Computing the Cardinal Direction Development between Moving Points in Spatio-temporal Databases

Tao Chen, Hechen Liu & Markus Schneider*

Department of Computer and Information Science and Engineering
University of Florida
Gainesville, FL 32611, USA
{tachen,heliu, mschneid}@cise.ufl.edu

Abstract. In the same way as moving objects can change their location over time, the spatial relationships between them can change over time. An important class of spatial relationships are cardinal directions like *north* and *southeast*. In spatial databases and GIS, they characterize the relative directional position between *static* objects in space and are frequently used as selection and join criteria in spatial queries. Transferred to a spatiotemporal context, the simultaneous location change of different moving objects can imply a temporal evolution of their directional relationships, called *development*. In this paper, we provide an algorithmic solution for determining such a temporal development of cardinal directions between two moving points. Based on the *slice representation* of moving points, our solution consists of three phases, the *time-synchronized interval refinement phase* for synchronizing the time intervals of two moving points, the *slice unit direction evaluation phase* for computing the cardinal directions between two slice units that are defined in the same time interval from both moving points, and finally the *direction composition phase* for composing the cardinal directions computed from each slice unit pair. Finally, we show the integration of spatio-temporal cardinal directions into spatio-temporal queries as spatio-temporal directional predicates, and present a case study on the hurricane data.

1 Introduction

Objects that continuously change their positions over time, so-called *moving objects*, have recently received a lot of interest. Examples are moving points like vehicles, mobile devices, and animals, for which the time-dependent position is relevant. Temporal movements of spatial objects induce modifications of their spatial relationships over time, called *developments*. In spatial databases and

* This work was partially supported by the National Science Foundation under grant number NSF-IIS-0812194 and by the National Aeronautics and Space Administration (NASA) under the grant number NASA-AIST-08-0081.

GIS, spatio-temporal queries are particularly interesting when they ask for temporal changes in the spatial relationships between moving objects. An important class of spatial relationships are cardinal directions like *north* and *southeast* that characterize the relative directional position between spatial objects. Cardinal directions between two static objects have been extensively studied and have been frequently used as selection and join criteria in spatial queries. Transferred to a spatio-temporal context, the simultaneous location change of different moving objects can imply a change of their directional relationships. For example, a fishing boat that is southwest of a storm might be north of it some time later. We call this a *cardinal direction development*. Such a development between two moving objects describes a temporally ordered sequence of cardinal directions where each cardinal direction holds for a certain time interval during their movements. A development reflects the impact of time on the directional relationships between two moving objects, and usually proceeds continuously over time if the movements of the two objects are continuous.

It is an open, interesting, and challenging problem to capture the cardinal direction development between moving objects. Consider a database containing information about weather conditions. The query whether a hurricane stayed all the time to the southeast of another hurricane, and the query whether a hurricane has ever moved to the southeast of another hurricane can be particularly interesting to hurricane researchers to understand dynamic weather movement patterns. To answer these queries with current approaches and systems, we would need to check the validity of the spatial directional predicate, e.g. *southeast*, at all time instances during the common life time of both hurricanes. However, this is not possible since the movements of the hurricanes are continuous. The fact that the traditional, static cardinal directions cannot describe continuous, time dependent relationships leads to the need for new modeling strategies.

We have proposed a modeling strategy for cardinal direction developments in our previous work, in which we have defined the development of cardinal directions over time as a sequence of temporally ordered and enduring cardinal directions. In this paper, we propose our solution from an algorithmic perspective. We base our solution on the *slice representation* of moving points, which represents the temporal development of a point with a sequence of timely ordered units called *slices*. We propose a three-phase solution for determining the developments of the directional relationships between two moving points. In a *time-synchronized interval refinement phase*, two moving points are refined by synchronizing their time intervals. As a result, each slice unit of the refined slice representation of the first moving point has a matching slice unit in the refined slice representation of the second moving point with the time interval. In the second phase, the *slice unit direction evaluation phase*, we present a strategy of computing cardinal directions between two slice units from both moving points. Finally, in the *direction composition phase*, the development of the cardinal direction is determined by composing cardinal directions computed from all slices pairs from both moving points.

Section 2 introduces the related work in the literature. In Section 3, we review our modeling stratergy for cardinal direction developments. We propose a three-phase approach to computing the developments of cardinal directions between two moving points in Section 4. Section 5 defines spatio-temporal directional predicates for integrating cardinal direction developments into spatial-temporal databases and query languages. We present a case study on the hurricane best track data collected from National Hurricane Center (NHC) in Section 6, and show how the cardinal direction developments can help hurricane researchers to identify interesting weather event patterns. In Section 7, we draw some conclusions and discuss future work.

2 Related Work

A number of spatio-temporal models have been proposed to represent and manage moving objects. Early approaches tried to extend the existing spatial data models with temporal concepts. One approach is to store the location and geometry of moving objects with discrete snapshots over time. In [1], a spatio-temporal object o is defined as a time-evolving spatial object whose evolution is represented by a set of triplets (o_{id}, s_i, t_i), where o_{id} identifies the object o and s_i is the location of o at time instant t_i. Another approach in [2] applies linear constraints for modeling spatio-temporal data. It associates the spatial features like location and geometry of a moving object with consecutive time intervals. A common drawback of the two approaches mentioned so far is that, ultimately, they are incapable of modeling continuous changes of spatial objects over time. New approaches have been proposed to support a more integrated view of space and time, and to incorporate the treatment of continuous spatial changes. In [3, 4], the concept of *spatio-temporal data types* is proposed as *abstract data types* (*ADTs*) whose values can be integrated as complex entities into databases. A temporal version of an object of type α is given by a function from time to α. Spatio-temporal objects are regarded as special instances of temporal objects where α is a spatial data type like *point* or *region*. A *point* (representing an airplane, for example) that changes its location in the Euclidean plane over time is called a *moving point*. In this paper, we follow the specification of *spatio-temporal data types*, particularly the moving point data type, and take it as our basis for modeling cardinal directions.

Qualitative spatial relationships have a long tradition in GIS and spatial databases. They can be grouped into three categories: *topological*, *directional* and *distance*. The same classification holds for the relationships between moving objects. The distinction is that spatial relationships between moving objects can have a temporal evolution, i.e. they may change over time. So far, the focus has been mainly on spatio-temporal topological relationships like *cross* and *enter* [5, 6], and spatio-temporal distance relationships like *moving towards*, *moving away from*, [7] and *opposite_direction* [6]. Cardinal directions in a spatio-temporal context have been largely neglected in the literature. Static cardinal directions like *north* and *northeast* represent important qualitative spatial relationships that

describe relative direction positions between static spatial objects. Many models follow a *projection-based* approach, where direction relationships are defined using projection lines orthogonal to the coordinate axes [8, 9]. Some models apply a *cone-based* approach that defines direction relations by using angular zones [10, 11]. Others like the *Minimum Bounding Rectangle* (*MBR*) model [12] make use of the minimum bounding rectangles of both operand objects and apply Allen's 13 interval relations to the rectangle projections on the x- and y-axes respectively. However, all existing cardinal direction models only consider static directional relationships, and when transferred to a spatio-temporal context, none of the models is capable of modeling directional relationships that continuously change over time. In [13], an attempt has been made to model *moving spatio-temporal relationships* (*mst-relation*), which includes both topological relations and directional relations. During a time interval I_k, the mst-relation between two moving objects A_i and A_j is expressed as A_i (α, β, I_k) A_j, where α is any topological relation among *Equal*, *Inside*, *Contain*, *Cover*, *Covered By*, *Overlap*, *Touch* and *Disjoint* and β is one of the 12 directional relations, *South*, *North*, *West*, *East*, *Northwest*, *Northeast*, *Southwest*, *Southeast*, *Left*, *Right*, *Below* and *Above*. Both A_i α A_j and A_i β A_j are true during the interval I_k. This model provides a way of describing the topological and directional relationships between two moving objects. However, it is not clear how the relationships are determined. There are currently no well established strategies for modeling cardinal directions between two moving objects, and it is the main goal of this paper to bridge this gap.

We have presented a modeling strategy for cardinal direction developments in [14], in which the cardinal direction development between two moving points is formally defined. In this paper, we focus on the design of algorithms for computing such a cardinal directional development.

3 A Review of the Modeling Strategy for Cardinal Direction Developments between Moving Points

The approach that is usually taken for defining cardinal directions between two static points in the Euclidean plane is to divide the plane into partitions using the two points. One popular partition method is the *projection-based* method that uses lines orthogonal to the x- and y-coordinate axes to make partitions [12, 8]. The point that is used to create the partitions is called the *reference* point, and the other point is called the *target* point. The direction relation between two points is then determined by the partition that the *target* object is in, with respect to the *reference* object. Let $Points$ denote the set of static point objects, and let $p, q \in Points$ be two static point objects, where p is the target point and q is the reference point. A total of 9 mutually exclusive cardinal directions are possible between p and q. Let CD denote the set of 9 cardinal directions, then CD=$\{$*northwest* (NW), *restrictednorth* (N), *northeast* (NE), *restrictedwest* (W), *sameposition*(SP), *restrictedeast* (E), *southwest* (SW), *restrictedsouth* (S), *southeast* (SE)$\}$. Let $dir(p, q)$ denote the function that returns the cardinal direction between two static points p and q where q is the reference point, then

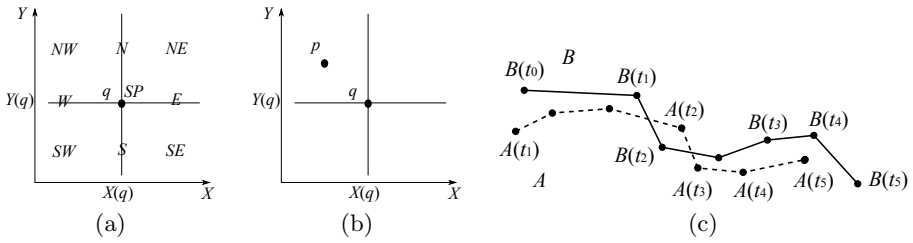

Fig. 1. Plane partitions using the reference object q (a), and an example in which p is *northwest* of q (b); the trajectories of moving points A and B in the time interval $[t_0, t_5]$ (c)

we have $dir(p, q) \in CD$. Figure 1a shows the partitions of the plane with respect to the reference point q, where each partition corresponds to the definition of one cardinal direction. The example in Figure 1b gives the case when p is to the *northwest* of q, i.e. $dir(p, q) = NW$.

When two points change their locations over time, the direction relation between them becomes time related, and may or may not change. First, we consider the cardinal directions at time instances. Let *time* denote the temporal data type representing time and *MPoints* denote the spatio-temporal data type that represents moving points. Figure 1c shows an example of two moving points A and B. For $A, B \in MPoints$, let $A(t)$ and $B(t)$ denote the snapshots of A and B at a time instance $t \in time$. If both A and B are defined at time t, then $A(t), B(t) \in Points$. The cardinal direction between A and B at t is therefore $dir(A(t),B(t)) \in CD$. For example, in Figure 1c, at time t_1 when A and B locate at $A(t_1)$ and $B(t_1)$, the cardinal direction between A and B at time instance t_1 is $dir(A(t_1), B(t_1))=SW$. At the time instance t_2 when A and B move to $A(t_2)$ and $B(t_2)$, the cardinal direction between them becomes $dir(A(t_2), B(t_2))=NE$. We propose our solution to determine what happened in between and to answer the question whether there exists a time instance t ($t_1 < t < t_2$) such that $dir(A(t), B(t))=W$ in the following sections. This scenario shows that within a common time interval, we may get different cardinal directions at different time instances. However, the change of time does not necessarily imply the change of cardinal directions between two moving points. In Figure 1c, from time t_3 to time t_4, A moves from $A(t_3)$ to $A(t_4)$ and B moves from $B(t_3)$ to $B(t_4)$. One observation that we can make is that although the positions of A and B have changed, the cardinal direction between A and B does not change. In this case, A is always to the *southwest* of B between t_3 and t_4. In other words, the cardinal direction between two moving points holds for a certain period of time before it changes. Based on this fact, we propose our modeling strategy. To determine the cardinal directions between two moving points during their life time, we first find out the *common life time intervals* between two moving points, on which both two moving points are defined. This is necessary because only when both moving points exist, we can determine the cardinal directions between them. In this case, the common life time interval between A and B in Figure 1c is $[t_1, t_5]$, and during the

time interval $[t_0, t_1]$, cardinal directions between A and B cannot be determined. Then, each common life interval is split into a list of smaller sub-intervals such that during each sub-interval the cardinal direction between two moving points does not change Further, on adjacent sub-intervals, different cardinal directions hold. Finally, we compose all cardinal directions determined on the common life time intervals of two moving points, and define it as the development of cardinal directions between the two moving points. Let $DEV(A, B)$ denote the function that computes the cardinal direction developments between two moving points A and B. Then we define $DEV(A, B)$ as $DEV(A, B) = d_1 \triangleright d_2 \triangleright ... \triangleright d_n$, where $d_i \in CD$ or $d_i = \perp$ ($1 \leq i \leq n$ and $\perp$ means undefined). Further, we restrain the transition between two adjacent cardinal directions to follow a so-called *state transition diagram*. The formal definitions and the detailed explanations can be found in [14].

4 Computing Developments between Moving Points

The concept we have introduced in the previous section serves as a specification for describing the changing cardinal directions between two moving points. However, issues like how to find common life time intervals and how to split them are left open. In this section, we overcome the issues from an algorithmic perspective. We first introduce the underlying data structure, called *slice representation*, for representing moving points. Then we propose a three phase strategy including the *time-synchronized interval refinement phase*, the *slice unit direction evaluation phase*, and *the direction composition phase*.

4.1 The Slice Representation for Moving Points

Since we take the specification of the moving point data type in [3, 4] as our basis, we first review the representation of the moving point data type. According to the definition, the moving point date type describes the temporal development of a complex point object which may be a point cloud. However, we here only consider the simple moving point that involves exactly one single point. A *slice representation* technique is employed to represent a moving point object. The basic idea is to decompose its temporal development into fragments called "slices", where within each slice this development is described by a simple linear function. A slice of a single moving point is called a *upoint*, which is a pair of values (*interval*, *unit-function*). The *interval* value defines the time interval for which the unit is valid; the *unit-function* value contains a record (x_0, x_1, y_0, y_1) of coefficients representing the linear function $f(t)$=($x_0 + x_1 t$, $y_0 + y_1 t$), where t is a time variable. Such functions describe a linearly moving point. The time intervals of any two distinct slice units are disjoint; hence units can be totally ordered by time. More formally, let A be a single moving point representation, *interval* = *time* $\times$ *time*, *real4* = *real* $\times$ *real* $\times$ *real* $\times$ *real*, and *upoint* = *interval* $\times$ *real4*. Then A can be represented as an array of slice units ordered by time, that is, $A = \langle (I_1, c_1), (I_2, c_2), ..., (I_n, c_n) \rangle$ where for $1 \leq i \leq n$ holds that

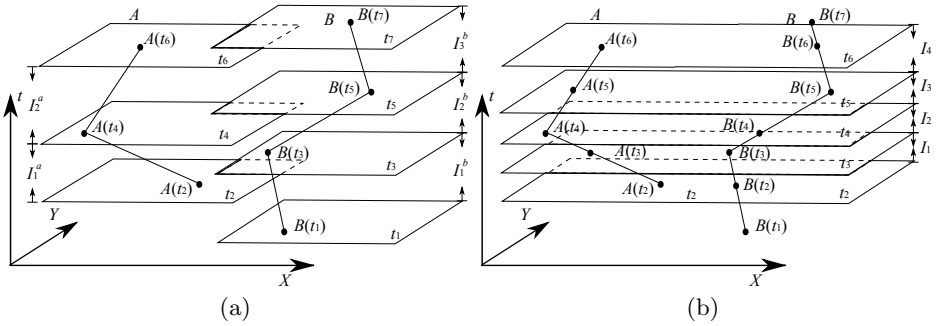

Fig. 2. An example of the slice representations of two single moving points A and B (a), and the time-synchronized slice representation of two moving points A and B (b)

$I_i \in interval$ and $c_i \in real4$ contains the coefficients of a linear unit function f_i. Further, we require that $I_i < I_j$ holds for $1 \leq i < j \leq n$.

Figure 2 shows the slice representations of two single moving points A and B. In this example, t_i $(1 \leq i \leq 7)$ is a time instance and for $1 \leq i < j \leq 7$, $t_i < t_j$. The moving point A is decomposed into two slices with intervals $I_1^A = [t_2, t_4]$ and $I_2^A = [t_4, t_6]$. Let the function f_1^A with its coefficients c_1^A and the function f_2^A with its coefficients c_2^A describe the movement of A in the intervals I_1^A and I_2^A respectively. Then A is represented as $A = \langle (I_1^A, c_1^A), (I_2^A, c_2^A) \rangle$. The snapshots $f_1^A(t_2)$ and $f_1^A(t_4)$ of the moving point A at the times t_2 and t_4 are the start and end points of the first slice, and $f_2^A(t_4)$ and $f_2^A(t_6)$ are the start and end points of the second slice. Similarly, the moving point B can be represented as $B = \langle (I_1^B, c_1^B), (I_2^B, c_2^B), (I_3^B, c_3^B) \rangle$ where c_1^B, c_2^B, and c_3^B contain the coefficients of the three linear functions f_1^B, f_2^B, and f_3^B that describe the linear movement of B in its three slice units. If the function f_i^A or f_i^B that in a slice unit maps a time instant t to a point value in A or B is not important, we allow the notations $A(t)$ and $B(t)$ respectively to retrieve the location of a moving point A or B at the time instant t.

Further, we introduce a few basic operations for retrieving information from the slice representation, which will be used by our algorithm later for computing cardinal directions between moving points.

The first set of operations is provided for manipulating moving points. The *get_first_slice* operation retrieves the first slice unit in a slice sequence of a moving point, and sets the current position to 1. The *get_next_slice* operation returns the next slice unit of the current position in the sequence and increments the current position. The predicate *end_of_sequence* yields *true* if the current position exceeds the end of the slice sequence. The operation *create_new* creates an empty *MPoint* object with an empty slice sequence. Finally, the operation *add_slice* adds a slice unit to the end of the slice sequence of a moving point.

The second set of operations is provided for accessing elements in a slice unit. The operation *get_interval* returns the time interval of a slice unit. The operation *get_unit_function* returns a record that represents the linear function of a slice

unit. The *create_slice* operation creates a slice unit based on the provided time interval and the linear function.

Based on the slice representation and the basic operations, we are now ready to describe our strategy for computing the cardinal directions between two moving points.

4.2 The Time-synchronized Interval Refinement Phase

Since a slice is the smallest unit in the slice representation of moving points, we first consider the problem of computing cardinal directions between two moving point slices. According to our definitions in [14] the cardinal directions only make sense when the same time intervals are considered for both moving points. However, matching, i.e., equal, slice intervals can usually not be found in both moving points. For example, in Figure 2, the slice interval $I_1^A = [t_2, t_4]$ of A does not match any of the slice intervals of B. Although the slice interval $I_1^B = [t_1, t_3]$ of B overlaps with I_1^A, it also covers a sub-interval $[t_1, t_2]$ that is not part of I_1^A, which makes the two slices defined in I_1^A and I_1^B incomparable. Thus, in order to compute the cardinal directions between two moving point slices, a *time-synchronized interval refinement* for both moving points is necessary.

We introduce a linear algorithm *interval_sync* for synchronizing the intervals of both moving points. The input of the algorithm consists of two slice sequences $mp1$ and $mp2$ that represent the two original moving points, and two empty lists $nmp1$ and $nmp2$ that are used to store the two new interval refined moving points. The algorithm performs a parallel scan of the two original slice sequences, and computes the intersections between the time intervals from two moving points. Once an interval intersection is captured, two new slices associated with the interval intersection are created for both moving points and are added to the new slice sequences of the two moving points. Let $I = [t_1, t_2]$ and $I' = [t'_1, t'_2]$ denote two time intervals, and let *lower_than* denote the predicate that checks the relationship between two intervals. Then we have $lower\_than(I, I') = true$ if and only if $t_2 < t'_2$. Further, let *intersection* denote the function that computes the intersection of two time intervals, which returns $\emptyset$ if no intersection exists. We present the corresponding algorithm *interval_sync* in Figure 3.

As a result of the algorithm, we obtain two new slice sequences for the two moving points in which both operand objects are synchronized in the sense that for each unit in the first moving point there exists a matching unit in the second moving point with the same unit interval and vice versa. For example, after the time-synchronized interval refinement, the two slice representations of the moving points A and B in Figure 2 become $A = \langle (I_1, c_1^A), (I_2, c_1^A), (I_3, c_2^A), (I_4, c_2^A) \rangle$ and $B = \langle (I_1, c_1^B), (I_2, c_2^B), (I_3, c_2^B), (I_4, c_3^B) \rangle$, where the c_i^A with $i \in \{1, 2\}$ contain the coefficients of the linear unit functions f_i^A, the c_i^B with $i \in \{1, 2, 3\}$ contain the coefficients of the linear unit functions f_i^B, and $I_1 = intersection(I_1^A, I_1^B) = [t_2, t_3]$, $I_2 = intersection(I_1^A, I_2^B) = [t_3, t_4]$, $I_3 = intersection(I_2^A, I_2^B) = [t_4, t_5]$, and $I_4 = intersection(I_2^A, I_3^B) = [t_5, t_6]$.

Now we analyze the complexity of the algorithm for function *interval_sync*. Assume that the first moving point mp_1 is composed of m slices, and the second

```
method interval_sync (mp1, mp2,
                      nmp1, nmp2)
  s1 ← get_first_slice(mp1)
  s2 ← get_first_slice(mp2)
  while not end_of_sequence(mp1)
  and not end_of_sequence(mp2) do
    i1 ← get_interval(s1)
    i2 ← get_interval(s2)
    i ← intersection(i1, i2)
    if i ≠ ∅ then
      f1 ← get_unit_function(s1)
      f2 ← get_unit_function(s2)
      ns1 ← create_slice(i, f1)
      ns2 ← create_slice(i, f2)
      add_slice(nmp1, ns1)
      add_slice(nmp2, ns2)
    endif
    if lower_than(i1, i2) then
      s1 ← get_next_slice(mp1)
    else
      s2 ← get_next_slice(mp2)
    endif
  endwhile
end
```

```
1  method compute_dir_dev(sl1, sl2)
2    dev_list ← empty list
3    s1 ← get_first_slice(sl1)
4    s2 ← get_first_slice(sl2)
5    slice_dir_list ← compute_slice_dir(s1,s2)
6    append(dev_list, slice_dir_list)
7    while not end_of_sequence(sl1)
8    and not end_of_sequence(sl2) do
9      (b, e) ← get_interval(s1)
10     s1 ← get_next_slice(sl1)
11     s2 ← get_next_slice(sl2)
12     (b_new, e_new) ← get_interval(s1)
13     if e < b_new then
14       append(dev_list, ⟨⊥⟩)
15     endif
16     slice_dir_list ← compute_slice_dir(s1,s2)
17     last_dir ← get_last_in_list(dev_list)
18     new_dir ← get_first_in_list(slice_dir_list)
19     if last_dir = new_dir then
20       remove_first(slice_dir_list)
21     endif
22     append(dev_list, slice_dir_list)
23   endwhile
24   return dev_list
25 end
```

Fig. 3. The algorithm *interval_sync* that computes the time-synchronized interval refinement for two moving points, and the algorithm *compute_dir_dev* that computes the cardinal direction development for two moving points.

moving point mp_2 is composed of n slices. Since a parallel scan of the slice sequences from two moving points is performed, the complexity is therefore $O(m + n)$ and the result contains at most $(m + n)$ intervals.

4.3 The Slice Unit Direction Evaluation Phase

From the first phase, the *time-synchronized interval refinement* phase, we obtain two refined slice sequences of both moving points that contain the same number of slice units with synchronized time intervals. In the second phase, we propose a solution for computing the cardinal directions between any pair of time-synchronized slice units.

We adopt a two-step approach to computing the cardinal directions between two slice units. The first step is to construct a mapping and apply it to both slice units so that one of the slice units is mapped to a slice unit that consists of a point that does not change its location. We prove that the mapping is a *cardinal direction preserving mapping* that does not change the cardinal direction relationships between the two slice units. The second step is to determine the cardinal directions between the two mapped slice units.

The difficulty of computing cardinal directions between two slice units comes from the fact that the positions of the moving points change continuously in both slice units. A much simpler scenario is that only one slice unit consists of a moving point, whereas the other slice unit involves no movement. In this simpler case, the cardinal directions can be easily determined. Thus, the goal is to find a mapping that maps two slice units su_a and su_b to two new slice units su'_a and su'_b that satisfy the following two conditions: (i) su'_a and su'_b have the same cardinal directions as units su_a and su_b, that is, the mapping is a *cardinal direction preserving mapping*; (ii) either su'_a or su'_b does not involve movement.

In order to find such a mapping for two slice units, we first introduce a simple *cardinal direction preserving mapping* for static points. Let p and q denote two points with coordinates (x_p, y_p) and (x_q, y_q). Let $X(r)$ and $Y(r)$ denote the functions that return the x-coordinate and y-coordinate of a point r. We establish a simple translation mapping $M(r) = (X(r) - x_0, Y(r) - y_0)$, where x_0 and y_0 are two constant values. We show that the cardinal direction between p and q is preserved by applying such a mapping.

Lemma 1. *Given $p = (x_p, y_p)$, $q = (x_q, y_q)$, the mapping $M(r)$=($X(r) - x_0$, $Y(r) - y_0$), where r is a point and x_0 and y_0 are two constant values, and $p' = M(p)$ and $q' = M(q)$, we have $dir(p, q) = dir(p', q')$*

Proof. According to the definition in Section 3, the cardinal direction $dir(p, q)$ between two points p and q is based on the value of $X(p) - X(q)$ and the value of $Y(p) - Y(q)$. Since we have $X(p') - X(q') = X(p) - X(q)$ and $Y(p') - Y(q') = Y(p) - Y(q)$, we obtain $dir(p, q) = dir(p', q')$. □

In Figure 4(a), two points p and q are mapped to p' and q', and the cardinal direction is preserved after the mapping, i.e., $dir(p, q) = dir(p', q') = NW$.

Now we are ready to define a *cardinal direction preserving mapping* for two slice units. Let su^A and su^B denote two slice units (*upoint* values) from the time-synchronized moving points A and B where $su^A = (I, c^A)$ and $su^B = (I, c^B)$ with $I \in interval$ and $c^A, c^B \in real4$. Let f^A and f^B be the two corresponding linear unit functions with the coefficients from c^A and c^B respectively. We establish the following mapping M for a unit function $f \in \{f^A, f^B\}$: $M(f) = f - f^B$

We show in Lemma 2 that by mapping the unit functions of the slices su_A and su_B to two new unit functions, the cardinal directions between the slice units su_A and su_B are still preserved.

Lemma 2. *Let $su^A = (I, c^A) \in$ upoint and $su^B = (I, c^B) \in$ upoint, and let f^A and f^B be the corresponding linear unit functions with the coefficients from $c^A = (x_0^A, x_1^A, y_0^A, y_1^A)$ and $c^B = (x_0^B, x_1^B, y_0^B, y_1^B)$ respectively. We consider the mapping $M(f) = f - f^B$, where f is a linear unit function, and the translated* upoint *values $su_t^A = (I, c_t^A)$ and $su_t^B = (I, c_t^B)$ where c_t^A and c_t^B contain the coefficients of $M(f^A)$ and $M(f^B)$ respectively. Then, the cardinal directions between the slice units su_t^A and su_t^B are the same as the cardinal directions between the slice units su^A and su^B.*

Proof. Let $I = [t_1, t_2]$, $f^A(t) = (x_0^A + x_1^A t, y_0^A + y_1^A t)$, and $f^B(t) = (x_0^B + x_1^B t, y_0^B + y_1^B t)$. Then we have $M(f^A) = (x_0^A - x_0^B + (x_1^A - x_1^B)t, y_0^A -$

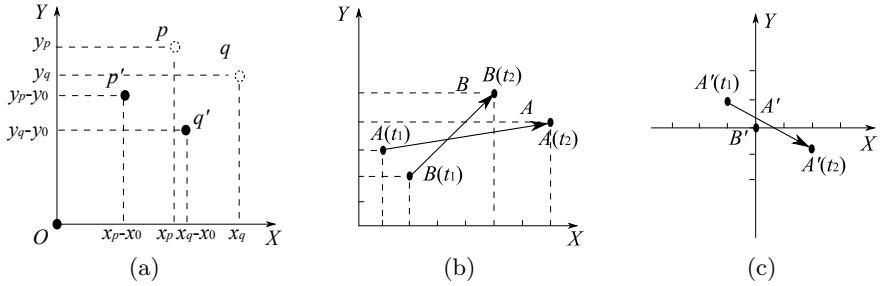

Fig. 4. A simple cardinal direction preserving mapping from p, q to p',q' (a); the cardinal direction preserving mapping from slice unit A, B (b) to A',B' (c).

$y_0^B + (y_1^A - y_1^B)t)$ and $M(f^B) = (0,0)$. Assume there exists a time t_0 $(t_1 \leq t_0 \leq t_2)$ such that $dir(f^A(t_0), f^B(t_0)) \neq dir(M(f^A)(t_0), M(f^B)(t_0))$. Let $x^B = x_0^B + x_1^B t_0$ and $y^B = y_0^B + y_1^B t_0$ denote two constant values. Since $M(f^A)(t_0) = (x_0^A - x_0^B + (x_1^A - x_1^B)t_0, y_0^A - y_0^B + (y_1^A - y_1^B)t_0)$ and $M(f^B)(t_0) = (0,0)$, we have $M(f^A)(t_0) = (X(f^A(t_0)) - x^B, Y(f^A(t_0)) - y^B)$ and $M(f^B)(t_0) = (X(f^B(t_0)) - x^B, Y(f^B(t_0)) - y^B)$. This matches the *cardinal direction preserving mapping* function $M(r) = (X(r) - x_0, Y(r) - y_0)$. Thus, the assumption $dir(f^A(t_0), f^B(t_0)) \neq dir(M(f^A)(t_0), M(f^B)(t_0))$ contradicts to Lemma 1. □

After applying the *cardinal direction preserving mapping* $M(f)$ to both unit functions f^A and f^B, we now obtain two new unit functions f'_a and f'_b as follows:

$$g^A(t) = M(f^A)(t) = (x_0^A - x_0^B + (x_1^A - x_1^B)t, y_0^A - y_0^B + (y_1^A - y_1^B)t)$$
$$g^B(t) = M(f^B)(t) = (0,0)$$

The unit function g^A describes a linear movement in the unit interval, while the unit function g^B describes a static point that holds its position during the entire unit interval. In other words, su^A is mapped to a new slice unit su_t^A which has a linear movement, and su^B is mapped to a new slice unit su_t^B that has no movement during the unit interval. Figure 4 shows an example of mapping the slice units A and B to slice units A' and B'. In this example, $A = [I, c^A]$ and $B = [I, c^B]$ where $I = [1,2]$, c^A and c^B contain the coefficients of the two unit functions f^A and f^B respectively, $f^A(t) = (-5 + 6t, 2 + t)$ and $f^B(t) = (-1 + 3t, -1 + 3t)$. Thus, $A(t_1) = f^A(1) = (1,3)$, $A(t_2) = f^A(2) = (7,4)$, $B(t_1) = f^B(1) = (2,2)$, and $B(t_2) = f^B(2) = (5,5)$. After applying the mapping, we obtain $g^A(t) = (-4 + 3t, 3 - 2t)$ and $g^B(t) = (0,0)$. Thus, $A'(t_1) = g^A(1) = (-1,1)$, $A'(t_2) = g^A(2) = (2,-1)$, and $B'(t_1) = B'(t_2) = (0,0)$.

So far, we have managed to reduce the problem of computing the cardinal directions between two moving slice units to the problem of computing the cardinal directions between one moving slice unit and one static slice unit. The second step is to compute the cardinal directions between su_t^A and su_t^B.

Since su_t^B is located constantly at $(0,0)$ during the time interval and since the trajectory of su_t^A is a linear function with respect to time t, we apply the

projection based approach (Section 3) to determining the cardinal directions. The idea is to take su_t^B as the reference point and to create partitions by using the x- and y-coordinate axes. Then we project the slice unit su_t^A to the xy-plane, and the cardinal directions are determined by the partitions that its trajectory intersects. Finally, the cardinal directions are ordered according to the time when they occurred and are stored into a list. For example, the cardinal directions between A' and B' in Figure 4b are NW, N, NE, E, and SE.

4.4 The Direction Composition Phase

Finally, in the direction composition phase, we iterate through all slice units, compose all cardinal directions that have been detected in slice units, and form a complete cardinal direction list in the temporal order. Further, we remove duplicates between consecutive cardinal directions.

We introduce the linear algorithm *compute_dir_dev* in Figure 3 for computing the final cardinal direction development (line 24) between two synchronized moving points. The input of the algorithm consists of two lists of slices $sl1$ and $sl2$ (line 1) that stem from the *time-synchronized interval refinement phase*. Since the two slice lists are guaranteed to have the same length, the algorithm takes a slice from each list (lines 3, 4, 10 and 11), determines the cardinal directions for each pair of slices (lines 5 and 16), which have the same unit interval, and traverses both lists in parallel (lines 7 and 8). For two consecutive pairs of slices, we have to check whether the slice intervals are adjacent (lines 9, 12, and 13). If this is not the case, we add the list with the single element $\perp$ to the global list *dev_list* in order to indicate that the cardinal direction development is undefined between two consecutive slice intervals (lines 13 to 15).

For each pair of slices, the function *compute_slice_dir* determines their cardinal directions according to the strategy discussed in Section 4.3 (lines 5 and 16). We maintain a list *slice_dir_list* to keep these newly computed cardinal directions from the current slice pair and compare its first cardinal direction with the last cardinal direction that has been computed from the last slice pair and is stored in the global list *dev_list* (lines 17 to 19). If both cardinal directions are the same, the first cardinal direction from the list *slice_dir_list* is removed in order to avoid duplicates (lines 19 to 21). The newly computed cardinal directions in the list *slice_dir_list* are added to the global list *dev_list* (lines 6 and 22).

The algorithm *compute_dir_dev* deploys a number of auxiliary list functions. The function *get_first_in_list* returns the first element in a list. The function *get_last_in_list* returns the last element in a list. The function *append* adds a list given as its second argument to the end of another list given as its first argument. The function *remove_first* removes the first element from a list.

Now we analyze the complexity of the algorithm for function *compute_dir_dev*. Assume that the first moving point mp_1 consists of m slices, and the second moving point mp_2 consists of n slices. The inputs of the function *compute_dir_dev* are two lists of slices generated from the time-synchronized interval refinement phase, thus each list contains at most $m + n$ slices. The function *compute_dir_dev* iterate through all slices in both

list and compose the cardinal directions computed. So the time complexity is $O(m+n)$.

5 Defining Spatial-temporal Direction Predicates within Databases

In this section, we discuss how cardinal direction developments can be integrated into spatio-temporal databases and query languages. This requires the formal definition of cardinal direction developments as binary predicates since it will make the query processing easier when using pre-defined predicates as selection conditions. In the following part, we define some important predicates which will be sufficient for most queries on cardinal direction developments between moving objects.

First of all, we give the definition of *existential direction predicates*. This type of predicates finds out whether a specific cardinal direction existed during the evolution of moving objects. For example, a query like "Find all ships that appeared north of ship Fantasy" belongs to this category. It requires a predicate named *exists_north* as a selection condition of a join. This predicate can be defined as follows,

Definition 1. *Given two moving points $A, B \in MPoints$, their cardinal direction development $DEV(A,B) = d_1 \triangleright d_2 \triangleright \ldots \triangleright d_n$ with $n \in \mathbb{N}$ and $d_i \in CD$ or $d_i = \perp$ for all $1 \leq i \leq n$. Then we define the* existential direction predicate exists_north *as*

$$exists\_north(A,B) = true \overset{\text{def}}{\Leftrightarrow} \exists 1 \leq i \leq n : d_i = N$$

Definition 1 indicates that the predicate *exists_north* is true if the direction *north* exists in the sequence of the cardinal direction development. It can help us define the above query. Assume that we have the following relation schema for ships

```
ships(id:integer, name:string, route:mpoint)
```

The query can be expressed using an SQL-like query language as follows:

```
SELECT s1.name FROM ships s1, ships s2
WHERE  s2.name = 'Fantasy' AND exists_north(s1.route, s2.route);
```

The other existential cardinal direction predicates *exists_south*, *exists_east*, *exists_west*, *exists_sameposition*, *exists_northeast*, *exists_southeast*, *exists_northwest*, and *exists_southwest* are defined in a similar way.

Another important category of predicates expresses that one moving object keeps the same direction with respect to another moving object. For example, assume that there is a group of ships traveling from north to south and each ship follows the ship in front of the group. Now the leader of the group wants to know which ships belong to the group. The problem is to find out which ships are keeping a northern position with respect to the leading ship.

Definition 2. *Given two moving points* $A, B \in MPoints$. *The predicate* keeps_north *is defined as*

$$\begin{aligned} keeps\_north(A,B) = {} & exists\_north(A,B) && \wedge \neg exists\_south(A,B) \\ & \wedge \neg exists\_southeast(A,B) && \wedge \neg exists\_east(A,B) \\ & \wedge \neg exists\_sameposition(A,B) && \wedge \neg exists\_northwest(A,B) \\ & \wedge \neg exists\_northeast(A,B) && \wedge \neg exists\_southwest(A,B) \\ & \wedge \neg exists\_west(A,B) \end{aligned}$$

Definition 2 shows that the relationship *keeps_north* between two moving objects implies that the only existential direction predicate in the cardinal direction development of these moving objects is *exists_north* without any other existential direction predicates. In other words, we have $DEV(A,B) = N$.

We consider the above example and assume that the identifier of the leader ship is 1001. Then the query "Find all ships keeping a position north of the leader ship 1001" can be expressed as

```
SELECT s1.id FROM ships s1, ships s2
WHERE  s2.id = '1001' AND keeps_north(s1.route, s2.route);
```

The other predicates that express that one moving object remains in the same direction with respect to another moving object are *keeps_south*, *keeps_east*, *keeps_west*, *keeps_sameposition*, *keeps_northeast*, *keeps_southeast*, *keeps_northwest*, and *keeps_southwest*.

Another useful predicate checks for the transition between two cardinal directions in a cardinal direction development. The transition can be either a direct change or an indirect change through a set of intermediate directions. We name this predicate as *from_to*. For example, the query "Find all ships that have traveled from the south to the north of the ship Fantasy" can be answered by using this predicate.

Definition 3. *Given two moving points* $A, B \in MPoints$, *their cardinal direction development* $\mathrm{DEV}(A,B) = d_1 \triangleright d_2 \triangleright \ldots \triangleright d_n$ *such that* $d_i \in CD$ *or* $d_i = \perp$ *for all* $1 \leq i \leq n$, *and two cardinal directions* $d', d'' \in \mathrm{CD}$. *We define the predicate* from_to *as follows:*

$$\begin{aligned} from\_to(A,B,d',d'') = true \overset{\text{def}}{\Leftrightarrow} {} & d' \neq \perp \ \wedge \ d'' \neq \perp \wedge \\ & \exists\, 1 \leq i < j \leq n : d_i = d' \ \wedge \ d_j = d'' \end{aligned}$$

We formulate the above query as follows:

```
SELECT s1.id FROM ships s1, ships s2
WHERE  s2.name = 'Fantasy' AND
       from_to(s1.route, s2.route, 'S', 'N');
```

Finally, we define the predicate *cross_north* which checks whether a moving point traverses a large extent of the region in the north of another moving point.

Definition 4. *Given two moving points* $A, B \in MPoints$ *and their cardinal direction development* $\mathrm{DEV}(A, B) = d_1 \triangleright d_2 \triangleright \ldots \triangleright d_n$ *such that* $d_i \in CD$ *or* $d_i = \perp$ *for all* $1 \leq i \leq n$. *We define the predicate* crosses_north *as follows:*

$$\begin{aligned} \mathit{crosses\_north}(A, B) = \mathit{true} \overset{\text{def}}{\Leftrightarrow} n \geq 3 \ \wedge \ \exists 2 \leq i \leq n-1 : \\ (d_{i-1} = \mathrm{NW} \ \wedge \ d_i = N \ \wedge \ d_{i+1} = \mathrm{NE}) \vee \\ (d_{i-1} = \mathrm{NE} \ \wedge \ d_i = N \ \wedge \ d_{i+1} = \mathrm{NW}) \end{aligned}$$

The query "Find all the ships that have crossed the north of ship Fantasy" can be expressed as follows:

```
SELECT s1.id FROM ships s1, ships s2
WHERE  s2.name = 'Fantasy' AND crosses_north(s1.route, s2.route);
```

The other predicates *cross_south*, *cross_east*, and *cross_west* can be defined in a similar way.

6 Case Study: Cardinal Direction Development in Hurricane Research

In this section, we apply our strategy to a real world application, and show how the evaluation of cardinal direction development can help with the hurricane research.

We have integrated the directional predicates into a moving object database (MOD) developed for the NASA workforce. The moving object database is a full-fledged database with additional support for spatial and spatiotemporal data in its data model and query language. It maintains tropical cyclone and hurricane data provided by public sources, and the weather data derived from the NASA mission sensor measurements. It also provides functionality in terms of spatiotemporal operations and predicates that can be deployed by decision makers and scientists in ad-hoc queries. By enabling the capability of evaluating cardinal direction developments among hurricanes, the scientists can have a better understanding of dynamic patterns on weather events. We establish our experiments on the historical hurricane data collected from National Hurricane Center (NHC). The original data is available on the web site of NHC [15]. The sensors collect six data points per day for a specific hurricane, i.e., at 00:00, 06:00, 12:00 and 18:00 UTC time. The data collected are the hurricane locations in terms of longitudes and latitudes, time, and other thematic data like wind speed and category. We load these data points into moving point types, and represent the trajectory of each hurricane as a moving point in MOD. In this paper, we present a case study on all hurricanes in year 2005 on the Atlantic Ocean. The following table is created in the database:

```
test_moving(id:integer, name:string, track:mpoint)
```

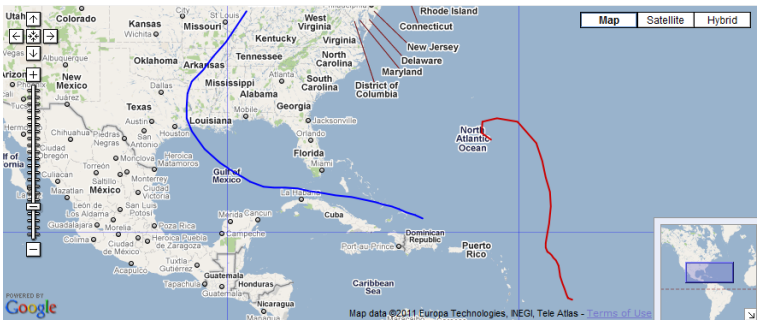

Fig. 5. The trajectories of hurricanes PHILIPPE and RITA.

In the schema *test_moving*, *name* is the attribute that stores hurricane names and *track* is a moving point type attribute that stores the trajectory of hurricanes. A total of 28 hurricanes that have been active on the Atlantic Ocean in the year 2005 are loaded in the data table. Due to the space limit, we evaluate the following two types of directional queries: the cardinal direction development query and the top-k query.

First, consider the query: "Find the cardinal direction development between PHILIPPE and RITA.", we can post the following SQL query:

```
SELECT m1.name, m2.name, mdir(m1.track, m2.track),
FROM   test_moving m1, test_moving m2
WHERE  m1.name = 'PHILIPPE' AND m2.name = 'RITA';
```

The function *mdir* is a user defined function registered at the database end that computes the cardinal direction developments between two moving points. A string representation is returned as the result. In this case, we obtain the following result:

```
NAME     NAME  MDIR(M1.TRACK,M2.TRACK)
-------- ----- ---------------------------------
PHILIPPE RITA  ->undefined[2005091712,2005091800)
               ->NW[2005091800,2005092212)
               ->W[2005092212,2005092212]
               ->SW(2005092212,2005092402)
               ->W[2005092402,2005092402]
               ->NW(2005092402,2005092406)
               ->undefined[2005092406,2005092606)
```

The result is a list of timely ordered cardinal directions. In the time interval [2005-09-17 12:00:00,2005-09-18 00:00:00), RITA is not evolved yet, thus the cardinal direction is *undefined*. When RITA is "born", it starts from the northwest of PHILIPPE, moves to the north of PHILIPPE. Then it crosses the west of PHILIPPE and moves to the southwest of PHILIPPE on date 2005-09-22.

In the following two days, it moves back to the northwest of PHILIPPE. The visualization of the two hurricane is shown in Figure 5. The result shows an interesting movement pattern between the two hurricanes, which may suggest the hurricane researchers to investigate the correlations in terms of wind speed, air pressure, and ocean currents during a certain time interval between the two hurricanes.

Another type of query that is intersecting to hurricane researchers is the top-k query. Here, the top-k evaluates the lasting time of cardinal directions between two hurricanes. Thus, given two hurricanes, we are able to find the top-k cardinal directions between them. Let us consider the query: "find top 2 cardinal directions between MARIA with other hurricane tracks". We can formulate the SQL query as follows:

```
SELECT m1.name, m2.name, topKDir(m1.track,m2.track,3)
FROM   test_moving m1, test_moving m2
WHERE  m1.name='MARIA' AND m1.name<>m2.name
AND    topKDir(m1.track,m2.track,2) <> ' '
```

The function $topKDir(m1.track, m2.track, 2)$ returns the top 2 cardinal directions (excluding the undefined direction) between two moving points that last the longest, and it returns empty string if there does not exist defined cardinal directions between them. We get the following result:

```
NAME   NAME     TOPKDIR(M1.TRACK,M2.TRACK,2)
------ -------- --------------------------
MARIA  LEE      NW NE
MARIA  NATE     SW
MARIA  OPHELIA  SW
```

The result shows that the top two cardinal directions lasting the longest between MARIA and LEE are *NW* and *NE*. NATE and OPHELIA are always to the *SW* of MARIA. From this result, we can observe that during the life time of MARIA, two hurricanes spent most of their time moving in the southwest of MARIA and one hurricane spent most of its time in the northwest of MARIA. No hurricanes exists in the other directions like *SE* or *NE* of MARIA. This observation may raise the intersects of hurricane researchers to investigate the causes and the facts that lead to the pattern, or to make conclusions from this pattern.

7 Conclusions and Future Work

In this paper, we present a three-phase solution for computing the cardinal directions between two moving points from an algorithmic perspective. We show the mapping of cardinal direction developments between moving points into spatio-temporal directional predicates and the integration of these predicates into the spatio-temporal query language of a moving objects database. We present a case

study on the hurricane data to show a real world application for the cardinal direction development. In the future, we will implement a comprehensive set of predicates for querying cardinal direction development. We will also extend our concept to more complex moving objects like moving regions and moving lines.

References

1. Theodoridis, Y., Sellis, T.K., Papadopoulos, A., Manolopoulos, Y.: Specifications for Efficient Indexing in Spatiotemporal Databases. In: 10th Int. Conf. on Scientific and Statistical Database Management (SSDBM). (1998) 123–132
2. Grumbach, S., Rigaux, P., Segoufin, L.: Spatio-temporal Data Handling with Constraints. GeoInformatica (2001) 95–115
3. Erwig, M., Güting, R.H., Schneider, M., Vazirgiannis, M.: Spatio-temporal Data Types: an Approach To Modeling and Querying Moving Objects in Databases. GeoInformatica **3**(3) (1999) 269–296
4. Forlizzi, L., Guting, R., Nardelli, E., Schneider, M.: A Data Model and Data Structures for Moving Objects Databases. In: ACM SIGMOD Int. Conf. on Management of Data. (2000) 319–330
5. Erwig, M., Schneider, M.: Spatio-temporal Predicates. IEEE Trans. on Knowledge and Data Engineering (TKDE) **14**(4) (2002) 881–901
6. Su, J., Xu, H., Ibarra, O.H.: Moving Objects: Logical Relationships and Queries. In: 7th Int. Symp. on Spatial and Temporal Databases (SSTD). (2001) 3–19
7. de Weghe, N.V., Bogaert, P., Delafontaine, M., Temmerman, L.D., Neutens, T., Maeyer, P.D., Witlox, F.: How To Handle Incomplete Knowledge Concerning Moving Objects. In: Behaviour Monitoring and Interpretation. (2007) 91–101
8. Frank, A.: Qualitative Spatial Reasoning: Cardinal Directions As an Example. International Journal of Geographical Information Science **10**(3) (1996) 269–290
9. Skiadopoulos, S., Koubarakis, M.: Composing Cardinal Direction Relations. Artificial Intelligence **152** (2004) 143–171
10. Haar, R.: Computational Models of Spatial Relations. Technical Report: TR-478 (MSC-72-03610) (1976)
11. Skiadopoulos, S., Sarkas, N., Sellis, T., Koubarakis, M.: A Family of Directional Relation Models for Extended Objects. IEEE Trans. on Knowledge and Data Engineering (TKDE) **19** (2007)
12. Papadias, D., Theodoridis, Y., Sellis, T.: The Retrieval of Direction Relations Using R-trees. In: Int. Conf. on Database and Expert Systems Applications (DEXA). (1994) 173–182
13. Li, J.Z., Ozsu, M.T., Tamer, M., Szafron, D., Ddi, S.G.: Modeling of Moving Objects in a Video Database. In: IEEE International Conference on Multimedia Computing and Systems. (1997) 336–343
14. Chen, T., Liu, H., Schneider, M.: Evaluation of Cardinal Direction Developments between Moving Points. In: ACM Symp. on Geographic Information Systems (ACM GIS). (2010) 430–433
15. : NHC Archive of Hurricane Seasons. `http://www.nhc.noaa.gov/pastall.shtml`