Using a Sequential Index in Terrain-Aided Navigation

Ling Lin
Department of Computer Science
Linkoping University, Sweden
linli@ida.liu.se
http://www.ida.liu.se/~linli

Tore Risch
Department of Computer Science
Linkoping University, Sweden
torri@ida.liu.se
http://www.ida.liu.se/~torri

Abstract

Terrain-aided navigation is a database application in which an aeroplane locates itself by matching the height trajectory with a terrain-elevation map on board the aircraft. Without an index a matching algorithm has to be applied to the whole map. This paper shows that an indexing technique, termed the IP-index, can efficiently filter out sub-areas of the map to provide starting positions for the matching algorithm (to improve the efficiency). Performance measurements of the IP-index in terrain-aided navigation are given in this paper. The IP-index is a dynamic indexing technique for large 1-D sequences on the value domain. It supports interpolation assumptions on the sequences, i.e., it can be used to query implicit values in addition to explicit values. It can be implemented by regular ordered indexes such as B-trees.

Keywords: sequences, indexing, interpolation, query processing, terrain-aided navigation.

1 Introduction

Terrain-aided navigation is to use the terrain height over the mean sea level, the terrain elevation, to draw conclusions about the position of an aircraft. The idea is: A map with sampled terrain elevation measured in a uniformly spaced grid is stored on board the aircraft. Flying over an area, the aircraft altitude over mean sea-level is measured with a barometric sensor and the ground clearance is measured with a radar. The difference between the altitude and the ground clearance is an estimate of the terrain elevation, which can be compared to the stored values in the map to find out the position of the aircraft, see Fig. 1.1.

Gathering samples as the aircraft flies over an area will give a trajectory of measured elevations. The more samples gathered the more likely it is that the trajectory is unique and that a good position estimate may be found when comparing the trajectory with the stored elevations in the map.

Errors and uncertainty exist in the measurements of the barometric sensor and radar. For example, flying through different local weather conditions will cause the barometric sensor to produce biased errors. Thus, the measured terrain elevation is an approximation of the *real* terrain elevation. In order to locate the position of the aircraft in the map, a non-linear matching algorithm which takes into account the probability density function of the measured elevation has been developed by [6]. This matching algorithm is applied to every grid of the map by processing each new measured elevation in the trajectory recursively.

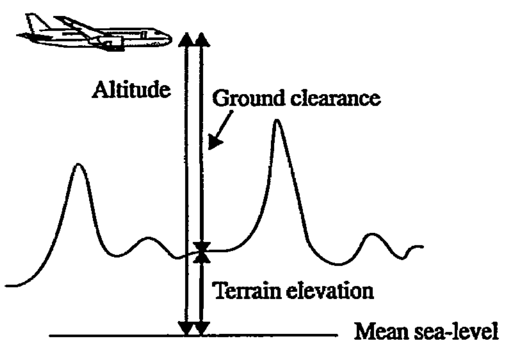

Fig. 1.1: Illustration of the terrain-aided navigation

Since the matching algorithm is expensive and has to be applied to every grid of the map, it becomes very inefficient to implement when the number of grids in the map becomes large (see [6]). Therefore, we propose an index that can filter out those sub-areas whose terrain elevations are in some range around the measured values *h*. We use a confidence interval (*h*-10, *h*+10) where the probability that the true elevation value is inside this interval is higher than 99.99% according to the probability density function. Then, the sub-areas whose terrain elevations are inside the interval (*h*-10, *h*+10) can be used as starting positions of the above matching algorithm instead of the whole map.

In short, the problem is: given a map which stores sampled terrain elevation in each grid, how can we *efficiently* find out those sub-areas whose terrain elevations are inside interval (*h'*, *h"*)? One may argue that a regular ordered secondary index can return all the grids that have terrain elevation inside the interval (*h'*, *h"*). But to group the set of grids returned into sub-areas is not a trivial task (for detailed explanation see Section 5). In this paper we propose using a novel indexing technique, *IP-index* [11], which can return the sub-areas of the map much more efficiently than a regular ordered secondary index.

The IP-index is a dynamic indexing technique for large 1-D sequences. It aims at supporting interpolation assumptions on sequences, i.e., it can be used to query implicit values in addition to explicit values. It supports range queries as efficiently as exact queries. It can be implemented by any regular ordered indexes such as B-trees.

This paper is organized as following: Section 2 explains why the IP-index is needed in real-life applications and discusses related work. Section 3 gives an overview of the indexing technique. Section 4 shows how to process different kinds of queries on data sequences by using the IP-index. Section 5 shows how the IP-index can be applied to terrain-aided navigation and improve the efficiency of the matching algorithm. Section 6 gives the performance measurements. Conclusions and future work are given in Section 7.

2 Related Work

Sequential data brings new challenges to DBMSs. Examples of sequential data include: 1) time sequences such as temperature reading generated by sensors in scientific measurements; 2) stock price index in business applications; 3) medical data such as a patient's temperature reading or cardiology data, 4) multispectral satellite image data. The terrain elevation map in the navigation application (see Section 1) is a special kind of sequential data where each row of the map is a sequence of terrain elevations. Sequential data require special storage and accessing techniques that do not appear in conventional database systems. For example, large amount of research has been dedicated to similarity search for sequences, i.e., finding those (sub-)sequences that match a given pattern with some error distance [1][3][4][13][15]. The applications can be found in identifying companies with similar growth patterns, products with similar selling patterns, stocks with similar price movement, images with similar weather patterns, etc.

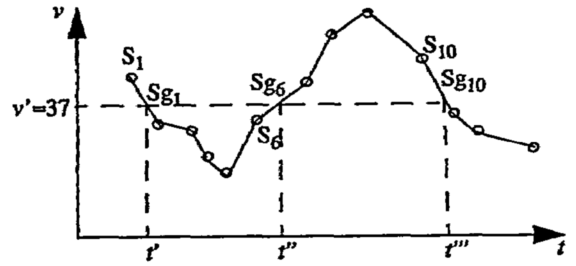

Fig. 2.1: Illustration of a value query

However, we found out that some applications require searches for *values* instead of the shape of sequences. For example, if the time sequence in Fig. 2.1 represents a patient's temperature reading over time, the physician will be interested in the query like "At what time period did the patient have a temperature higher than 37°C (i.e., have a fever)?". Generally, these kind of queries appear as "when was the value equal to (or above, below) some threshold v'?", or "when was the value in the range (v', v'')?". We term these kind of queries *value queries* in contrast to *shape queries*. It is not trivial to answer value queries since: 1) In most applications, there are some kind of interpolation functions associated with the discrete time sequence. In other words, a time sequence is regarded as a general function that produces values for any implicit time points in addition to those explicit points. When queries concern the implicit values, a conventional index does not help. 2) For range value queries on a time sequence, it is not efficient to use a conventional index either. We will explain this more in Section 5.

In order to deal with value queries that involve interpolation assumptions on time sequences, we developed a new indexing technique termed the IP-index [11]. The intuition behind the IP-index is illustrated in Fig. 2.1. The original time sequence is viewed as a sequence of states S_i. By connecting each neighbour states we get segments $Sg_i=[S_i, S_{i+1}]$. If we can find all the segments Sg_i that intersect the line $v=v'$ (i.e., the segments Sg_1, Sg_6, Sg_{10} in Fig. 2.1), then we can calculate the time points when the value was equal to v' (i.e., t', t'' and t''' in Fig. 2.1). This is because the time point t' can be computed by applying the interpolation function on the neighbour states around Sg_1. Likewise, the time point t'' can be computed by applying the interpolation function on the neighbour states around Sg_6, and the time point t''' can be computed by applying the interpolation function on the neighbour states around Sg_{10}.

Thus, a value query for a time sequence is transformed into an interval intersecting problem. There have been several indexing methods proposed for k-dimensional spatial search, e.g. k-d trees[14], R-trees[9] and SR-Tree[10]. There are also some index trees proposed in computational geometry to deal with interval problems, e.g., Interval Trees[7], and Segment Trees[5]. However, none of the above methods are suitable for value queries on time sequences. The reasons are: 1) Time sequences consist of large sets of intervals $[S_i, S_{i+1}]$ which are often dynamically growing, while most spatial data structures assume a fixed search space. 2) The intervals in time sequences have a special property that the end point of Sg_i is the starting point of Sg_{i+1} (i.e. S_{i+1}). This property makes the IP-index much more simple compared to spatial search trees such as R-trees.

Although the IP-index [11] was intended to deal with value queries for time sequences, it turned out to work well for any 1-D data sequences. For example, in this paper the IP-index will be applied to the terrain elevation map to find out the sub-areas in the map whose terrain elevations are inside the interval (h', h''). The only assumption the IP-index made is that the data points in the sequence are ordered, which is true for all sequential data.

3 The IP-index

This section illustrates the idea of the IP-index. Although we use time sequences in the illustration, the idea is applicable to any 1-D data sequences (as long as the sequence is ordered).

A time sequence (TS) is viewed as a sequence of states, where each state is $S_i=(t_i, v_i)$. If we project each value v_i on the v-axis, we get non-overlapping intervals $I_j=[k_j, k_{j+1})$, where each k_j is a distinct value of v_i (see $k_1...k_4$ in Fig. 3.1). We can see that all values that belong to one interval have the same sequence of intersecting segments (marked to the left in Fig. 3.1). In the IP-index we associate with each interval [k_j,

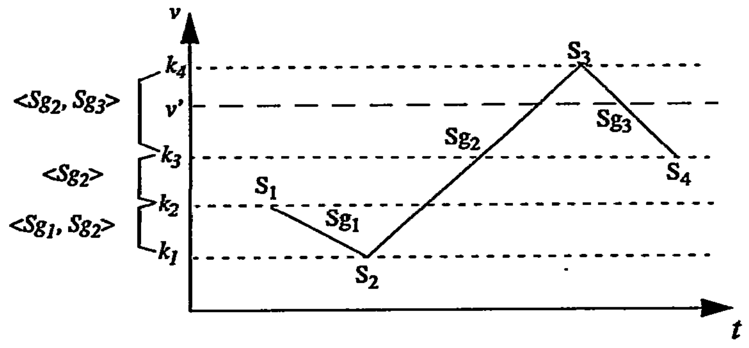

Fig. 3.1: Illustration of the IP-index

k_{j+1}) all segments Sg_i that span[1] it. A simple illustration of the IP-index is shown in Fig. 3.1, where we associate each interval $[k_j, k_{j+1})$ with the sequence of spanning segments Sg_i.

Since each segment Sg_i is uniquely identified by its starting state S_i, we use S_i to represent the segment Sg_i in the IP-index. We term the starting states of each segments that intersect the line $v=v'$ as the *anchor-states* of v'. Then, the sequence of intersecting segments can be represented as the sequence of anchor-states, which is termed the *anchor-state sequence*. The anchor-state sequence is a state sequence ordered by time.

Since each interval $[k_j, k_{j+1})$ is uniquely identified by its starting point k_j, we use k_j to represent the interval $[k_j, k_{j+1})$ in the IP-index.

Suppose that $k_1<k_2<...<k_j<...$ are the ordered distinct values of v_i in the *TS*. Then each index entry N_i in the IP-index has the form [*key, anchors*] where

- $N_i.key=k_j$.
- $N_i.anchors$ is the anchor-state sequence for all v' such that $v' \geq k_j$ and $v'<k_{j+1}$. It is also denoted as *anchors*(k_j).

For example, the IP-index for the simple *TS* in Fig. 3.1 is:

anchors$(k_1)=<S_1, S_2>$
anchors$(k_2)=<S_2>$
anchors$(k_3)=<S_2, S_3>$
anchors$(k_4)=nil$

To verify our ideas we implemented the IP-index in a main-memory database [8] using an AVL-tree [1]. The performance measurements [11] show that the IP-index dramatically improves the performance of value queries. For periodic sequences that have a limited range and precision on their value domain (most application sequences have this property), the IP-index has an upper bound for insertion and search time.

4 Processing Queries

To show how the IP-index can be used in terrain-aided navigation (to find out those sub-areas whose terrain elevations are inside the interval (h', h'')), let us start from a simple example. Suppose that a time sequence represents a patient's temperature reading in a hospital. There are several kinds of value queries which can be answered given the IP-index.

1. When did the patient have the temperature equal to 37°C?
   We term this kind of query as an *exact query*, expressed as $F^{-1}(v')$.
2. During what time period did the patient have the temperature higher than 37°C?
   We term this kind of query as an *inequality query*, expressed as $F^{-1}(v>v')$ (or $F^{-1}(v<v')$).
3. When did the patient have a temperature *around* 37°C?
   We term this kind of query as an *interval query*, expressed as $F^{-1}(v'-e<v<v'+e)$.

We will see that exact queries can be computed *directly* by using an IP-index. Inequality queries can be computed given that we can compute exact queries. Interval queries can be computed given that we can compute inequality queries. In Section 5 we will show that by posing interval queries to the terrain elevation map we can get all sub-areas whose terrain elevations are inside the interval (h', h'').

4.1 Exact Query

This section shows how to use the IP-index to calculate $F^{-1}(v')$. Suppose that *ifn* is the interpolation function for the time sequence and *ip-index* is the IP-index built for the time sequence. Then, as illustrated in Fig. 2.1, we need to find all anchor-states S_i for v', then apply the interpolation function on the states around S_i to calculate the time points t'. These anchor-states are stored in *ip-index*. Below we give notations which are used in this section:

- N_i — an index entry in the IP-index.
- $N_i.key$ — the key of the index entry N_i.
- $N_i.anchors$ — the anchor-state sequence associated with the index entry N_i.

To calculate an exact value query $F^{-1}(v')$, we search the IP-index to find the index entry N_L where

$$N_L.key = max\{N_i.key \mid N_i.key \leq v_i\}$$

Then $N_L.anchors$ contains the anchor-state sequence for the

1. We say a segment Sg_i spans an interval I_i when the projection of Sg_i on the v-axis spans the interval I_i, i.e. if $Sg_i=((t_s, v_s), (t_e, v_e))$ and $I_i=(v_a, v_b)$, then $v_s \leq v_a$ and $v_e \geq v_b$.

value v'. For example, in Fig. 3.1, the index entry N_L for the value v' is [k_3, *anchors*] where *anchors*=<S_2, S_3>. The reason is that k_3 (=v_4) is the first key which is "below" or equal to v'.

Therefore, given the IP-index and the interpolation assumption *ifn* for a time sequence, the pseudo-code for computing an exact query $F^{-1}(v')$ will look like the following:

```
exact_query(ip-index, v', ifn):
  $F^{-1}(v')$=nil          /* initialize the result */
  find the index entry $N_L$ in ip-index where
      $N_L$.key=max{$N_i$.key | $N_i$.key≤v'}            (1)
          /* find the index entry which stores the anchor-state
                 sequence for v' */
  For each state $S_i$ in $N_L$.anchors                  (2)
      $F^{-1}(v')$=$F^{-1}(v')$+
              ifn$^{-1}$(v', surrounding_states($S_i$))
  end for each
  return $F^{-1}(v')$
```

Figure 4.1: The algorithm of $F^{-1}(v')$

Note that there are two steps marked (1) and (2) in this algorithm. Step (1) searches the index to find the anchor-state sequence, step (2) applies the inverse interpolation function ifn^{-1} on each anchor-states to get $F^{-1}(v')$. The function *surrounding_states*(S_i) is dependent on the interpolation function *ifn*. For example, if *ifn* is linear interpolation, then *surrounding_states*(S_i)={S_i, S_{i+1}}.

4.2 Inequality Query

Given an example time sequence TS=<S_1, S_2,...S_8> as illustrated in Fig. 4.2, according to the algorithm in Fig. 4.2, we have $F^{-1}(v')$=<t'_1, t'_2, t'_3, t'_4>. It can be seen in Fig. 4.2 that:

- $F^{-1}(v>v')$=<(t'_1, t'_2), (t'_3, t'_4)>
- $F^{-1}(v<v')$=<(t_s, t'_1), (t'_2, t'_3), (t'_4, t_e)>

where t_s is the first time point of TS (i.e., S_1.*time*) and t_e is the last time point of TS (i.e., S_8.*time*).

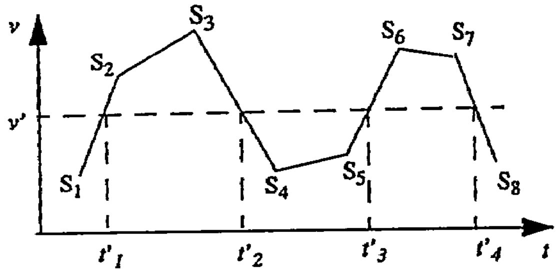

Figure 4.2: Illustration of an inequality query

The observations are: 1) $F^{-1}(v>v')$ (or $F^{-1}(v<v')$) is a sequence of time intervals. 2) Each interval of $F^{-1}(v>v')$ (or $F^{-1}(v<v')$) is composed only by those time points returned by $F^{-1}(v')$ (plus t_s and t_e). Now let us see how to compose the time points of $F^{-1}(v')$ into corresponding time intervals of $F^{-1}(v>v')$ (or $F^{-1}(v<v')$).

In step (2) of the algorithm of $F^{-1}(v')$ in Fig. 4.1, the inverse interpolation function is applied to each anchor-state S_i to get t'. At the same time we can check the "direction" of t', i.e.,

- *direction*(t')='+' if S_{i+1}.*value*>S_i.*value*
- *direction*(t')='-' if S_{i+1}.*value*<S_i.*value*

This is illustrated in Fig. 4.3. Notice that we do not store segments with S_{i+1}.*value*=S_i.*value* in the IP-index since we only record intersecting (non-horizontal) segments Sg_i.

It can be seen from Fig. 4.2 that:

- $F^{-1}(v>v')$=(t'_i, t'_{i+1})* where *direction*(t_i)='+'
- $F^{-1}(v<v')$=(t'_i, t'_{i+1})* where *direction*(t_i)='-'

(The time intervals concerning t_s and t_e have to be treated specially by comparing S_1.*value* and S_8.*value* with v'.)

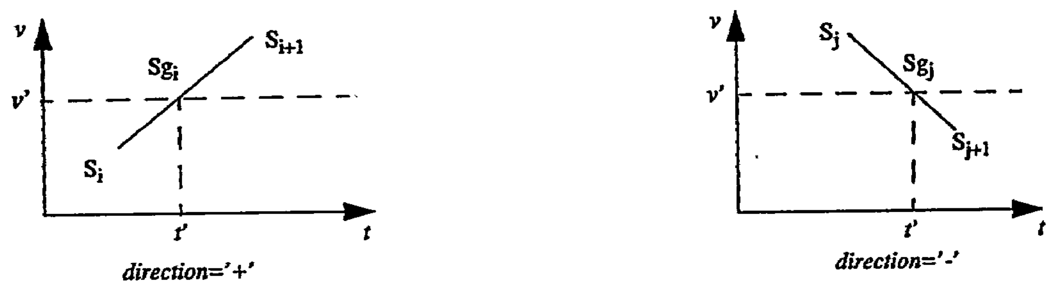

Figure 4.3: Direction of the interpolated time points

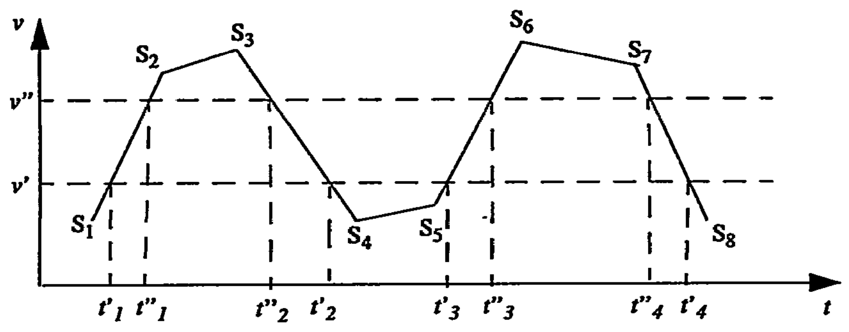

Figure 4.4: Illustration of an interval query

In some applications (e.g., the terrain-aided navigation) it is desirable to return the "state intervals" instead of the time intervals of $F^{-1}(v>v')$ (or $F^{-1}(v<v')$). For example, in Fig. 4.4 the state intervals of $F^{-1}(v>v')$ are $[S_2, S_3]$ and $[S_6, S_7]$. This is trivial given that we can calculate $F^{-1}(v>v')$ (or $F^{-1}(v<v')$). These "state intervals" can be found by rounding[1] the time intervals. For example, rounding (t'_1, t'_2) in Fig. 4.4 results in $[S_2, S_3]$.

4.3 Interval Query

Interval queries can be calculated given that we can calculate inequality queries. This is because $F^{-1}(v'<v<v'')=F^{-1}(v>v') \cap F^{-1}(v<v'')$ where '$\cap$' means "interval intersection". For example, in Fig. 4.4, we have $F^{-1}(v>v')=<(t'_1, t'_2), (t'_3, t'_4)>$, $F^{-1}(v<v'')=<(t_s, t''_1), (t''_2, t''_3), (t''_4, t_e)>$. There we see that $F^{-1}(v'<v<v'')=<(t'_1, t''_1), (t''_2, t'_2), (t'_3, t''_3), (t''_4, t'_4)>$, which is the interval intersection of $F^{-1}(v>v')$ and $F^{-1}(v<v'')$.

So, to calculate $F^{-1}(v'<v<v'')$, we do the following:

1. Calculate $F^{-1}(v>v')$.
2. Calculate $F^{-1}(v<v'')$.
3. Apply interval intersection to the results returned from 1 and 2.

We will show in next section that by posing interval queries on the terrain elevation map we can get all sub-areas whose terrain elevations are inside an interval (h', h'').

5 Using the IP-index in Terrain-aided Navigation

This section shows how the IP-index can improve the efficiency of the matching algorithm in terrain-aided navigation. First we show how the IP-index can be applied to the terrain elevation map to filter out those sub-areas whose terrain elevations are inside the interval (h', h'').

Since the IP-index is designed for 1-D sequences, it cannot be directly applied to the two-dimensional map. The approach is: we view each row of the map as a time sequence (see Fig. 5.1). That is, each grid corresponds to a state in a time sequence, the position identifier ij corresponds to the timestamp t, and the elevation h_{ij} corresponds to the value v. For each time sequence, we pose the interval query $F^{-1}(h'<v<h'')$ to get the position intervals (ij_pos', ij_pos'') where the values inside these intervals are inside the range (h', h''). As mentioned in Section 4.2, by rounding these intervals we can get corresponding column intervals $[ij', ij'']$. These column intervals are the sub-areas in this row whose terrain elevations are inside the interval (h', h'').

Now let us look at how the IP-index can improve the efficiency of the matching algorithm in the terrain-aided navigation. As we mentioned in the introduction, the matching algorithm finds the true position of the aircraft by applying a non-linear algorithm to the whole map recursively for each measured elevation in the trajectory. Taking a trajectory of elevation measurements $h_1, h_2, ...h_k$, we can use the IP-index to find the sub-areas in the map whose terrain elevation are inside the interval $(h_1\text{-}10, h_1\text{+}10)$ (this interval is based on the probability density function as explained in the introduction), thus providing starting positions for the matching algorithm. Since the matching algorithm does not have to be applied to the whole map, efficiency is improved.

We define the *cardinality* of an interval $(h_i\text{-}10, h_i\text{+}10)$ as the number of grids in the map whose terrain elevation are inside this interval. In the measurements we found out that for a trajectory $h_1, h_2, ...h_k$, the cardinalities of $(h_i\text{-}10, h_i\text{+}10)$ vary very much (see Fig. 6.2). Since the matching algorithm does not really have to start from the first measurement h_1, i.e., it can start from any (sub-areas returned from) the interval $(h_i\text{-}10, h_i\text{+}10)$ and apply the non-linear algorithm backwards $(h_{i-1}, h_{i-2}, ...h_1)$ and forwards $(h_{i+1}, h_{i+2}, ...h_k)$, we do not always have to take the first interval $(h_1\text{-}10, h_1\text{+}10)$ to return the sub-areas. Instead we take an interval $(h_i\text{-}10, h_i\text{+}10)$ in the trajectory which returns a small cardinality (this will be further illustrated in the next section). The matching algorithm starts from these small areas (and applies the non-linear algorithm backwards and forwards in the trajectory) will be much more efficient than starting from the whole map.

The cardinality of the interval $(h_i\text{-}10, h_i\text{+}10)$ can be computed easily from the IP-index by adding a new field *cardinality* in each index entry. That is, an index entry (see Section 3) is

1. Here "rounding" means finding the largest state interval that is inside the time interval.

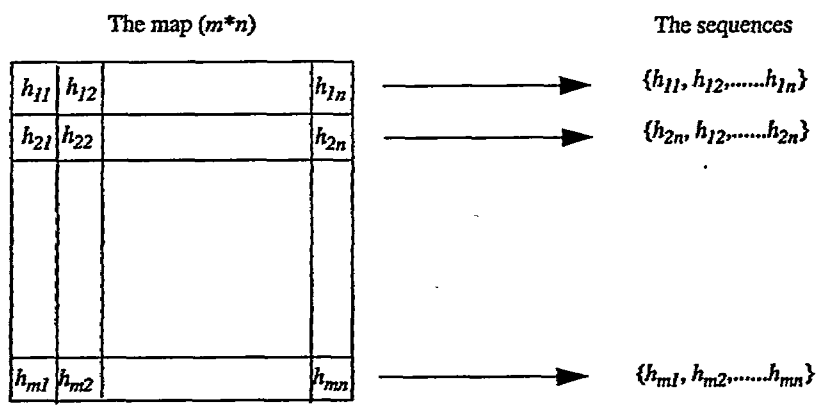
The map ($m*n$)
The sequences
h_{11}
h_{12}
h_{1n}
h_{21}
h_{22}
h_{2n}
h_{m1}
h_{m2}
h_{mn}
$\{h_{11}, h_{12}, \ldots\ldots h_{1n}\}$
$\{h_{2n}, h_{12}, \ldots\ldots h_{2n}\}$
$\{h_{m1}, h_{m2}, \ldots\ldots h_{mn}\}$

Fig. 5.1: Transformation between the map and the data sequences

modified to [*key, anchors, cardinality*] where

- $N_i.key=k_j$.
- $N_i.anchors$ is the anchor-state sequence for all v' such that $v' \geq k_j$ and $v' < k_{j+1}$. It is also denoted as $anchors(k_j)$.
- $N_i.cardinality$ is the number of states in the sequence whose values are equal to $N_i.key$ (k_j). It is also denoted as $cardinality(k_j)$.

Then, the IP-index insertion algorithm [11] should also be slightly modified (for this application) to increment $N_i.cardinality$ by 1 whenever a new state arrives whose value is equal to $N_i.key$ (k_j).

Then, the cardinality of the interval (h'-10, h''+10) is computed by adding together $cardinality(k_j)$ where k_j are inside the interval (h'-10, h''+10). This is efficient since the IP-index is an ordered index.

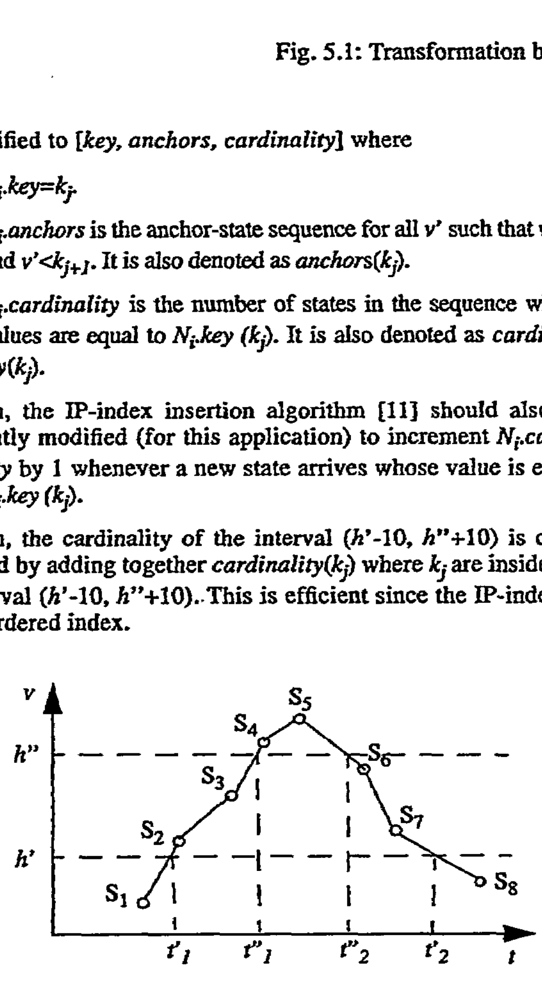

Fig. 5.2: Comparing to a conventional ordered secondary index

Before we show the measurement results, we would like to point out that a conventional ordered secondary index is not sufficient to return the sub-areas whose terrain elevations are inside the interval (h'-10, h''+10) (though it seems like so). Suppose that Fig. 5.2 represents a row of the terrain-elevation map. Using an IP-index we get $F^{-1}(h'<v<h'')=\{(t'_1, t''_1), (t''_2, t'_2)\}$, by rounding these intervals we get the state sequences (sub-sequences) $\{[S_2, S_3], [S_6, S_7]\}$. By contrast, a conventional ordered secondary index will return a set of states S_2, S_7, S_3, S_6 which are ordered by their *values* instead of by their order in the sequence. To group these states S_2, S_7, S_3, S_6 into sub-sequences ($\{[S_2, S_3], [S_6, S_7]\}$) is *not* a trivial task, especially when the number of sub-sequences returned is large (In Fig. 5.2 there are only two sub-sequences returned) or the number of states inside a sub-sequence is large. Another disadvantage of a conventional secondary index is that it has to search for all keys that are between the interval (h', h'') in order to find all states whose values are inside the interval (h', h''). This indicates that when the interval is large (this is the case with the navigation application), the IP-index will perform stably while a conventional secondary index will deteriorate.

6 Measurements

In this section we measure how much the IP-index can improve the efficiency of the matching algorithm in the terrain-aided navigation. To make the measurements as close to reality as possible, we use a real map over a part of Sweden (see Fig. 6.1) which consists of *101 by 101 samples in a uniformly spaced grid*. (It is sampled with 50 m distance between each sample point, yielding an area of 25 square kilometres of terrain.) The elevation sample for each grid in the map is not the average value over the grid but the measured terrain elevation rather exactly in the center of the grid. Some interpolation method (e.g., linear interpolation) could easily be applied to the map to produce any terrain elevation between the sampled points. As seen in Fig. 6.1 the terrain elevation has different characteristics in different parts of the map. The flat area in the upper left corner of the map is a lake.

50 elevation track files were randomly generated in order to cover different parts of the map. The starting positions of these tracks were uniformly distributed along the leftmost line of the map. Likewise their final positions were uniformly distributed along the parallel finish line on the other side of the grid. Each track file represents a trajectory of an aircraft and it contains 80 sampled terrain elevation measurements.

For a trajectory $h_1, h_2, \ldots\ldots h_k$, the cardinality of each interval (h_i-10, h_i+10) can vary very much, as shown in Fig. 6.2. As

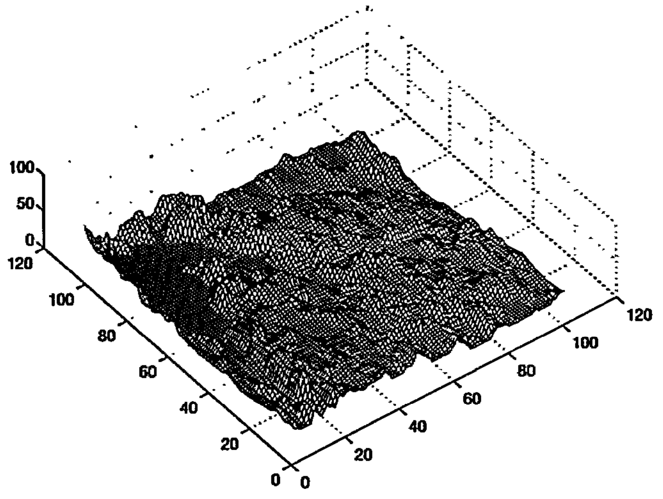

Fig. 6.1: The real map

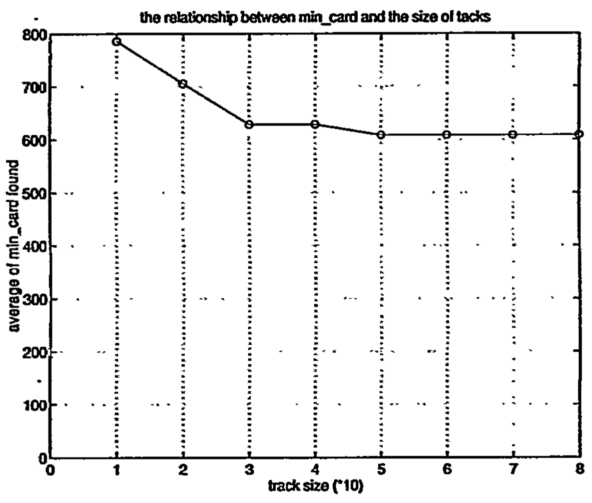

Fig. 6.3: The relationship between the number of samples taken and the average of the minimum cardinality found

we mentioned in Section 5, we would like to take an interval (h_i-10, h_i+10) in the trajectory which returns a rather small cardinality. The more samples (in a trajectory) we take, the higher the probability will be to find a small cardinality. To measure the relationship between the minimum cardinality found and the size of the tracks, we calculate the cardinalities (using the IP-index as mentioned in Section 5) for the first i*10 (i=1...8) samples of the 50 tracks. We recorded the minimum cardinality found and the corresponding size of the tracks. Thus, for each track we get a sequence of minimum cardinality min_i (i=1...8) and the corresponding $size_i$=i*10 (i=1...8). Fig. 6.3 shows the $avg(min_i)$ over the 50 tracks. It shows that: 1) The minimum cardinality decreases with the size of the tracks; 2) After approximate 30 samples, the average of the minimum cardinality reaches a stable value (i.e., a value around 609).

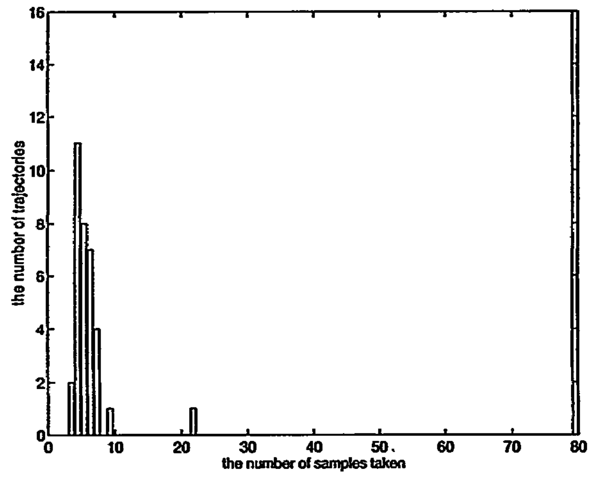

Fig. 6.4: The histogram of the settling time of the IP-index

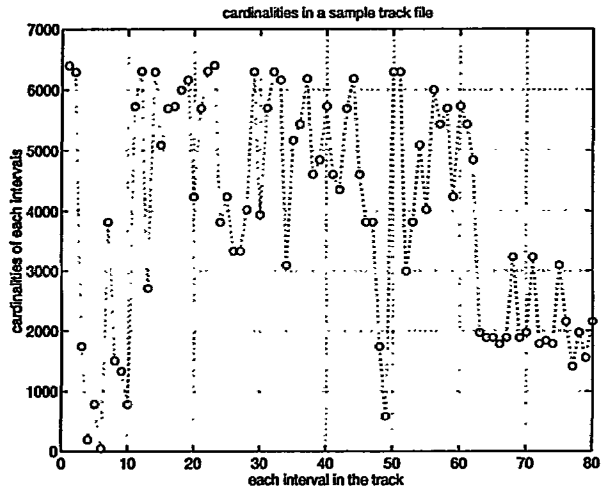

Fig. 6.2: Cardinality distribution for a sample track

Thus we can use the value 609 as the "converge" threshold for the IP-index, i.e., we say the IP-index has settled when it finds a cardinality less than 609. Taken a trajectory of elevation measurements h_1, h_2,...h_k, it is interesting to see how fast the IP-index settles. We tested the IP-index on the 50 tracks and record the number of samples needed to reach the converge threshold 609. The histogram is shown in Fig. 6.4.

Fig. 6.4 shows that the settling time of the IP-index is generally small. For most tracks less than 10 samples are needed. Notice that when the track does not cover areas with small cardinalities (for example, when the aircraft is flying over flat areas such as lakes), the cardinality threshold cannot be reached even if the whole track is checked. That is the reason why some tracks are located in the rightmost line (where the settling time is 80) in Fig. 6.4.

Notice that the measurement results are affected by the application data. For example, the value of the "converge" threshold (609) used in these measurements are dependent on the terrain-variation map and the position of the tracks (in the map). It should be tuned for different application data set.

This value in turn affects the settling time of the IP-index.

Nevertheless, these measurements show how the IP-index should be used in the navigation application to find good starting positions for the matching algorithm. That is: for any track file we calculate the cardinality for each sample h_i, and stop either when the first cardinality is found which is lower than the threshold (i.e. 609 in the example), or when the IP-index reaches its "converge threshold" (i.e. 30 samples in the example since the minimum cardinality will not continue to decrease based on statistics shown in Fig. 6.3). Suppose we stop at the sample h_s in the trajectory, then we pose the range query $F^{-1}(h_s-10<v<h_s+10)$ to find the sub-areas in the map whose terrain elevations are inside the interval (h_s-10, h_s+10). These sub-areas are returned to the matching algorithm to serve as the starting positions (recall in Section 5 that the matching algorithm can work backwards and forwards). Since the number of grids inside these areas are guaranteed to be small, the matching algorithm will always be *much* more efficient than starting from the whole map.

7 Conclusions and Future Work

Many applications require value queries concerning implicit interpolation assumptions on data sequences. For this purpose we developed a novel indexing technique termed the IP-index [11], which supports implicit value queries including exact queries $F^{-1}(v')$ and range queries $F^{-1}(v'<v<v'')$. In this paper we show how the IP-index can be used in terrain-aided navigation and dramatically improve the efficiency of the matching algorithm.

Terrain-aided navigation requires efficient filtering out of sub-areas in a map whose terrain elevations are inside an interval (h', h''). A conventional ordered secondary index is not efficient in returning these sub-areas. By contrast, if we build an IP-indexes for each row of the map, we can get the sub-areas efficiently by posing a range query $F^{-1}(h'<v<h'')$ to each row. Also, by associating cardinality information in the IP-index we can efficiently find those sub-areas with small cardinality to serve as starting positions of the matching algorithm.

In future work we plan to extend the IP-index to index on two-dimensional sequences. In that case we can apply the IP-index on the whole map directly without having to build an IP-index for each row. Also we would like to implement the IP-index for huge maps that can only be stored in disks and be able to cash part of the index (for a sub-area of the map) to main-memory when required by the application.

Acknowledgements

The authors would like to thank Niclas Bergman for cooperation in the terrain-aided navigation and providing relevant materials.

References

1. G. M. Adelson-Velskii and E. M. Landis. *Doklady Akademia Nauk SSSR*, 146, (1962), pp. 263-266; English translation in *Soviet Math*, 3, pp. 1259-1263.

2. R. Agrawal, C. Faloutsos and A. Swami, "Efficient Similarity Search in Sequence Databases". In *Proc. of the Fourth International Conference on Foundations of Data Organization and Algorithms*, pp. 69-84, Chicago, Oct. 1993.

3. R. Agrawal, K. Lin, H. S. Sawhney, K. Shim, "Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases". In *Proc. VLDB, Conf.*, pp. 490-501, 1995.

4. R. Agrawal, G. Psaila, D. L. Wimmers and M. Zaït, "Querying Shapes of Histories". In *Proc. 21st VLDB Conf.*, pp. 502-514, 1995.

5. J. L. Bently, *"Algorithms for Klee's Rectangle Problems"*, Computer Science Department, Carnegie-Mellon University, Pittsburgh, 1972.

6. N. Bergman, *"A Bayesian Approach to Terrain-Aided Navigation"*, Technical Report, LiTH-ISY-R-1903, Linköping University, Oct. 1996.

7. H. Edelsbrunner, *"Dynamic Rectangle Intersection Searching"*, Institute for Information Processing, Rept. 47, Technical University of Graz, Graz, Austria.

8. G. Fahl, T. Risch and M. Sköld. An architecture for Active Mediators. In *Proc. Intl. Workshop on Next Generation Information Technologies and Systems*, pp. 47-53, Haifa, Israel, 1993.

9. A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching". In *Proc. ACM SIGMOD Conf.*, June 1984, Boston, MA.

10. C. P. Kolovson and M. Stonebraker, "Segment Indexes: Dynamic Indexing Techniques for Multi-Dimensional Interval Data". In *Proc. ACM SIGMOD Conf.*, pp. 138-148, 1991.

11. L. Lin, T. Risch, M. Sköld, D. Badal, "Indexing Values of Time Sequences", in *Proceedings of 5th International Conference on Information and Knowledge Management*, pp. 223-232, Rockville, Maryland, Nov. 1996.

12. L. Lin, "A Value-Based Indexing Technique For Time Sequences", Lic. Thesis No 597, Linköping University, Jan., 1997, ISBN 91-7871-888-0.

13. C. S. Li, P. S. Yu and V. Castelli. HierachyScan, "A Hierachical Similarity Search Algorithm for Databases of Long Sequences". In *Proc. Data Engineering Conf.*, pp. 546-553, Feb. 1996.

14. K. Ooi, B. McDonell and R. Sacks-Davis, "Spatial kd-tree: Indexing mechanism for spatial database". In *IEEE COMPSAC 87*, 1987.

15. H. Shatkay, S. B. Zdonik, "Approximate Queries and Representations for Large Data Sequences". In *Proc. Data Engineering Conf.*, pp.536-545, Feb. 1996.