A Flexible Spatio-temporal Indexing Scheme for Large-scale GPS Track Retrieval

Longhao Wang, Yu Zheng, Xing Xie, Wei-Ying Ma
Microsoft Research Asia, 4F, Sigma Building, NO.49 Zhichun Road, Beijing 100080, China
{v-lowang, yuzheng, xingx, wyma}@microsoft.com

Abstract

The increasing popularity of GPS device has boosted many Web applications where people can upload, browse and exchange their GPS tracks. In these applications, spatial or temporal search function could provide an effective way for users to retrieve specific GPS tracks they are interested in. However, existing spatial-temporal index for trajectory data has not exploited the characteristic of user behavior in these online GPS track sharing applications. In most cases, when sharing a GPS track, people are more likely to upload GPS data of the near past than the distant past. Thus, the interval between the end time of a GPS track and the time it is uploaded, if viewed as a random variable, has a skewed distribution. In this paper, we first propose a probabilistic model to simulate user behavior of uploading GPS tracks onto an online sharing application. Then we propose a flexible spatio-temporal index scheme, referred to as Compressed Start-End Tree (CSE-tree), for large-scale GPS track retrieval. The CSE-tree combines the advantages of B+ Tree and dynamic array, and maintains different index structure for data with different update frequency. Experiments using synthetic data show that CSE-tree outperforms other schemes in requiring less index size and less update cost while keeping satisfactory retrieval performance.

1. Introduction

In recent years, with decreasing price and increasing locality accuracy, GPS devices have become more and more prevalent in modern life. Hence, as never before, lots of GPS track data, e.g. users' GPS tracks, have been accumulated both continuously and unobtrusively.

The large amounts of GPS data have given rise to a generation of novel applications on the Web. One major trend is online GPS track sharing applications which allows users to upload, share and browse GPS track and related multimedia content [2, 3, 4, 5, 6, 7] over Web maps. Compared to traditional text-based description, visualizing users' GPS tracks over Web maps can provide a more explicit, concise and fancy approach to express their life experience. Consequently, users can recall their own past events better, and obtain more information from other people's experience when they browse a GPS track.

As more and more GPS tracks are accumulated, how to manage and index these GPS data become an important issue in these Web applications. Meanwhile, users also need an efficient approach to retrieve the specific GPS tracks they are interested in. However, existing search method by tags, like activity tags and region tags [2, 3, 7], offered by these Web applications cannot satisfy users' needs. For instance, a user in New York City may want to find a relaxing walking trail near his house in an ordinary weekend. In this scenario, search by region tags like NYC would be too large for the user, and tags are also insufficient to express the temporal query of the user. Thus, spatio-temporal search function, which allows users to retrieve the trajectories within a given spatial range and temporal interval, would provide an effective way of managing GPS data and improving user experience.

Although some spatio-temporal indexing schemes for trajectory data have been proposed in the past years, they are not optimized for GPS track sharing applications on the Web. We observe that users tend to upload GPS tracks of the near past more frequently than GPS tracks of the distant past. For instance, users are more likely to upload GPS tracks of today than those of months ago. This observation also holds in other data sharing applications such as Flickr and YouTube. People are more likely to upload the images/videos they took recently than those of long time ago. Thus, traditional spatio-temporal indexing schemes, like R tree or its variants, are not optimal to handle the skewed nature of accumulative GPS tracks.

In this paper, we propose Compressed Start-End Tree (CSE-tree) for the GPS data sharing applications based on users' uploading behavior. In this scheme, we first partition the space into disjoint cells that cover the whole spatial region, and then maintain a flexible temporal index for each spatial cell. To insert a new GPS track, we divide

the track into segments according to spatial partition. Then each segment is inserted into the temporal index of corresponding spatial grid. For all segments in a temporal index, they are divided into several groups according to end time of the segment. We observed that for different groups, the frequency of new updates is different. CSE-tree uses B+ tree index for frequently updated groups and sorted dynamic array for rarely updated ones. The update frequency of a group may change as time goes by, so we transform a B+ tree index into a sorted dynamic array if update frequency of a group drops below a threshold. In all, the contributions of our work lie in that:

- A stochastic process model is proposed to simulate user behavior of uploading GPS tracks to online sharing applications. This model can also be applied to other data sharing applications on the Web.
- A novel indexing scheme is optimized to the user behavior of uploading GPS tracks. Our scheme requires less index space and less update cost while keeping satisfactory retrieval performance.

The rest of this paper is organized as follows: We make a survey of related work in Section 2, and introduce the preliminaries of this paper, including application, term definition and query type in Section 3. Then we present our model for user behavior of uploading GPS tracks in Section 4 and our indexing scheme is proposed in Section 5. After presenting the experiment result in Section 6, we conclude our paper in Section 7.

2. Related work

Many online GPS track sharing applications already exist on the Web. "GPS exchange forum" [4] provides a public space where users can exchange plain GPS track files by posting literal descriptions of the original creator and other people's comments. "GPS sharing" [5] improves the user interface by displaying the uploaded GPS track on a digital map and marking the positions where the user has taken an image. It also computes basic statistics of the track, including length, date and ascent. "Mountain Bike" [2] not only presents tracks and images on map, but also allows users to add their own tags to describe the track, such as "hilly", "muddy", and "relaxing". Users can search tracks by tag names. Tag-based search function is also provided in "Wikiwalki" [7], and it focuses on "country" and "activity". People are fond of these novel applications for good reasons. For his/her personal use, an individual can better recall and enjoy his/her past events. For public use, it is more convenient to share other people's life experience. For example, when they are planning an outdoor activity like jogging and cycling, they would like to find out whether a certain track is suitable for them. One who wants to relax would avoid a track labeled "tiresome" or "challenging", while these tracks may be attractive for those who want to exercise more. Another use is to discover other users with similar interests of travelling and form online communities, as what [7] is trying to do. However, none of these applications offers spatial and temporal search functions that allow users to specify his/her query range over maps.

Another work that is related to ours is indexing trajectory data (such as GPS track). Such indices largely fall into two categories: indices that optimize queries about the future positions of spatiotemporal objects, and those that optimize historical queries. The later is more related to ours. For index of historical queries, there are mainly three approaches. One straightforward way of indexing historical trajectory data is by using any multidimensional access method like R-tree and its variants [9, 10, 11, 12, 13]. But these approaches do not take advantage of the special properties of time dimension, thus introduces excessive overlapping between index nodes and therefore leads to decreased query performance. The second approach uses overlapping and multi-version structures, such as MR-tree [14], HR-tree [15] and MV3R-tree [16]. This approach builds a separate R-tree for each time stamp and shares common parts between two consecutive R-trees. Though this approach allows efficient temporal query, it also requires excessive storage. The third approach divides the spatial dimension into grids, and then builds separate temporal index for each grid. This category includes SETI [17], SEB-tree [18] and MTSB-tree [19]. Since this approach has been proved to have better insertion and query performance [17, 18, 19], the index proposed in this paper also use spatial partition method.

For temporal index of spatial partition method, SETI uses R-tree in each grid. MTSB-tree uses TSB-tree which assumes historical data will not be updated, thus unsuitable for our scenario. SEB-tree represents track segment within a spatial grid by its start and end time stamp, thus it can index these segments as points in a two dimensional plane. Our CSE-tree differs from SEB-tree in two aspects. First, we divide track into groups using a different criterion. Second, we employ different index strategies for data with different update frequency.

3. Preliminary

This section first introduces an application scenario in which spatio-temporal search functions is leveraged to improve state-of-the-art GPS track sharing applications. Then we formulate the GPS track search problem with definitions of several key terms and query types used throughout this paper.

3.1. Application

The work described in this paper is a part of research of our project GeoLife [1], which focuses on visualization,

organization, fast retrieval and effectively mining of GPS track data for both personal and public use. We have implemented an online GPS track sharing platform on which users can upload GPS data, browse their own information, search and share others' GPS tracks. Figure 1 shows the user interface of spatio-temporal search function in GeoLife. The CSE-tree proposed in this paper has been deployed in our platform to support efficient online insertion and query of GPS tracks. In Figure 1(a), users can specify temporal query range by inputting start time and end time. In Figure 1(b), users can specify spatial query by clicking the map to input two diagonal points of a rectangle.

We have also performed a user study to testify the effectiveness of spatio-temporal search function in improving user experience. Fifty users carrying GPS device over a period of six months have uploaded 3,236 GPS tracks to our system. These GPS tracks are further leveraged to model user behavior in Section 4 and generate synthetic data in Section 6.

(a) Temporal search (b) Spatial search

Figure 1. Spatio-temporal search of GeoLife

3.2. Definition

Figure 2 gives an illustration of a typical GPS track. Each track consists of a sequence of track entries. A track entry can be formulated as <latitude, longitude, time stamp>. Connecting these entries sequentially forms a track, as depicted in the right part of Figure 2.

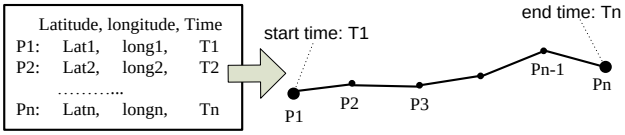


Figure 2. GPS track and corresponding track

Let T_s denote the start time of the track, and T_e denote the end time of the track. The duration of a GPS track (T_{dur}) is the time interval between the end time (T_e) and the start time (T_s) of the track. Put formally,

$$T_{dur} = T_e - T_s \qquad (1)$$

Since the user cannot upload a track onto the Web immediately after it ends, there must a time interval between T_e and the time it is uploaded onto a website (T_{up}). This interval is defined as T_{int}. Put formally,

$$T_{int} = T_{up} - T_e \qquad (2)$$

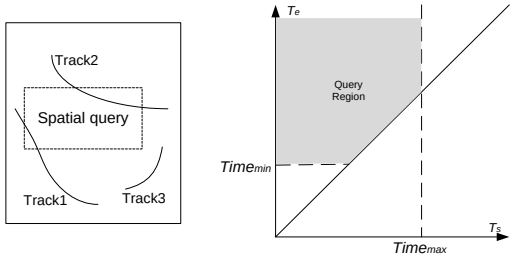
3.3. Query

A spatio-temporal query should include both spatial and temporal predicate. The spatial predicate specifies a rectangular region represented by its four boundaries. The temporal predicate specifies a time span between the minimum and the maximum temporal boundary of the query, symbolized as $Time_{min}$ and $Time_{max}$.

For spatial predicate, the index should retrieval all tracks that overlap with the query's spatial input. In Figure 3(a), spatial query is to retrieve all tracks that intersect with spatial query input, such as *Track1* and *Track2*. For temporal predicate, the desired result is to retrieve all GPS tracks that satisfy:

$$\begin{cases} T_e > Time_{min} \\ T_s < Time_{max} \end{cases} \qquad (3)$$

On a two-dimensional plane where the horizontal axis denotes Ts and vertical axis denotes Te, a track can be represented as a point on the plane. Therefore, the temporal query specified by $< Time_{min}, Time_{max} >$ is to retrieve all points that fall in the grey "Query Region" part of the plane in Figure 3(b).



(a) Spatial query (b) Temporal query

Figure 3. Spatial and temporal query

4. Modeling user behavior

In order to model user behavior of uploading GPS tracks, we need to investigate the distribution of the following three random variables: T_{up}, T_{int} and T_{dur}. Then we propose a model to simulate user behavior.

4.1. Uploading process

Similar to the webpage requests to a server, uploading GPS tracks to an online application can be viewed as random events in time that occur independently of one another. Thus according to queuing theory, these events can be modeled as a Poisson process, i.e. for intensity λ, the number of uploaded GPS tracks in time interval (t, t+ τ] follows Poisson distribution with parameter λτ:

$$P\big[\big(N(t+\tau) - N(t)\big) = k\big] = \frac{e^{-\lambda\tau}(\lambda\tau)^k}{k!}$$
$$k = 0,1,2 \ldots \qquad (4)$$

4.2. Distribution of T_{dur}

The duration of GPS track T_{dur} can be viewed as a random variable. Figure 4 shows the distribution of duration of tracks collected in our user study.

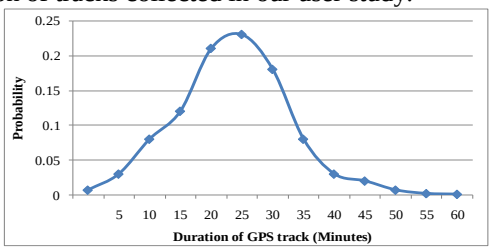


Figure 4. Track Duration Distribution

Intuitively, people would neither move very far nor very near outdoors before they stop. From the data collected in the user study mentioned in Section 3.1, we assume that the distribution of T_{dur} follows normal distribution. Put formally,

$$T_{dur} \sim N(\mu, \sigma^2) \qquad (5)$$

4.3. Distribution of T_{int}

If viewing Tint as a random variable, we observe that it has a skewed distribution. The reason is as follows. First, when sharing a GPS track, people are more likely to upload GPS data of the near past than the distant past. For instance, users are more likely to upload a GPS track of today than that of months ago. Second, people do need some time to transfer the track data from GPS device to a website. The distribution of T_{int}, therefore, should be a combination of the two aforementioned effects.

Although we cannot perform statistical analysis on the large-scale GPS tracks from existing online applications directly, as shown in Figure 5, the distribution of Tint can be inferred from our analysis on users' upload behavior on Flickr. The data is summarized from 57,243 images which were uploaded by different users from July 1st, 2007 to November 8th, 2007. In Figure 5, the solid line represents our analysis of Flickr data. The horizontal axis denotes the time span between the images' taken time and uploaded time.

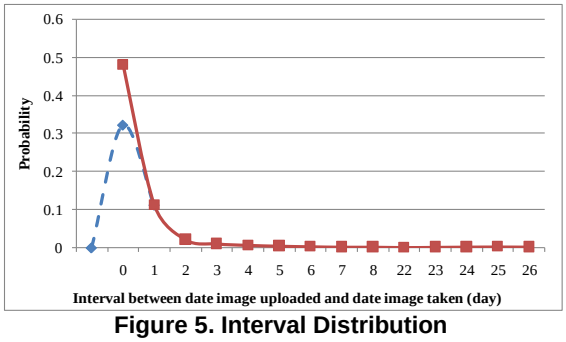


Figure 5. Interval Distribution

However, since the smallest time unit maintained by Flickr is day, the probability of different hours within a day cannot be inferred from statistical evidence. Intuitively, if we look into the hours in the first day, there must be a time gap between the end time of GPS track and the time it is uploaded onto an online sharing application. Thus we append the broken line showing the distribution of probability in the first day. We assume that users' motivation of uploading GPS tracks varies in a way similar to radio signal transmitting in propagation environment. i.e., Tint follows Rayleigh distribution.

$$T_{int} \sim Rayleigh(\lambda) \qquad (6)$$

4.4. Generating Synthetic data

Given the information above, user behavior of uploading GPS tracks can be simulated by five steps:

1. Generate a series of T_{up} using Poisson process.
2. Generate T_{int} by Rayleigh distribution, and deduce T_e by using the transform of equation (1).
$$T_e = T_{up} - T_{int}$$
3. Generate T_{dur} by normal distribution.
4. Generate T_s by using the transform of equation (2)
$$T_s = T_e - T_{dur}$$
5. Using T_s and T_e as the first and last time stamp, we can generate all GPS points of the track with GSTD method [20].

5. Index design

5.1 Overall Structure

Similar to [17, 19], we divide the space into disjoint cells that cover the whole spatial region, and then maintain a temporal index for each spatial cell, as illustrated in Figure 6. The whole spatial region can be partitioned into uniform cells using grid based indexing method or uneven grid using quad-tree indexing method.

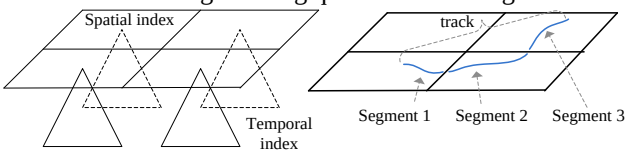


Figure 6. Overall Structure and Track Segmentation

Each uploaded GPS track is assigned a unique trackID when it enters the system. In order to insert a track into the index, we first partition the track into segments by spatial grids. Then each segment is inserted into the temporal index of corresponding spatial grid.

To execute a query, first we retrieve the spatial grids that overlap with query's spatial predicate, and a candidate cell list is produced. Then for each cell in candidate cell

list, its local temporal index is searched to find tracks that intersect the temporal predicate and trackIDs are returned. Finally, trackIDs from different cells are merged to remove duplicates and we get the final result.

While the overall insertion and query operations are similar to previous index structures [17, 18], the major difference lies in temporal index, which is named "Compressed Start End Tree" (CSE-tree). Further detail of local temporal index is explicated in the sections below.

5.2. CSE-tree

CSE-tree is essentially a two-dimensional index structure for temporal index of each spatial grid. After a track is partitioned into segments by spatial grids, each segment is inserted into corresponding temporal index, i.e., CSE-tree of the spatial grid. In a two-dimensional plane where the horizontal coordinate denotes Ts and the vertical coordinate denotes Te, a track segment can be represented by a point on the plane.

The overall structure of a CSE-tree is illustrated in Figure 7. Since T_e must be greater than the T_s, all points presenting a segment fall in the upper left part of the plane. All points are divided into several groups by T_e. For each group, a *Start Time Index* is built to index points within the group. For all groups, one *End Time Index* is built to index different groups.

The *End Time Index* is implemented as a B+ tree for indexing <*key, value*> pair. The *key* is the numerical value of the dividing lines of different groups, like t_0, t_1, t_2, t_3, …, t_i and t_{i+1}. The *value* is a pointer to *Start Time Index* like S_0, S_1 and S_i. The partition criterion is determined in insertion process. For an index entry in *End Time Index* whose key is t_i, it should cover all track segments whose T_e is between [t_i, t_{i+1}) in the spatial grid and the entry's value is the pointer to *Start Time Index* S_i.

The *Start Time Index* is also organized as a B+ tree structure indexing <*key, value*> pair. The *key* is T_s of the segment and *value* is < T_e, trackID> pair of the segment.

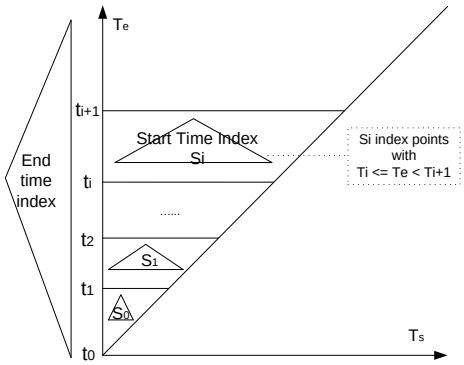
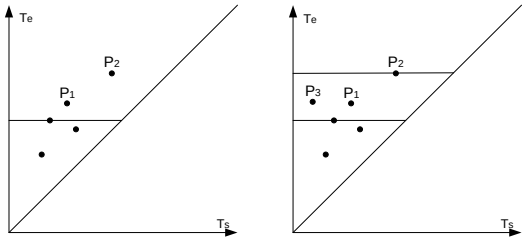


Figure 7. CSE-tree structure

Three operations including insertion, search, and compression are designed for CSE-tree. The insertion operation is performed to add new GPS track segments into CSE-tree while the search operation is implemented to handle temporal queries on CSE-tree. The compression operation is based on our observation that insertion into the index is not distributed evenly, as we have discussed in Section 4.3. Therefore, we can use different index structures for the frequently updated and rarely updated part of data. The compression operation is called when the part of data's update frequency drops below a threshold to make the conversion between different index structures.

5.2.1. Insertion One important parameter in insertion process is *Partition Threshold T*. A dividing line in end time dimension is drawn as soon as the number of items in the last *Start Time Index* reaches *T*. Upon the insertion of a new track segment, CSE-tree first looks up the *End Time Index* with T_e to find the largest key smaller than T_e. The corresponding value of the <*key, value*> pair is the pointer to a *Start Time Index*. Then the *Start Time Index* is updated using key T_s and value < T_e, *trackID*>. Finally the last *Start Time Index* is checked to see if the number of items in this index exceeds *Partition Threshold T*. If so, a routine is called to create a new *Start Time Index* and insert a <*key, value*> pair into *End Time Index* where *key* is the maximum T_{end} in the last *Start Time Index* and *value* is the pointer to the newly created *Start Time Index.*

Figure 8 gives a brief illustration of the insertion process. Assume that *Partition Threshold T* equals three. When the first two points (P_1, P_2) in the last division is inserted, no partition line is drawn (Figure 8(a)). When the third point comes (P_3), after it is inserted into corresponding *Start Time Index*, the item number in the last *Start Time Index* reaches *Partition Threshold T*, so a new dividing line is drawn and a new *Start Time Index* is created. The numerical value of the dividing line equals the maximum value of T_e in the last *Start Time Index* (P_2's T_e). We then insert *End Time Index* with a <*key, value*> pair whose *key* is the value of P_2's T_e and *value* is pointer to newly created *Start Time Index.* (Figure 8(b))



(a) Before inserting P_3 (b) After inserting P_3

Figure 8. An example of CSE-tree insertion

The insertion algorithm can be further optimized by an analysis of distribution of insertion frequency. In Figure 9, due to the Rayleigh distribution of T_{int}, the portion of data whose end time is near "now" is updated more frequently. Therefore, we can first check up the last few *Start Time Indexes* like S_{j+1} and S_j. If the new insertion can be inserted into the last few *Start Time Indexes*, then we can avoid the cost of searching the entire *End Time Index*.

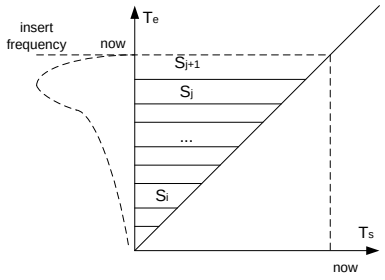


Figure 9. CSE-tree insertion frequency

5.2.2 Search In a CSE-tree, in order to find all GPS tracks that intersect with the temporal predicate of a query, we need to perform the three steps. First, we look up the *End Time Index* with $Time_{min}$ and we can get *Start Time Indexes* that contain tracks whose end time is greater than $Time_{min}$. In Figure 10, Given query $<Time_{min}, Time_{max}>$, the first step should return *Tree1*, *Tree2* and *Tree3*. Second, we use $Time_{max}$ to look up each *Start Time Index* to find the tracks whose start time is less than $Time_{max}$. Finally, we filter out undesired tracks in the *Start Time Indexes* that intersect with $Time_{min}$. In Figure 10, *Tree1* contains some tracks whose T_e is less than $Time_{min}$, and they are filtered out before returning results to the users.

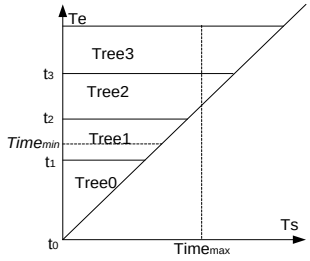


Figure 10. CSE-tree search operation

5.2.3 CSE-tree compression In the previous discussion, we assume that *Start Time Index* and *End Time Index* are implemented using B+ tree. However, as data accumulate, the tree structure would grow larger and larger. Although B+ tree performs well for frequent updates, this advantage is quite unnecessary for archive data. According to the analysis of user behavior in Section 4, though uploading a GPS track a long time ago is possible, its frequency is low and higher insertion cost for these archive data would not affect overall performance significantly. Hence the index for archive data only needs to guarantee speedy response to query, but not insertion. Therefore, the B+ tree index can be replaced by a simple dynamic array structure for archive data that is uploaded long time ago and the index size can be greatly reduced. An illustration is given in the Figure 9. The current time "now" is shown as a broken line that is constantly moving as time elapses. S_i, S_j and S_{j+1} represent three *Start Time Indexes*. S_j and S_{j+1} have a higher frequency of being updated and S_i have a lower frequency. Therefore, S_j and S_{j+1} may be organized using B+ trees while S_i uses a sorted dynamic array.

The operation that transforms a B+ tree into a dynamic array is defined as "compression". It is implemented by reading all data from B+ tree leaves and creating a new sorted dynamic array containing all these data. Since this compression operation is time-consuming, it is carried out in an off-line manner to ensure high index performance. We also maintain a "time flag" for each *Start Time Index*, recording the last time the corresponding *Start Time Index* is updated. The system checks these time flags periodically. When the difference between the current time and the time of last update exceeds a threshold, i.e., the *Start Time Index* has not been updated for a long time, we assume that future update on this index is also unlikely, and we can call the "compress" routine and transform a B+ tree index to a dynamic array.

6. Experiment

Since spatial grid partition is already thoroughly discussed and compared to other index structures such as 3DR-tree [17, 18], we only present the experiment results concerning our temporal index and compare CSE-tree to the R-tree [17] and SEB-tree [18].

6.1. Settings

The experiments are carried out on a PC with 3.00 GHz Intel Pentium 4 CPU, Windows XP SP2 platform, and 0.99 GB RAM. The inner node size of B+ tree is 64 byte, which is the cache line size of Pentium 4 CPU. According to [21], making the inner node size equal to the size of CPU cache line can increase B+ tree cache hits and decrease cache miss, thus improving the overall performance. Leaf size of B+ tree is 1024 byte.

Test data is generated following the steps in Section 4.4. We use synthetic data instead of real data because data in GeoLife is collected in our user study and uploaded according to a systematic plan, thus cannot reflect arrival process of new upload in real world. The time unit hereafter is hour. For the Poisson process that simulates arrival of uploaded tracks, we used intensity values of 100, 300, 500 and 700, meaning there are 100, 300, 500 and 700 uploaded tracks to the website on average. The total duration of the process is 2400, meaning 2400 hours, or 100 days. The parameter of Rayleigh distribution for T_{int} is 1.07. The mean of T_{dur} is 0.42 and variance is 0.98. All these parameters are estimated from the result of our user study in Section 3.1.

6.2. Results

Our indexing scheme is compared and evaluated from three aspects, namely, index size, average number of node

access in one insertion and average number of node access in one query. The horizontal axis of the following figures denotes the intensity of arrival of uploaded tracks.

6.2.1. Index Size We insert all synthetic data into the R-tree, SEB-tree and CSE-tree. Index size is shown below.

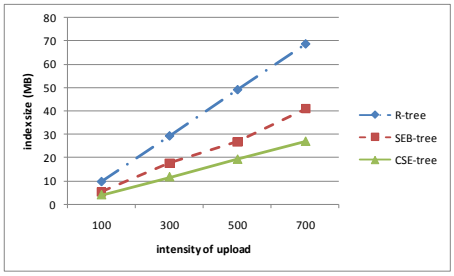


Figure 11. Index size comparison

The result shows that R-tree has the largest index size, because overlapping areas between nodes introduce excessive redundancy. SEB-tree takes up less index size than R-tree since the area covered by node are mutually exclusive, and therefore no overlapping between nodes exists. CSE-tree further reduces index size by converting a B+ tree into a dynamic array when update of the corresponding portion of data becomes less frequent.

6.2.2. Insertion Performance The number of node access for one insertion is illustrated in Figure 12. We count the average number of node access of 500 insertions, after the index is built with data in Section 6.2.1. R-tree has the largest number of node access, because R-tree is balanced tree, but distribution of new insertions is skewed. As a result, much inner node split is incurred and the overall performance deteriorates. CSE-tree requires less node access in one insertion than SEB-tree due to the optimization mechanism discussed in Section 5.2.1.

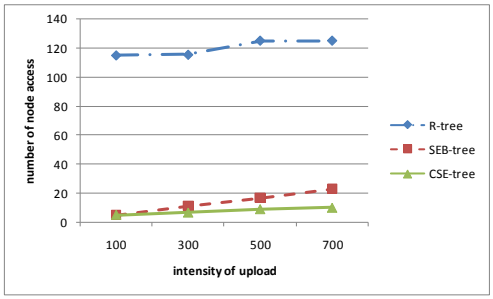
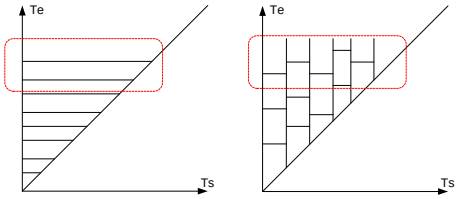


Figure 12. Mean number of node access in one insertion

As shown in Figure 13, the portion of data surrounded by the broken line has greater frequency of new insertions. For CSE-tree, the majority of new insertions will happen to the last few *Start Time Indexes*, thus we can avoid looking up *End Time Index* by simple checking the last few *Start Time Indexes*. This optimization process mentioned in Section 5.2.1 significantly reduces number of node access incurred in the insertion. The insertion algorithm of SEB-tree, however, has to first find the vertical column, and then searches within the vertical column, thus results in more node access.



(a) CSE-tree (b) SEB-tree

Figure 13. Comparison of insertion

6.2.3. Query Performance The selectivity of a temporal query is defined as the fraction of the total temporal span covered by the query. In our experiment, the total temporal span is 2400 hours. So a query with selectivity 0.1% spans 2.4 hours. We generated totally 30,000 queries, of which one third has selectivity 0.01%, one third has 0.1% and one third has 1%. The queries are evenly distributed over the whole temporal span. The mean number of node access is illustrated in Figure 14. Our results show that R-tree has the worst performance because if inner nodes have overlapping areas, the R-tree has to search all candidates respectively. Both CSE-tree and SEB-tree require much less node access than R-tree. CSE-tree requires slightly more node access than SEB-tree due to compression operation to CSE-tree. In a compression operation, a less frequently updated B+ tree is replaced by a dynamic array in order to save index space, but a binary search on a dynamic array incurs slightly more node access than search on a B+ tree.

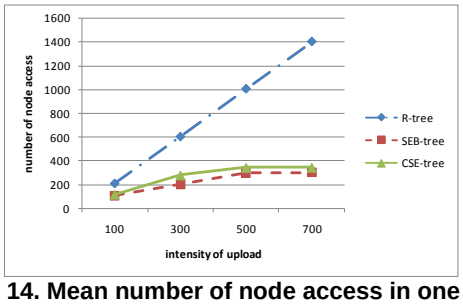
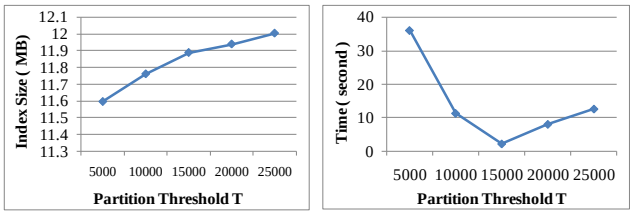


Figure 14. Mean number of node access in one query

6.2.4. Partition Threshold T of CSE-Tree Defined in Section 5.2.1, *Partition Threshold T* of CSE-Tree determines the time a new *Start Time Index* is created during insertion process. We have tested the effect of *Partition Threshold T* on the performance of CSE-Tree. Intensity of upload arrival rate is 500. Generation of data and query is the same with Section 6.2.3. The experimental results are shown in Figure 15. The query time in Figure 15 (b) is the total time of 30000 queries. Insertion time is more or less the same for different *T* values, thus omitted for the sake of concision.

Our first observation is that index size increases as *T* gets larger. The reason is that larger *T* value has two effects on the size of CSE-tree. First, larger *T* means larger size of each *Start Time Index*. Second, the number of *Start Time Index* will be smaller. But the first effect is more important to the overall size of CSE-tree.

Another observation is that search time first drops then increases as *T* value grows. Our reason is that when *T* is small, there would be a greater number of *Start Time Indexes*, and the search process has to access more *Start Time Indexes* to find the desired result. However, if *T* is too large, there are more tracks in each *Start Time Index*. As a result, the time cost of search within each *Start Time Index* would be high. Also, for the *Start Time Index* that intersect with a query's $Time_{min}$, it would take more time to filter out undesired segment. Take the query in Figure 10 for instance, if *Partition Threshold T* increases, *Tree1* will becomes larger, and it will consume more time to search within *Tree1* and to filter out tracks whose T_e is less than $Time_{min}$. The overall query performance of CSE-tree is decided by the two above-mentioned effects. When *T* is small, the former effect is greater, but the later one becomes dominant when *T* value exceeds 15000.



(a) Index size (b) Search

Figure 15. Effect of Partition Threshold T

7. Conclusion

In this paper, motivated by the observation on user behavior of uploading GPS tracks onto Web applications, we proposed a flexible spatio-temporal indexing scheme, CSE-tree, to help system effectively manage large volume of GPS tracks and provide fast retrieval service for Web users. A model that simulates such user behavior has been proposed based on stochastic process theory as well as statistical analysis on the data collection uploaded by users in real world. Over the synthetic data generated by the model, CSE-tree is compared with SEB-tree and R-tree. Experiments show that CSE-tree incurs slightly more node access than SEB-tree for query but requires less index size and cost less node access for insertion. CSE-tree also requires less index size than R-tree and cost less node access for both insertion and query than R-tree.

In the future, we would further investigate how to make a smooth transition between a B+ tree, which has faster insertion but requires more index size, and a sorted dynamic array, which has slower insertion but requires less index size. This would enable us to improve our compression algorithm with a series of intermediate stages between a B+ tree and a dynamic array, varying according to the frequency of new updates.

8. Reference

[1] Y. Zheng, et al, Learning Transportation Mode from Raw GPS Data for Geographic Application on the Web. In Proc. of WWW'08, Beijing, China. 2008

[2] Mountain Bike. http://www.mtb-tracks.co.uk/northyorkmoors/default.aspx

[3] Bikely: http://www.bikely.com/

[4] GPS Track route exchange forum: http://www.gpsxchange.com/

[5] GPS sharing: http://gpssharing.com/

[6] SlamXR. http://www.msslam.com/slamxr/slamxr.htm

[7] Wikiwalki. http://www.wikiwalki.com

[8] H. Marios, G. Kollios, V. J. Tsotras, D. Gunopulos. Indexing Spatialtemporal Archives. The VLDB Journal, 15(2): 143-164, 2006

[9] G. Kollios, V. J. Tsotras, D. Gunopulos, A. Delis, M. Hadjieleftheriou. Indexing animated objects using spatiotemporal access methods. IEEE Trans. Knowl. Data Eng. 13(4), 758-777(2001)

[10] C. Kolovson, M. Stonebraker. Segment Indexes: Dynamic indexing techniques for multi-dimensional interval data. In: Proceedings of the ACM SIGMOD Conference on Management of Data, pp. 138–147 (1991)

[11] A. Kumar, V. J. Tsotras, C. Faloutsos. Designing access methods for bitemporal databases. IEEE Trans. Knowl. Data Eng. 10(1), 1-20 (1998)

[12] B. Salzberg, V. J. Tsotras. Comparison of access methods for time-evolving data. Commun. ACM 31(2), 158–221 (1999)

[13] Y. Theodoridis, T. Sellis, A. Papadopoulos, Y. Manolopoulos. Specifications for efficient indexing in spatiotemporal databases. In: Proceedings of the International Conference on Scientific and Statistical Database Management, pp. 123–132 (1998)

[14] X. Xu, J. Han, and W. Lu. RT-Tree: An Improved R-Tree Indexing Structure for Temporal Spatial Databases. In Proc. of the Intl. Symp. on Spatial Data Handling, SDH, pages 1040–1049, July 1990.

[15] M. A. Nascimento, J. R. O. Silva. Towards historical R-trees. In Proc. of the ACM Symp. on Applied Computing, SAC, pages 235–240, Feb. 1998.

[16] Y. Tao, D. Papadias. MV3R-Tree: A spatio-temporal access method for timestamp and interval queries. In: Proceedings of the International Conference on Very Large Data Bases, pp. 431–440(2001)

[17] V. P. Chakka, A. Everspaugh, and J. M. Patel. Indexing Large Trajectory Data Sets with SETI. In Proc. of the Conf. on Innovative Data Systems Research(CIDR), 2003

[18] Z. Song, N. Roussopoulos. SEB-tree: An Approach to Index Continuously Moving Objects. Proceedings of International Conference of Mobile Data Management, pages 340–344, Jan. 2003.

[19] P. Zhou et al. Close Pair Queries in Moving Object Databases. Proceedings of ACM GIS, page 2-11, 2005

[20] Y.Theodoridis, J. R. O. Silva, M. A. Nascimento. On the Generation of Spatiotemporal Datasets. Advances in Spatial Databases, 6th International Symposium, Lecture Notes in Computer Science, Springer, page 147–164, 1999

[21] J. Rao, K. A. Ross, Making B+-tree Cache Sensitive in Main Memory. Proceedings of ACM SIGMOD Conference, 2000, pp. 475-486.