other proposed approach ([7]) for KNN queries in SNDB by up to one order of magnitude.

They also show that the size of the candidate set generated by our proposed filter step is smaller than that of the approaches designed for spatial index structure.

4 Continuous Nearest Neighbor Queries

Continuous nearest neighbor queries are defined as determining the K nearest neighbors of any object on a given path. An example of this type of query is shown in Figure 2 where a moving object (e.g., a car) is traveling along the path (A, B, C, D) (specified by the dashed lines) and we are interested in finding the first 3 closest restaurants (restaurants are specified in the figure by $\{r_1, ..., r_8\}$) to the object at any given point on the path. The result of a continuous NN query is a set of *split points* and their associated KNNs. The split points specify the locations on the path where the KNNs of the object change. In other words, the KNNs of any object on the segment (or interval) between two adjacent split points is the same as the KNNs of the split points. The challenge for this type of query is to efficiently find the location of the split point(s) on the path. The current studies on continuous NN queries are focused on spaces where the distance function between two objects is one of the Minkowski distance metrics (e.g., Euclidean). However, the distance function in spatial network databases is usually defined as their shortest path (or shortest time) which has a computationally complex function. This renders the approaches that are designed for Minkowski distance metrics, or the ones that are based on utilization of vector or metric spatial index structures, impractical for SNDB.

In this section, we discuss our solutions for C-KNN queries in spatial network databases. We first present our approach for the scenarios when only the first NN is desired (i.e., C-NN), and then discuss two solutions for the cases where the KNN of any point on a given path are requested.

4.1 Continuous 1-NN Queries

Our solution for CNN queries, when only the first nearest neighbor is requested, is based on the properties of VN^3. As we showed in Section 3, a network Voronoi polygon of a point of interest p_i specifies all the locations in space (space is limited to the roads in SNDB), where p_i is their nearest neighbor. Hence, in order to specify the C-NN of a given path, we can first specify the intersections of the path with the NVPs of the network. Subsequently, we can conclude that p_i is the C-NN of the segments of the path that are contained in $NVP(p_i)$. Note that this approach cannot be extended to C-KNN queries since the polygons are first order NVPs, i.e., they can only specify the first nearest neighbor of an object.

For example, assume the network Voronoi diagram shown in Figure 1 where $\{p_1, ..., p_7\}$ are the points of interest. As depicted in the figure, the path from S to

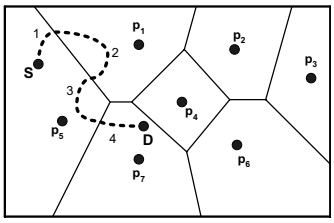


Figure 1: Continuous first nearest neighbor

D crosses $NVP(p_1)$, $NVP(p_5)$ and $NVP(p_7)$ and can be divided to 4 segments. We can conclude that p_5, p_1, and p_7 are the first nearest neighbors of any point on segments $\{1, 3\}$, $\{2\}$, and $\{4\}$, respectively.

4.2 Continuous KNN Queries

In this section, we discuss two approaches to address continuous KNN queries. Our first solution, IE, is based on examining the KNNs of all the nodes on a path, while our second approach, UBA, eliminates the KNN computation for the nodes that cannot have any split points in between.

4.2.1 Intersection Examination: IE

Our first approach to address continuous KNN queries in SNDB is based on finding the KNNs of the intersections on the path. We describe the intuition of our IE approach by defining the following properties:

Property 1: Let $p(n_i, n_j)$ be the path between two adjacent nodes n_i and n_j, o_1 and o'_1 be the first nearest points of interest (or neighbors) to n_i and n_j, respectively, and assume that $p(n_i, n_j)$ includes no point of interest, then the first nearest point of interest to *"any"* object on $p(n_i, n_j)$ is either o_1 or o'_1.

Proof: This property can be easily proved by contradiction. Assume that the nearest point of interest to a query object q on $p(n_i, n_j)$ is $o_k \notin \{o_1, o'_1\}$. We know that the shortest path from q to o_k, $p(q, o_k)$, must go through either n_i or n_j. Suppose $p(q, o_k)$ goes through n_i and hence $distance(q, o_k) = distance(q, n_i) + distance(n_i, o_k)$. However, we know that $distance(n_i, o_k) > distance(n_i, o_1)$ since o_1 is the first nearest point of interest to n_i and hence its distance to n_i is smaller than the distance of any other point of interest to n_i. Subsequently, we can conclude that $distance(q, o_k) > distance(q, o_1)$ which means that o_1 is closer to q than o_k, contradicting our initial assumption.

As an example, this property suggests that in Figure 2, the first nearest restaurant to any point between A and B can be either r_1 or r_3 since r_1 and r_3 are the first nearest neighbors of A and B respectively.

Property 2: Let $p(n_i, n_j)$ be the path between two adjacent nodes n_i and n_j, $O = \{o_1, ..., o_k\}$ and $O' = \{o'_1, ..., o'_k\}$ be the k nearest points of interest to n_i and n_j, respectively, and assume that $p(n_i, n_j)$ includes no point of interest, then the k nearest points of interest to *"any"* object on $p(n_i, n_j)$ is a subset of $\{O \bigcup O'\}$.

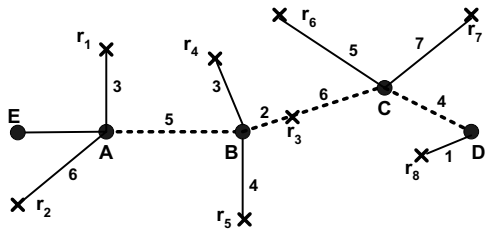


Figure 2: Example of continuous K nearest neighbor query

Proof: This property is in fact the extension of property 1 and can be similarly proved by contradiction.

As an example, this property suggests that in Figure 2, since the three nearest restaurants to A and B are $\{r_1, r_2, r_3\}$ and $\{r_3, r_4, r_5\}$, respectively, the three nearest restaurants to any object between A and B can only be among $\{r_1, r_2, r_3, r_4, r_5\}$.

From the above properties, we can conclude that:

- If two adjacent nodes have similar KNNs, every object on the path between the nodes will have similar KNNs as the nodes, meaning that there will be no split points on the shortest path between the nodes. That is simply because O and O' (in property 2) are the same and hence, $\{O \bigcup O'\} = O = O'$.
- In order to find the continuous KNN of any point on a path, we can first break the path into smaller segments where each segment obeys the above properties. We then find the continuous KNN for each segment, and finally, the union of the results for all segments generates the result set for the entire path.

The above properties are not valid for a path p if p includes one or more points of interest (e.g., the path between B and C in Figure 2 which includes r_3). We can address this issue in two alternative ways: 1) breaking p to smaller segments where each segment does not include any points of interest. For example, we can break the path (B, C) of Figure 2 to (B, r_3) and (r_3, C). This will require that in addition to B and C, the KNNs of r_3 be determined as well which incurs additional overhead. However, in the real world data sets, the points of interest usually constitute a very small percentage of the nodes in the graph (e.g., in the State of California, restaurants that have a density of less than 5%, are the points of interest with the maximum density). Hence, the incurred overhead is negligible. 2) Similar to Properties 1 and 2, we can also easily prove that by including points of interest that are on p in the candidate set, the above properties will again become valid for p. For example, this solution suggests that the three closest restaurant to any point on the path (B, C) is a subset of $\{r_3 \bigcup \{r_3, r_4, r_5\} \bigcup \{r_6, r_8, r_3\}\}$. For the rest of this paper, we use the first alternative.

Once the KNNs of the nodes in the network are specified, we need to find the location and the KNNs of the split points on each segment (i.e., path between two adjacent nodes). Note that split point(s) only exist on the segments where the nodes of that segment have different KNNs. We divide the KNNs of a node n_i to two increasing and decreasing groups: the NNs that their distances to a query object q, which is traveling from n_i in a specific direction, increases as the distance of q and n_i increases, are called increasing NNs and vice versa. Note that whether a NN is increasing or decreasing depends on the direction that q is traveling. We can specify whether a point of interest is considered as increasing or decreasing NN using the following property:

Property 3: Let n_i and n_j be two adjacent nodes, $d(x, y)$ specifies the length of the shortest path between objects x and y, and $O = \{o_1, ..., o_k\}$ be the set of points of interest in the network, then the shortest path from n_i to $o_a \in O$ goes through n_j if and only if $d(n_i, o_a) = d(n_i, n_j) + d(n_j, o_a)$.

Proof: This property is self-evident and we omit its proof.

We formally define increasing/decreasing NNs as:

Definition: A point of interest o is called increasing for the direction $n_i \rightarrow n_j$ if the shortest path from n_i to o does not pass through n_j, and it is called decreasing otherwise.

An example of the above definition and property suggests that in Figure 2, r_3 is considered as a decreasing NN for a query object that is traveling from A toward B (since B is on the path between A and r_3), but it is considered an increasing NN when the query is traveling from A toward E.

We can now describe our approach to find the location of the split points between two nodes, and their KNNs, using the following example: suppose that in Figure 2, we are interested to find the three closest restaurants to any point on the path (A, B, C, D).

Step 1: The first step is to break the original path (A, B, C, D) to smaller segments. For the given example, the resulting segments will be (A, B), (B, r_3), (r_3, C) and (C, D).

Step 2: Once the segments are specified, the KNNs of the nodes of each segment must be determined. We use VN^3 approach to efficiently find the KNNs of the nodes. To illustrate our technique, we focus on the first segment (i.e., AB). The other subsequent segments can be treated similarly. The three nearest restaurants to A and B and their distances as $\{(r_1, 3)(r_2, 6)(r_3, 7)\}$ and $\{(r_3, 2)(r_4, 3)(r_5, 4)\}$, respectively. We now know that there must be split point(s) between A and B and the KNNs of any point on segment (A, B) is a subset of the candidate list $\{r_1, r_2, r_3, r_4, r_5\}$.

Step 3: From the above candidate list, we can easily generate a sorted list of the nearest neighbors for the starting point of the segment, A. We also specify

whether each NN is an increasing or decreasing NN using $\uparrow$ and $\downarrow$ symbols, respectively. For the given example, the sorted candidate list for A is $\{\uparrow (r_1, 3), \uparrow (r_2, 6), \downarrow (r_3, 7), \downarrow (r_4, 8), \downarrow (r_5, 9)\}$.

Step 4: We now specify the location of the first split point by: 1) we find the location of the split point for any two subsequent members of the sorted list, o_i and o_{i+1}, where the first and second members have increasing and decreasing distances to A, respectively. The distance of the split point for o_i and o_{i+1} to A can be easily found as: $\frac{d(A,o_{i+1})+d(A,o_i)}{2} - d(A, o_i) = \frac{d(A,o_{i+1})-d(A,o_i)}{2}$. Note that since o_{i+1} is lower than o_i on the sorted candidate list, $d(A, o_{i+1})$ is always greater than $d(A, o_i)$, meaning that the location of the split point is always between A and B, 2) we then select the split point with the minimum distance to A as the first split point. For the given example, the only two subsequent members of the candidate list that satisfy the above condition are $\uparrow r_2$ and $\downarrow r_3$ with split point $p_1 = \frac{6+7}{2} = 6.5$ and a distance of $(6.5 - 6 =)0.5$ to A.

Note that we ignore other combinations of any two subsequent members of the sorted list because: a) if two subsequent members both have increasing (or decreasing) distances to A (e.g., $\uparrow (r_1, 3)$ and $\uparrow (r_2, 6)$, or $\downarrow (r_3, 7)$ and $\downarrow (r_4, 8)$), the differences between their distances to a query object moving from A to B will remain constant, meaning that there will be no split points between them, and b) if the first member has a decreasing and the second member has an increasing distance to A, when the query object is traveling from A to B, the distance of the query to the first member will be decreased further and the distance to the second member will be increased further, which means there will be no split points between the members.

Step 5: We can easily update the sorted candidate list to reflect their distances to the first split point p_1 by adding/subtracting the distance of A and p_1 to/from the members that have increasing/decreasing distances to A. The sorted candidate list for p_1 will then become $\{\uparrow (r_1, 3.5), \downarrow (r_3, 6.5), \uparrow (r_2, 6.5), \downarrow (r_4, 7.5), \downarrow (r_5, 8.5)\}$.

Step 6: We now treat p_1 as the beginning node of a new segment, (p_1, B), and repeat steps 4 to 6: we first determine the split points for $(\uparrow r_1, \downarrow r_3)$ and $(\uparrow r_2, \downarrow r_4)$ pairs as $np_1 = \frac{3.5+6.5}{2} = 5$ and $np_2 = \frac{6.5+7.5}{2} = 7$, then find their distances to A as $d(np_1, p_1) = 5 - 3.5 = 1.5$ and $d(np_2, p_1) = 7 - 6.5 = 0.5$, and finally select np_2 as the next split point p_2. We continue executing steps 4 to 6 until the new split point has similar KNNs as B.

Table 1 shows the results of repeating the above steps for the segment (A, B): the KNNs of any point on (A, p_1) interval is equal to the KNNs of A (and p_1), for any point on (p_1, p_2) segment is equal to KNNs of p_1 (and p_2), and so on. Note that the distance from a query object, which is between two split points, to its KNNs can be similarly computed. Subsequently, the results for other segments can be similarly found.

Split Point	Distance to A	Candidate Set
p_1	0.5	$\uparrow (r_1, 3.5)$, $\downarrow (r_3, 6.5)$, $\uparrow (r_2, 6.5)$, $\downarrow (r_4, 7.5)$, $\downarrow (r_5, 8.5)$
p_2	1.0	$\uparrow (r_1, 4)$, $\downarrow (r_3, 6)$, $\downarrow (r_4, 7)$, $\uparrow (r_2, 7)$, $\downarrow (r_5, 8)$
p_3	1.5	$\uparrow (r_1, 4.5)$, $\downarrow (r_3, 5.5)$, $\downarrow (r_4, 6.5)$, $\downarrow (r_5, 7.5)$, $\uparrow (r_2, 7.5)$
p_4	2.0	$\downarrow (r_3, 5)$, $\uparrow (r_1, 5)$, $\downarrow (r_4, 6)$, $\downarrow (r_5, 7)$, $\uparrow (r_2, 8)$
p_5	2.5	$\downarrow (r_3, 4.5)$, $\downarrow (r_4, 5.5)$, $\uparrow (r_1, 5.5)$, $\downarrow (r_5, 6.5)$, $\uparrow (r_2, 8.5)$
p_6	3	$\downarrow (r_3, 4)$, $\downarrow (r_4, 5)$, $\downarrow (r_5, 6)$, $\uparrow (r_1, 6)$, $\uparrow (r_2, 9)$

Table 1: Split points for segment (A, B) of Figure 2

This approach, although provides a precise result set, is conservative and may lead to unnecessary execution of KNN queries. For example, suppose that in Figure 1, we are interested in the first NN of any point on the traveling path from S to D. Clearly, there are only three split points on this path. However, if we utilize IE approach to address this query, the 1-NN query will be issued for all the intersections of the path. We address this issue with our second approach.

4.2.2 Upper Bound Algorithm: UBA

The UBA approach works similar to IE. However, while IE performs KNN queries for every node (e.g., intersection) on the path, the UBA approach only performs the computation of KNN queries for the nodes that is required and hence, provides a better performance by reducing the number of KNN computations. The intuition for UBA is similar to what is discussed in [12]: when a query object is moved slightly, it is very likely that its KNNs remain the same. Song et. al. in [12] define a threshold value as $\delta = \min(d(o_{i+1}, q) - d(o_j, q))$ where q is the query object and $o_{i+1} \in (K+1)NNs(q)$. The defined δ specifies the minimum difference between the distances of any two subsequent KNNs of q. It can be shown that if the movement of q is less than $\frac{\delta}{2}$, the KNNs of q remain the same. This approach is designed for Euclidean spaces but we apply it to spatial network databases. In addition, we propose a less conservative bound, δ', which improves the performance of our approach further.

We first discuss the extension of the approach described in [12] to SNDB using the example in Figure 2. Let us assume that a query object q is traveling from D toward C and we are interested in finding the three closest restaurants to q. The above approach suggests to first find the four closest restaurants to D, $\{(r_8, 1), (r_6, 9), (r_3, 10), (r_7, 11)\}$. The value of δ is then computed ($\delta = 1$), and finally it is concluded that while the distance of q and D is less than or equal to $(\frac{\delta}{2} =)$ 0.5, the 3NNs of q are the same as the 3NNs of D. The next (3+1)NN query must then be issued at the point that the distance of q and D becomes 0.5.

As we discussed in Section 4.2.1, depending on the traveling path of a query object q, its KNNs can be

divided to two increasing and decreasing groups. We showed that if two subsequent members of the candidate list are both increasing or decreasing, or if the first one is decreasing and the second one is increasing, they cannot generate any split points on the path. This property is in fact the basis of UBA algorithm.

We define the new threshold value as $\delta' = \min(d(o_{i+1}, q) - d(o_i, q))$ where q is the query object, $o_{i+1} \in (K+1)NNs(q)$, and o_i and o_{i+1} have increasing and decreasing distances to q, respectively. The reason for this is similar to the discussion presented in the step 4 of Section 4.2.1. Our defined δ' specifies the minimum difference between the distances of *only* the NNs that can generate a split point on the traveling path. If the movement of q is less than $\frac{\delta'}{2}$, the KNNs of q remain the same. Otherwise, a new (K+1)NN query must only be issued at the intersection point that is immediately before the point that has a distance of $\frac{\delta'}{2}$ to the initial location of q. For example, in Figure 2, if the traveling path is (D, C, B, A) and the point specified by some δ' is between C and B or between D and C, a new (K+1)NN query must be issued at point C and then the split points between C and D are specified similar to IE. In this case, UBA will perform similar to IE. However, if the point specified by some δ' is between B and A, then a new (K+1)NN query must be issued at point B which means UBA eliminates the overhead of computing KNN for C. Note that δ' is always greater than or equal to δ and hence, provides a better bound for our method.

We now discuss the same example using our UBA approach. The four nearest neighbors of D are $\{\uparrow (r_8, 1), \downarrow (r_6, 9), \downarrow (r_3, 10), \downarrow (r_7, 11)\}$. Note that in addition to specifying the KNNs of an object, the VN^3 approach can also be used to specify the direction (i.e., increasing or decreasing) of the neighbors. This can be achieved by determining the immediate connected node, n, to the object that is on the shortest path from the object to its K^{th} neighbor, r_k. Consequently, r_k is decreasing if n is on the traveling path. With our approach, we only examine $\uparrow r_8$ and $\downarrow r_6$ to compute δ' since they are the only subsequent members of the list that satisfy our condition. The value of δ' for this example will then become $9-1 = 8$, which means that when q starts moving from D toward C, as long its distance to D is less than or equal to $(\frac{8}{2} =)4$, the 3NNs of q will remain similar to the 3NNs of D. This means that once the (3+1)NNs of D are determined, there is no need to compute (3+1)NNs of any other point on the $(D \rightarrow C)$ path. Moreover, as we discussed in Section 4.1, the first nearest neighbor of a moving point remains the same as long as the point stays in the same NVP. Hence, we ignore the comparison of the first and second nearest neighbors (if it is necessary) and check any changes in the first nearest neighbor only by locating the intersection of the path with the NVPs of the network. Consequently, the new value of i in our formula for δ' varies from 2 to K, which may lead to even a higher bound value for δ'.

Function IE(Path P)

1. *Break P to segments that satisfy property 2:* $P = \{n_0, ..., n_m\}$
2. *Start from* $n_i = n_0$, *for each segment* (n_i, n_{i+1}):
2.1 *Find KNN*(n_i) *and KNN*(n_{i+1})
2.2 *Find the directions of KNNs*(n_i)
2.3 *Find the split points of the segment* (n_i, n_{i+1})

Function UBA(Path P)

1. *Break P to segments that satisfy property 2:* $P = \{n_0, ..., n_m\}$
2. *Start from* $n_i = n_0$, *while* $n_i \neq n_m$:
2.1 *Find (K+1)NN(* n_i *) and their directions*
2.2 *compute* δ'
2.3 *Find* n_p, n_q *where* δ' *is between* (n_p, n_q)
2.4 *If* $n_q = n_{i+1}$:
2.4.1 *IE* (n_i , n_{i+1})
2.5 $n_i = n_{i+1}$

Figure 3: Pseudo code of IE and UBA

Figure 3 shows the pseudo code of our IE and UBA approaches.

5 Performance Evaluation

We conducted several experiments to compare the performance of the proposed approaches for the continuous KNN queries. The data set used for the experiments is obtained from NavTeq Inc., used for navigation and GPS devices installed in cars. The data represents a network of approximately 110,000 links and 79,800 nodes of the road system in downtown Los Angeles. We performed the experiments using different sets of points of interest (e.g., restaurants, shopping centers) with different densities and distributions. The experiments were performed on an IBM ZPro with dual Pentium III processors, 512MB of RAM, and Oracle 9.2 as the database server. We calculated the number of times that the KNN query must be issued and the required times, for different values of K and different lengths of traveling paths. We present the average results of 100 runs of continuous K nearest neighbor queries.

Table 2 shows the query response time (in seconds) for IE and UBA approaches when the length of the traveling path is 5KM and the value of K varies from 1 to 20. Also, the numbers inside parenthesis in columns under UBA specify the number of nodes a (K+1)NN query is issued at. Note that for the given data set, the average length of the segments between two adjacent intersections is about 147 meters, meaning that (on average) there are 34 intersections in a 5KM path. As shown in the table, UBA always outperforms IE. This is because as shown in the table, the number of nodes a (K+1)NN query is issued at in UBA method is always less than 34, i.e., the number of nodes a KNN query must be issued at in IE method. How-

Entities	Qty (density)	K=1	K=3		K=5		K=10		K=20	
			IE	UBA (No. of nodes)	IE	UBA (No. of nodes)	IE	UBA (No. of nodes)	IE	UBA (No. of nodes)
Hospital	46 (0.0004)	0.6	153	22.5 (5)	217	82 (13)	476	294 (21)	952	670 (24)
Shopping Centers	173 (0.0016)	0.68	85	20 (8)	110	58 (18)	231	149 (22)	493	377 (26)
Parks	561 (0.0053)	0.7	34	11 (11)	51	16.5 (20)	91.8	62 (23)	187	143 (26)
Schools	1230 (0.015)	0.92	17	7 (14)	23.8	14.7 (21)	48.5	37 (26)	119	94.5 (27)
Auto Services	2093 (0.0326)	1.0	15.3	6.7 (15)	22.1	14.3 (22)	48	38 (27)	85	72 (29)
Restaurants	2944 (0.0580)	1.0	13.6	6.0 (15)	19.8	13.4 (23)	47.6	39.2 (28)	81.6	74.5 (31)

Table 2: Query processing time (in seconds) of IE vs. UBA, Traveling Path = 5KM

ever, the advantage of UBA over IE is minimal when the points of interest are distributed densely in the network (e.g., restaurants). The reason for this is that in these cases, the value of δ' is relatively close to the average length of the segments. That is, the points determined by the UBA approach on which the next (K+1)NN queries must be performed, are (usually) located between two adjacent nodes. Hence, the UBA approach can only eliminate the computation of KNNs for a small number of adjacent nodes. However, for the points of interest that are sparsely distributed in the network (e.g., hospitals), the value of δ' is usually much larger than the average length of the segments. This means that the UBA approach can filter out several adjacent nodes from the computation of KNNs and hence, significantly outperforms IE.

Also note that the performance of UBA becomes close to IE when the value of K increases. This is also because there are more number of subsequent increasing/decreasing neighbors that must be examined when the value of K increases. This will lead to a smaller value for δ', which will lead to KNN query for more number of nodes. When the value of K is equal to 1, the query response time becomes smaller when the points of interest are sparse. This is because the number of NVPs in the network are less when the points of interest are sparse and hence, the intersection of a line with the NVPs can be determined faster. The experiments for traveling paths of 1, 2, 5, 10, and 20 KM show similar trends.

6 Conclusion

In this paper we presented alternative solutions for continuous K nearest neighbor queries in spatial network databases. These solutions efficiently find the location and KNNs of split point(s) on a path. We showed that the continuous 1NN queries can be simply answered using the properties of our previously proposed VN^3 approach: the split points are the intersection(s) of the path with the network Voronoi polygons of the network. We also proposed two solutions for the cases where continuous K nearest neighbors are requested. With our first solution, IE, we showed that the location of the split points on a path can be determined by first computing the KNNs of all the nodes on the path, and then examining the adjacent nodes that have different KNNs. Our second solution, UBA, improves the performance of IE by eliminating the computation of KNNs for the adjacent nodes that cannot have any split points in between. Our experiments also confirmed that UBA outperforms IE.

7 Acknowledgement

This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), IIS-0238560 (CAREER), IIS-0324955 (ITR), and in part by a grant from the US Geological Survey (USGS).

References

[1] J. Feng and T. Watanabe. "A Fast Method for Continuous Nearest Target Objects Query on Road Network". In *VSMM'02 pp.182-191 Sept. 2002, Korea.*

[2] G. R. Hjaltason and H. Samet. "Distance Browsing in Spatial Databases". *TODS*, 24(2):265–318, 1999.

[3] C. S. Jensen, J. Kolrvr, T. B. Pedersen, and I. Timko. "Nearest Neighbor Queries in Road Networks". In *ACM-GIS03, New Orleans, Louisiana, USA.*

[4] M. Kolahdouzan and C. Shahabi. "Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases". In *VLDB 2004, Toronto, Canada.*

[5] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. "Fast Nearest Neighbor Search in Medical Image Databases". In *VLDB 1996, Mumbai (Bombay), India.*

[6] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *"Spatial Tessellations, Concepts and Applications of Voronoi Diagrams".* John Wiley and Sons Ltd., 2nd edition, 2000.

[7] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. "Query Processing in Spatial Network Databases". In *VLDB 2003, Berlin, Germany.*

[8] N. Roussopoulos, S. Kelley, and F. Vincent. "'Nearest Neighbor Queries". In *SIGMOD 1995, San Jose, California.*

[9] T. Seidl and H.-P. Kriegel. "Optimal Multi-Step k-Nearest Neighbor Search". In *SIGMOD 1998, Seattle, Washington, USA.*

[10] S. Shekhar and J. S. Yoo. "Processing in-Route Nearest Neighbor Queries: a Comparison of Alternative Approaches". In *ACM-GIS03, New Orleans, Louisiana, USA.*

[11] P. Sistla, O. Wolfson, S. Chamberlain, and D. S. "Modeling and Querying Moving Objects". In *IEEE ICDE 1997.*

[12] Z. Song and N. Roussopoulos. "K-Nearest Neighbor Search for Moving Query Point". In *SSTD 2001, Redondo Beach, CA, USA.*

[13] Y. Tao and D. Papadias. "Time Parameterized Queries in Spatio-Temporal Databases". In *SIGMOD 2002.*

[14] Y. Tao, D. Papadias, and Q. Shen. "Continuous Nearest Neighbor Search". In *VLDB 2002, Hong Kong, China.*

Managing Trajectories of Moving Objects as Data Streams

Kostas Patroumpas Timos Sellis

School of Electrical and Computer Engineering
National Technical University of Athens
Zographou 15773, Athens, Hellas
{kpatro, timos}@dbnet.ece.ntua.gr

Abstract

The advent of modern monitoring applications, such as location-based services, presents several new challenges when dealing with continuously evolving spatiotemporal information. Frequent updates in the positions of moving objects, unexpected fluctuations in data volume and the requirement for real-time responses to continuous spatiotemporal queries indicate the limitations of traditional database systems. We attempt to model management of moving objects with the underlying assumption that their trajectories are essentially continuous, time-varying and possibly unbounded data streams. We propose a basic framework for managing trajectory streams, and suggest the introduction of enhanced constructs for advanced query capabilities. Our first experience with querying moving objects in two data stream prototype systems is promising for the feasibility and extensibility of this approach.

1 Introduction

In monitoring applications, like sensor networks, financial tickers, Web search engines etc., data arrives into the system from multiple sources as *online data streams*. This persistent data flow through a network requires real-time responses to users' requests. Multiple *continuous queries* remain active for long and they must be evaluated incrementally, keeping up with the arrival rate of the data. The typical pull-based model for transactions and queries utilized in the conventional DBMS paradigm gives way to a push-based architecture, which has been adopted by several prototype systems currently being developed for data stream management [ACC+03, BBD+02, CCD+03].

Applications for *location-based services* (e.g. automatic vehicle location) have also greatly benefited from recent advances in the fields of telecommunications and Global Positioning System (GPS). The possibility to locate objects in continuous movement (e.g., vehicles or humans) in real-time and with improved accuracy has boosted up the trend for developing moving objects databases, and consequently, has assisted research in the broader area of spatiotemporal phenomena. In particular, topics such as representation, indexing and querying moving objects have attracted much research effort, mainly towards the foundation of an appropriate model for the efficient management of their *trajectories*.

We believe that the research fields of data streams and management of moving objects can naturally come together. In particular, ideas and techniques originally proposed for data streams can also apply to trajectories:

- Why not maintain trajectory elements in memory for immediate processing, rather than just storing them in a spatiotemporal database for offline management?
- Is it possible to formulate continuous queries over trajectories and provide their results incrementally?
- If the current status of movement is what matters most, would it then be reasonable to ignore the details concerning remote parts of trajectories and instead focus on windows of the most recent features?

We think that the answer to these indicative questions is affirmative according to the model we propose.

Indeed, by sampling the trajectories of moving objects and thus compiling their successive positions across time, a data stream of spatiotemporal features can be created. Sampling rates may fluctuate, data could get lost during transmission to the system, or even superfluous (i.e. very dense) measurements might be observed at locations. Apparently, a DBMS cannot store data in its entirety, since the arrival rate is unpredicted and the expected amount of data rather high, as tuples are piling up continuously into the processing mechanism from their sources. Possible variations at the arrival rate of incoming data might impose excessive requirements in system resources, especially with respect to memory allocation. It is also likely that the system cannot cope with a sudden burst of data and thus become unable to provide responses

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

to queries in time; techniques such as *load shedding* [ACC+03] or *approximations* [BBD+02] have been devised in an attempt to remedy the problem. Clearly, a traditional DBMS, yet enhanced with special constructs for spatiotemporal data management, would prove inadequate to this purpose, especially in terms of online processing of queries that remain active for a long time.

Our fundamental assumption about streams of trajectory elements paves the way towards the formulation of a query language for manipulating *trajectory streams*. Noticeably, spatial information is not simply some invariable coordinates, as in the case of sensor networks covering an area. On the contrary, it is precisely the continuous movement that is of great interest, as well as any spatiotemporal interactions among moving objects. Hence, this language should bring together the clarity and the processing functionality of a stream query language along with operations particularly designed to capture interesting spatiotemporal interrelationships, e.g. "objects crossing the boundaries of a polygon area".

This paper describes a novel approach to the management of trajectories of moving objects, considering them as online data streams. To the best of our knowledge, there is no prior work making out the case for utilizing a stream query language so as to express continuous queries over dynamic spatiotemporal information. We believe that a declarative language is the most appropriate for the trajectory stream model proposed here. The semantics of an SQL-like language can be comfortably modified so as to apply to streams, whereas spatiotemporal operations may be incorporated with proper adjustments or additions. New language constructs, namely several types of *windows*, are worth introducing so as to facilitate managing and querying trajectories as data streams.

The remainder of this paper is structured as follows. Section 2 presents a brief overview of previous work in the research areas of data streams and moving objects databases. In Section 3, we introduce elements of a model for representing trajectories as data streams, which we call trajectory streams. In Section 4, we briefly report on formulating indicative spatiotemporal continuous queries over trajectories in two data stream prototype systems. In Section 5, we discuss the necessity for new language constructs, mainly special-purpose window operators for trajectory streams. Finally, Section 6 concludes the paper and suggests ideas for future research.

2 Related Work

Several data stream prototypes are already at an advanced implementation stage, introducing a lot of new concepts with regard to the architecture of such a data management system. In parallel, several suggestions have been made for a stream query language. Most researchers tend to adopt a declarative SQL-like language. A *Continuous Query Language* (CQL) is being developed for the STREAM prototype [STR04], supporting management of dynamic streams and static relations, as well as mappings between them [ABW03]. In addition, CQL provides the means for query optimization, a crucial issue when numerous continuous queries are active at the same time. Similarly, TelegraphCQ introduces *StreaQuel* [TEL04], which extends semantics of relational operators to streams and allows transformation between streams. There is also support for windows that specify portions of the infinite stream. Conversely, AURORA [AUR04] follows the data flow paradigm, and makes use of a built-in query algebra *SQuAl* in order to construct a network of operators. These primitive operators take part in the global execution plan maintained for all active continuous queries.

Concerning management of spatiotemporal information, there are two different perspectives regarding time. From a historical point of view, an algebra for moving objects has been proposed [KSF+03] with data types, operators and predicates over trajectories. These constructs adhere to SQL and they may be implemented as an extension to a traditional DBMS [EGSV99]. In contrast, other models attempt to determine the current status of objects or to predict their future positions. To this goal, a *Future Temporal Logic* language was introduced [SWCD97], enhanced with spatiotemporal constructs for suitable query specification. Indexing techniques for moving objects or trajectories abound, with scalability being the main challenging task. Indeed, the larger the number of moving objects, the more expensive the maintenance of indexing structures due to frequent position updates.

Some recent papers focus on evaluating continuous queries over continuously moving objects. Most of them deal with specific query types, such as *k-nearest neighbors* [ISS03] or *range* queries [GL04]. However, the streaming behavior of trajectories is all but implicit. For instance, in [MXA04], shared query execution is based on hashing with a predefined decomposition of space. Evaluation of spatiotemporal queries is outside the scope of this paper.

Finally, Stream Query Repository [SQR04] offers some example queries involving coordinates; nonetheless, spatial references are used as stationary information only, without examining potential interactions among the corresponding entities.

3 Modeling Trajectories as Data Streams

The scope of our model is currently restricted to point objects only, thus spatial entities are considered to have no extent: the shape of objects is reduced to their centroid. A *trajectory* is obtained by recording the successive positions a point object takes across time. This continuous data flow from a number of sources (e.g. GPS receivers in cars) to a central processing system can be suitably modeled as a *trajectory stream*. So trajectories are viewed from a historical perspective, not dealing at all with predictions about the future position of the moving objects. On the other hand, the model conveniently enables calculation of derived attributes, such as speed or acceleration.

3.1 Basic Entities

The entities appearing in the trajectory stream model can be drawn from the following principal components:

- *Relations*, which can be updated from time to time, thus allowing several temporal versions. Of particular interest are *spatial relations* that need be supported for stationary entities (e.g. area boundaries). These are stored in attributes of a primitive data type (point, line, and region). Each such relation must contain features of the same spatial data type, avoiding by all means any mixture of dissimilar spatial entities.
- Pure *data streams*, modeled as ordered multisets of tuples. The ordering attribute (timestamp) is based on valid time values registered at the data sources. An identifier needs to be included in the schema of attributes to distinguish the origin of the incoming tuples, as the union of those tuples creates the stream.
- *Trajectory streams* can be considered a specialized class of streams, which models the continuous movement of spatial point entities. The spatial reference evolves with time, following the successive positions taken by each moving object. Therefore, there is a close connection between spatial and temporal coordinates, which coexist simultaneously and create a unique spatiotemporal reference. This duality between space and time is very significant in formulating queries, since a combination of temporal and spatial predicates may be required.
- *Derived streams*, produced when operations and predicates are applied to the contents of *base streams* [ABW03] originating from data sources. Apart from local views used in continuous queries (as will be demonstrated in subsection 4.1), the principal form of derived information is *summaries* or *synopses* that approximate the contents of the (trajectory) stream, possibly by sampling or aggregating specific attributes. These can be produced after a query or a subquery is applied over the tuples of a base stream.
- *Query language*, for expressing continuous queries on the contents of streams and relations in a declarative manner. Obviously, this language can be implemented according to the algebraic structures prescribed for operations applicable to streams.

Informally speaking, if data evolves with time, then it is modeled as a data stream; if spatial information also varies with time, a trajectory stream is produced. Permanence in space and time is captured with relations. Observe that trajectory streams may collapse to data streams after projecting out the spatial reference attribute. So, it is possible that a data stream can be derived from a trajectory stream, but not vice versa. For example, a data stream containing the speed of moving objects can be calculated for every position, retaining only timestamp information (but not spatial features) in the resulting tuples. Consequently, by projecting out the timestamp attribute as well, a relation is derived. Relations may even include temporal and/or spatial attributes; however, as long as order cannot be defined, they may be considered as degenerate (static) streams or trajectory streams.

As in the relational model, we may define:

Definition 1 (Schema of tuples): The *tuple schema E* of the data is represented by a set of elements ($e_1, e_2, \ldots e_N$). Each element e_i is termed *attribute*, it has a name A_i and its values are drawn from a –possibly infinite– atomic data type domain D_i. The finite number N of the attributes is called the *arity* of the schema. A *tuple* is an instance of the schema and it is described by the values of the respective attributes. □

3.2 Temporal Modeling

Time is represented as an ordered sequence of distinct moments. A *timestamp* is attached to the set of coordinates (and other attributes relative to the measurement) at data sources. As a result, each timestamp marks the actual time the item was recorded and refers to what is termed *valid time* in temporal databases. Another option is to introduce for each tuple a second timestamp based on *transaction time* (hence marking the time instants that tuples enter into the system for processing) [BBD+02]. Timestamps based on valid time need to be included in the schema of tuples (*explicit timestamps*), since they are indispensable to the calculation of derived attributes. More formally:

Definition 2 (Time Domain): The *Time Domain T* is regarded as an ordered, infinite set of discrete values (*time instants*) $\tau \in T$. A *time interval* $[\tau_1, \tau_2] \subset T$ consists of all distinct time instants of *T between* τ_1 and τ_2. □

For clarity [ABW03], T may be considered similar to the set of natural numbers $\mathbb{N}$. For each timestamp, a *Data Stream S* is an unbounded multiset (bag) of items (i.e. allowing duplicates as in the definition at [ABW03]):

Definition 3 (Data Stream): A *Data Stream S* is a mapping $S : T \rightarrow 2^R$ from the Time Domain T to the powerset of the set R of tuples with schema E. One of the attributes A_τ is designated as the *timestamp* of each tuple, taking its ever-increasing values τ from T. □

From a historical perspective, a Data Stream may be regarded as an *ordered sequence of tuple values evolving in time*, whereas an instance of the stream at a specific time (e.g., `NOW` or even in the future) is a finite set of tuples. Only one attribute is used as timestamp, i.e., a unique time reference for the entire tuple. As further explained in subsection 3.4, each tuple maps to exactly one timestamp, but more than one tuples can have identical timestamp values. Timestamps cannot be assigned a `NULL` value.

3.3 Spatial Modeling

As mentioned before, the proposed model deals only with *point entities* moving in two spatial dimensions. Despite

this simplification, interactions are allowed between moving objects and other shapes with extent (i.e. lines or regions), which remain stationary over time. Therefore, the model attempts to capture spatiotemporal relationships among trajectories or between trajectories and static spatial entities. It is also envisaged that the model is extensible to higher dimensionality of spatial entities.

Definition 4 (Point Domain): The *Spatial Point Domain* P contains all possible (hence infinite) pairs of values $\langle x, y \rangle$, with real planar coordinates $x, y \in \mathbb{R}$. Thus, P may be regarded similar to $\mathbb{R}^2$. □

Concerning spatial data types, those originally introduced in [GBE+00] are deemed sufficient. More specifically, `point`, `points`, `line` and `region` with their obvious interpretation may appear in relations, whereas only type `point` is allowed in trajectory streams.

We then make use of several primitive *predicates* for expressing topological relations. Each of these binary spatial predicates applies to a discrete instance of the trajectory of a moving point in conjunction either with another moving point or with a stationary spatial feature:

- a *point* (predicate `MEET` when the two points coincide),
- a *directed line* (so that predicates `LEFT` and `RIGHT` can be defined for the moving point, or `MEET` when the point is along the line), and
- a *region*, with predicates `INSIDE` for moving points that fall within its interior (U^o) and `MEET` for those touching its boundary (∂U).

Complex predicates like `ENTER`, `LEAVE`, `CROSS` and `BYPASS` [PJT00] can be constructed from primitive ones.

Spatial *operators* are also needed for the formulation of queries. We designate four basic ones (note that the first three of them refer to a single object):

- `length` as the Euclidean distance between two arbitrary point positions of the same trajectory,
- `direction` as the angle between the vector of the object's movement and the horizontal axis,
- `heading` which denotes the general orientation of the movement ({`E`, `NE`, `N`, `NW`, `W`, `SW`, `S`, `SE`}), and
- `distance` for calculating the Euclidean distance between two distinct moving point objects at the same time instant.

Additionally, when formulating queries, simple shapes might be necessary to predicates, e.g. for defining the area of interest for a range query. We mention four basic *type constructors* for spatial entities:

- `Point(x,y)`,
- `Circle(P,r)`, where `P` is an instance of the `Point` type (the center) and `r` $\in \mathbb{R}$ the radius,
- `Rectangle(P1,P2)`, where `P1`, `P2` are instances of `Point` (the edge-points of a diagonal) and
- `Triangle(P1,P2,P3)`, where `P1`, `P2`, `P3` are the three vertices defining the triangle.

Notice that the *uncertainty* caused from positional inaccuracies should be taken into account (with a *fault tolerance*) when applying the predicates and operators mentioned above.

Finally, we extend the concept of data streams by defining a *Point Data Stream S* as an unbounded bag of elements with varying spatiotemporal properties:

Definition 5 (Point Data Stream): A *Point Data Stream* S is a mapping $S : T \times P \rightarrow 2^R$ from Time Domain T and Point Domain P to the powerset of the set R of tuples with schema E. In addition to an attribute A_T used for *time-stamp values* $\tau \in T$, another attribute A_P acts as the *spatial reference* for the represented entity and obtains its values $p \in \mathbb{R}^2$ exclusively from the Point Domain P. Pair $\langle p, \tau \rangle$ may be regarded as a composite *space-time-stamp*. □

3.4 Spatiotemporal Modeling

Movement in 3D-space (one temporal and two spatial dimensions) can be regarded as a sequence of discrete observations of positions across time. Once coherence in time domain is retained (i.e., no vacuums when recording locations), then contiguity in spatial domain for each object is also likely to be preserved. Redundancies in the collected data are inevitable, if sampling rate is too high and speed too low. On the other hand, it is not at all certain that an object's movement remains linear between two successive observations, as the model suggests. As a result, the trajectory itself should be considered as an error-prone approximation of the actual movement.

Definition 6 (Trajectory Stream): Let O denote the set of continuously moving point objects. The changing position of each distinct object $o \in O$ is actually a function of time $p_o: T \rightarrow P$. The Point Data Stream S composed of the successsive tuple values obtained by monitoring the movement of the points from O is a *Trajectory Stream*. □

Therefore, a trajectory stream for point objects may be viewed as an *ordered sequence of tuple values concurrently evolving in space and time*. An instance of this sequence is a set of tuples with positional and temporal indications; the former are simply some 2D-coordinates $\langle x, y \rangle$ in a common reference system, whereas the latter are instants drawn from a common Time Domain. Spatial and temporal references cannot be assigned a `NULL` value. This *space-time-stamp* uniquely determines the status of each object in space and time, and is explicitly included in the schema of tuples E. This is essential for calculations concerning derived attributes, such as speed. Updates to older data are not allowed, so trajectory streams are considered to be composed of *append-only* tuples. We emphasize the difference between point and trajectory streams through the following example.

Example 3.1. Imagine N mobile stations, which measure air pollution in a city. Collected values are transmitted to a control center along with their location and the time instant they were recorded. Obviously, incoming tuples

are compiled into a *trajectory stream*, as every item captures an instance of an object's movement. It is likely that a user might submit a continuous query that retrieves tuples referring to the k out of N stations where the higher concentrations of a specific gas (e.g., CO) are observed at any time instant. As one might expect, these *top-k* stations will vary across time, so the selected tuples will form a data stream with changing spatial references for possibly different objects. Therefore, this derived information is a *point data stream*, not a trajectory stream; the notion of a sequence of object transitions from one location to another does not hold anymore in that case. □

Although *total ordering* among tuples of a trajectory stream cannot be achieved based on their space-time-stamps, a *temporal total order* among tuples belonging to the same trajectory can be defined according to their timestamp values only. Obviously, elements referring to the same trajectory should have advancing timestamps, even if the object stands still (i.e. identical spatial positions, assuming no errors in measurement). Formally:

Definition 7 (Ordering): A *temporal ordering* f_O is defined as a many–to–one mapping from the *type domain* D_S of the tuples s of the trajectory stream S to the *Time Domain* T, with the following properties:

- $\forall\ s \in S, \exists\ \tau \in T$, such that $f_O(s) = \tau$.
- $\forall\ s_1, s_2 \in S$, instances of the trajectory stream S, and their values $\tau_1, \tau_2 \in T$ at timestamps $s_1.A_T$ and $s_2.A_T$ respectively, if $\tau_1 \leq \tau_2$, then $f_O(s_1) \leq f_O(s_2)$. □

The first property of the definition above states that a timestamp need be associated to every tuple of the stream. Many tuples may map to the same timestamp; however, there may exist time instants where no tuples are recorded, e.g. for a particular time period no positional updates were received from a GPS. The second property establishes *monotonicity*, i.e. tuples of the trajectory stream are in increasing temporal order as time advances. Hence, tuples referring to the same trajectory can be ordered by simply comparing their time indications. This inherent and valuable characteristic of the time dimension has a subtle effect: after a while, older tuples may be considered as obsolete, so they may be either purged from memory completely freeing space or stored in synopses. Observe that the latter property holds for any two stream tuples (i.e. not necessarily belonging to the same trajectory), provided that time indications have identical granularities with values drawn from a common Time Domain T.

Note that an attribute of the schema may be designated as the identifier *id* of each point object. This can be proven useful in case the contents of the trajectory stream need to be split in disjoint *substreams*, one for each point object. We are currently investigating potential inclusion of velocity vectors in the model. Intuitively, it is the mobility of data sources in both space and time that produces a trajectory stream; this is best captured by a vector (like velocity or acceleration) that emphasizes change.

3.5 Types of Query Operators

As we will demonstrate in the following section, we make use of several types of operators in the framework of two stream prototypes, applying their semantics to trajectories. These constructs can either be adapted from the relational or data stream models, or may be introduced especially for managing trajectories (as we suggest in Section 5):

- *Relational*, such as selection or projection, modified properly to apply on trajectory streams as well.
- *Windowing*, for determining specific parts of the streams and converting them to relations for subsequent processing. The *scope* of the windows can be fixed, variable or sliding.
- *Transform*, for conveniently mapping relations (e.g. tuples produced by other operators) to streams.
- *Spatial*, for determining interactions between moving points and other stationary spatial entities, e.g. points, lines, regions.
- *Temporal*, for processing timestamps, which apply either on a time-point basis (individual timestamps) or for intervals (range of consecutive timestamps).
- *Spatiotemporal*, for effectively capturing evolving properties and interactions between moving points.

4 Querying Trajectories as Data Streams

In this section we briefly report our first experience in formulating continuous queries over trajectories in a SQL-like declarative language. Due to lack of space, we give just a few indicative queries submitted to the trial versions of two stream prototypes (STREAM, TelegraphCQ) that have recently been made publicly available. Despite variations at the idiom of SQL utilized, as well as certain limitations in expressing certain types of queries, our overall impression is that the streaming paradigm is powerful enough for managing moving objects. Our focus was on expressiveness, leaving aside query performance.

A synthetic dataset was used, containing positions of several types of vehicles (taxis, ambulances, private and police cars) moving in the road network of the greater area of Athens. After applying a shortest path algorithm at 1000 origin-destination pairs and sampling each route every second, in total 3.6 million records were created. Some adjustments were necessary to the schema of tuples, in order to properly represent spatiotemporal information, according to the restrictions imposed by each prototype.

4.1 Continuous Trajectory Queries in STREAM

We first give a short overview of the Continuous Query Language [ABW03] that is being developed for the STREAM prototype. Queries in CQL are composed from three classes of operators:

- *stream-to-relation* operators (particularly windows) are applied over streaming tuples and return relations.
- *relation-to-relation* are common SQL operators, such as selection or projection.

- *relation-to-stream* operators take a set of relational tuples as input and provide their output in the form of streams at every time instant. ISTREAM streams inserts to its input relation (i.e., tuples that did not exist in the previous time instant), DSTREAM streams deletes from its input relation (tuples that ceased to appear in the relation since the previous instant), whereas RSTREAM streams all current tuples of its input relation.

Thus, query semantics apply to streams and relations alike. Continuous queries have the following general form:

```
SELECT <select_list>
FROM <relations>, <streams_with_windows>
WHERE <predicates>
GROUP BY <expressions>
```

Windows are always specified for streams (in the FROM clause of the CQL query). Supported types include:

- *sliding windows* (RANGE <time interval>) that specify the most recent tuples according to their timestamps,
- *tuple-based* (ROWS <value expression>) that determine a given number from the most recent stream tuples and
- *partitioned windows* (PARTITION BY <attribute list> ROWS <value expr>), which partition the recent portion of a stream based on a subset of its attributes.

Since the current release of the STREAM prototype is Web-based, and our focus was not on performance issues, only a small subset of the data was used for convenience. The schema of tuples is Vehicles (vID, vType, x, y, t, TS), where TS is used for timestamps attached to all tuples.

Query Q1: For instance, a spatiotemporal *range* query, such as "*Find all taxis found within the area of interest (e.g., a rectangle) sometime in the last 10 minutes*" can be expressed with a sliding temporal window, as follows:

```
SELECT V1.vID, V1.vType, V1.x, V1.y
FROM Vehicles V1 RANGE 10 MINUTES
WHERE V1.x>=475000 AND V1.x<=480000
AND V1.y>=4204000 AND V1.y<=4208000
AND V1.vType="TAXI"
```

□

Note that CQL has no built-in types for any spatial entities, so only simple shapes can be defined (from their coordinates). Nested subqueries are not yet supported, but views may be defined over streams or relations. So, *aggregation* (GROUP BY) operations can be performed using views, since the HAVING clause is not available:

Query Q2: "*Alert when more that 10 police cars are located simultaneously within the area of interest*". This query has to be expressed with two intermediate views:

```
InRegionCnt: SELECT TS, COUNT(vID) AS cnt
             FROM Vehicles NOW
             WHERE x>=475000 AND x<=480000
             AND y>=4204000 AND y<=4208000
             AND vType="POLICE"
             GROUP BY TS

PoliceCnt: ISTREAM (SELECT * FROM InRegionCnt)
```

The first view returns a *relation* containing the number of police cars within the area for each time instant, whereas the second one transforms (ISTREAM) these intermediate results into a *stream*. NOW is a shortcut for a sliding window that returns only tuples with the current timestamp value. The final query provides the expected results:

```
SELECT * FROM PoliceCnt NOW WHERE cnt>10
```

□

Even *nearest neighbor queries* can be expressed in CQL, but in a complex style with many intermediate views. Things get even more complicated, because only simple arithmetic functions are supported, so geometric ones, like distance, must be simulated in several steps. However, expressiveness of CQL helps in formulating queries for complex spatial predicates, such as ENTER into region:

Query Q3: *"Find all vehicles entering now into the area of interest"*. Spatial operation ENTER can be simulated as an ANTISEMIJOIN[1] between two temporary relational views. The former (InsideAreaNow) stores vehicles located now inside the area; the latter (InsideAreaRec) keeps those recorded within the area in the last two time instances, utilizing a partitioning window for each object:

```
InsideAreaNow: SELECT vID, t
               FROM Vehicles NOW
               WHERE x>=475000 AND x<=480000
               AND y>=4204000 AND y<=4208000

InsideAreaRec: SELECT vID, t + 1 AS t1
               FROM Vehicles PARTITION BY vID ROWS 2
               WHERE x>=475000 AND x<=480000
               AND y>=4204000 AND y<=4208000

SELECT InsideAreaNow.vID, InsideAreaNow.t
FROM InsideAreaNow ANTISEMIJOIN InsideAreaRec
```

Note that we take advantage of the monotonicity of time, assuming tacitly that positional information is registered at every time instant. □

4.2 Continuous Trajectory Queries in TelegraphCQ

TelegraphCQ is based on PostgreSQL DBMS, with many necessary adjustments to achieve adaptivity of operators in dynamically changing query load or data flow rate. Continuous queries are executed as cursors, which collect resulting tuples and fetch them to users in an incremental fashion. Only *sliding windows* are currently supported for streams; a WINDOW clause is specified for streams after standard SELECT-FROM-WHERE commands [TEL04]:

```
SELECT <select_list>
FROM <relation_and_<pstream_list>
WHERE <predicate>
GROUP BY <group_by_expressions>
WINDOW stream [time interval], ...
ORDER BY <order_by_expressions>
```

Neither nested subqueries nor self-joins can yet be expressed in TelegraphCQ. However, PostgreSQL offers several built-in *spatial* operators, functions and data types (point, polygon, etc.), which prove particularly valuable in formulating continuous queries over trajectories.

[1] *Antisemijoin* of *R* and *S* is defined as the multiset of tuples in *R* that do not agree with any tuple of *S* in the attributes common to both *R* and *S*.

The schema of tuples used for the test queries was `Vehicles (vID, vType, pos, t, TCQTIME)`, where `TCQTIME` is the attribute reserved for timestamp values. Next, we give some SQL expressions submitted to TelegraphCQ, equivalent to the queries of the previous subsection:

Query Q1: A temporal sliding window is specified (as in STREAM), but we take advantage of spatial data types and operations (@ denotes "point `INSIDE` region"):

```
SELECT vID, vType, TCQTIME
FROM Vehicles V1
WHERE (V1.pos @ polygon '(475000, 4204000,
                480000, 4208000)') = TRUE
AND V1.vType ='TAXI'
WINDOW V1 ['10 minutes']                    □
```

Query Q2: Aggregation is carried out on timestamp values, but the `HAVING` clause simplifies query expression:

```
SELECT TCQTIME, COUNT(V1.vID) AS cnt
FROM Vehicles V1
WHERE (V1.pos @ polygon '(475000, 4204000,
                480000, 4208000)') = TRUE
AND V1.vType='TAXI'
GROUP BY TCQTIME
HAVING COUNT(V1.vID)>=10
WINDOW V1 ['1 seconds']                     □
```

However, due to lack of support for nested subqueries or views, other types of queries cannot be expressed at all. This is the case for nearest neighbors or complex spatial predicates (like `ENTER` into region, which can be expressed in STREAM, as showed for **Query Q3**). On the contrary, queries involving spatial functions, like *distance* (denoted `<->` in PostgreSQL) are formulated rather simply:

Query Q4: *"Find all vehicles that are now within a distance of 500 meters from a point of interest"* (this point is specified here by its coordinates):

```
SELECT vID, vType, TCQTIME, (V1.pos <->
           Point '(475750,4201500)') AS distance
FROM Vehicles V1
WHERE (V1.pos <-> Point '(475750,4201500)')<=500
WINDOW V1 ['1 seconds']                     □
```

5 Special Structures for Trajectory Streams

Only *time-based*, *tuple-based*, and *partitioned windows* have been implemented for STREAM and TelegraphCQ. As we have just demonstrated, these structures are also valid for the trajectory stream model. In addition, we advocate for the adoption of four special-purpose window operators, with semantics applied to trajectories.

5.1 Window Specification Types

i. Binary windows. Sliding windows of size two time units may be needed to examine changes in motion, e.g., an object crossing the boundary of a stationary area. Window constructs proposed for data streams tend to ignore the *order* of tuples once the contents of the window have been specified. In contrast, when dealing with trajectories, the current and the oldest tuple representing the boundaries of the temporal window are of particular interest. Based on this information, several interesting entities can be derived, such as velocity or acceleration. Further, *window edge functions* may be utilized in `SELECT` clauses, such as `First_Value` and `Last_Value` (used in SQL-99). These two functions can take as arguments specific attributes and return the values of the most remote and the most recent tuple within the window, respectively.

ii. Landmark windows. The starting edge of this window remains fixed over time, while its end point matches the present time instant. Hence, such a window returns an always-increasing number of tuples. Evidently, this window type can be defined based on time units only; therefore a syntax like `[AFTER t]` seems plausible and reminiscent of that for time-based windows. However, instead of a time interval, a specific time instant `t` is specified, no matter if any tuple exists with exactly this timestamp. A similar window type can be defined even when all tuples *before* a specific time instant `t` are needed, with a syntax like `[BEFORE t]`. Note that window contents remain unchanged as soon as current timestamp `NOW` exceeds value `t`. Interestingly enough, combination of these two language constructs could provide a *band window* that determines a time period with the syntax `[AFTER t1 AND BEFORE t2]` (if $t1 \leq t2$).

iii. Area-based windows. While any sliding or landmark window refers solely to timestamps, this particular window type applies only to trajectory streams and exploits their spatial contents. Intuitively, an area-based window extracts tuples from a trajectory stream whose positional reference falls inside the interior of the area defined. The area boundaries can be obtained either from a stationary spatial relation or by employing a type constructor (such as `Circle` or `Rectangle`). The former results in a stationary area, whereas the latter enables even a *moving area* to be defined, e.g. a circle of known (or time-varying) radius around a moving point object. A clause such as `[AREA A]` must be used *always* in combination with another window type based on timestamps (*sliding* or *landmark*); or else, it would select all tuples belonging to objects that crossed area `A` anytime in the past.

iv. Trajectory-based windows. Clearly, this is a value-based window that examines any but spatiotemporal attributes of a trajectory stream and extracts elements according to the criteria specified. Syntax `[WHERE <trajectory conditional expression>]` may be used so as to isolate qualifying tuples of certain trajectories, e.g. `WHERE id IN (24, 89, 425)`. A possible use of this window type might be to split a trajectory stream into separate *substreams* based on objects' identities. Such a clause need be combined with time-based, partitioned or area-based (but not

tuple-based) windows, thus limiting the number of tuples returned. In contrast to the `WHERE` clause used in `SELECT` queries, this construct is applied to trajectory streams prior to any other processing for a given query (e.g. joins to another stream or relation).

5.2 Issues in Continuous Queries over Trajectories

We now informally point out certain issues related to query processing of continuous queries on moving objects. The notion of *punctuations* [TMSF02] could prove particularly constructive in query evaluation. For instance, extra tuples may be interleaved in a trajectory stream so as to indicate that a moving object has just crossed the boundary of an area or that it was found very close to another object or a known reference location.

The presence of numerous continuous queries over the same trajectory segments puts forward the need for *multiple query optimization* [Sel88]. In particular, it seems reasonable that expensive spatial operations (e.g. intersections) common to several queries need not be carried out in isolation from each other. Instead, grouping similar predicates or operators together, evaluating them and finally disseminating intermediate results to active queries, could cut execution costs considerably.

Last, but not least, *trajectory sketches* may be worth introducing. Due to the large volume of positional data flowing into the system, a compressed synopsis could provide an acceptable approximation for each trajectory (using sampling, wavelets or other summary structures).

6 Conclusions and Future Work

This paper considers streams as first-class concepts in a data management system and presents a new approach in modeling moving point objects by representing their continuous motion as trajectory streams. Basic predicates and operators are identified, under the assumption that movement takes place in one temporal and two spatial dimensions. Formulation of continuous queries over trajectories turns out to be feasible and intuitive, when window constructs are incorporated in a query language developed for managing data streams. Preliminary attempts to formulate continuous queries on moving objects in two data stream prototypes has showed encouraging results, and some challenging issues as well.

We believe that this approach is a promising area for future research. We plan to further study modeling of moving objects, introducing algebraic constructs for windows and proposing syntax rules for query language. Indexing trajectories by utilizing summaries as auxiliary structures in the presence of continuous positional updates is also important for improved query evaluation. Finally, shared execution and multiple query optimization of various spatiotemporal predicates is considered a major challenge in such a dynamic environment.

Acknowledgements. We greatly appreciate availability of data stream prototype systems, STREAM from Stanford University and TelegraphCQ from UC Berkeley. Comments by anonymous reviewers have improved the presentation of this paper.

References

[ACC+03] 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, August 2003.

[ABW03] A. Arasu, S. Babu, and J. Widom. An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations. In *Proceedings of the 9th Int'l Conference on Data Base Programming Languages*, Berlin, Germany, Sept. 2003.

[AUR04] AURORA Project: Website available at `http://www.cs.brown.edu/research/aurora/`

[BBD+02] B. Babcock, S. Babu, M.Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *Proceedings of the 21st ACM Symposium on Principles of Database Systems*, pp.1-16, Madison, Wisconsin, May 2002.

[CCD+03] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S.R. Madden, V. Raman, F. Reiss, M.A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proceedings of the 1st Conference on Innovative Data Systems Research (CIDR'03)*, Asilomar, California, January 2003.

[EGSV99] M. Erwig, R.H. Güting, M. M. Schneider, and M. Vazirgiannis. Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. *Geoinformatica*, 3(3):269-296, 1999.

[GL04] B. Gedik, and L. Liu. MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System. In *Proceedings of the 9th Int'l Conference on Extending Database Technology*, Heraklion, Greece, March 2004

[GBE+00] R.H. Güting, M.H. Böhlen, M. Erwig, C.S. Jensen, N.A. Lorentzos, M. Schneider, and M. Vazirgiannis. A Foundation for Representing and Querying Moving Objects. *ACM Transactions on Database Systems*, 25 (1): 1-42, 2000.

[ISS03] G. S. Iwerks, H. Samet, and K. Smith. Continuous k-Nearest Neighbor Queries for Continuously Moving Points with Updates. In *Proceedings of the 29th Int'l Conference on Very Large Data Bases*, Berlin, Germany,September 2003.

[KSF+03] M. Koubarakis, T. Sellis, A. Frank, S. Grumbach, R.-H. Güting, C. Jensen, N. Lorentzos, E. Nardelli, B. Pernici, Y. Manolopoulos, B. Theodoulidis, N. Tryfona, H.-J. Schek, and M. Scholl (eds.) *Spatiotemporal Databases: The CHOROCHRONOS Approach*, LNCS vol. 2520, Springer, 2003.

[MXA04] M.F. Mokbel, X. Xiong, W. G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In *Proceedings of 23rd ACM SIGMOD Int'l Conference on Management of Data*, Paris, June 2004.

[PJT00] D. Pfoser, C. Jensen, and Y. Theodoridis. Novel Approaches in Query Processing for Moving Objects. In *Proceedings of the 26th Int'l Conference on Very Large Data Bases,* Cairo, Egypt, September 2000.

[SWCD97] A. Prasad Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Proceedings of IEEE Int'l Conference on Data Engineering (ICDE'97)*, pp. 422-432, 1997.

[Sel88] T. K. Sellis. Multiple-Query Optimization. *ACM Transactions on Database Systems*, 13 (1): 23-52, 1988.

[SQR04] SQR – A Stream Query Repository. Available at `http://www-db.stanford.edu/stream/sqr`.

[STR04] STREAM Project: Website available at `http://www-db.stanford.edu/stream/`

[TEL04] TelegraphCQ Project: Website available at `http://telegraph.cs.berkeley.edu/telegraphcq/v0.2/`

[TMSF02] P. Tucker, D. Maier, T. Sheard, and L. Fegaras. Enhancing Relational Operators for Querying over Punctuated Data Streams. In *Proceedings of the 28th Int'l Conference on Very Large Data Bases*, Hong Kong, China, 2002.

Indexing Query Regions for Streaming Geospatial Data

Quinn Hart

CalSpace
University of California, Davis
Davis, CA 95616, U.S.A.
qjhart@ucdavis.edu

Michael Gertz

Department of Computer Science
University of California, Davis
Davis, CA 95616, U.S.A.
gertz@cs.ucdavis.edu

Abstract

This paper introduces the Dynamic Cascade Tree (DCT), a structure designed to index query regions on multi-dimensional data streams. The DCT is designed for a stream management system with a particular focus on Remotely-Sensed Imagery (RSI) data streams. For these streams, an important query operation is to efficiently restrict incoming geospatial data to specified regions of interest. As nearly every query to an RSI stream has a spatial restriction, it makes sense to optimize specifically for this operation. In addition, spatial data is highly ordered in it's arrival. The DCT takes advantage of this trendiness. The problem generalizes to solving many *stabbing point* queries. While the worst case performance is quite bad, the DCT performs very well when the stabbing point exhibits certain trending characteristics that are common in RSI data streams. This paper describes the DCT, discusses performance issues, and provides extensions of the DCT.

1 Introduction

New methods for processing streaming data [1, 2, 5] have a great deal of potential impact for remotely-sensed geospatial image data originating from the various satellites orbiting the Earth. Besides its typically large bandwidth, Remotely-Sensed Imagery (RSI) data has a number of characteristics that are different from generic streaming data. One important difference is that streaming RSI data is highly organized with respect to it's spatial components. This organization varies for different data streams, but generally image data will arrive in contiguous packets of data. These packets may be individual pixels, rows of pixels, or small images, depending on the instrument. Within a packet, the organization of the pixels is well defined. Consecutive data packets from an RSI data stream are usually close to one another spatially. Also, an RSI data stream is arriving at a high data rate, but usually only at one or a small number spatial locations at a time. In addition, most queries against an RSI data stream include operations to restrict the geospatial data to be processed to specified regions of interest. Therefore, an RSI stream management system needs to efficiently intersect incoming geospatial image data with a possibly large number of query regions.

In this paper, we present a method for intersecting incoming geospatial image data with multiple spatial restrictions, that is, queries that request incoming data for particular regions only. For this, we introduce the Dynamic Cascade Tree (DCT), a structure to index query regions and to provide for efficient insertions and deletions of queries. The DCT supports *stabbing point queries* [3] for a single moving point. A stabbing query is a simple query that, for a given point, will identify all indexed regions that contain that point. For an incoming RSI data stream, the structure is used to efficiently determine what queries are interested in that data. The trendiness inherent to most types of streaming RSI data is exploited to build a small index that is especially efficient when multiple stabbing queries are in close proximity. Based on the information provided by the DCT, query plans can be generated and incoming data can be pipelined to respective query operators, thus providing the basis for multiple-query processing models for streaming RSI data.

The remainder of the paper is structured as follows. Section 2 describes related research for similar problem domains. Section 3 outlines the data and query model underlying RSI. Section 4 describes the *DCT* in detail. Section 5 discusses the performance of the *DCT*, and implications of input regions and stabbing point trends. Section 6 describes extensions and modifications of the *DCT*.

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

2 Related Work

Many data structures have been developed for one and two dimensional stabbing queries including, among others, interval trees, priority search trees, and segment trees [3]. Space partitioning methods for answering stabbing queries include quadtrees, hashes, and numerous variants of R-trees.

The two most common methods for solving stabbing queries in two dimensions are multi-level segment trees [3] and R-trees [4]. Using multi-level segment trees, one dimension of the region is stored in a segment tree, while the second dimension is indexed with an associated interval structure for each node in the first segment tree. Storage for these structures can be $O(n \lg n)$, with stabbing query times of $O(\lg^2 n)$. Dynamic maintenance of such a structure is more complicated and requires larger storage costs [12]. It is difficult to modify the multi-level segment tree to improve results for trending data. If the input stabbing point moves a small distance, which doesn't change the query results, it still would take $O(\lg n)$ time to respond. That is because even if every node in the multi-level segment tree maintains knowledge of the previous point, it would still take $\lg(n)$ time to traverse the primary segment tree to discover that no changes to the query occurred.

R-trees solve the stabbing query problem by recursively traversing through successive minimum bounding rectangles that include the extent of all regions in the sub-tree, generally with good performance. Since these rectangle regions can overlap, there can be no savings from knowing the previous stabbing query, as there is no way to know if an entirely new path through the segment tree needs to be traversed. R^+-trees[11] can have better performance for these trending stabbing points, since the minimum bounding rectangles are not allowed to overlap and so maintaining the previous query can help verify a query hasn't left a particular region. R^+-trees have problems with redundant storage, dynamic updates, and potential deadlocks [7].

Probably the approach most similar to the *DCT* described in this paper is that of the Query index [6, 9]. The Query index builds a space partitioning index on a set of static query regions, and at each time interval, it allows a number of moving objects to probe the index to determine overlapping queries. Main memory implementations show that grid-based hashing of query regions generally outperform R-tree or quad-tree based methods. SINA [8] describes an incremental method to solving the problem of intersecting moving objects. However, much of the approach involves efficient integration with disk-based static queries, and a complete main-memory implementation would be more similar to the query index approach.

All the indices described above anticipate a large number of moving objects to be indexed against the query regions. The RSI application described above is different in the sense that the *DCT* index is designed for a single or small number of moving objects, where the input rate of data for that moving object is very high. In this application, the desire is for a small index that can efficiently route a high volume data stream to the query regions, rather than indices that are interested in the location of the moving objects.

In one sense, the *DCT*is basically a method for dynamically maintaining a region around a current point for which the current set of query regions is valid, and identifying where this result set is no longer valid. Another method for dynamically describing a neighborhood of validity for a stabbing query using R-trees was proposed by Zhang et al. [14]. This method builds an explicit region of validity around a current point, which can then be used to verify that a stabbing point will not result in a different response. The technique makes a number of additional queries to the R-tree index in order to build this region. Like the technique described in this paper, this could result in cost savings if many subsequent stabbing queries are located within the region of validity.

3 Data and Query Model

Our data model for RSI data is based on raster images. To allow for different types of objects called image, we employ some concepts from the Image Algebra [13] and extend these concepts to account for the specifics of streaming RSI data.

An image consists of a set of points and values associated with these points. The *point set* of an image is a set of points and an associated measure of distance between points. As our interest is in RSI, we choose as point set $\mathbf{X}$ a subset of $\mathbb{R}^3$, with a point $\mathbf{x} \in \mathbf{X}$ of the form $\mathbf{x} = \{x, y, t\}$. The pair $\{x, y\}$ denotes a spatial location in some spatial reference system, and t denotes a timestamp. Thus, a point set exhibits spatio-temporal characteristics. For example, weather satellites continuously transmit images of clouds over one hemisphere of the earth. Given such an image, the *point set* corresponds to the actual location for each point in the image, along with the time that the image was acquired.

A *value set* $\mathbb{V}$ provides the values associated with points in a given point set. For the weather satellite example, the *value set* includes all the intensity levels in the image. Value sets can be complex, in the case of color images, $\mathbb{V}$ is a subset of $\mathbb{Z}^3$ for the red, green, and blue components. For gray-scale images, it is a subset of $\mathbb{Z}$. Based on the concepts of point and value sets, we can now give a functional representation of an image.

Definition 3.1 Given a point set $\mathbf{X}$ and value set $\mathbb{V}$. A $\mathbb{V}$-valued image $\mathbf{i}$ is a function from $\mathbf{X}$ to $\mathbb{V}$, denoted $\mathbf{i} = \{(\mathbf{x}, \mathbf{i}(\mathbf{x})) \mid \mathbf{x} \in \mathbf{X}\}$. The pair $(\mathbf{x}, \mathbf{i}(\mathbf{x}))$ is called a *pixel* of $\mathbf{i}$. $\mathbf{x}$ is the spatio-temporal component of the pixel and $\mathbf{i}(\mathbf{x}) \in \mathbb{V}$ is the *pixel value* at point $\mathbf{x}$.

Different types of RSI have different orderings and structures. Figure 1 shows these structures. Airborne cameras obtain imagery in an image-by-image manner. Some sensors, such as NOAA's Geostationary Operational Environmental Satellite (GOES), obtain RSI data basically in a row-by-row fashion. Although conceptually the data collected by GOES can be viewed as a stream of images, the images are actually obtained in a *row-scan order* in which pixels are delivered a few lines at a time. Still other types of sensors gather data on a pixel-by-pixel basis.

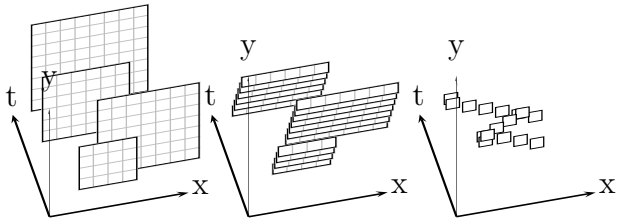


Figure 1: Examples of different point set orderings for streaming RSI data: image-by-image, line-by-line, and point-by-point.

All the above scenarios describing how RSI data is obtained show an important characteristics we aim to exploit in our approach: *the points in a point set exhibit certain trends*. That is, consecutive points in a stream of RSI data have a close spatial and temporal proximity. The only exception is where the last point of a line in an image is followed by the first point of a new image (scenario on the left in Figure 1). As we will show in the following section, knowing about the *trendiness* of incoming geospatial point data can have a significant influence on how queries against a stream of RSI data are processed.

Queries against a stream of RSI data are typically continuous queries that run for a long time and may include complex operators, such as spatial transforms or aggregates (see [13] for operations on point sets). However, since most applications are not interested in the complete region covered by a sensor, *spatial restriction* is a type of operation common to all queries. A spatial restriction specifies a *region of interest*, primarily in the form of a rectangle, and typically precedes other operations on point data. The Dynamic Cascade Tree (DCT) includes an index structure and algorithms to efficiently determine what query regions are affected by incoming RSI image data, and to pass those images to the appropriate queries.

4 The Dynamic Cascade Tree (DCT)

The problem of quickly answering multiple queries on a stream of RSI data is basically solving a normal *stabbing query* [3] for a point. That is, as query result, a stabbing query determines all query regions that contain the current point delivered by the RSI data stream. For RSI data, the stabbing points are special in that the next stabbing point is typically very close to the previous stabbing point. The goal is to take advantage of the trendiness of stabbing points and to develop index structures that improve the search performance for subsequent stabbing queries.

The structure proposed in the following builds an index that is dynamically tuned to the current location of RSI data. For a given point, the *DCT* maintains the regions around that point where the query result will change. Stabbing queries can be answered in constant time if the new stabbing point has the same result as the previous query and will incrementally update a new result set based on the previous set when the result is different. The structure is designed to be small and quickly allow for insertions and deletions of new query regions. It assumes some particular characteristics of the input stream, notably that the stream changes in such a way that many subsequent incoming RSI data will contribute to the same result set(s) to region queries as the current point. Therefore, the cost of maintaining a dynamic structure can be amortized over a large set of queries. Section 5 describes in more detail the performance implications of the regions and input data stream.

4.1 DCT Components

Figure 2 gives an overview of the data structure employed, which we term a *Dynamic Cascade Tree (DCT)*. The figure shows a set of query regions $a, b, \ldots, f$, the node cn, denoting the most recent stabbing point from the data stream, and the associated structures for the *DCT*. The figure describes a *DCT* that indexes two dimensions. There is no required order in how the dimensions are referenced, and the example shows the vertical (y) dimension being the first dimension indexed in the *DCT*.

The components of *DCT* are pleasantly simple extensions to a binary tree. In the example and following pseudo-code, we assume that we have two simple search structures, *List* and *2-Key-List*. *List* supports INSERT(KEY,VALUE), DELETE(KEY), and ENUMERATE(). Keys in *List* are unique for each value. In our approach, we use a simple skip list [10] to implement *List*. The *2-Key-List* is incrementally more complex. It supports INSERT($key_1, key_2, value$), DELETE(key_1, key_2) and ENUMERATE(key_1) using two keys. The combination of two keys is unique for each value. ENUMERATE(key_1) enumerates all the values in the *2-Key-List*, entered with key_1. An implementation of *2-Key-List* could be a skip list using key_1, where each node has an associated skip list using key_2. With this implementation, for *2-Key-List* .DELETE(key_1, key_2), if the deletion causes an empty set in the associated key_1 node, then that entire node is deleted.

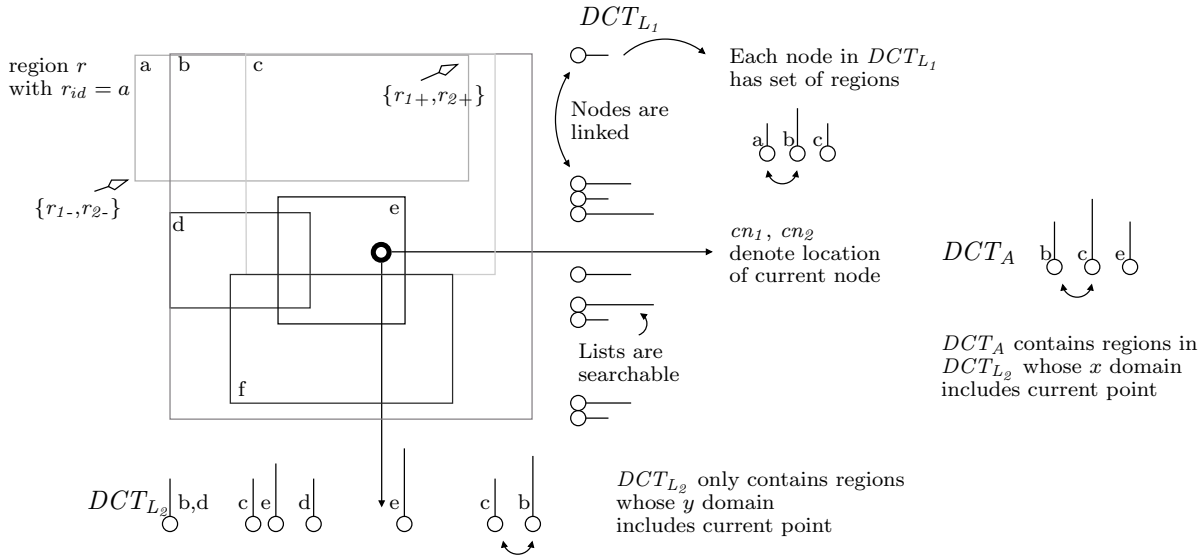


Figure 2: Dynamic Cascade Tree (DCT) with query regions $a, b, \ldots, f$; a query region, r, is described by it's minimum (lower left) corner $\{r_{1-}, r_{2-}\}$ and maximum (upper right) corner $\{r_{1+}, r_{2+}\}$

List leaf nodes contain region ids, r_{id} as keys with a pointer to the query region, r as the value. For the *2-Key-List*, key_1 is the value of the endpoint, and key_2 is the id of the query region, denoted r_{id}. The leaf nodes of a *2-Key-List* correspond to the half-open line segments between two endpoints. Leaf nodes of a *2-Key-List* also have pointers to the next and previous nodes in sorted order, allowing for linked list traversal to leaf nodes. One reason for choosing a skip list implementation is that the forward pointers already exist, and only an additional back pointer is added to a normal skip list.

The *DCT* maintains a separate *2-Key-List* for each dimension of the individual query regions. In Figure 2, these are DCT_{L_1} and DCT_{L_2}. In addition, the *DCT* maintains a *List*, DCT_A, of all query regions that overlap the current stabbing point. A second structure, *cn*, maintains pointers to nodes within each *2-Key-List*, corresponding to the most current stabbing point.

The *2-Key-List* for the first dimension, DCT_{L_1} contains the minimum and maximum endpoints in the 1st (y) dimension, $\{r_{1-}, r_{1+}\}$, for every query region r.

The next *2-Key-List*, DCT_{L_2} contains keys on the endpoints in the 2nd (x) dimension and r_{id}. DCT_{L_2} does not contain the endpoints of all the regions in *DCT*, but only the regions whose 1st dimension (y) overlap with the current node, cn_1.

If the query regions contain more dimensions, additional *2-Key-List* structures are added to the *DCT*, where each subsequent *2-Key-List* only indexes those regions that overlap the current point up to that dimension.

In Figure 2, *cn* contains two pointers, cn_1 and cn_2 to nodes within both DCT_{L_1} and DCT_{L_2} corresponding to the location of the most recent stabbing point.

DCT_A is the final *List* that contains all the currently selected query regions that correspond to the current stabbing point query. Just as DCT_{L_2} contains only a subset of the regions of DCT_{L_1} that contain the cn_1 node, DCT_A contains the subset of DCT_{L_2} where the cn_2 node is contained by the 2nd dimension of each region. The DCT_{L_1}, DCT_{L_2}, and DCT_A structures make up a cascade of indexes, each a subset of the previous index.

4.2 Updating query regions in the DCT

The *DCT* is initialized by creating the *2-Key-List* and *List* structures, adding a starting node for DCT_{L_2} and DCT_{L_1} outside their valid range, and assigning cn_2 and cn_1 to those nodes.

Algorithm 1 shows the pseudo-code for inserting query regions into *DCT*. Insertion and deletion are simple routines. For insertion, a region is first inserted into DCT_{L_1} and then successively into DCT_{L_2} and DCT_A if the region overlaps the current node *cn* in the other dimensions. DELETE-REGION is similar to the insertion, taking a region r as input. It should be clear that the structures DCT_{L_2} and DCT_A need to be maintained when new regions are inserted and deleted, and for each new stabbing point. Since DCT_{L_2} contains regions overlapping the current node *cn*, when a new stabbing point arrives where a y boundary for any region in the *DCT* is crossed, then the DCT_{L_2} structure needs to be modified to account for the regions to be included or deleted from consideration. A similar method needs to be associated with boundary crossings in the x dimension while traversing DCT_{L_2} and modifying DCT_A.

Algorithm 1 Inserting Query Regions in DCT

```
INSERT-REGION(DCT, cn, r)
1  ▷ Input: DCT,
2  ▷        current node cn= {cn_2,cn_1}
3  ▷        region, r = {r_2-, r_1-, r_2+, r_1+}
4  INSERT-ITH(DCT, cn, r, 1)

INSERT-ITH(DCT, cn, r, i)
1  ▷ Input: DCT,cn,r same as INSERT-REGION
2  ▷        dimension, i
3  I ← DCT_{L_i} ▷ ith 2-Key-List
4  if (i > dimensions of r)
5     then DCT_A.INSERT(r_id, r)
6     else I.INSERT(r_i-, r_id, r)
7          I.INSERT(r_i+, r_id, r)
8          if (r_i- ≤ cn_i . key and r_i+ > cn_i . key)
9             then INSERT-ITH(DCT, cn, r, i + 1)
```

4.3 Querying the DCT

The algorithm for reporting selected (active) query regions for a new stabbing point $np = \{np_1, np_2\}$ begins by traversing the *2-Key-List* DCT_{L_1} in the y direction from the current node $cn = \{cn_1, cn_2\}$ to the node containing np_1 going through every intermediate node using the linked list access on the leaf nodes of DCT_{L_1}. At each boundary crossing, as regions are entered or exited, those regions need to be added to or deleted from the DCT_{L_2} *2-Key-List*. When the point has traversed to the node containing np_1, traversal begins in the x direction, moving from cn_2 to the node containing np_2. As with DCT_{L_1}, when the traversal hits x boundary points, the entered query regions are added to DCT_A and the exited regions are deleted from DCT_A. When the traversal reaches np, cn contains pointers to the nodes containing np, DCT_{L_2} contains x endpoints to all the regions with y domains that contain np_1, and DCT_A lists all regions that contain np. DCT_A is then enumerated to report all the query regions that are affected by the new stabbing point np.

Figure 3 shows an example of an update of the structures within DCT on reporting regions for a new stabbing point. This extends the example of Figure 2. In this example, the new point has crossed a y boundary that contains two region endpoints, c and f. As the current point traverses in the y direction to this new point, the x endpoints of region c are removed from DCT_{L_2}, and the endpoints of f are added to DCT_{L_2}. When the endpoints of these regions are deleted, the regions themselves are also deleted from DCT_A. In the example, c is deleted from and f inserted into DCT_A. After reaching np_1, DCT_{L_2} is traversed in the x direction. In the example, this results in e being deleted from DCT_A. Finally, DCT_A is enumerated, completing the procedure.

Algorithm 2 describes the REPORT-REGIONS procedure, which reports query regions for a new stabbing point, while updating the structures of the DCT. REPORT-REGIONS simply calls UPDATE-ITH on the first dimension and then reports all regions in the DCT_A *List*. The procedure UPDATE-ITH recursively visits each dimension in the DCT and adds and deletes regions from the associated *2-Key-List* for that dimension. In UPDATE-ITH, Lines 7 to 12 traverse the dimension backwards. At each endpoint, the corresponding region is either added or removed from *2-Key-List* in the next dimension. Lines 13 to 18 execute a similar traversal in the forward direction, also adding and removing regions from the next *2-Key-List*. Only one of the **while** loops is executed at each invocation. After traversing to np_i, UPDATE-ITH is called again for the next indexed dimension, $i + 1$. This continues through all dimensions of the DCT. Note that INSERT-ITH will add regions into the DCT_A *List*when traversing the final dimension of the DCT. When all dimensions have been traversed, DCT_A contains all the regions that overlap np in all dimensions.

Algorithm 2 Stabbing queries in DCT

```
REPORT-REGIONS(DCT,cn,np)
1  ▷ Input: DCT
2  ▷        current node(s) cn= {cn_1,cn_2,...}
3  ▷        new stabbing point, np= {np_1,np_2,...}
4  ▷ Output: List of query regions containing np.
5  UPDATE-ITH(DCT,cn,np,1)
6  return DCT_A.ENUMERATE

UPDATE-ITH(DCT, cn, np, i)
 1  ▷ Input: same as REPORT-REGIONS
 2  ▷        dimension, i
 3  ▷ Output: List of query regions containing np.
 4  if (i > max dimension of r)
 5     then return
 6  I ← DCT_{L_i} ▷ i-th 2-Key-List of DCT
 7  while (np_i < cn_i . key)
 8        do for r ∈ I.ENUMERATE(cn_i)
 9           if r_i- = cn_i . key
10              then DELETE-ITH(DCT,cn,r,i + 1)
11              else INSERT-ITH(DCT,cn,r,i + 1)
12        cn_i ← cn_i .PREV
13  while (np_i > cn_i .NEXT. key)
14        do cn_i ← cn_i .NEXT
15           for r ∈ I.ENUMERATE(cn_i)
16           if (r_i+ = cn_i . key)
17              then DELETE-ITH(DCT,cn,r,i + 1)
18              else INSERT-ITH(DCT,cn,r,i + 1)
19  UPDATE-ITH(DCT,cn,np,i + 1)
```

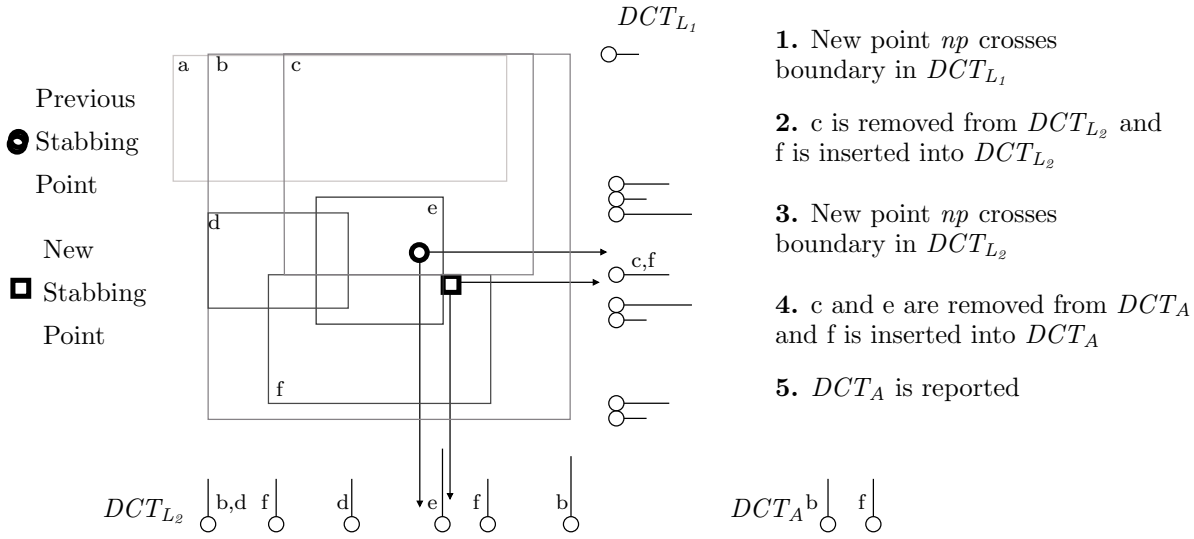


Figure 3: Stabbing point moving in Report-Regions

5 DCT Performance

The performance of Report-Regions is highly dependant on the location of the regions, the trending properties of the stream data, and the interaction of the two parameters. For executions of Report-Regions, with n being the total number of query regions and k being the average number of resultant regions, the average time of execution can range from $O(k)$ in the best case to $O(n \lg n + k)$ in the worst case. Reasonable experiments could be designed that would approach either of these limits. Instead, there are rules to consider for the application of the DCT.

5.1 Insertions and deletions of query regions

The DCT data structure is small and robust to many insertions and deletions of query regions. Insertions and deletions take $O(\lg n)$ time as the query region is potentially added to the structures DCT_{L_1}, DCT_{L_2}, and DCT_A. *List* and *2-Key-List* are simple to maintain dynamically in $O(n)$ space.

5.2 Number of boundaries crossed

The DCT is designed for trending data, which can most quantitatively be measured by the number of region boundary crossings from one stabbing point to the next. This structure works best when the number of query boundaries crossed on subsequent input points is not large. When no boundaries are crossed, then no internal lists are modified, and Report-Regions runs in $O(k)$ time. When a boundary is crossed in the i-th dimension, then each region in the crossed DCT_{L_i} node needs to be inserted into or deleted from the $(i+1)$-th *2-Key-List* structure. This is true for regions whose domains in subsequent dimensions do not overlap the new stabbing point *np*, and thus do not contribute to the DCT_A structure. The cost of Report-Regions in this case can be as high as $O(n \lg n + k)$, since many non-overlapping regions are inserted into the DCT_{L_2} structure. There is no performance difference in whether the crossing occurs at a single boundary with many regions at that node, or over many boundary crossings with few regions.

The DCT data structure indexes lazily in the sense that for insertions of new regions that do not overlap cn, it only indexes a new region on its first dimension, and not on all dimensions. Thus, boundary crossing costs must take some time to index on these dimensions of the regions. The problem with the DCT is that these costs can occur many times in the travel of the input stabbing points. Rather than indexing these values once, the DCT re-indexes a subset of points multiple times as boundaries are crossed. The hope is that in a new set of regions, many subsequent stabbing points will be in these areas, and the low cost of those stabbing points will make up for the extra cost of maintaining a dynamic index.

5.3 Trajectory of the trending data point

Another aspect affecting performance is the trajectory of the input stabbing points. For example, consider a DCT in two dimensions, like the one shown in Figure 2, with trajectories that are increasing or decreasing monotonically in the x and y dimensions. In these cases, regions are put into the DCT_{L_2} structure at most one time. The total time maintaining the DCT_{L_2} structure then is at most $O(n \lg n)$, fixing a bound on the dynamic maintenance costs of the DCT. The total cost of m stabbing queries over that trajectory would be $O(n \lg n + mk)$, where k is the average number of regions per stabbing point. For a two-dimensional segment tree implementation, the total cost would be $O(n \lg n + m \lg^2 n + mk)$, which includes the static cost of maintaining a segment tree and does not include extra costs for dynamically maintaining that tree.

On the other hand, data with a more erratic trajectory can result in poor performance. Consider a point that repeatedly crosses a single boundary containing all n region boundaries. Again, each iteration would require $O(n \lg n)$ time, as the DCT_{L_2} and DCT_A structures are both repeatedly made up and torn down.

Also, the DCT as described in the Figures above, which indexes on y and then x, favors stream data points that trend in the x direction over data that trend in the y direction. The reason for this is that more regions added into the DCT_{L_2} structure end up being reported, and the dynamic structure building is not wasted. Also, there are fewer insertions and deletions in the DCT_{L_2} structure in the first place. When the stabbing point np crosses a boundary in the x direction, it still takes $O(\lg n)$ time to insert, as it updates DCT_A, but this is more useful work in updating DCT_A than a y crossing boundary, which can spend wasteful time adding points into DCT_{L_2} that might never be used. Where a stabbing query for a y trending trajectory can have a worst case time in REPORT-REGIONS of $O(n \lg n + k)$, the worst case time in x trending stabbing points is $O(n + k)$, when worthless insertions into DCT_A are skipped, as described below.

This shows that order in the cascade is very important, and dimensions that see more boundary crossings for subsequent stabs into the DCT should be pushed deeper into the structure. Boundary crossings are of course dependent on the trajectory of the stabbing point and the organization of the regions in the DCT.

5.4 Skipping worthless insertions

As mentioned above, when the next point of the input stream trends a long way with respect to the number of query boundaries traversed, then the time for REPORT-REGIONS goes up to at least the number of regions contained in all the boundaries crossed. Regions that are both entered and exited in the course of a single traversal to np are even worse. Their endpoints are needlessly added, then deleted from DCT_{L_2}, at a cost of up to $O(\lg n)$, and never queried. This can easily be remedied, but for clarity was left out of the initial UPDATE-ITH algorithm. When encountering a region at a boundary crossing, check that the region will remain a valid region when np has finished it's traversal before inserting into the *2-Key-List* structure. This prevents wasted index modifications, but does not help with the basic problem of long traverses of np or input points that cross back and forth across expensive boundaries. Algorithm 3 shows the modifications made to UPDATE-ITH in lines 7 to 12 in Algorithm 2. A similar modification would be made to the forward loop in UPDATE-ITH.

Algorithm 3 Stabbing query modification

```
 8  ...
 9  ▷ replaces lines 7 to 12 in UPDATE-ITH
10  old = cn_i.key
11  while (np_i < cn_i . key)
12        do for r ∈ I.ENUMERATE(cn_i)
13           if (r_i- = cn_i . key) and (old < r_i+)
14              then
15                   DELETE-ITH(DCT,cn,r,i + 1)
16           elseif (r_i+ = cn_i . key) and (r_i- < np_i)
17              then
18                   INSERT-ITH(DCT,cn,r,i + 1)
19        cn_i ← cn_i .PREV
20           ...
```

6 DCT Extensions and Modifications

In Section 4, the discussion centered on answering a simple stabbing query for a single point and a number of regions in a two dimensional space. This basic framework can undergo some simple modifications to handle a number of similar types of queries.

6.1 Window queries

The first general area is for regions other than points. The stabbing query can easily be changed from a single point to a constant size rectangle. Constant size rectangles are common in RSI data. In this case, track the center location of the stabbing rectangle, and when inserting new regions of interest, extend the boundaries by half the width and height of the rectangle queries. Intersections of the modified regions and the stabbing point will coincide with intersections of the original regions and the rectangle query.

Queries on stabbing rectangles with more dynamic extents are possible, too. For example, in the two dimensional case, track both edges of the stabbing rectangle in the DCT_{L_1} and DCT_{L_2} structures. The leading edge of the lines in the y dimension will track insertions into DCT_{L_2}, and the trailing edges will track deletions from DCT_{L_2}. Leading and trailing are with respect to the previous stabbing rectangle. Rectangles that are growing in size from the previous rectangle may have two leading edges, and shrinking rectangles will have two trailing edges. A similar strategy is used for mapping the DCT_{L_2} structure to the DCT_A structure.

Many remote-sensing image data come in a row-by-row scheme (see Figure 1). For this special case, use and maintain the DCT_{L_1} structure as in the stabbing point example, and only modify the DCT_{L_2} structure for different lengths of the individual rows of data.

6.2 Adding dimensions

Adding an additional dimension is a simple extension by adding another intermediate *2-Key-List* to the DCT_{L_1}, DCT_{L_2}, and DCT_A cascade. For example, a time dimension on a rectangle query, or in this instance a cube query, could be added. As discussed

in Section 5, it is best to order the dimensions so the most varying is on the deeper levels of the *DCT*. The monotonic increase of time would make it a good candidate for the level before the DCT_A structure. However, if a system contains query regions with a mostly unbounded temporal dimension, then it could also be located at the first level. One nice feature of making time the first structure of the cascade is that it also does double duty in providing a structure to prune regions that have expired. For geospatial data streams, if incoming pixels are timestamped, then new stabbing points that have identified regions whose time extent has ended can be removed from the time structure as well as the other cascaded structures.

6.3 Non-spatial multi-dimensional data

The focus of the *DCT* has been on spatial data. However, these techniques could be applied to a general multi-dimensional data space. The obvious modifications could be made and extended to n dimensions as outlined above. One important issue to address is the order of the cascade of *2-Key-List* structures. As described in Section 5, if regions are dispersed equally, it is generally best to move the most varying parameter to the end of the cascade, and move the least varying to the top. This structure is also appropriate for range queries over a single dimension over trending data by maintaining only the top *2-Key-List*. Since the index sizes are relatively small, it is conceivable that a system could consistently maintain one dimensional *DCT* structures, and then dynamically begins to build two or n dimensional structures when queries requesting such regions are instantiated. The advantage of this method is that the structures are no longer maintained as the queries requesting those regions are deleted.

7 Conclusions and Future Work

In this paper, we have presented the Dynamic Cascade Tree (DCT), a simple data structure designed to follow trending geospatial data points that constitute streaming geospatial image data. The focus has been on two dimensional stabbing queries, but we have offered modifications to a number of related problems. Theoretical and rule of thumb performance bounds have been discussed. Initial further work will focus on more experimental tests of the *DCT* for various realistic scenarios.

We are currently implementing the *DCT* as part of a query processing architecture to support complex continuous queries over streams of remotely-sensed geospatial image data. The proposed *DCT* will build an important component to facilitate the optimization of multiple queries against such a stream.

Acknowledgments

This work was partially supported by the NSF grant IIS-0326517.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom. Models and issues in data stream systems. In *Proc. of the 21st Symposium on Principles of Database Systems*, 1–16, 2002.

[2] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, S. Zdonik. Monitoring streams - a new class of data management applications. In *VLDB 2002*, 215–226.

[3] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2000.

[4] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *In Proc. ACM SIGMOD Int. Conf. on Management of Data*, 47–57, 1984.

[5] J. M. Hellerstein, M. J. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, M. A. Shah. Adaptive query processing: Technology in evolution. *IEEE Data Engineering Bulletin*, 23(2):7–18, 2000.

[6] D. V. Kalashnikov, S. Prabhakar, S. E. Hambrusch. Main memory evaluation of monitoring queries over moving objects. *Distrib.& Parallel Databases*, 15(2):117–135, 2004.

[7] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, Y. Theodoridis. R-Trees have grown everywhere. Unpublished Techn. Report, 2003.

[8] M.F.Mokbel, X. Xiong, W.G. Aref. SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 623–634, 2004.

[9] S. Prabhakar, Y. Xia, D.V. Kalashnikov, W.G. Aref, S.E. Hambrusch. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE Trans. on Computers*, 51(10), 2002.

[10] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *CACM*, 33(6):668–676, 1990.

[11] T. K. Sellis, N. Roussopoulos, C. Faloutsos. The R^+-tree: A dynamic index for multi-dimensional objects. In *VLDB 1987*, 507–518.

[12] M. J. van Kreveld, M. K. Overmars. Concatenable segment trees. Technical report, Rijksuniversiteit Utrecht, 1988.

[13] J. N. Wilson, G. X. Ritter. *Handbook of Computer Vision – Algorithms in Image Algebra*. CRC Press, 2nd edition, 2001.

[14] J. Zhang, M. Zhu, D. Papadias, Y. Tao, D. L. Lee. Location-based spatial queries. In *Proc. of the 2003 ACM SIGMOD International Conference on Management of Data*, 443–454, 2003.

Condition Evaluation for Speculative Systems: a Streaming Time Series Case

X. Sean Wang
CS Dept., Univ. of Vermont
xywang@cs.uvm.edu

Like Gao
CS Dept., Univ. of Vermont
lgao@cs.uvm.edu

Min Wang
IBM T.J. Watson
min@us.ibm.com

Abstract

Application systems often need to react with certain actions whenever some preset conditions are satisfied. In many cases, the evaluation of these conditions takes long time, but some prediction of the results can be obtained rather quickly. In this situation, speculation may be a good idea. That is, the system takes predictions (speculation) to prepare (such as prefetch) for the possible reaction. Obviously, the risk is wasted efforts due to false alarms. Higher precision prediction results in less waste, but takes longer time and may reduce/eliminate the opportunity for speculation. A balance needs to be struck. A quality-driven prediction subsystem is thus necessary, so that the "user" of the prediction subsystem can impose quality (in terms of precision and response-time) requirements.

This paper focuses on such a prediction subsystem with conditions on streaming time series. Two problems need to be solved: how to predict the precision and how to achieve the required precision in an optimized way. The paper introduces a prediction model to tackle the first problem, and presents an algorithm to attack the second. Experiments show that the prediction subsystem works well.

1 Introduction

A speculative system is one that uses some kind of risk taking mechanism to achieve an overall gain. Prefetching in various scenarios, such as in processors, operating systems, and database systems, is a typical example. To speculate is generally to carry out some activity based on the prediction of the forthcoming requests, thereby saving some time when the actual requests are processed. The risk of speculation is wasted efforts while the gain is the potential overall increase in throughput.

Obviously, a number of factors affect the performance of a speculative system. Prediction precision is an important one. Usually, the higher the prediction precision is, the more the system gains. However, in certain situations, higher prediction precision takes longer time to produce. An overall strategy is needed that trades off response time with precision.

In this paper, we study a condition-driven system that takes reactive measures (by a reactive subsystem) whenever some preset conditions are satisfied (evaluated by a parallel condition evaluation subsystem). We assume the conditions evaluation subsystem contains a prediction subsystem that is responsible to provide a prediction to the truth values of the conditions. We focus on this prediction subsystem.

Since precision and response time of the prediction (by the prediction subsystem) are usually positively correlated, a balance must be struck between them in order to achieve overall gain. It is thus imperative for the prediction subsystem to have the ability to trade time with precision. In other words, we require that the prediction subsystem be able to satisfy some QoS (quality of service) requirements.

More specifically, given a set of conditions, the prediction subsystem will partition them into two sets, one true-set and one false-set. We measure the precision of the two sets by false positive/negative ratios. The smaller these values are, the more precise the prediction is. Another view is that the predictions are approximate answer to the evaluation of the conditions.* The response time is the time when the last condition in the true-set is produced by the prediction subsystem. This is due to the fact that most systems respond to conditions that become true (otherwise, it is usually not difficult to negate conditions).

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

*Hence, we will use *prediction* and *evaluation* interchangeably when no confusion arises.

In order to implement a quality-driven prediction subsystem, it is important to have the ability to measure the (intermediate) result quality during the evaluation process. While it is straightforward to measure the response time, it is impossible to measure the accuracy (i.e., false positive and negative ratios) precisely when an approximation method is used. Indeed, the precise accuracy can only be measured *a posteriori*, i.e., only after we know the actual evaluation results of the conditions. So we advocate measuring accuracy in an *a priori* manner, i.e., the false ratios are estimated based on prior knowledge. To do this, we build a prediction model based on historical data analysis. At the evaluation time, we use this prediction model to derive precision estimates. When the precision based on the prediction model is not enough to satisfy the "user" requirements, the prediction subsystem needs some processing to increase the precision.

Since the evaluation procedure is based on the prediction model that is probabilistic in nature, our system cannot guarantee the satisfaction of accuracy in a strict sense. Instead, similar to the *soft* quality of service (QoS) concept in computer network [9], the system satisfies the accuracy requirements in a "soft" manner. That is, the system guarantees with enough confidence that the *expected* accuracy will not exceed given thresholds. Specifically, the system allows the "user" (i.e., the reactive subsystem) to impose the following quality constraints:

- *Response time constraint*: All evaluation results must be reported within a given time limit.

- *Accuracy constraints*: The expected false positive and negative ratios must not exceed the given thresholds, with enough confidence (the confidence is deduced from the prediction model).

Since the prediction subsystem may not be able to satisfy both constraints simultaneously, especially when the constraints are strict and system resource is limited. We choose to let the user (i.e., the reactive subsystem) impose a precision requirement, and let the prediction subsystem optimize on the response time. In other cases, the user may want to impose a response time requirement (and then decide what to do based on the prediction precision), but this is beyond the scope of this workshop paper.

The rest of the paper is organized as follows. In Section 2, we briefly discuss an application where a speculative system with a condition prediction subsystem can be useful. In Section 3, we give some formally definitions of our problem, and in Section 4, we introduce our prediction model. We outline our evaluation algorithm in Section 5 and present experimental results in Section 6. We review related work in Section 7, and conclude the paper with Section 8.

2 Speculating Situation Manager

In [3], active systems are studied with an underlying situation manager, where a situation is a reactive entity that receives events as an input, combines composition filtering with content filtering, and detects situations as an output [3]. Many applications require reactions to situations (rather than single events). The high-level architecture of an application that uses the situation manager can be like that in Figure 1 [3]. An example situation can be that "within the same day for at least 5 times, a customer sells after buys the same stock" [3].

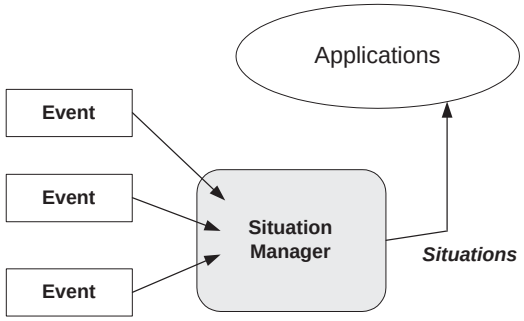


Figure 1: Situation manager high-level architecture.

A situation can take a long time to emerge from the situation manager due to two possible reasons. Firstly, the evaluation of conditions that involve complex operations may take a long time after all the events have arrived.† For some complex operators, some kind of approximation can be used to produce a prediction for the conditions. Secondly, the temporal distance between the first and the last events for the situation can be great. In the above example, only when the 5^{th} time that the customer sells after buys the same stock, the situation appears. This gives room for a speculative application system. Indeed, in the above example, when the 5^{th} buy of a stock by the same customer occurs, we probably have enough confidence to speculate that the 5^{th} sell will occur. In this case, we can tell the application with some confidence that the situation will occur so that the application can start to prepare reactive measures for the situation. If the situation does occur, the application can react faster. If the situation in the end does not occur, then the application just wasted some efforts (and rollback may be necessary). Depending on the application, this may be a risk worth taking.

In other words, when situations take long time to emerge and can be predicted with some confidence, it may be advantageous to predict certain situations. In this case, the situation manager becomes a speculating one, and applications become speculative. See Figure 2. In this case, the speculative application should

†AMIT does not allow complex operators like nearest neighbor search.

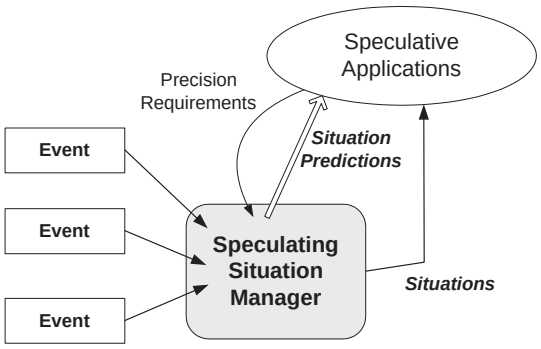


Figure 2: Speculating situation manager.

tell the situation manager how precise the prediction should be in order to achieve the best overall gain.

The question we explore in this paper is how to provide predictions to the true-set and false-set for the given conditions. Obviously, given enough time, the predictions can be made as precise as possible (the "worst" case is to wait until the conditions are evaluated fully, and "predictions" become actual values). The question is how to respond in the fastest way as long as the precision requirement is satisfied. In this paper, we assume the conditions are on streaming time series as defined below.

3 Basic Definitions

Streaming time series

A *time series* is a finite sequence of real numbers and the number of values in a time series is its *length*. A *streaming time series*, denoted s, is an infinite sequence of real numbers. At each time position i, however, the streaming time series takes the form of a finite sequence, assuming the last real number is the one that arrived at time i.

Condition on streaming time series

In general, a condition can be any user-defined predicate on streaming time series, and needs to be evaluated after every data arrival.

As an example, given (finite) time series x and y of the same length l, we may define the following correlation coefficient function:

$$corr(x,y) = \frac{\sum_{i=1}^{l}(x_i-\overline{x})(y_i-\overline{y})}{\sqrt{\sum_{i=1}^{l}(x_i-\overline{x})^2}\sqrt{\sum_{i=1}^{l}(y_i-\overline{y})^2}},$$

where $\overline{x}$ and $\overline{y}$ are the mean values of x and y, respectively. Correlation coefficient quantitatively measures the degree of correlation between two time series in terms of how closely they are related to each other.

When dealing with streaming time series s^1 and s^2, we only look at their sub-series within a sliding window of size w. That is, for each time t ($t \geq w$), we define the function values of s^1 and s^2 *at time* t by using $x = s^1[t-w+1,\ldots,t]$ and $y = s^2[t-w+1,\ldots,t]$. Correspondingly, we use $corr(s^1, s^2, w)$ to denote the function with sliding window size of w on streaming time series s^1 and s^2. The function yields a value at each given time t ($t \geq w$). Using this function, we can naturally form conditions on streaming time series, which are commonly used to monitor interesting trend of streaming time series [25].

- $|corr(s^1, s^2, w)|>0.75$: the (absolute) correlation between two streaming time series is high;
- $|corr(s^1, s^2, w_1)| > 2|corr(s^1, s^3, w_2)|$: the (absolute) correlation between s^1 and s^2 is two times greater than that between s^1 and s^3.

Note that the above conditions are quite trivial in terms of computation, but will be used in this paper for illustrative purposes. In real application, conditions can be quite complex, and can sometime involve accesses to large databases. For example, a condition may specify that one of the near neighbors (among a large collection of time series objects) of the streaming time series shows a particular property.

Quality measures

We denote a set of conditions as $C = \{c_1, c_2, \ldots, c_n\}$ and the reported evaluation result of condition c_i at time position t as $r(c_i, t)$. We denote the precise evaluation result (the actual value of the given condition when a precise evaluation process is used) of c_i at time position t as $R(c_i, t)$. Obviously, we have $r(c_i, t) \in \{\mathsf{True}, \mathsf{False}\}$, and $R(c_i, t) \in \{\mathsf{True}, \mathsf{False}\}$.[‡] Note that $r(c_i, t)$ may not be equal to $R(c_i, t)$ due to the approximate nature of the system. Let C_T denote all the conditions in C whose reported results are True at time position t, i.e., $C_T = \{c_i | c_i \in C \text{ and } r(c_i, t) = \mathsf{True}\}$. Similarly, let $C_F = \{c_i | c_i \in C \text{ and } r(c_i, t) = \mathsf{False}\}$. We call C_T and C_F the reported-True and reported-False sets, respectively. The two sets are disjoint and $C = C_T \bigcup C_F$.

Using the above notation, we define the following three parameters to measure the quality of an evaluation system at each time position t, with a smaller value meaning better quality.

1. *Response Time, RT,* is the duration from the data arrival time t to the time when last condition c_i, $r(c_i, t) = \mathsf{True}$, is reported.
2. *False Positive Ratio, FPR, of a reported-*True *set* C_T is the fraction of the conditions (among all the conditions in C_T) whose actual values are False. We define $FPR = 0$ if C_T is an empty set.

[‡]In the rest of the paper, we may omit t and use $r(c_i)$ ($R(c_i)$) to denote the reported evaluation result (actual value) of condition c_i at time position t when the context is clear.

3. *False Negative Ratio, FNR, of a reported-*True *set* C_F is the fraction of the conditions (among all conditions in C_F) whose actual values are True. We define $FNR = 0$ if C_F is an empty set.

Note that response time is defined only on conditions that are reported true. This is because the particular situation we are dealing with in which the situation manager only needs to enact actions when corresponding conditions become true, and does nothing in other cases. Other situations may call for different definitions, such as averaged elapsed time over all reported true conditions, etc.

4 Prediction Model

While it is easy and straightforward to measure the quality parameter RT, it is difficult to measure the other two quality parameters, FPR and FNR. For example, according to the definition of FPR, we need to know the actual values of all conditions in a reported-True set C_T when to calculate FPR. Of course, this is an unrealistic requirement since the underlying assumption is that we do not know the actual values of all the conditions. A practical alternative is to build a prediction model using historical evaluation results and calculate the *expected FPR* and FNR based on the model in a probabilistic manner.

Probability distribution of an evaluation result

For each condition c_i, we define a random variable X_i to state the outcome of its evaluation. Clearly, X_i follows Bernoulli distribution $X_i \sim B(\rho_i)$, that is,

$$X_i = \begin{cases} 1 & if\ R(c_i) = \mathsf{True} \\ 0 & if\ R(c_i) = \mathsf{False} \end{cases} \quad with \begin{cases} P(X_i = 1) = \rho_i \\ P(X_i = 0) = 1 - \rho_i \end{cases}$$

where the mean ρ_i is also called the expected value of X_i. Note in this paper, we make the simplifying assumption that all X_i's are mutually independent.

Depending on how the system treats c_i, we see three different cases for ρ_i: (1) c_i is precisely evaluated. In this case, we have $\rho_i = 1$ if c_i is evaluated to be True and $\rho_i = 0$ if c_i is evaluated to be False. (2) c_i's result is reported based on an approximation procedure, e.g., prediction. In this case, ρ_i cannot be known exactly. Instead, its estimate $\hat{\rho}_i$ will be used. This $\hat{\rho}_i$ is a random variable following certain distribution, due to the fact that the approximation procedure gives the estimate with some confidence. (3) Other cases where ρ_i is unknown. We now discuss how to model the distribution of ρ_i for the latter two cases.

In Case (2), assume c_i is estimated by an approximation procedure that is based on N historical evaluation results (samples). Then the estimate of ρ_i, $\hat{\rho}_i$, can be approximated by a normal distribution function $Norm(\mu_i, \sigma_i^2)$, where the mean value $\mu_i = \bar{X}_i$ and the variance $\sigma_i^2 = \frac{(1-\mu_i)\mu_i}{N}$, i.e., $\hat{\rho}_i \sim Norm(\mu_i, \frac{(1-\mu_i)\mu_i}{N})$. (Here, $\bar{X}_i$ denotes the sample mean.)

For Case (3), since ρ_i is unknown, its estimate $\hat{\rho}_i$ is used again. However, different from Case (2), the distribution of $\hat{\rho}_i$ is modeled as $Norm(0.5, 0.25)$. Mean value 0.5 implies that the chances of c_i being true or false are the same. The variance value 0.25 is the maximum variance that an estimate of ρ can have.

For all the three cases, the estimate of ρ_i can be viewed as a random variable that follows normal distribution. (Case (1) is a special case of normal distribution.)

Example prediction model

To get $\hat{\rho}_i$ when condition c_i is predicted, we propose to build a prediction model by analyzing the historical evaluation results for each condition. We adopt the data mining approach in [14] and histogram techniques in [20] to build the prediction model. Without loss of generality, we assume each atomic condition is in the form of $f() > \gamma$, where f is an arithmetic expression with the two basic functions and γ is a constant. We partition the range of f (values of f) into buckets. Each bucket is denoted by its boundaries $(v_1, v_2]$.

We build an *i-step look-ahead* prediction model based on historical data, which are the precise evaluation results in a long run. At each time position, function f is evaluated precisely. For a function f, a bucket $(v_1, v_2]$, and a fixed look-ahead step i, we obtain a count as follows: find all the time positions in the long run when the value of f falls into the bucket $(v_1, v_2]$. Randomly pick N of these positions, say, $t_1, \ldots, t_N$. Then check the condition $f() > \gamma$ at time positions $t_j + i$ for $j = 1, \ldots, N$, and record the percentage of times the condition is true. Hence, we can construct a histogram for each f and i: the buckets correspond to a partition of the range of f, and each bucket is associated with a percentage value which is obtained in the way discussed above. This histogram can be refined via histogram building techniques (e.g., splitting and merging) [2].

The histogram can be represented by a curve (interpolated) instead of bars for clarity: the x-axis represents the buckets (or the range of f), and the y-axis represents the percentages (which we call μ). Fig. 3 shows six example histograms for the condition $|corr(s^1, s^2, 50)| > 0.75$. (In these examples, the history has 20,000 time positions and we take 50 samples for each bucket in the histogram.) As an example, assume at time t, we obtain $|corr(s^1, s^2, 50)| = 0.85$ through a precise evaluation procedure. If we choose to look ahead 5 steps, we look up the corresponding curve in Fig. 3 and obtain $\mu = 0.8$ for $f = 0.85$. This means if the correlation value is 0.85 (or -0.85) at time t, we can predict that this condition (absolute correlation value is greater than 0.75) is true at time $t + 5$ with a probability of 0.8.

The above only gives an example on how a prediction model might be obtained. Different kind of con-

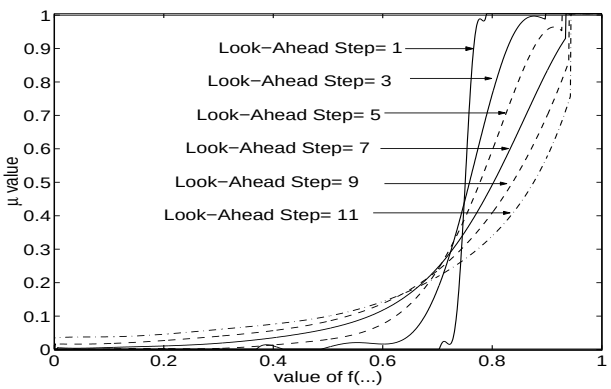


Figure 3: A sample prediction model.

ditions call for different models, and different ways to build the models.

In general, a prediction model for each condition is built based on some feature values (like in the above example). We assume that when a condition is precisely evaluated, the corresponding feature values (used for prediction) are extracted. When the prediction of a condition is required, we will look back in time to find the nearest time position when the condition was precisely evaluated. We use then-extracted feature values and the prediction model to predict the probability for the condition to be true.

FPR and *FNR* constraints

With the prediction model, given a reported-True set of size m, we can derive its expected false positive ratio, denoted $E(FPR)$. Indeed, we can prove

$$E(FPR) \sim Norm(\mu, \sigma^2)$$

where $\mu = \sum_{i=1}^{m}(1-\mu_i)/m$ and $\sigma^2 = \sum_{i=1}^{m}\sigma_i^2/m^2$. Here μ_i and σ_i take values from the corresponding cases.

Similarly, we can prove that the expected *FNR* of a reported-False set of size m is also a normally distributed random variable with $\mu = \sum_{i=1}^{m}\mu_i/m$ and $\sigma^2 = \sum_{i=1}^{m}\sigma_i^2/m^2$.

We are now ready to define false positive ratio (*FPR*) constraint and false negative ratio (*FNR*) constraint by using the expected *FPR* and *FNR*.

Definition An *FPR*-constraint is in the form of a pair $\tau_{FPR} = (\theta_E, \alpha)$ $(0 \le \theta_E, \alpha \le 1)$. A set of reported-True conditions C_T satisfies τ_{FPR} if $P\{E(FPR) \le \theta_E\} \ge \alpha$. We call θ_E and α the *expected-mean threshold* and the *confidence threshold*, respectively.

Intuitively, the smaller the θ_E value and the greater the α value, the "tighter" an *FPR*-constraint τ_{FPR} is. Formally, we define a partial order $\tau'_{FPR} \le \tau''_{FPR}$ if $\theta'_E \le \theta''_E$ and $\alpha' \ge \alpha''$, and we say τ'_{FPR} is a tighter *FPR*-constraint than τ''_{FPR}. Among all such *FPR*-constraints that a reported-True set C_T satisfies, we call the "tightest" one as the *FPR-quality* of set C_T. For simplicity, we fix the value of α in all constraints, and thus, the one having the smallest θ_E will always give the highest *FPR*-quality.

Symmetrically, we can define *FNR*-constraint τ_{FNR} and *FNR*-quality of a reported-false set C_F.

5 Evaluation algorithm

As mentioned earlier, in our speculative situation manager, an important decision is how precise (in terms of false-positive/negative ratios) the underlying condition evaluation system is. This decision is based on how much saving the approximated algorithm can save (in terms of response time). A basic problem for the underlying condition evaluation system is the following optimization problem: use the minimum amount of time to achieve the required precision.

More precisely, given accuracy constraints for both C_T and C_F (i.e., τ_{FPR} and τ_{FNR}), we need to minimize the response time. We use a greedy algorithm for this optimization problem, as shown in Fig. 4. The basic idea is to increase the size of C_T and C_F aggressively, and at the same time, try to report as early as possible those conditions in C_T.

In the algorithm, we assume that if a condition is reported true, then its μ value must be greater than threshold 0.5. That is, we don't want to risk it if the chance of condition to be true is small. Likewise, if a condition is reported to be false, its μ value must be less than (or equal to) 0.5. Different system may require other threshold values other than 0.5.

The algorithm starts with using `InitExpandC`$_T$ to get initial report-True set C_T. The procedure `InitExpandC`$_T$ is to take all the conditions with the highest μ values (no less than 0.5) as long as the *FPR*-constraint is satisfied. The conditions in C_T are reported to be True. These are the conditions that can be reported True without doing any precise evaluation. This is Step 1.

After Step 1, we need to precisely evaluate conditions in order to report them true (without violating either τ_{FPR} or $\mu > 0.5$). In Step 2, as a greedy algorithm, we pick up the condition having the next highest μ value. This is the one immediately after the conditions in the initial C_T in the μ**List**. Hence, we pick it (i.e., c_{i_T}) up to precisely evaluate. If the condition is evaluated True, we add it to C_T and try to expand C_T with no precise evaluation again (by calling a Procedure `ExpandC`$_T$, which basically tries to grab the conditions with the next highest μ values as long as the *FPR*-constraint is satisfied), report the conditions in the expended C_T and keep going. If the condition is evaluated False, then we just keep going to precisely evaluate the next condition, since we are still not able to expand C_T without a condition evaluated true.

Consts.:	τ_{FPR} and τ_{FNR} constraints
Goal:	minimize response time (RT)
Step 1.	Form and report the reported-True set: Initialize $z_T = 0$ and $z_F = 0$; $[z_T, i_T] = \texttt{InitExpandC}_T(z_T)$; Report $c_1, \ldots, c_{i_T-1}$ as True;
Step 2.	Process all conditions c_i with $(\mu_i > 0.5)$: Do loop until $(\mu_{i_T} \leq 0.5)$ or $(i_T > n)$ { - Precisely evaluate c_{i_T}. Two outcomes: - If c_{i_T} is evaluated False, update z_F with an extra reported-False condition, continue the loop with $i_T = i_T + 1$; - If c_{i_T} is evaluated True, then - Update z_T w/ an extra rpt.-True cond.; - $[z_T, \Delta_T] = \texttt{ExpandC}_T(z_T, i_T + 1)$ - Report $c_{i_T}, \ldots, c_{i_T+\Delta_T}$ as True; - Update $i_T = i_T + \Delta_T + 1$; }
Step 3.	Form the reported-False set: $[z_F, i_F] = \texttt{InitExpandC}_F(z_F)$
Step 4.	Process all conditions c_{i_T} with $(\mu_{i_T} \leq 0.5)$: Continue to use the same i_T from Step 2, do loop until $(i_T > i_F)$. { - Precisely evaluate c_{i_T}. Two outcomes: - If c_{i_T} is evaluated True, report c_{i_T} as True and continue the loop with $i_T = i_T + 1$; - If c_{i_T} is evaluated False, then o update z_F w/ an extra rpt.-False cond.; o $[z_F, \Delta_F] = \texttt{ExpandC}_F(z_F, i_F)$; - update $i_F = i_F - \Delta_F$ and $i_T = i_T + 1$; }
Step 5.	Report as False all those conditions that were not reported True.

Figure 4: Algorithm `QualEval`.

During Step 2, if we run out of conditions in μ**List**, we can stop (just report all the conditions that were evaluated False as false and thus achieve $FNR = 0$). If the μ**List** is not exhausted, then we need to reach the first condition in the μ**List** such that its μ value is no greater than 0.5.

Once we only have conditions with μ no greater than 0.5, we need to precisely evaluate them and report them as soon as they are evaluated True. However, there is a chance we may be able to report them False. Therefore, Step 3 tries to get the maximum set of conditions to report False without precise evaluation (note that all the conditions that were evaluated False need to be taken into account, hence the z_F value may not start with 0 in Step 3). The procedures $\texttt{InitExpandC}_F$ and $\texttt{ExpandC}_F$ are analogous to $\texttt{InitExpandC}_T$ and $\texttt{ExpandC}_T$, respectively.

After Step 3, if we still have conditions that need to be processed (i.e., if $i_T \leq i_F$), we will pick them up to evaluate. Since we want to minimize the response time for the conditions in C_T, we precisely evaluate the conditions starting from those with greater μ values. Again, if any condition is evaluated False, we will try to expand C_F.

It is easily seen that the algorithm is correct since both FPR and FNR-constraints are both satisfied.

6 Experimental Results

In this section, we present our experimental results, showing that the algorithm effectively achieves its optimization goal and satisfies the quality requirements.

We first describe the data set, condition set, and performance parameters we used in our experiments.

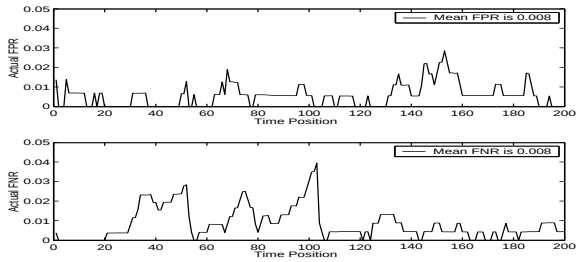
Data set: We generate synthetic data for experiments. The data set consists of 100 streaming time series. Each time series is independently generated with a random walk function. For stream s, $s_i = s_{i-1} + rand$, where $rand$ is a random variable uniformly distributed in the range of $[-0.5, 0.5]$.

Condition set: Our condition set includes 400 conditions defined over these 100 streams. Each condition may contain one or more correlation functions. Each correlation function is defined on two streams that are picked up randomly from the 100 streams, with its sliding window size being randomly chosen from $[50, 1000]$.

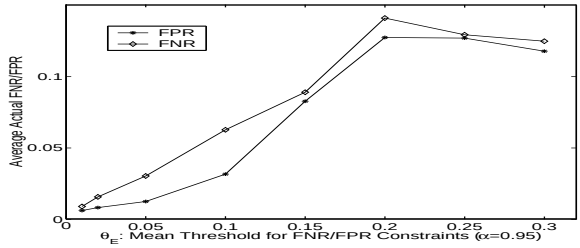
Performance parameters: We use the four quality parameters described in Section 3 (i.e., RT, FPR and FNR) to measure the performance of our algorithms. Note that FPR and FNR are all real numbers in $[0, 1]$ and can be computed precisely by comparing the reported evaluation results with the precise evaluation results (done for the purpose of performance evaluation). The response time is measured by the number of conditions that are precisely evaluated (either to True or False) before all the conditions in the reported-True set are reported. By using this measure (instead of using real time), we can clearly separate the overhead of the optimization procedure and the condition evaluation time.

We now show the performance of `QualEval`. In our experiment, we set the confidence threshold $\alpha = 95\%$ for both FPR- and FNR-constraints. We vary the expected-mean threshold θ_E from 0.05 to 0.3 in different runs, and execute the algorithm for 1,000 time positions in each run.

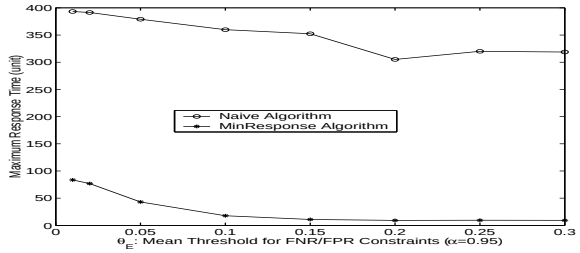
Fig. 5(a) and (b) show the evaluation quality achieved in terms of actual FPR and FNR. The two plots of Fig. 5(a) present the actual FPR and FNR values at each time position for 200 time positions with $\theta_E = 0.01$ (for both τ_{FPR} and τ_{FNR}). We can see that these actual FPR (FNR) values are in the range $[0.01, 0.04]$ with a mean of 0.008 (which is very close to the given $\theta_E = 0.01$). Fig. 5(b) presents how well FPR (FNR) constraints with various mean thresholds (varying from 0.01 to 0.3) are satisfied by our algorithm. We calculate the average of the actual FPR (FNR) values over 1000 time positions for each run, and we can see



(a) Actual *FPR* and *FNR* for one run with $\theta_E = 0.01$



(b) Average actual *FPR* and *FNR* for various runs



(c) Response time comparison (with naive algorithm)

Figure 5: Performance of `QualEval`.

that the average is either below or very close to the corresponding given expected-mean threshold θ_E for all the runs. Note that when the given θ_E becomes large enough (i.e., greater than 0.2 in Fig. 5(b)), the actual *FPR* (*FNR*) values tend to become some constants that are determined by the prediction model. The little drop of the curves in this figure is due to experimental variance.

Fig. 5(c) shows the performance of `QualEval` in terms of response time. For comparison, a naive algorithm is implemented: It randomly picks up a condition to evaluate precisely, until it has reported k Trues, where k is the number of real Trues in the reported-True set from `QualEval` (i.e., k is the number of c_is such that $R(c_i) =$ True and $c_i \in C_T$). This is to make the naive algorithm report the same number of true conditions. We compare the response time of `QualEval` with this naive algorithm for different runs with θ_E values in $[0.01, 0.3]$. We can see that `QualEval` consistently outperforms the naive algorithm. Note that the response time of `QualEval` decreases as θ_E increases, because the greater the θ_E value, the coarser approximation is allowed, and thus fewer precise evaluations are needed.

The performance gain of `QualEval` is significant. For example, given $\theta_E = 0.01$, `QualEval` only takes about 1/15 time of the naive algorithm, but maintains the quality of *FPR* and *FNR* at around 1%. When θ_E is set to higher values, the performance gain becomes more significant. Note that the confidence threshold α is set to 95% in all the experiment reported here. Given the same expected-mean threshold θ_E for both *FPR*- and *FNR*-constraints, smaller α will yield faster response. We omit the experimental results on using various α settings in this paper.

7 Related Work

Speculation has been used in various applications such as processor design, cache implementation, and buffer management in OS. Recently, Polyzotis and Ioannidis [22] described a speculative query processing system, where the system anticipates the user queries by predicting from partially entered query statements. Our design follows the same approach.

The quality-driven aspect of our work is similar to the QoS concept in computer network [12, 19]. QoS in computer network allows the end users to specify their requirements on different quality metrics (e.g., service availability, delay, delay variation, throughput, and packet loss rate). The network system needs to guarantee certain levels of service quality based on these requirements. This paper adapts the QoS concept into condition evaluation on streaming time series and presents a basic design strategy for developing such a quality-driven system.

Aurora [24, 1, 8] seems to be the only data stream processing system that contains a QoS component. In Aurora, a user may register an application with a QoS specification that provides the user's preference on the performance and quality of this application. These QoS specifications serve to drive policies for scheduling and load shedding.

With limited resources, using approximation techniques in processing continuous queries on data streams has been studied in [11, 13, 10, 15, 25]. However, most approximate evaluation strategies, and even some precise evaluation strategies, only consider one quality aspect and neglect the others. For example, Chain [6] minimizes the memory usage without considering the response time at all. Our work differs from all such work in that our strategy takes different user-specified quality requirements into consideration to guide the evaluation procedure. In other words, the satisfaction of quality requirements in our system is provided by run-time dynamic adjustments instead of by static algorithm design. This advantage allows our system to handle different constraints.

In Statstream system [25], Zhu and Shasha presented an efficient approximation method to calculate all pair-wise correlations for a set of streaming time series. In this paper, we use correlation function to

construct conditions. While [25] aims at building an approximate evaluation system that can process correlations efficiently, we focus on satisfying the user's quality requirements.

8 Conclusion

In this paper, we studied a condition prediction subsystem that considers user-specified quality requirements. We argued that such a system is necessary for speculative systems. We used statistical analysis to derive the likelihood of a condition to be true at a time position. By using this likelihood and the associated confidence (due to finite sampling), we estimated the quality of our predictions. Based on this prediction method, we designed an algorithm to produce predictions satisfying precision requirements. Our experiments showed that the algorithm is effective.

The prediction subsystem presented in this paper works well with simple correlation conditions and synthetic data sets. To make this subsystem applicable to real applications, it would be interesting to extend the prediction subsystem to handle more general conditions such as those in AMIT, or those involving complex operators such as nearest neighbor searches. As future work, it is also interesting to study how to improve the subsystem when some assumptions made in this paper do not hold anymore (e.g., streaming time series come from some common sources and are not independent). Another interesting research direction is to design algorithms that satisfy other types of quality requirements. For example, a user may want to impose a response time limit while requiring the system to achieve the best precision.

References

[1] D. J. Abadi et. al. Aurora: A data stream management system. In *SIGMOD*, 2003.

[2] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In *SIGMOD*, pages 181–192, 1999.

[3] A. Adi, D. Botzer, and O. Etzion. The Situation Manager Component of Amit - Active Middleware Technology. In the 5^{th} International Workshop on Next Generation Information Technologies and Systems (NGITS), pages. 158-168, 2002.

[4] D. Anderson et. al. Detecting unusual program behavior using the statistical component of the next-generation intrusion detection expert system (NIDES). Tech. Report SRI-CSL-95-06, SRI, 1995.

[5] S. Arya, D. M.Mount, N. S. Netanyahu, R. Silverman and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. In *Journal of the ACM*, 45(6), pages 891–923, 1998.

[6] B. Babcock, S. Babu, M. Datar, and R. Motwani. Chain: Operator scheduling for memory minimization in data stream systems. In *SIGMOD*, pages 253–264, 2003.

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

[8] D. Carney et. al. Monitoring streams - a new class of data management applications. In *Very Large Data Bases*, 2002.

[9] CISCO. Quality of Service (QoS). On-line. http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/qos.htm, 2003.

[10] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *SIGMOD*, pages 40–51, 2003.

[11] A. Dobra, M. N. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *SIGMOD*, pages 61–72, 2002.

[12] P. Ferguson and G. Huston. *Quality of Service: Delivering QoS on the Internet and in Corporate Networks*. John Wiley & Sons, 1998.

[13] S. Ganguly, M. N. Garofalakis, and R. Rastogi. Processing set expressions over continuous update streams. In *SIGMOD*, pages 265–276, 2003.

[14] L. Gao, M. Wang, X. S. Wang, and S. Padmanabhan. A learning-based approach to estimate statistics of operators in continuous queries: a case study. In *DMKD*, June 2003.

[15] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB*, pages 79–88, 2001.

[16] J. Kang, J. F. Naughton, and S. D. Viglas. Evaluating window joins over unbounded streams. In *ICDE*, 2003.

[17] L. Lim, M. Wang, and J. S. Vitter. SASH: A self-adaptive histogram set for dynamically changing workloads. In *VLDB*, 2003.

[18] Y. Matias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *VLDB*, pages 101–110, 2000.

[19] D. McDysan. *QoS and Traffic Management in IP and ATM Networks*. McGraw-Hill Osborne Media, 1999.

[20] V. Poosala. *Histogram-based Estimation Techniques in Databases*. PhD thesis, University of Wisconsin, 1997.

[21] P. A. Porras and P. D. Neumann. EMERALD: Event monitoring enabling response to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, 1997.

[22] N. Polyzotis, and Y. Ioannidis. Speculative Query Processing. First Biennial Conference on Innovative Data Systems Research (CIDR), 2003.

[23] S. Ross. *A First Course in Probability*. Prentice Hall, 2001.

[24] N. Tatbul, U. etintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *VLDB*, pages 309–320, 2003.

[25] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, pages 358–369, 2002.

Continuous Query Processing of Spatio-temporal Data Streams in PLACE

Mohamed F. Mokbel Xiaopeng Xiong Moustafa A. Hammad Walid G. Aref*

Department of Computer Sciences, Purdue University, West Lafayette, IN 47907-1398
{mokbel,xxiong,mhammad,aref}@cs.purdue.edu

Abstract

The tremendous increase of cellular phones, GPS-like devices, and RFIDs results in highly dynamic environments where objects as well as queries are continuously moving. In this paper, we present a continuous query processor designed specifically for highly dynamic environments (e.g., location-aware environments). We implemented the proposed continuous query processor inside the PLACE server (Pervasive Location-Aware Computing Environments); a scalable location-aware database server currently developed at Purdue University. The PLACE server extends data streaming management systems to support location-aware environments. Such environments are characterized by the wide variety of continuous spatio-temporal queries and the unbounded spatio-temporal streams. The proposed continuous query processor mainly includes: (1) Developing new *incremental* spatio-temporal operators to support a wide variety of continuous spatio-temporal queries, (2) Extending the semantic of sliding window queries to deal with spatial sliding windows as well as temporal sliding windows, and (3) Providing a shared execution framework for scalable execution of a set of concurrent continuous spatio-temporal queries. Preliminary experimental evaluation shows the promising performance of the continuous query processor of the PLACE server.

*This work was supported in part by the National Science Foundation under Grants IIS-0093116, EIA-9972883, IIS-9974255, IIS-0209120, 0010044-CCR, and EIA-9983249.

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

1 Introduction

The wide spread of cellular phones, handheld devices, and GPS-like technology enables environments where virtually all objects are aware of their locations. Such environments call for new query processing techniques to efficiently support location-aware servers. Unlike traditional database servers, location-aware servers have the following distinguished characteristics: (1) Data are received from moving and stationary objects in the form of unbounded spatio-temporal streams, (2) Large number of continuous stationary and moving spatio-temporal queries, and (3) Any delay of the query response results in an obsolete answer. Consider a query that asks about the moving objects that lie in a certain region. If the query answer is delayed, the answer may be outdated where objects are continuously changing their locations.

Existing techniques for handling continuous spatio-temporal queries in location-aware environments (e.g., see [3, 16, 18, 31, 34, 36, 39, 40]) focus on developing specific high level algorithms that utilize traditional database servers. In this paper, we go beyond the idea of high level algorithms, instead, we present a continuous query processor that aims to modify the database engine to support a wide variety of continuous spatio-temporal queries. Our continuous query processor is implemented inside the PLACE (Pervasive Location-Aware Computing Environments) server; currently developed at Purdue University [2, 24]. The PLACE server extends both the PREDATOR relational database management system [30] and the NILE streaming database management system [15] to support efficient continuous query processing of spatio-temporal streams. In particular, the continuous query processor of the PLACE server has the following distinguishing characteristics:

1. **Incremental evaluation.** The PLACE continuous query processor employs an *incremental* evaluation paradigm by continuously updating the query answer. We distinguish between two types of updates; namely *positive* and *negative* updates [23]. A positive/negative update

indicates that a certain object needs to be added to/removed from the query answer.

2. **Spatio-temporal operators.** The PLACE continuous query processor employs a new set of spatio-temporal incremental operators (e.g., INSIDE and kNN operators) that support incremental evaluation of a wide variety of continuous spatio-temporal queries.

3. **Predicate-based Sliding Windows**: We extend the notion of sliding windows beyond time-based and tuple-count windows to accommodate for predicate-based windows (e.g., an object expires from the window when it appears again in the stream).

4. **Scalability.** We use a *shared execution* paradigm as a means of achieving scalability in terms of the number of outstanding continuous spatio-temporal queries.

The rest of the paper is organized as follows: Section 2 highlights the challenges we faced in building the continuous query processor of the PLACE server along with the related work of each challenge. In Section 3, we present an overview of the data model and SQL language used by the PLACE server. Section 4 presents different methods of expiring incoming tuples in the PLACE server. The incremental processing of continuous queries is discussed in Section 5. Section 6 discusses the shared execution of concurrent continuous queries. The graphical user interface (GUI) of the PLACE server is presented in Section 7. Section 8 introduces preliminary experimental results from the PLACE server. Finally, Section 9 concludes the paper.

2 Challenges and Related Work

In this section, we go through some of the challenges we faced while building the continuous query processor of the PLACE location-aware server. With each challenge, we present its related work.

2.1 Challenge I: Incremental Evaluation of Continuous Queries

Most of spatio-temporal queries are continuous in nature. Unlike snapshot queries that are evaluated only once, continuous queries require continuous evaluation as the query result becomes invalid with the change of information [37]. A naive way to handle continuous queries is to abstract the continuous query into a series of snapshot queries executed at regular interval times. Existing algorithms for continuous spatio-temporal queries aim to optimize the time interval between each two instances of the snapshot queries. Mainly, three different approaches are investigated: (1) The validity of the results [39, 40]. With each query answer, the server returns a *valid time* [40] or a *valid region* [39] of the answer. Once the valid time is expired or the client goes out of the valid region, the client resubmits the continuous query for reevaluation. (2) Caching the results. The main idea is to cache the previous result either in the client side [31] or in the server side [18]. Previously cached results are used to prune the search for the new results of k-nearest-neighbor queries [31] and range queries [18]. (3) Precomputing the result [18, 34]. If the trajectory of query movement is known apriori, then by using computational geometry for stationary objects [34] or velocity information for moving objects [18], we can identify which objects will be nearest-neighbors [34] to or within a range [18] from the query trajectory. If the trajectory information changes, then the query needs to be reevaluated.

With the large number of continuous queries, reevaluating a continuous spatio-temporal query, even with large time intervals, poses a redundant processing for the location-aware servers. In the PLACE continuous query processor, we go beyond the idea of reevaluating continues spatio-temporal queries. Instead, we provide an incremental evaluation paradigm, where only the updates of the result are reported to the user.

2.2 Challenge II: Wide Variety of Continuous Queries

Most of the existing query processing techniques focus on solving special cases of continuous spatio-temporal queries, e.g., [31, 34, 39, 40] are valid only for *moving queries on stationary objects*, [5, 9, 11, 25] are valid only for *stationary range queries*. Other work focus on aggregate queries [11, 32, 33] or k-NN queries [16, 31]. Trying to support such wide variety of continuous spatio-temporal queries in a location-aware server results in implementing a variety of specific algorithms with different access structures.

In the PLACE continuous query processor, we avoid using tailored algorithms for each kind of continuous spatio-temporal queries. Instead, we furnish the PLACE server with a set of primitive pipelined query operators that can support a wide spectrum of continuous spatio-temporal queries.

2.3 Challenge III: Large Number of Concurrent Continuous Queries

Most of the existing spatio-temporal algorithms focus on evaluating only one spatio-temporal query (e.g., [3, 16, 18, 31, 34, 36, 39, 40]). In a typical location-aware server [2, 21, 24], there is a huge number of concurrently outstanding continuous spatio-temporal queries. Handling each query as an individual entity dramatically degrades the performance of the location-aware server.

Although there is a lot of research in sharing the execution of continuous web queries (e.g., see [8]) and

continuous streaming queries (e.g., see [6, 7, 14]), optimization techniques for evaluating a set of continuous spatio-temporal queries are recently addressed for centralized [25] and distributed environments [5, 9]. The main idea of [5, 9] is to ship part of the query processing down to the moving objects, while the server mainly acts as a mediator among moving objects. In centralized environments, the Q-index [25] is presented as an R-tree-like [10] index structure to index the queries instead of objects. However, the Q-index is limited in two aspects: (1) It performs reevaluation of all the queries (through the R-tree index) every T time units. (2) It is applicable only for stationary queries. Moving queries would spoil the Q-index and hence dramatically degrade its performance.

2.4 Challenge IV: Indexing Moving Objects/Queries

Most of the existing spatio-temporal index structures [22] aim to modify the traditional R-tree [10] to support the highly dynamic environments of location-aware servers. In particular, two main approaches are investigated: (1) Indexing the future trajectories such that the existing tree would last longer before an update is needed. Examples of this category are the TPR-tree [29], R^{EXP}-tree [28], and the TPR*-tree [35]). (2) Modifying the deletion and insertion algorithms for the original R-tree to support frequent updates. Examples of this category include the Lazy-update R-tree [17] and the Frequently-updated R-tree [19]

Even with the proposed modifications of the R-tree structures, highly dynamic environments degrades the performance of the R-tree and results in a bad performance. In the PLACE continuous query processor, we avoid using R-tree-like structure. Instead, we use a grid-like index structure [23] that is simple to update and retrieve. Moreover, fixed grids are space-dependent, thus there is no need to continuously change the index structure with the continuous insertion and deletion.

3 The PLACE Server

In this section, we present the data modelling and SQL language used by the PLACE server.

3.1 Data Model

By subscribing with the PLACE server, moving objects are required to send their location updates periodically to the PLACE server. A location update from the client (moving object) to the server has the format (OID, x, y), where OID is the object identifier, (x, y) is the location of the moving object in the two-dimensional space. An update is timestamped upon its arrival at the server side. Once an object stops moving (e.g., an object reaches to its destination or the cellular phone is shut down) it sends to the server a *disappear* message which indicates that the object is no further moving.

Due to the highly dynamic nature of location-aware environments and the infinite size of incoming spatio-temporal streams, we cannot store all incoming data. Thus, the PLACE server employs a three-level storage hierarchy. First, a subset of the incoming data streams is stored in in-memory buffers. In-memory buffers are associated with the outstanding continuous queries at the server. Each query determines which tuples are needed to be in its buffer and when these tuples are expired, i.e., deleted from the buffer. Second, we keep an in-disk storage that keeps track with only one reading of each moving object and query. Since, we cannot update the disk storage every time we receive an update from moving objects, we sample the input data by chosing every kth reading to flush to disk. Moreover, we cache the readings of moving objects/queries and flush them once to the secondary storage every T time units. Data on the secondary storage are indexed using a simple grid structure [23]. Third, every $T_{archive}$ time units, we take a snapshot of the in-disk database and flush it to a repository server. The repository server acts as a multi-version structure of the moving objects that supports historical queries. Stationary objects (e.g., gas stations, hospitals, restaurants) are preloaded to the system as relational tables that are infrequently updated.

3.2 Extended SQL Syntax

As the PLACE server [24] extends both PREDATOR [30] and NILE [15], we extend the SQL language provided by both systems to support spatio-temporal operators. Mainly, we add the $INSIDE$ and kNN operators to support continuous range queries and k-nearest-neighbor queries respectively. A continuous query is registered at the PLACE server using the SQL:

```
REGISTER QUERY query_name AS
SELECT select_clause
FROM from_clause
WHERE where_clause
INSIDE inside_clause
kNN knn_clause
WINDOW window_clause
```

The `REGISTER QUERY` statement registers the continuous query at the PLACE server with the *query_name* as its identifier. The *select_clause*, *from_clause*, and *where_clause* are inherited from the PREDATOR [30] database management statement. The *window_clause* is inherited from the NILE [15] stream query processor to support continuous sliding window queries [14]. A continuous query is dropped form the system using the SQL: `DROP QUERY` *query_name*.

The *inside_clause* can represent stationary rectangular or circular range queries by specifying the two corners or the center and radius of the query region, respectively. If the first parameter to the *inside_clause* is set to M, then the query is moving and the second parameter represents the ID of the *focal* object of the query. Similarly, the *knn_clause* can represent stationary as well as moving k-nearest-neighbor queries.

4 Tuple Expiration

With the unbounded incoming spatio-temporal streams, it becomes infeasible to store all incoming tuples. However, some input tuples may be buffered in memory for a limited time. The choice of the stored tuples are mainly query dependent, i.e., we store only the tuples of interest. Since the queries are continuously changing, there should be a mechanism to expire (delete) some of the stored tuples and replace them with other tuples that becomes more relevant to the outstanding continuous spatio-temporal queries. In the PLACE continuous query processor, we employ three types of tuple expiration, namely, *temporal* expiration, *spatial* expiration, and *predicate-based* expiration.

4.1 Temporal Expiration

Most of the data stream management systems use the concept of temporal expiration as a mechanism to answer continuous sliding window queries. A sliding widow query involves a time window w. Any object that has a timestamp within the current sliding window of any outstanding query Q is in-memory buffered with the associated buffer of Q.

An example for a sliding window query submitted to the PLACE server is: Q_1: *"Continuously, report the number of cars that passed by region R in the last hour"*.

```
SELECT COUNT(ObjectID)
FROM MovingObjects
WHERE type = Car
INSIDE R
WINDOW 1 hour
```

Notice that Q_1 buffers all incoming tuples during the previous hour. A tuple is expired (i.e., deleted from the query buffer) once it goes out of the sliding time window (i.e., if it becomes more than one hour old).

4.2 Spatial Expiration

The PLACE server introduces a new type of expiration that depends on the spatial location of the moving objects instead of their timestamps. An incoming tuple o is stored in the in-memory buffer associated with a query Q only if o satisfies the spatial window (e.g., region) of Q.

An example of spatial expiration query is: Q_2: *"Continuously, report the number of cars in a certain area."*. Notice that unlike Q_1, in Q_2, we are concerned about the actual current number of cars not the number of cars in the recent history. The SQL of Q_2 is similar to that of Q_1 with only the removal of the window statement.

4.3 Predicate-based Expiration

Due to the nature of spatio-temporal streams, other forms of tuple expiration may arise. For example, consider the query Q_3: *"For each moving object, continuously report the elapsed time between each two consecutive readings"*. Such a query contains a self join where objects from the stream of moving objects are self joined based on the object identifier. The query buffer needs to maintain only the latest reading of each moving object. Once the reading of a certain object is reported, the previous reading is expired. We call such kind of expiration as *predicate-based* where it is mainly dependent on the query semantic.

5 Incremental Evaluation

To avoid reevaluating continuous spatio-temporal queries, we employ an *incremental evaluation* paradigm in the PLACE continuous query processor. The main idea is to only report the changes of the answer from the last evaluation time. By employing *incremental evaluation*, the PLACE server achieves the following goals: (1) Fast query evaluation, since we compute only the updates of the answer not the whole answer. (2) In a typical location-aware server, query results are sent to the users via satellite servers [1, 12]. Thus, limiting the amount of transmitted data to the updates only rather than the whole query answer saves in network bandwidth. (3) When encapsulating incremental algorithms into physical pipelined query operators, limiting the tuples that go through the whole query pipeline to only the updates reduces the flow in the pipeline. Thus, efficient query processing is achieved.

To realize the incremental evaluation processing in the PLACE server, we go through three main steps. First, we define the high level concept of incremental updates, by defining two types of updates; *positive* and *negative* updates [20, 23]. Second, we encapsulate the processing of incremental algorithms into pipelined query operators. Third, we modify traditional pipelined query operators (e.g., distinct and join) to deal with the concept of negative tuples [13].

5.1 Positive/Negative Updates

Incremental evaluation is achieved through updating the previous query answer. Mainly, we distinguish between two types of updates; *positive* updates and *negative* updates. A positive/negative update indicates

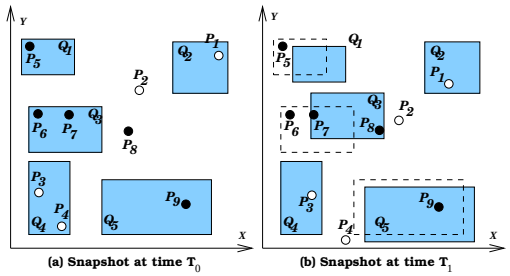


Figure 1: Incremental evaluation of range queries

that a certain object needs to be added to/removed from the query answer. A query answer is represented in the form $(QID, OList)$, where QID is the query identifier and $OList$ is the query answer. The PLACE server continuously updates the query answer with updates of the form $(QID, \pm, OID)$ where $\pm$ indicates the type of the update and OID is the object identifier.

Figure 1 gives an example of applying the concept of positive/negative updates on a set of continuous range queries. The snapshot of the database at time T_0 is given in Figure 1a with nine moving objects, p_1 to p_9, and five continuous range queries, Q_1 to Q_5. The answer of the queries at time T_0 is represented as (Q_1, P_5), (Q_2, P_1), (Q_3, P_6, P_7), (Q_4, P_3, P_4), and (Q_5, P_9). At time T_1 (Figure 1b), only the objects p_1, p_2, p_3, and p_4 and the queries Q_1, Q_3, and Q_5 change their locations. As a result, the PLACE server reports the following updates: $(Q_1, -P_5)$, $(Q_3, -P_6)$, $(Q_3, +P_8)$, and $(Q_4, -p_4)$.

5.2 Spatio-temporal Incremental Pipelined Operators

Two alternative approaches can be utilized in implementing spatio-temporal algorithms inside the PLACE server: using SQL *table functions* [26] or encapsulating the algorithms in physical query operators. Since there is no straightforward method for pushing query predicates into table functions [27], the performances is limited and the approach does not give enough flexibility in optimizing the issued queries. In the PLACE server we encapsulate our algorithms inside physical pipelined query operators that can be part of a query execution plan. By having pipelined query operators, we achieve three goals: (1) Spatio-temporal operators can be combined with other operators (e.g., distinct, aggregate, and join operators) to support incremental evaluation for a wide variety of continuous spatio-temporal queries. (2) Pushing spatio-temporal operators deep in the query execution plan reduces the number of tuples in the query pipeline. This reduction comes from the fact that spatio-temporal operators act as filters to the above operators. (3) Flexibility in the query optimizer where multiple candidate execution plans can be produced.

The main idea of spatio-temporal operators is to keep track of the recently reported answer of each query Q in a query buffer termed $Q.Answer$. Then, for each newly incoming tuple P, we perform two tests: Test I: Is P part of the previously reported $Q.Answer$? Test II: Does P qualify to be part of the current answer? Based on the results of the two tests, we distinguish among four cases:

- **Case I:** P is part of $Q.Answer$ and P still qualify to be part of the current answer. As we process only the updates of the previously reported result, P will not be processed.
- **Case II:** P is part of $Q.Answer$, however, P does not qualify to be part of the answer anymore. In this case, we report a *negative* update P^- to the above query operator. The *negative* update indicates that P is *spatially* expired from the answer.
- **Case III:** P is not part of $Q.Answer$, however, P qualifies to be part of the current answer. In this case, we report a *positive* update to the above query operator.
- **Case IV:** P is not part of $Q.Answer$ and P still does not qualify to be part of the current answer. In this case, P has no effect on Q.

5.3 Traditional Operators

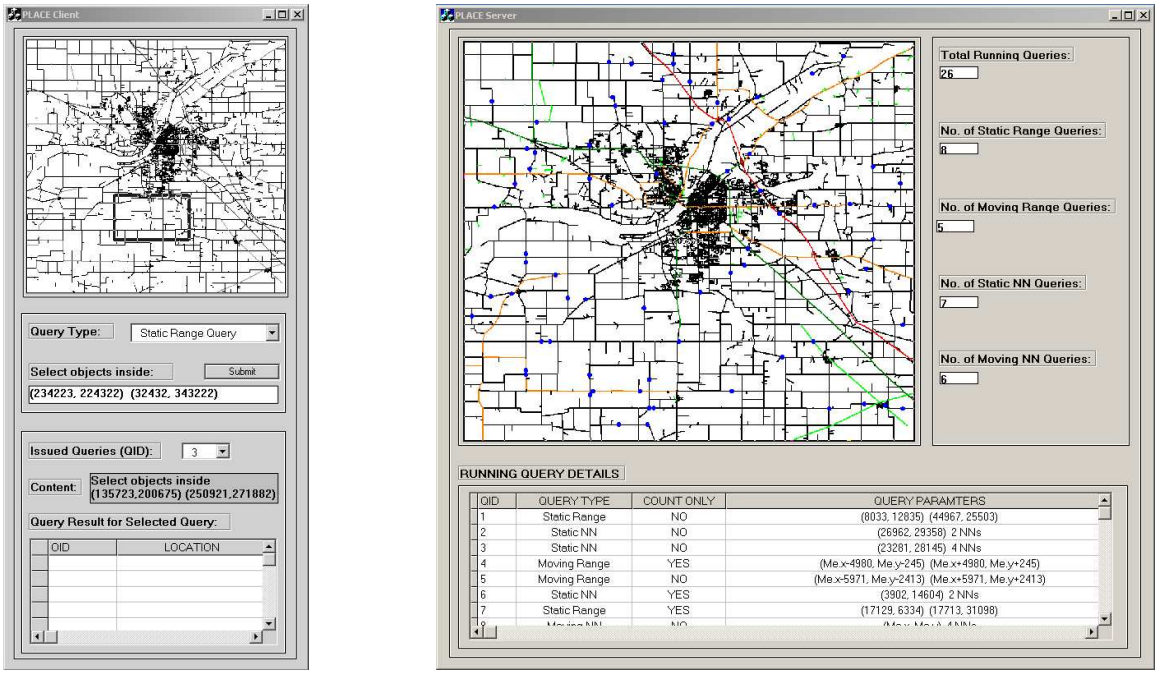
Having the spatio-temporal operators at the bottom or at the middle of the query evaluation pipeline requires that all the above operators be equipped with special handling of *negative* tuples. The NILE query processor [15] handles *negative* tuples in pipelined operators as follows:

- *Selection* and *Join* operators handle *negative* tuples in the same way as *positive* tuples. The only difference is that the output will be in the form of a *negative* tuple.
- *Aggregates* update their aggregate functions by considering the received *negative* tuple.
- The *Distinct* operator reports a *negative* tuple at the output only if the corresponding *positive* tuple is in the recently reported result.

For more details about handling the *negative* tuples in various query operators, the reader is referred to [13].

6 Scalability

The PLACE continuous query processor exploits a *shared execution* paradigm [21, 23, 38] as a means for achieving scalability in terms of the number of concurrently executing continuous spatio-temporal queries.



(a) Client GUI

(b) Server GUI

Figure 3: Snapshot of the PLACE client and server

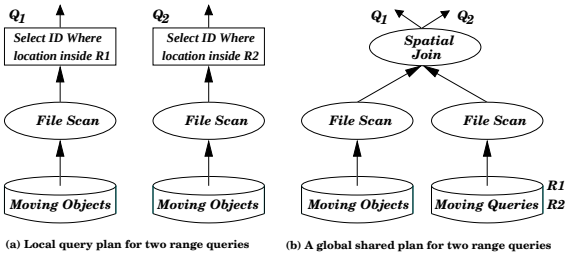


Figure 2: Shared execution of continuous queries.

The main idea is to group similar queries in a query table. Then, the evaluation of a set of continuous queries is modelled as a spatial join between moving objects and moving queries. Similar ideas of shared execution have been exploited in the NiagaraCQ [8] for web queries and PSoup [6, 7] for streaming queries.

Figure 2a gives the execution plans of two simple continuous spatio-temporal queries, Q_1: *"Find the objects inside region R_1"*, and Q_2: *"Find the objects inside region R_2"*. With *shared execution*, we have the execution plan of Figure 2b. Shared execution for a collection of spatio-temporal range queries can be expressed in the PLACE server by issuing the following continuous query:

```
SELECT Q.ID, O.ID
FROM QueryTable Q, ObjectTable O
WHERE O.location inside Q.region
```

7 User interface in PLACE

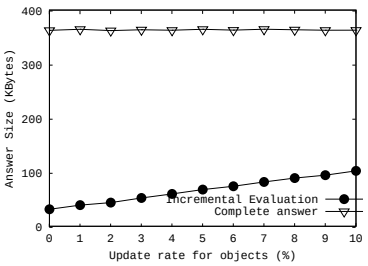
Figure 3 gives snapshots of the client and server graphical user interface (GUI) of PLACE. The client GUI simulates a client end device used by the users. Users can choose the type of query from a list of available query types. The spatial region of the query can be determined using the map of the area of interest[1] (the bold plotted rectangle on the map). By pressing the *submit* button, the client translates the query into SQL language and transmits it to the PLACE server. Once the query is submitted to the server, the result appears to the query as a list at the bottom of Figure 3a. A client can send multiple queries of different types to the PLACE server.

The PLACE server GUI is for the purpose of administration at the server side. The main idea is to keep track of the concurrently executing continuous queries from each type. All the processed queries along with their parameters are displayed in the bottom left of the screen. In addition, the server GUI contains a regional map showing the movement of objects, and the parameters of the selected queries.

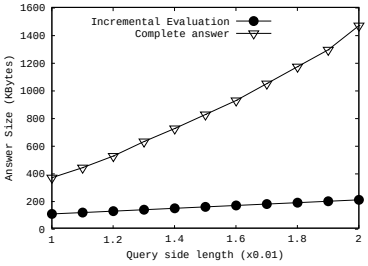
8 Performance Evaluation

In this section, we present preliminary experiments that show the promising performance of the continuous query processor in the PLACE server. We use the

[1]The map in Figure 3 is for the Greater Lafayette, IN, USA.

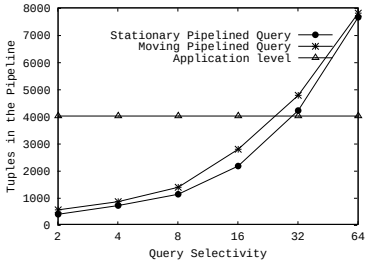


(a) Moving objects (%)



(b) Query size

Figure 4: The answer size



(a) $INSIDE$ Operator

Figure 5: Pipelined operators.

Network-based Generator of Moving Objects [4] to generate a set of 100K moving objects and 100K moving queries. The output of the generator is a set of moving objects that move on the road network of a given city. We choose some points randomly and consider them as centers of square range queries.

8.1 Size of Incremental Answer

Figure 4 compares between the size of the incremental answer returned by utilizing the incremental approach and the size of the complete answer. The location-aware server buffers the received updates from moving objects and queries and evaluates them every 5 seconds. Figure 4a gives the effect of the number of moving objects that reported a change of location within the last 5 seconds. The size of the complete answer is constant and is orders of magnitude of the size of the worst-case incremental answer. In Figure 4b, the query side length varies from 0.01 to 0.02. The size of the complete answer increases dramatically to up to seven times that of the incremental result. The saving in the answer size directly affects the communication cost from the server to the clients.

8.2 Pipelined Spatio-temporal Operators

Consider the query Q: *"Continuously report all trucks that are within MyArea"*. *MyArea* can be either a stationary or moving range query. A high level implementation of this query has only a selection operator that selects only the *"trucks"*. Then, a high level algorithm implementation would take the selection output and incrementally produce the query result. However, an encapsulation of $INSIDE$ algorithm into a physical operator allows for more flexible plans.

Figure 5 compares the high level implementation of the above query with pipelined operators for both stationary and moving queries. The selectivity of the queries varies from 2% to 64%. The selectivity of the selection operator is 5%. Our measure of comparison is the number of tuples that go through the query evaluation pipeline. When algorithms are implemented at the application level, the performance is not affected by the selectivity. However, when $INSIDE$ is pushed before the *selection*, it acts as a filter for the query evaluation pipeline, thus, limiting the tuples through the pipeline to only the incremental updates. With $INSIDE$ selectivity less than 32%, pushing $INSIDE$ before the selection greatly affects the performance.

9 Conclusion

In this paper, we present the continuous query processor of the PLACE (Pervasive Location-Aware Computing Environments) server; a database server for location-aware environments currently developed at Purdue University. The PLACE server extends both the PREDATOR database management system and the NILE stream query processor to deal with unbounded spatio-temporal streams. In addition to the temporal tuple expiration defined in sliding window queries, we maintain other forms of tuple expirations (e.g., spatial expiration). To efficiently handle large number of continuous queries, we employ an incremental evaluation paradigm that contains: (1) Defining the concept of *positive* and *negative* updates, (2) Encapsulating the algorithms for incremental processing into pipelined spatio-temporal operators, and (3) Modifying traditional query operators (e.g., distinct and join) to deal with the *negative* updates that comes from the spatio-temporal operators. Shared execution is employed by the continuous query processor as a means of achieving scalability in terms of the number of concurrently continuous queries. Preliminary experimental results show the promising performance of the PLACE continuous query processor.

References

[1] S. Acharya, M. J. Franklin, and S. B. Zdonik. Disseminating Updates on Broadcast Disks. In *VLDB*, 1996.

[2] W. G. Aref, S. E. Hambrusch, and S. Prabhakar. Pervasive Location Aware Computing Environments (PLACE). http://www.cs.purdue.edu/place/, 2003.

[3] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis. Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. In *IDEAS*, 2002.

[4] T. Brinkhoff. A Framework for Generating Network-Based Moving Objects. *GeoInformatica*, 6(2), 2002.

[5] Y. Cai, K. A. Hua, and G. Cao. Processing Range-Monitoring Queries on Heterogeneous Mobile Objects. In *Mobile Data Management, MDM*, 2004.

[6] S. Chandrasekaran and M. J. Franklin. Streaming Queries over Streaming Data. In *VLDB*, 2002.

[7] S. Chandrasekaran and M. J. Franklin. PSoup: a system for streaming queries over streaming data. *VLDB Journal*, 12(2):140–156, 2003.

[8] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *SIGMOD*, 2000.

[9] B. Gedik and L. Liu. MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System. In *EDBT*, 2004.

[10] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*, 1984.

[11] M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. J. Tsotras. On-Line Discovery of Dense Areas in Spatio-temporal Databases. In *SSTD*, 2003.

[12] S. E. Hambrusch, C.-M. Liu, W. G. Aref, and S. Prabhakar. Query Processing in Broadcasted Spatial Index Trees. In *SSTD*, 2001.

[13] M. A. Hammad, W. G. Aref, M. J. Franklin, M. F. Mokbel, and A. K. Elmagarmid. Efficient execution of sliding-window queries over data streams. Technical Report TR CSD-03-035, Purdue University Department of Computer Sciences, Dec. 2003.

[14] M. A. Hammad, M. J. Franklin, W. G. Aref, and A. K. Elmagarmid. Scheduling for shared window joins over data streams. In *VLDB*, 2003.

[15] M. A. Hammad, M. F. Mokbel, M. H. Ali, W. G. Aref, A. C. Catlin, A. K. Elmagarmid, M. Eltabakh, M. G. Elfeky, T. M. Ghanem, R. Gwadera, I. F. Ilyas, M. Marzouk, and X. Xiong. Nile: A Query Processing Engine for Data Streams (Demo). In *ICDE*, 2004.

[16] G. S. Iwerks, H. Samet, and K. Smith. Continuous K-Nearest Neighbor Queries for Continuously Moving Points with Updates. In *VLDB*, 2003.

[17] D. Kwon, S. Lee, and S. Lee. Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree. In *Mobile Data Management, MDM*, 2002.

[18] I. Lazaridis, K. Porkaew, and S. Mehrotra. Dynamic Queries over Mobile Objects. In *EDBT*, 2002.

[19] M.-L. Lee, W. Hsu, C. S. Jensen, and K. L. Teo. Supporting Frequent Updates in R-Trees: A Bottom-Up Approach. In *VLDB*, 2003.

[20] M. F. Mokbel. Continuous Query Processing in Spatio-temporal Databases. In *Proceedings of the ICDE/EDBT PhD Workshop*, 2004.

[21] M. F. Mokbel, W. G. Aref, S. E. Hambrusch, and S. Prabhakar. Towards Scalable Location-aware Services: Requirements and Research Issues. In *GIS*, 2003.

[22] M. F. Mokbel, T. M. Ghanem, and W. G. Aref. Spatio-temporal Access Methods. *IEEE Data Engineering Bulletin*, 26(2), 2003.

[23] M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In *SIGMOD*, 2004.

[24] M. F. Mokbel, X. Xiong, W. G. Aref, S. Hambrusch, S. Prabhakar, and M. Hammad. PLACE: A Query Processor for Handling Real-time Spatio-temporal Data Streams (Demo). In *VLDB*, 2004.

[25] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *IEEE Trans. on Computers*, 51(10), 2002.

[26] B. Reinwald and H. Pirahesh. Sql open heterogeneous data access. In *SIGMOD*, 1998.

[27] B. Reinwald, H. Pirahesh, G. Krishnamoorthy, G. Lapis, B. T. Tran, and S. Vora. Heterogeneous query processing through sql table functions. In *ICDE*, 1999.

[28] S. Saltenis and C. S. Jensen. Indexing of Moving Objects for Location-Based Services. In *ICDE*, 2002.

[29] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In *SIGMOD*, 2000.

[30] P. Seshadri. Predator: A Resource for Database Research. *SIGMOD Record*, 27(1):16–20, 1998.

[31] Z. Song and N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *SSTD*, 2001.

[32] J. Sun, D. Papadias, Y. Tao, and B. Liu. Querying about the Past, the Present and the Future in Spatio-Temporal Databases. In *ICDE*, 2004.

[33] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias. Spatio-Temporal Aggregation Using Sketches. In *ICDE*, 2004.

[34] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *VLDB*, 2002.

[35] Y. Tao, D. Papadias, and J. Sun. The TPR*-Tree: An Optimized Spatio-temporal Access Method for Predictive Queries. In *VLDB*, 2003.

[36] Y. Tao, J. Sun, and D. Papadias. Analysis of Predictive Spatio-Temporal Queries. *TODS*, 28(4), 2003.

[37] D. B. Terry, D. Goldberg, D. Nichols, and B. M. Oki. Continuous Queries over Append-Only Databases. In *SIGMOD*, 1992.

[38] X. Xiong, M. F. Mokbel, W. G. Aref, S. Hambrusch, and S. Prabhakar. Scalable Spatio-temporal Continuous Query Processing for Location-aware Services. In *SSDBM*, June 2004.

[39] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based Spatial Queries. In *SIGMOD*, 2003.

[40] B. Zheng and D. L. Lee. Semantic Caching in Location-Dependent Query Processing. In *SSTD*, 2001.

Towards A Streams-Based Framework for Defining Location-Based Queries

Xuegang Huang Christian S. Jensen

Department of Computer Science, Aalborg University
Fredrik Bajers Vej 7E, DK-9220, Aalborg, Denmark
{xghuang,csj}@cs.aau.dk

Abstract

An infrastructure is emerging that supports the delivery of on-line, location-enabled services to mobile users. Such services involve novel database queries, and the database research community is quite active in proposing techniques for the efficient processing of such queries. In parallel to this, the management of data streams has become an active area of research.

While most research in mobile services concerns performance issues, this paper aims to establish a formal framework for defining the semantics of queries encountered in mobile services, most notably the so-called continuous queries that are particularly relevant in this context. Rather than inventing an entirely new framework, the paper proposes a framework that builds on concepts from data streams and temporal databases. Definitions of example queries demonstrates how the framework enables clear formulation of query semantics and the comparison of queries. The paper also proposes a categorization of location-based queries.

Keywords: Location-based service, data stream, continuous query, skyline query, range query, nearest-neighbor query.

1 Introduction

The emergence of mobile services, including mobile commerce, is characterized by convergences among new technologies, applications, and services. Notably, the ability to identify the exact geographical location of a mobile user at any time opens to range of new, innovative services, which are commonly referred to as location-based services (LBSs) or location-enabled services.

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

In an LBS scenario, the service users are capable of continuous movement, and changing user locations are sampled and streamed to a processing unit, e.g., a central server. The notion of a data stream thus occurs naturally. Service requests result in queries being issued against the data streams and other, typically relational, data.

Conventional queries are one-time queries, i.e., queries that are simply issued against the state of the database as of the time of issue, upon which they, at a single point in time, return a result. In our scenario, so-called continuous queries are also natural. Such queries are "active" (i.e., being re-evaluated) for a duration of time, and their results are kept up-to-date as the database changes during this time. As an example, an in-vehicle service may display the three nearest, reasonably priced hotels with rooms available along the route towards the vehicle's destination. The vehicle's location (a data stream) together with data about hotels (relational data) are continuously queried to provide the result (a data stream).

Significant results on the processing of location-based queries (LBQs) has already been reported. As LBQs are defined in different settings, no direct means are available for classifying and comparing these queries. As more and more work, considering more and more different kinds of queries, is reported, the need for comparison increases.

This paper presents a general framework within which the semantics of LBQs can be specified. This enables the definition of LBQs in a single framework, which in turn enables the comparison of queries. The framework is well defined—it is based on precise definitions of data structures and operations on these. The framework has the following characteristics.

- Streams as well as relations are accommodated.
- Because queries often involve ranked results, relations are defined to include order.
- Relational algebraic operators are extended to also apply to streams, by using mappings of streams to relations, and, optionally, mappings of relations to streams.

The result is an expressive yet semantically simple framework that may be extended with additional operators and

mappings. To illustrate the extensibility, a new operator, the skyline operator, is introduced.

Rather than listing and defining all possible location-based queries, this paper represents several prominent ones, such as a range query, a nearest-neighbor query, and a location-based skyline query; and it discusses categorizations of LBQs.

The research area of stream data is quite active and has produced a number of interesting concepts in relation to the semantics of continuous queries. Specifically, significant research results have been reported on query processing for data streams (e.g., [3, 6, 22, 29]). Some works consider queries over data streams together with relations (e.g., [1, 18]), but only few works consider the formalization of queries over streams and relations.

Similarly, location-based query processing is an active area of research, and many interesting results have appeared. Much attention has been given to the indexing and query processing for moving objects. Numerous index structures and algorithms have been proposed for a variety of location-based queries (e.g., [4, 9, 12, 13, 14, 17, 19, 20, 23, 24, 26, 27, 28]), such as nearest neighbor queries, reverse neighbor queries, spatial range queries, distance joins, and closest-pair queries. A new type of query, the skyline query, has recently received attention [5, 8, 15, 21]. However, only little attention has been paid to query processing in relation to spatial data streams [16]. To the best of our knowledge, no formal frameworks have been proposed for the definition of location-based queries against relations and data streams.

Recently, Arasu et al. [1] have offered an interpretation of continuous queries over streams, by formalizing streams, relations, and mapping operators among them. We build on their general approach. To accommodate ordering as well as duplicates, we use list-based relations and a variant of the list-based relational algebra proposed by Slivinskas et al. [25]. To be able to express query semantics precisely, our approach also accommodates the notions of activation and deactivation times and reevaluation granularity.

The paper is outlined as follows. Section 2 defines the data structures underlying the framework and presents the application scenario. The next section completes the framework, by defining the operators that map between the different operators in the framework. Section 4 uses the framework to define different location-based queries and also discusses the categorization of location-based queries. The last section summarizes and offers directions for future research.

2 Data Structures and Application Scenario

2.1 Data Model Definition

Building on the relation concept defined by Slivinskas et al. [25], we define relations as lists to capture duplicates and ordering. We define schemas, tuples, and relation instances, then define the same concepts for streams.

Definition 2.1. A *relation schema* (Ω, Δ, dom) is a three-tuple where Ω is a finite set of attributes, Δ is a finite set of domains, and $dom : \Omega \to \Delta$ is a function that associates a domain with each attribute.

obj_id	obj_loc	obj_type
301	$(20, 35)$	police station
302	$(30, 80)$	hospital
303	$(65, 75)$	fi re department
304	$(80, 120)$	hospital
305	$(70, 80)$	police station

Figure 1: Relation r_{obj}

Relation r_{obj} in Figure 1 has schema (Ω, Δ, dom), where $\Omega = \{obj\_id, obj\_loc, obj\_type\}$, $\Delta = \{\text{number}, \text{location}, \text{string}\}$, and $dom = \{(obj\_id, \text{number}), (obj\_loc, \text{location}), (obj\_type, \text{string})\}$.

Definition 2.2. A *tuple* over *schema* $\mathcal{S} = (\Omega, \Delta, dom)$ is a function $t : \Omega \to \bigcup_{\delta \in \Delta} \delta$, such that for every attribute A of Ω, $t(A) \in dom(A)$. A *relation* over $\mathcal{S}$ is a finite sequence of tuples over $\mathcal{S}$.

The definition of a relation corresponds to the definition of a list or a sequence. A relation can thus contain duplicate tuples, and the ordering of tuples is significant. Relation r_{obj} from Figure 1 is the list $\langle t_1, t_2, t_3, t_4, t_5 \rangle$, where, e.g., $t_1 = \{(obj\_id, 301), (obj\_loc, (20, 35)), (obj\_type,$ "police station")$\}$.

Definition 2.3. A *stream schema* is a relation schema (Ω, Δ, dom), where Ω includes a special attribute T, Δ includes the time domain $\mathbb{T}$, and $dom(\mathrm{T}) = \mathbb{T}$.

We assume that domain $\mathbb{T}$ is totally ordered. While, for simplicity, we use the non-negative numbers as the time domain in the sequel, other domains may be used. For example, the real or natural numbers, the TIMESTAMP domain of the SQL standard, or one of the domains proposed by the temporal database community may be used.

Stream s_{usr} in Figure 2 has schema (Ω, Δ, dom), where $\Omega = \{usr\_id, usr\_v, usr\_loc, \mathrm{T}\}$, $\Delta = \{\text{number}, \text{velocity}, \text{location}, \mathbb{T}\}$, and $dom = \{ (usr\_id, \text{number}), (usr\_v, \text{velocity}), (usr\_loc, \text{location}), (\mathrm{T}, \mathbb{T})\}$.

usr_id	usr_v	usr_loc	T
...	...	...	...
1004	$(10, -15)$	$(90, 80)$	10
1002	$(-25, 25)$	$(200, 10)$	12
1003	$(-12, -11)$	$(60, 80)$	10
1004	$(0, 0)$	$(100, 60)$	12
1003	$(-10, 3)$	$(40, 58)$	12
1001	$(0, 1)$	$(16, 38)$	9
1004	$(20, 35)$	$(100, 60)$	15
...	...	...	...

Figure 2: Stream s_{usr}

Definition 2.4. A *stream* is a possibly infinite multiset of tuples over *stream schema* $\mathcal{T}$.

For a stream tuple t_i, the time $\tau_i = t_i(\mathrm{T})$ indicates when the tuple became available in the stream. While a relation is ordered, we have chosen to not introduce an inherent order

on streams. Streams come with the natural (partial) order implied by their time attribute.

Stream s_{usr} in Figure 2 is the possibly infinite multiset $s_{usr} = \{\!\{ \ldots, (1004, (10, -15), (90, 80), 10), \ldots, (1004, (20, 35), (100, 60), 15), \ldots \}\!\}$.

While a query is issued against an entire relation state, intuitively, a query issued at some time τ_q will only see either what has appeared in the stream so far, i.e., all tuples with timestamp less than or equal to τ_q, or what has appeared in the stream between some past time and τ_q. The latter may be assumed if what has appeared in the stream so far does not fit in the available memory.

2.2 Discussion

As we pointed out earlier, we use streams for modeling the locations of moving objects such as pedestrians, cars, and buses. We use relations for modeling aspects of an application domain that change discretely.

As we aim for a generic framework, we make no assumptions about the representations of the geographical locations and extents of objects that limit the applicability of the framework. However, to be specific, we assume that positions are simply points (x, y) in two-dimensional Euclidean space; in accord with this, a velocity vector is given by (v_x, v_y). We note that in some application scenarios, positions of objects are given in terms of road networks, using linear referencing [11]. The framework is also applicable in the context of this kind of positioning.

In the example we use throughout, stream s_{usr} in Figure 2 captures positions and velocities of moving users. Attribute usr_id records the ID of a user, and usr_v and usr_loc record the velocity and location of the user at the time instant recorded in attribute T. In a real-world application, multiple streams may well be present. For example, users moving by bicycle and by car may be captured by separate streams. For simplicity, we only use one stream.

Relation r_{usr} in Figure 3 captures discretely changing properties of the service users. As before, usr_id records the ID of a user; and attributes usr_name capture the first name of a user.

usr_id	usr_name
1001	Kate
1002	Bill
1003	Joan
1004	Tom

Figure 3: Relation r_{usr}

Finally, relation r_{obj} in Figure 1 records the points of interest that service users may query. Attribute obj_id captures the ID of a point of interest, obj_loc records its location, and obj_type records its type.

In real-world applications, additional attributes and relations may of course be used, beyond the ones introduced above.

In our scenario, the users of the services that issue the queries are moving, and the points of interests being queried are static. However, in other equally valid scenarios, a static user can query moving objects, e.g., a supermarket wants to know all the potential customers who are near the supermarket between 8:00 a.m. and 5:00 p.m. Also, a moving user may query other moving users—this may be typical of location-based games.

3 Mapping Operators

Queries are either one-time or continuous, they apply to relations and streams, and their results are either relations or streams.

Relations are well known, and the semantics of queries against relations are generally agreed upon. In contrast, what the appropriate semantics of queries against streams should be and how these should be defined are less obvious. Following Arasu et al. [1], we aim to maximally reuse

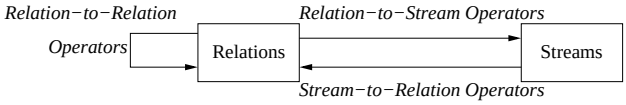


Figure 4: Mapping Operators

the relational setting in defining the semantics of queries against streams. We do this by introducing mapping operations between streams and relations, so that a query against a stream can be defined by mapping the stream to a relation, then applying a relational query, and then, optionally, mapping the result to a stream. This results in the framework of representations and operators outlined in Figure 4. Note that direct *stream-to-stream* operators are absent.

3.1 Relation-to-Relation Operators

3.1.1 Basic Algebra Operators

A relation-to-relation operator takes one or more relations $r_1, \cdots, r_n$ as arguments and produces a relation r as a result. As our relations are ordered, we use operators introduced by Slivinskas [25] as our relation-to-relation operators: selection (σ), projection (π), union-all ($\sqcup$), Cartesian Product ($\times$), difference ($\backslash$), duplicate elimination ($rdup$), aggregation (ξ), sorting ($sort$), and top (top).

These carry their standard meanings when applied to relations without order. As an example of how the operators are defined, consider selection σ. Based on the definitions in Section 2, we use $\mathcal{R}$ to be the set of all relations and let $r = \langle t_1, t_2, \ldots, t_n \rangle \in \mathcal{R}$. We let $p \in \mathcal{P}$, where (following standard practice) $\mathcal{P}$ is the set of all selection predicates (also termed propositional formulas, see, e.g., [2, pp. 13–14]) that take a tuple as argument and return True or False.

The selection operator $\sigma : [\mathcal{R} \times \mathcal{P} \rightarrow \mathcal{R}]$ is defined using λ-calculus rather than tuple relational calculus, to contend with the order. Being a parameter, argument p is expressed as a subscript, i.e., $\sigma_p(r)$.

$$\sigma \triangleq \lambda r, p.(r = \bot) \rightarrow r,$$
$$(tail(r) = \bot) \rightarrow (p(head(r)) \rightarrow head(r), \bot),$$
$$(p(head(r)) \rightarrow head(r), \bot) \mathbin{@} \sigma_p(tail(r))$$

The arguments are given before the dot, and the definition is given after the dot. Thus, if r is empty (denoted as $\bot$), the operation returns it. Otherwise, if r contains only one tuple (the remaining part of the relation, $tail(r)$, is empty), we apply predicate p to the (first) tuple, $(head(r))$. If the predicate holds, the operation returns the tuple; otherwise, it returns an empty relation. If these conditions do not hold,

the operation returns the first tuple or an empty relation (depending on the predicate), with the result of the operation applied to the remaining part of r appended (@). The common auxiliary functions $head$, $tail$, and @ are defined elsewhere (e.g., [25]). Since the objective is to obtain an expressive framework, the framework is kept open to the introduction of such auxiliary functions, although they may increase the conceptual complexity.

3.1.2 Skyline Operator

We proceed to demonstrate how a *skyline* operator, which is of particular interest in location-based services, can be expressed in the framework.

To understand the operator, consider a set of points in l-dimensional space. One point p_1 dominates another point p_2 if p_1 is at least as good as p_2 in all dimensions and is better than p_2 in at least one dimension [5]. It is assumed that a total order exists on each dimension, and "better" in a dimension is defined as smaller than (alternatively, larger than) with respect to the dimension's total order. Next, we assume a relation r with attributes $\{a_1, \ldots, a_l, b_1, \ldots, b_m\}$ so that the sub-tuples corresponding to attributes $\{a_1, \ldots, a_l\}$ make up the l-dimensional points. The skyline operator then returns all tuples in r that are not dominated by any other tuple in r.

To be precise, we first define two auxiliary functions. Let $\mathcal{T}$ denote the set of all tuples of any schema. The first function is $Dmnt$: $[\mathcal{T} \times \mathcal{R} \times \Omega^l] \rightarrow \{\text{True}, \text{False}\}$, which returns True if there exists a tuple in the (second) relation argument that dominates the first argument tuple with respect to the argument attributes.

$$
\begin{aligned}
&Dmnt \triangleq \lambda t, r, a_1, \ldots, a_l.(r = \perp) \rightarrow \text{False},\\
&\quad Eql(t, head(r), a_1, \ldots, a_l) \rightarrow Dmnt(t, tail(r), a_1, \ldots, a_l),\\
&\qquad Comp(t, head(r), a_1, \ldots, a_l) \rightarrow \text{True},\\
&\qquad\quad Dmnt(t, tail(r), a_1, \ldots, a_l)
\end{aligned}
$$

Function Eql returns True if the two argument tuples are identical on all argument attributes $a_1, \ldots, a_l$. Function $Comp$ returns True if the second argument tuple is no worse than the first argument tuple on any of the argument attributes.

In the first line, if r is empty, the operation returns False. Otherwise, if the first argument tuple t is the same as the head of argument relation r on the argument attributes, the operation continues to consider the rest of r. Else, the third line checks if $head(r)$ is no worse than t. If so, t is dominated by $head(r)$, and the operation returns True. Otherwise, the operation proceeds with the rest of r.

Next, we define auxiliary function $Fltr$: $[\mathcal{R} \times \mathcal{R} \times \Omega^l] \rightarrow \mathcal{R}$. For two relations r_1 and r_2 having the same attributes $a_1, a_2, \ldots, a_l$, $Fltr$ collects all the tuples in r_1 that are not dominated by any tuple in r_2 with respect to attributes $a_1, a_2, \ldots, a_l$.

$$
\begin{aligned}
&Fltr \triangleq \lambda r_1, r_2, a_1, \ldots, a_l.(r_1 = \perp) \rightarrow r_1,\\
&\qquad Dmnt(head(r_1), r_2, a_1, \ldots, a_l) \rightarrow\\
&\qquad\qquad Fltr(tail(r_1), r_2, a_1, \ldots, a_l),\\
&\qquad head(r_1) \mathbin{@} Fltr(tail(r_1), r_2, a_1, \ldots, a_l)
\end{aligned}
$$

Here, if r_1 is empty, the operation returns it. Otherwise, if the head of r_1 is dominated by any tuples in r_2 on the argument attributes, the operation continues with the rest of r_1. Else, it returns the $head(r_1)$ with the result of the operation applied to $tail(r_1)$ appended.

The skyline operator $skyline$: $[\mathcal{R} \times \Omega^l] \rightarrow \mathcal{R}$ is defined next. Arguments Ω^l are parameters and are expressed as subscripts, i.e., $skyline_{a_1,\ldots,a_l}(r)$.

$$skyline \triangleq \lambda r, a_1, \ldots, a_l.(r = \perp) \rightarrow r, Fltr(r, r, a_1, \ldots, a_l)$$

3.2 Stream-to-Relation Operators

A *stream-to-relation* operator takes a stream as input and produces a relation. As relations are finite while streams can be infinite, windowing is commonly used to extract a relation from a stream [3]. We describe three types of sliding windows [1]: time-based, tuple-based, and partitioned. Other types of windows can be easily incorporated into the framework, as this does not affect other parts of the framework. The stream-to-relation operators map multisets into lists. We assume that each operator described next orders its result according to the time attribute T (tuples with the same time value may be in any order).

3.2.1 Time-Based Windows

A time-based sliding window operator $\mathcal{W}^a$, with absolute or now-relative time parameter τ_s, on a stream s returns all tuples $t \in s$ for which $\tau_s \leq t(\mathrm{T}) \leq \tau_c$, where τ_c is the current time.

usr_id	usr_v	usr_loc
1001	$(0, 1)$	$(16, 38)$
1004	$(10, -15)$	$(90, 80)$
1003	$(-12, -11)$	$(60, 80)$
1002	$(-25, 25)$	$(200, 10)$
1004	$(0, 0)$	$(100, 60)$
1003	$(-10, 3)$	$(40, 58)$

Figure 5: Result of $\mathcal{W}^a_9(s_{usr})$ at Time 12

Note that $\mathcal{W}^a_{\tau_c}(s)$ consists of tuples that made their appearance in s at time τ_c, while $\mathcal{W}^a_0(s)$ consists of all tuples that appeared in the stream so far. For stream s_{usr} in Figure 2, suppose $\tau_c = 12$ and $\tau_s = 9$. The result of $\mathcal{W}^a_9(s_{usr})$ can be seen in Figure 5.

3.2.2 Tuple-Based Windows

A tuple-based sliding window operator $\mathcal{W}^b$, with positive integer parameter N, on a stream s returns the N most recent tuples in s, i.e., the tuples $t \in s$ for which $t(\mathrm{T}) \leq \tau_c$ and such that no other tuples exist in S that have larger time

usr_id	usr_v	usr_loc
1003	$(-12, -11)$	$(60, 80)$
1002	$(-25, 25)$	$(200, 100)$
1004	$(0, 0)$	$(100, 60)$
1003	$(-10, 3)$	$(40, 58)$

Figure 6: Result of $\mathcal{W}^b_4(s_{usr})$ at Time 12

values (that do not exceed τ_c). If ties exist, tuples are chosen at random among the ties. Note also that fewer than N qualifying tuples may exist.

A tuple-based window is specified as $\mathcal{W}^b_N(s)$. Note that $\mathcal{W}^b_\infty(s) = \mathcal{W}^a_0(s)$. As an example, recall s_{usr} in Figure 2 and let $\tau_c = 12$. Then $\mathcal{W}^b_4(s_{usr})$ is given in Figure 6.

3.2.3 Partitioned Windows

A partitioned sliding window over stream s takes a positive integer N and a subset of s's attributes, $\{A_1, \ldots, A_m\}$, as parameters. This operation first partitions S into substreams based on the argument attributes, then computes a tuple-based sliding window of size N independently on each substream, and then returns the union of these windows.

usr_id	usr_v	usr_loc
1001	$(0, 1)$	$(16, 38)$
1002	$(-25, 25)$	$(200, 10)$
1004	$(0, 0)$	$(100, 60)$
1003	$(-10, 3)$	$(40, 58)$

Figure 7: Result of $\mathcal{W}_{1,usr\_id}(s_{usr})$

Using $\mathcal{W}$ as the operator name, the partitioned window can be expressed as $\mathcal{W}_{N,A_1,\ldots,A_m}(s)$. To exemplify, consider s_{usr} in Figure 2, let $\tau_c = 12$, $N = 1$, and let the set of attributes be $\{usr\_id\}$. Then the result of $\mathcal{W}_{1,usr\_id}(s_{usr})$ is given in Figure 7.

3.3 Relation-to-Stream Operators

A relation r may be subject to updates, so that its state varies across time. We use the notation $r(\tau)$ to refer to the state of r at time τ. With this definition, we can specify the two relation-to-stream operators Istream and Rstream (adapted from [1]). The operators $\sqcup$, $\times$, and $\setminus$ are the algebra operators defined in Section 3.1.1.

Istream ("Insert" stream) maps relation R into a stream S so that a tuple $t \in r(\tau) \setminus r(\tau - 1)$ is mapped to $(t, \tau) \in s$. Next, Rstream maps relation r into stream s by tagging each tuple in r with each time that it is present in r. Assuming that 0 is the earliest time instant, the operators are defined as follows.

$$\text{Istream}(r) = \sqcup_{\tau>0}((r(\tau) \setminus r(\tau - 1)) \times \{\tau\}) \sqcup (r(0) \times \{0\})$$
$$\text{Rstream}(r) = \sqcup_{\tau \geq 0}(r(\tau) \times \{\tau\})$$

Assume that a moving tourist wants to continuously know the nearest hospitals. The result, which is subject to change as the tourist moves, may be returned as a stream produced using one of the windowing operators and a relation-to-stream operator. We discuss nearest-neighbor queries in the next section.

We have so far defined relational operators and mapping operators between streams and relations. As location-based queries involve operations on spatial data, spatial operators are intrinsic to such queries. We treat spatial operators as black boxes and simply assume a set of such operators. Specifically, we will use spatial operators proposed by the OpenGIS Consortium [7].

3.4 One-Time and Continuous Queries

A one-time query is a combination of stream-to-relation and relation-to-relation operators, while a continuous query is a possibly infinite numbers of one-time queries that are run repeatedly within a specified time interval according to a specified time granularity. The result of a continuous query can either be relations or streams. To generate a stream result, relation-to-stream operators are naturally employed by the continuous query; see Figure 8.

Let a one-time query be expressed as $LBQ_p(s, r)$, where s and r are argument streams and relations and p is the parameters of the query. Then a continuous query can be expressed as $CLBQ_p(s, r)[T_s, T_e, \mathcal{G}]$, where T_s and T_e are the start and end time of the continuous query and $\mathcal{G}$ is the time granularity of the query. A relation-to-stream operator may also be included in this expression to map the results into stream.

Since relations and the associated algebraic operators accommodate duplicates and order, any queries that can be expressed using traditional relational algebra can be presented in the framework; and the framework is open to new kinds of queries, as new algebraic and mapping operators can be added.

4 Location-Based Queries

The literature covers the processing of quite a few kinds of LBQs, including range, nearest-neighbor, and reverse nearest-neighbor queries, as well as closest pair queries, spatial joins, and spatial aggregate queries. Queries can concern past states of reality, or present and (anticipated) future states. Our focus will be on queries that concern the current state.

We proceed to demonstrate how the semantics of three queries can be specified: spatial range and nearest neighbor queries, and a new location-based skyline query. We end by discussing the categorization of location-based queries.

4.1 Spatial Range Query

Various kinds of spatial range queries are used commonly. A range query may be used for finding all moving objects within a circular region around a point of interest; and a continuous range query may be used for the monitoring of a region.

We assume a spatial range sr is given and define a range query (RQ) and continuous range query (CRQ) using a combination of stream-to-relation, relation-to-relation, and relation-to-stream operators.

To define the range query, we first obtain the most recent positions of all users. This is done by applying a partition window $\mathcal{W}_{1,usr\_id}(s_{usr})$ to stream s_{usr} and by applying a function CurLoc, using an extended projection. The function takes as argument a tuple $t \in s_{usr}$ that records the movement of an object, and it returns the location at time τ_c of the object.

$$r_s = \pi_{usr\_id, usr\_v, \text{CurLoc}(t) \text{ AS loc}}(\mathcal{W}_{1,usr\_id}(s_{usr}))$$
$$\text{CurLoc}(t) = t(usr\_loc) + t(usr\_v) \cdot (\tau_c - t(\text{T}))$$

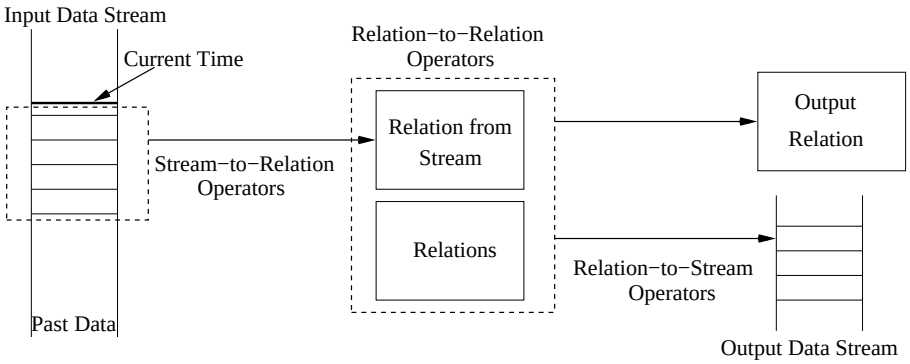


Figure 8: The Working of a Location-Based Query

Then a selection retrieves all users that are inside the spatial region. Operator within(sr, loc) returns true if loc is within spatial range sr. The one-time range query is then given as follows.

$$\mathrm{RQ}_{sr}(s_{usr}) = \sigma_{\mathrm{within}(sr,\mathrm{loc})}(r_s)$$

Next, assume that the start and end times of the continuous range query are T_s and T_e, and that the time granularity is $\mathcal{G}$. By applying the Rstream operator, the definition of the continuous range query is given next.

$$\mathrm{CRQ}_{sr}(s_{usr})[\mathrm{T}_s, \mathrm{T}_e, \mathcal{G}] = \mathsf{Rstream}(\sigma_{\mathrm{within}(sr,\mathrm{loc})}(R_s))[\mathrm{T}_s, \mathrm{T}_e, \mathcal{G}]$$

To exemplify, let spatial range sr be a circle with radius 50 and center $(20, 35)$, and let $\mathrm{T}_s = 0$, $\mathrm{T}_e = 20$, and $\mathcal{G} = 1$. Then the result of the continuous range query against stream s_{usr} is given in Figure 9. Note that the result is a stream.

usr_id	usr_v	loc	T
...	...	...	...
1001	$(0, 1)$	$(16, 38)$	9
1001	$(0, 1)$	$(16, 39)$	10
1003	$(-12, -11)$	$(48, 69)$	11
1001	$(0, 1)$	$(16, 40)$	11
...	...	...	...

Figure 9: Result Stream of $\mathrm{CRQ}_{sr}(s_{usr})[0, 20, 1]$

According to the above definition, all the users' locations are calculated at each time instant. However, it may be that some users have not reported location data for several hours, rendering their location data useless. This suggests an alternative definition where only location data that has arrived within some time duration from the current time is used.

Using the same T_s, T_e, and $\mathcal{G}$ as above and assuming that we are only interested in location data that arrived since time τ_s, an alternative definition follows.

$$\mathrm{CRQ}_{sr,\tau_s}(s_{usr})[\mathrm{T}_s, \mathrm{T}_e, \mathcal{G}] = \mathsf{Rstream}(\sigma_{\mathrm{within}(sr,\mathrm{loc})}(r'_s))[\mathrm{T}_s, \mathrm{T}_e, \mathcal{G}]$$

In this definition, r'_s is r_s where s_{usr} is replaced by $\mathsf{Rstream}(\mathcal{W}^a_{\tau_s}(s_{usr}))$. Intuitively, this query may miss some users who are actually inside the spatial range, but have not reported their location for some time.

4.2 Nearest Neighbor Query

The k nearest neighbor query (kNNQ) is another basic LBQ. Example uses include locating the nearest hospitals or emergency vehicles. To formulate the query that finds the k nearest neighbors of an object m_id, we first define several auxiliary functions.

A partitioning window query $\mathcal{W}_{1,usr\_id}(s_{usr})$ first retrieves the most recent position data for each object from the stream. Then a selection with predicate $usr\_id = m\_id$ is applied to retrieve the position data for our object. Let r_l denote the relation resulting from this selection.

To compute the k objects nearest to m_id, we calculate, using a spatial operator "dist," the distance between m_id's current location and the locations of all other objects, which are stored in attribute obj_loc of r_{obj}. As the next step in computing the query, we apply a generalized projection to associate the distance to the user object with each other object:

$$r_s = \pi_{obj\_id, obj\_loc, obj\_type, \mathrm{dis}}(r_{obj} \times r_l)$$

Here, "dis" denotes $\mathrm{dist}(obj\_loc, \mathrm{CurLoc}(t))$, function $\mathrm{CurLoc}(t)$ was defined earlier, and t denotes a tuple from the argument relation.

Then we apply the $sort$ and top operators to r_s to express the query.

$$\mathrm{kNNQ}_{m\_id,k}(s_{usr}, r_{obj}) = top_k(sort_{\mathrm{dis}}(r_s))$$

Let $r_p = \sigma_{obj\_type=\text{'police station'}}(r_{obj})$ contain all police stations and consider the query $\mathrm{kNNQ}_{1003,1}(s_{usr}, r_p)$ issued at time $\tau_c = 11$. The query finds the one police station nearest to user 1003. Using the definition, a window operator extracts all the most recent tuples for each user from stream s_{usr}. Then the tuple with $usr\_id = 1003$, $usr\_v = (-12, -11)$, $usr\_loc = (60, 80)$, and $\tau = 10$ is selected. The current location is approximated as $(60, 80) + (-12, -11) \cdot (11 - 10) = (48, 69)$. (We define the distance between points (x_1, y_1) and (x_2, y_2) as $|x_2 - x_1| + |y_2 - y_1|$). Among all objects in relation r_p, the object with $obj\_id = 304$ is selected.

If the user issues the same kind of query continuously while moving, a relation-to-stream operator may be used to map the result of each one-time query to a stream, which is expressed as follows:

$$\text{CkNNQ}_{m\_id,k}(s_{usr}, r_{obj})[\mathrm{T}_s, \mathrm{T}_e, \mathcal{G}] = \\ \text{Rstream}(\mathrm{kNNQ}_{m\_id,k}(s_{usr}, r_{obj}))[\mathrm{T}_s, \mathrm{T}_e, \mathcal{G}]$$

The result of $\text{CkNNQ}_{1003,1}(s_{usr}, r_p)[0, 20, 1]$ is shown in Figure 10.

obj_id	obj_loc	obj_type	dis	τ
...	...	...	...	...
304	$(70, 80)$	police station	10	10
304	$(70, 80)$	police station	33	11
301	$(20, 35)$	police station	43	12
301	$(20, 35)$	police station	36	13
...	...	...	...	...

Figure 10: CkNNQ over Relation r_p

4.3 Location-Based Skyline Query

The query assumes the following scenario. A user drives along a pre-defined route towards a destination. The user wants to visit one or several points of interest enroute. The most attractive of the qualifying points of interest are those that are nearest to the user's current location and that result in the smallest detour. The detour is the extra distance traveled if the user visits the point of interest and then travels to the destination.

Let r_l, CurLoc and t be as defined earlier. We assume that spatial operator "dist" takes into account the user's route, and we denote the user's destination by $dest$. Then the detour fe can be expressed as follows.

$$fe(obj\_loc, t, dest) = \mathrm{dist}(obj\_loc, \mathrm{CurLoc}(t)) + \\ \mathrm{dist}(obj\_loc, dest) - \mathrm{dist}(\mathrm{CurLoc}(t), dest)$$

Next, a (generalized) projection is applied to the Cartesian product of r_{obj} and r_l to get all the objects' distances and detours to the user:

$$r_s = \pi_{obj\_id, obj\_type, \mathrm{dis}, \mathrm{det}}(r_{obj} \times r_l)$$

Here, "dis" denotes $\mathrm{dist}(obj\_loc, \mathrm{CurLoc}(t))$ and "det" denotes $fe(obj\_loc, t, dest)$.

Finally, the skyline operation generates the result.

$$\text{SQ}_{m\_id,dest}(s_{usr}, r_{obj}) = skyline_{\mathrm{dis}, \mathrm{det}}(r_s)$$

Following the scenario described above, let $\tau_c = 15$ and assume a user with $usr\_id = 1002$ wants to go to a hospital or a police station enroute to the destination, the location of which is $(10, 90)$. For simplicity, we use direct line segments as routes between two points. Using the calculation above, the current location of the user is $(200, 10) + (-25, 25) \cdot (15 - 12) = (125, 85)$. For all static objects with usr_type "police station" and "hospital," the distance and detour are listed in Figure 11. The skyline operator returns the last three tuples.

obj_id	obj_type	dis	det
301	police station	155	100
302	hospital	100	10
304	hospital	80	60
305	police station	60	20

Figure 11: Intermediate Result

4.4 Towards a Categorization of Location-Based Queries

As it is obviously impossible to define all possible LBQs, we proceed to explore the space of possible LBQs by presenting several orthogonal categorizations of such queries.

First, queries can be categorized based on whether they refer to data concerning the past, present, or future states of reality.

Second, queries can be categorized according to whether they are one-time or continuous queries. Continuous queries may be classified further, based on whether they are constant or time-parameterized. The latter occurs when a query refers to the (variable) current time. An example is a continuous query that retrieves all objects currently within a spatial range. A corresponding constant query might retrieve all objects that are (currently believed to be) within a spatial range at some fixed near future time. Constant continuous queries have been termed "persistent" in the literature.

Third, queries may be classified as being either "one-to-many" or "many-to-many." The former queries apply one predicate to many objects, returning one set, multiset, or list of objects. The latter conceptually repeatedly applies many different predicates to many objects, potentially retrieving many objects for each predicate.

A simple selection is thus an example of the former. The k nearest neighbor query in Section 4.2 retrieves (up to) k objects that are the nearest to some (i.e., "one") specified object; it is thus also a "one-to-many" query. In contrast, joins are "many-to-many" queries: The predicate involving one (left hand side) object is applied to many (right hand side) objects, and this repeated many times. The so-called "closest pair" query, which finds pairs of objects from two different groups that are closest, is also a "many-to-many."

Fourth, LBQs may be categorized based on whether they involve "topological," "directional," or "metric" predicates.

Fifth, based on the time at which a query is registered to the system, it can be "pre-defined," meaning that it is present before the streams it uses start, or it can be "ad hoc," meaning that it is registered after at least one of its streams has started.

5 Summary and Future Work

Substantial research has been reported on query processing in relation to mobile services, in particular location-enabled mobile services. Different techniques are applicable to different kinds of queries. Based on results from stream and temporal databases, this paper proposes a framework for capturing the semantics of the diverse kinds of queries that are relevant in this context. By enabling the definition of queries in a single framework, the paper's proposal enables the comparison of queries.

The framework consists of data types, relations and streams, as well as algebraic operations on relations and

operations that map between streams and relations. The specific representations of spatial data and the associated operations on these are treated as black boxes, in order to enable applicability across different such representations and operations. The extensibility of the framework was exemplified by adding a skyline operator.

The use of the framework was illustrated by the definition of three location-based queries, a spatial range query, a nearest-neighbor query, and a location-based skyline query. Toy examples were given for illustrating these queries. Focus has been on the capture of the semantics of LBQs, and how to use one-time or continuous queries in actual location-based services is beyond the scope of the paper.

This paper represents initial work, and future work may be pursued in several directions. First, the framework may be enriched in various ways. One is to introduce explicit representations of the space within which the spatial objects are located and move, e.g., road networks. Second, while this paper has given one definition of each of three queries, it would be worthwhile to explore the different possible semantics that may be given to queries within the framework. Such a study may reveal whether or not desirable semantics can be specified in all cases. Third, more work on taxonomies for location-based queries is desirable. Interesting initial steps have been taken in this direction (e.g., [23]), but much more detail is desirable.

References

[1] A. Arasu, S. Babu, and J. Widom. An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations. Technical Report. Department of Computer Science, Stanford University, 12 pages, 2002.

[2] P. Atzeni and V. De Antonellis. *Relational Database Theory.* Benjamin/Cummings, 1993.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *Proc. PODS*, pp. 1–16, 2002.

[4] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Šaltenis. Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. In *Proc. IDEAS*, pp. 44–53, 2002.

[5] S. Borzsonyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *Proc. ICDE*, pp. 421–430, 2001.

[6] D. Carney, U. Centintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring Streams: A New Class of Data Management Applications. In *Proc. VLDB*, pp. 215–226, 2002.

[7] E. Clementini and P. D. Felice. Spatial Operators. In *SIGMOD Record*, **29** (3), pp. 31–38, 2000.

[8] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with Presorting. In *Proc. ICDE*, pp. 717–816, 2003.

[9] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Closest Pair Queries in Spatial Databases. In *Proc. SIGMOD Conf.*, pp. 189–200, 2000.

[10] L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. A Data Model and Data Structures for Moving Objects Databases. In *Proc. SIGMOD Conf.*, pp. 319–330, 2000.

[11] C. Hage, C. S. Jensen, T. B. Pedersen, L. Speičys, and I. Timko. Integrated Data Management for Mobile Services in the Real World. In *Proc. VLDB*, pp. 1019–1030, 2003.

[12] A. Hinze and A. Voisard. Location- and Time-Based Information Delivery in Tourism. In *Proc. SSTD*, pp. 489–507, 2003.

[13] S. Idreos and M. Koubarakis. P2P-DIET: Ad-hoc and Continuous Queries in Peer-to-Peer Networks Using Mobile Agents. In *Proc. SETN*, pp. 23–32, 2004.

[14] D. V. Kalashnikov, S. Prabhakar, W. G. Aref, and S. E. Hambrusch. Effi cient Evaluation of Continuous Range Queries on Moving Objects. In *Proc. DEXA*, pp. 731–740, 2002.

[15] D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *Proc. VLDB*, pp. 275–286, 2002.

[16] X. Liu, H. Ferhatosmanoglu. Effi cient k-NN Search on Streaming Data Series. In *Proc. SSTD*, pp. 83–101, 2003.

[17] H. Mokhtar and J. Su. Universal Trajectory Queries for Moving Object Databases. In *Proc. MDM*, pp. 133–145, 2004.

[18] S. Madden, M. Shah, J. Hellerstein, and V. Raman. Continuously Adaptive Continuous Queries Over Streams. In *Proc. SIGMOD Conf.*, pp. 49–60, 2002.

[19] M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In *Proc. SIGMOD Conf.*, 2004.

[20] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group Nearest Neighbor Queries. In *Proc. ICDE*, pp. 301–312, 2004.

[21] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An Optimal and Progressive Algorithm for Skyline Queries. In *Proc. SIGMOD Conf.*, pp. 467–478, 2003.

[22] L. Qiao, D. Agrawal, and A. E. Abbadi. Supporting Sliding Window Queries for Continuous Data Streams. In *Proc. SSDBM*, pp. 85–94, 2003.

[23] A. Y. Seydim, M. H. Dunham, and V. Kumar. Location Dependent Query Processing. In *Proc. MobiDE*, pp. 47–53, 2001.

[24] S. Šaltenis and C. S. Jensen. Indexing of Moving Objects for Location-Based Services. In *Proc. ICDE*, pp. 463–472, 2002.

[25] G. Slivinskas, C. S. Jensen, and R. T. Snodgrass. Query Plans for Conventional and Temporal Queries Involving Duplicates and Ordering. In *Proc. ICDE*, pp. 547–558, 2000.

[26] Z. Song and N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *Proc. SSTD*, pp. 79–96, 2001.

[27] S. Shekhar and J. S. Yoo. Processing In-Route Nearest Neighbor Queries: A Comparison of Alternative Approaches. In *Proc. ACMGIS*, pp. 9–16, 2003.

[28] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *Proc. VLDB*, pp. 287–298, 2002.

[29] D. Zhang, D. Gunopulos, V. J. Tsotras, and B. Seeger. Temporal and Spatio-Temporal Aggregations over Data Streams Using Multiple Time Granularities. In *INF. SYST.* **28** (1–2), pp. 61–84, 2003.