

Short Paper

Efficient Data Compression in Location Based Services

Yuni Xia
Department of Computer Science
Indiana University - Purdue University Indianapolis
Indianapolis, IN 46202
yxia@cs.iupui.edu

Yi-Cheng Tu Mikhail Atallah Sunil Prabhakar
Department of Computer Science
Purdue University
West Lafayette, IN 47907
{tuyc, mja, sunil}@cs.iupui.edu

ABSTRACT

In location based services, large amount of location data is generated constantly due to the continuous movements of the objects. The data is usually required to be stored for a fairly long time in order to answer window or history queries. This poses challenges to the efficiency of data storage and retrieval. In this paper, we observe that the redundancy among the moving object data is huge due to various factors including the ubiquitous periodicity of movements, the quasi-static moving feature of many objects and the large amount of common or shared segments among the moving object trajectories. We propose new mechanisms for compressing and storing moving object data. The constantly changing positions of moving objects are taken as time series. Time series data mining techniques are applied to discover periodic patterns within each series and similar or shared patterns among them. Based on the data mining results, the moving object data can be compressed substantially while maintaining the same or similar query performance.

1. INTRODUCTION

The combination of GPS, Internet, wireless communications, location techniques, and mobile devices have given rise to a wide range of location based services (LBS). Location based services are mobile data services that employ location as a key element to customize the data provided to the users. They utilize location-sensitive technologies such as GPS to find the geographical locations of the mobile devices and

then provide services based on the location information.

In a location-aware computing environment, new positions of the moving objects keep streaming into the database. The historic location data is required to be stored for many applications. This huge amount of data poses challenges to the efficiency of data storage and retrieval. However, due to the behavioral pattern of most moving objects and the topology of the environment, there are large amount of redundancy in the data. Data compression can be applied to reduce such redundancy. Compressing data to be stored or transmitted reduces storage and/or communication costs. Reducing the amount of data to be transmitted has the same effects as increasing the capacity of the communication channels. Similarly, compressing a file to half of its original size is equivalent to doubling the capacity of the storage medium. It may then become feasible to store the data at a higher, thus faster, level of the storage hierarchy and reduce the load on the input/output channels of the computer system.

For the past few years, tremendous amount of work has been done in building and deploying location based services and applications. Some of these focus on providing tools to the wireless telecommunication carriers, some focus on handset manufacturing, while others provide a suite of functions that contain geoprocessing applications and web services. Comparing to that, less effort has been made on the compression of moving object data.

In our work, the mobile database is viewed as a collection of a large number of time series. Each time series corresponds to a trajectory of a moving object. We noticed that time series in mobile database often share many common characteristics or patterns that have physical meanings. For example, the common segments of different time series means that moving objects move in the same road. In the field of time series, numerous work has been done on various issues such as classification, clustering [10], representation [13], similarity-based query [2] [1] [3] [12], whole-sequence matching or sub-sequence matching [6] [7], indexing the time sequence [11], statistical monitoring [15], anomaly detection, and so on. We go beyond the idea of time series analysis

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2002 ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

and mining and use the mining results for data compression. Our research emphasizes on finding and utilizing the pattern information to improve efficiency of storing and retrieving location data.

2. DATA REDUNDANCY IN LBS

In order to support location based services, the system needs to collect large amount of location data, which are generated every few minutes or even seconds due to the constant movements of the objects. Although position data can become obsolete after a short while, they are often required to be stored for a fairly long time in order to answer history or window queries. For example, which vehicles followed this trajectory; which people entered and exited the building during the past 24 hours? We observe that there is large amount of redundancy among the movements of moving object data due to the following features:

1. The movements of objects usually contains periodic patterns. The periodicity is ubiquitous among moving object data. For example, city buses repeat the same routes every hour or so. Most people tend to repeat the same or similar moving patterns every weekday and follow other patterns during the weekends. If we can identify the periodic patterns in the location data, we may extract them and avoid storing them repeatedly.
2. A majority of objects stay in a quasi-static state for a long time. They tend to stay in a state that is not exactly static, and move within a short range, for which we call the quasi-static state. For example, many people move within an office building during the day and stay at home at night for a long period of time. When the movement of an object is smaller than the precision requirement of the location based services, we can regard the object as static for a period and avoid storing multiple locations which are almost the same.
3. There are large number of common or shared segments among the moving object trajectories. For example, numerous vehicles take the same freeway; a large number of people, especially in big cities, take the same subway/bus/train/ferry route every day. Besides, many people stay in the same buildings for a long time. We can extract the common locations or trajectories and represent only once instead of store them for every object.

3. DATA COMPRESSION

In this section, we will explain how we identify the features and patterns in moving object data and use them for data compression.

3.1 Periodicity

Periodicity is ubiquitous among moving object data, however, extracting the periodic patterns base on the data is non-trivial. We propose to treat the positions of each object as a time series and apply time series mining techniques to discover the periodicities.

3.1.1 Periodicity Discovery

Finding periodic patterns in time series databases is an important data mining task with numerous applications.

Many methods have been developed for searching periodicity patterns in large data sets. Lots of them focus on mining full periodic patterns, where every point in time contributes to the cyclic behavior of the time series. However, in the real world, rarely a pattern is perfectly periodic (according to the strict mathematical definition of periodicity). Instead, most objects show the *partial periodic patterns*, which are patterns that are periodic over some but not all the points in it, or only some of the time episodes may exhibit periodic patterns. An example partial periodic pattern may state that every week day John was in the office from 8:00AM-12:00PM and 1:00-5:00PM, and on Road 69 from 7:30-8:00AM and 5:00-5:30PM, but his activities at other times do not have much regularity. Thus, partial periodicity is a looser kind of periodicity than full periodicity, and it exists widely in the real world.

Let Domain D be the set of locations. A moving object data sequence S is composed of elements from D and can be expressed as $S = e_0, e_1, \dots, e_i, \dots, e_{n-1}$, where i denotes the relative order of an element and n is the length of S . A pattern is a partial pattern if it contains the 'do not care' element '*'. Given a period T , a T -period pattern P is a sequence of elements $p_0, p_1, \dots, p_j, \dots, p_{T-1}$ ($0 \leq j \leq T$), where p_j can be an element from D or the wild card '*'. If $p_j = *$, then any element from D can be matched at the j -th position of P . A periodic fragment $s_i = e_i, e_{i+1}, \dots$ of a sequence S matches pattern $P = p_0, p_1, \dots, p_{T-1}$, if $\forall j$ ($0 \leq j < T$), we have either $p_j = '*'$ or $p_j = e_{i+j}$.

The support of a T -period pattern P , denoted as $SUP(P)$, in a sequence S is the number of periodic fragments that match P . A pattern P is regarded frequent with respect to a support parameter min_{sup} if $SUP(P) \geq \lfloor \frac{n}{T} \rfloor min_{sup}$, (support threshold). If there exists a frequent T -period pattern, we say that T is a frequent period. Element e is a frequent element if it appears in a frequent pattern [4]. The problem of mining partial periodic pattern can be defined as follows: Given a discrete data sequence S , a minimum support min_{sup} and a period window W , find:

1. the set of frequent periods T such that $1 \leq T \leq W$; and
2. all frequent T -period patterns w.r.t. min_{sup} for each T found in 1.

We adopt the algorithms for efficient mining of partial periodic patterns similar to the one proposed in [8], which explored interesting properties related to partial periodicity, such as the *apriori* property and the max-subpattern hit set property, by shared mining of multiple periods. The max-subpattern hit set property allows us to derive the counts of all frequent patterns from a relatively small subset of patterns existing in the time series. It can mine partial periodicities with only two scans over the time series database, even for mining multiple periods.

3.1.2 Compression Using Periodicities

After we obtain the periodic patterns, we can store the patterns only once. When they reoccur in future, we can only store links to the patterns instead of storing the whole patterns repeatedly.

If a periodicity is dominant and the sequences of a moving object for every period are very similar, we can use the compressed binary search tree to store the sequences. Com-

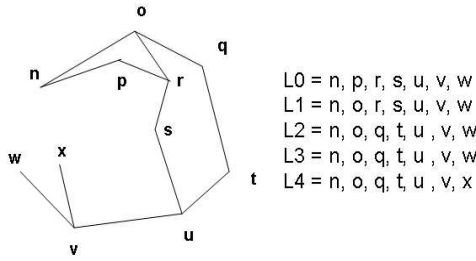


Figure 1: Trajectories

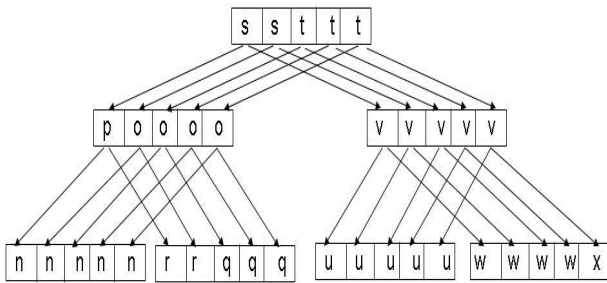


Figure 2: Simple Binary Search Tree

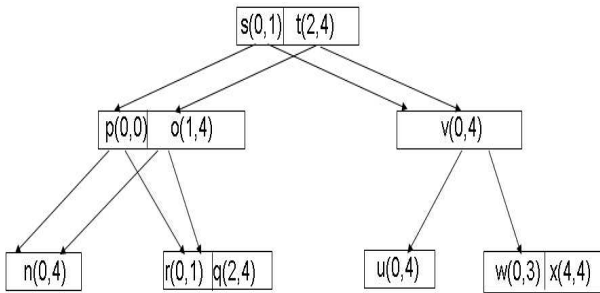


Figure 3: Compressed Binary Search Tree

pressed binary search tree was proposed to store similar sequences in [5]. An example of a set of similar time series are given in Figure 1. You may imagine L0-L4 are the moving trajectories of objects during each weekday morning. Suppose we have a set of k time series, each with n items. The sum of the differences between adjacent lists is h . The simple binary search tree, as shown in Figure 2 uses space $O(nk)$ and supports a query time of $O(\log n + h)$. The improvement of compressed binary search tree is to replace the sequence of tokens in one node storing the same item by just one triple (d, l, u) , where d is an item and $[l, u]$ represents the sequence of the lists between the l -th and the u -th lists. Figure 3 shows the binary search tree after compression. The compressed BST reduces the space requirement to $O(n + h)$.

3.2 Quasi-static State

We observe that a large number of moving objects in reality are in quasi-static states most of the time, for example, most people spend a long time at home or in an office, where their movements are small and slow. There are objects, of course, that move almost all the time, like taxis or city buses, but the proportion of these constantly moving objects is very small.

If you imagine the movements of an object over a period of time as a video, then the differences between two continuous snapshots or frames are very small. We can use this property for data compression. Similar to video compression, we can store only the differences between frames rather than all the contents in each frame.

In many applications, the precision requirements for the locations are not very high. For example, parents who track their children may only care if they are in the school or not; as to which classroom they are in, it does not matter. When the precision requirement is not very high, the moving object data can be further compressed. If the new position of a moving object does not differ from the old position by the precision threshold, the new position needs not to be stored and can be simply discarded. Furthermore, if the locator can determine that the new position of an object does not differ from the old position more than the precision threshold, it can choose not to report the new location and therefore reduce the communication cost.

We did some experiments to study this effect based upon data generated by the City Simulator 2.0 [9] developed at IBM. The City Simulator simulates the motion of people moving in a city. In our experiments, the number of people is set to 100,000 and the experiments run for 3030 seconds, during which each object updates its location for 100 times.

Figure 4 shows how the precision requirement affects the compression ratio. The compression ratio is computed as ratio of the amount of storage required for the scheme without compression to the one with compression. Obviously, the larger the precision threshold is, the more location changes fall within that range and thus, the higher the compression ratio will be. When the precision threshold is 10 meters, the compression ratio can be as high as 35, which means the compressed data is only around 3% of the original size and 97% of the data movements are within 10 meters and can be ignored. Even when the precision requirement is lowered to 2 meters, the compression ratio is still more than 3, which means the compressed data is less than 1/3 of the original data size.

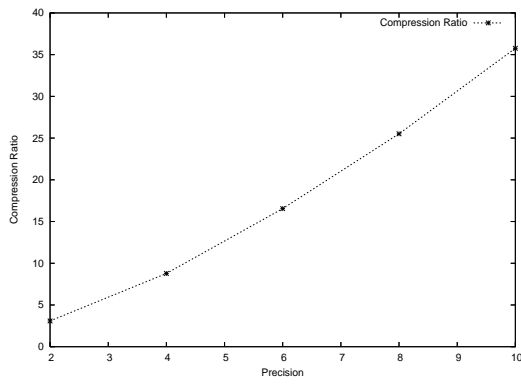


Figure 4: Compression Ratio vs. Precision

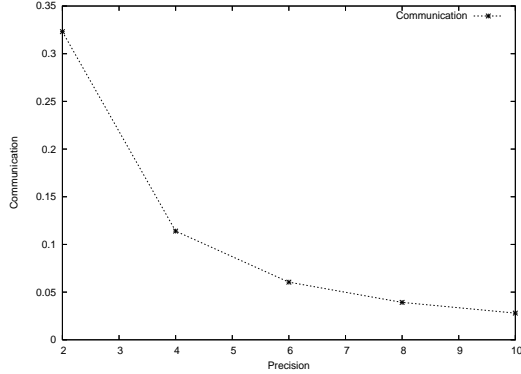


Figure 5: Communication vs. Precision

Figure 5 shows the relationship between the communication cost and the precision requirement. Here, the larger the precision threshold is, the more location changes fall within the precision range and need not be reported, therefore, the smaller the communication cost will be. When the precision is 2 meters, the communication cost is 32% of the original one, which means 2/3 of the location updates are within the precision requirement range and do not need reporting. As the precision threshold gets larger, the communication cost keeps decreasing. When the precision threshold reaches 10 meters, the communication cost is reduced to only 2.7% of the original one.

3.3 Shared Segments Among Trajectories

While periodicity and quasi-static features can help us exploit the redundancy that exists *within* each time series, discovering shared segments enables further compression by exploiting the redundancy *among* the time series.

Discovering shared segments among the time series is similar to finding out frequent sets or sequences. The frequent sets or sequences represent the hot areas or roads which repeat many times in the moving object database and should be represented in a more efficient way.

The problem of finding frequent sets is defined as follows: the input is a set of tuples (oid, S), where oid is the object ID and S is its position sequence. The absolute support of a sequence S_α in a sequence database is the number of tuples that contain S_α , denoted as $SUP(S_\alpha)$. Given a support threshold min_sup , a sequence S_α is a frequent sequence,

L0 = r, n, n, p, r
L1 = n, p, r, p
L2 = r, n, p, r
L3 = n, p, p, r, n

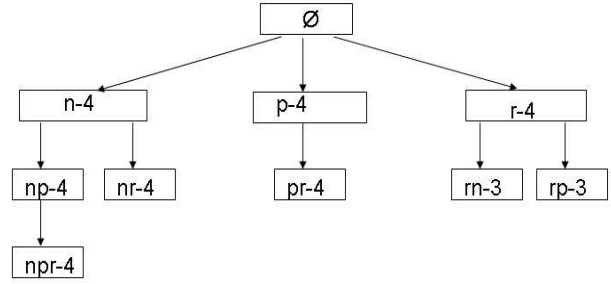


Figure 6: Frequent Sequence Tree

if $SUP(S_\alpha) \geq min_sup$. The problem of mining frequent sequence is to find the complete set of frequent sequences for an input sequence set, given a minimum support threshold min_sup .

The frequent sequences can be obtained by building a frequent sequence tree. Assume there is a lexicographical ordering among the set of items I in the input sequence database, conceptually the complete search space of sequence mining forms a sequence tree, which can be built in the following way: The root node of the tree is at the top level labeled as \emptyset , recursively we can extend a node N at level L in the tree by adding one item in I to get a child node at the next level $L + 1$ and the children of a node N are generated and arranged according to the chosen lexicographical ordering. By removing the infrequent sequences in the sequence tree, the remaining nodes in the lattice form a lexicographic frequent sequence tree, which contains the complete set of frequent sequences [14]. Figure 6 shows an example of 4 trajectories (L0-L3) and the corresponding lexicographic frequent sequence tree built from them, assuming $min_sup = 3$. Each node in Fig 6 contains a frequent sequence and its corresponding support.

Similarly, after getting the frequent sequences, we can store the sequences only once. If they reoccur in the future, only the links to the sequences (instead of the whole sequence) should be stored. Furthermore, the frequent sequences can be encoded using an Entropy Coding such as Huffman coding or Arithmetic coding to give additional compression. By entropy, we mean the amount of information present in the data, and an entropy coder encodes the given set of symbols with the minimum number of bits required to represent them.

4. CONCLUSIONS

In this paper, we propose new approaches for efficiently compressing and storing moving objects data for supporting location based services. The redundancy among moving object data is huge due to the ubiquitous periodicity of movements, the quasi-static moving feature of many objects and the large number of common or shared segments among the moving object trajectories. In this paper, the trajectories of each moving object data are taken as time series and we apply time series data mining techniques to find the periodic

patterns within each time series and the frequent patterns among them. The mining results can help compress the data and reduce the redundancy significantly.

5. REFERENCES

- [1] R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Twenty-First International Conference on Very Large Data Bases*, pages 490–501, Zurich, Switzerland, 1995. Morgan Kaufmann.
- [2] Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. Efficient Similarity Search In Sequence Databases. In D. Lomet, editor, *Proceedings of the 4th International Conference of Foundations of Data Organization and Algorithms (FODO)*, pages 69–84, Chicago, Illinois, 1993. Springer Verlag.
- [3] Rakesh Agrawal, Giuseppe Psaila, Edward L. Wimmers, and Mohamed Zaot. Querying shapes of histories. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *Twenty-first International Conference on Very Large Databases (VLDB '95)*, pages 502–514, Zurich, Switzerland, 1995. Morgan Kaufmann Publishers, Inc. San Francisco, USA.
- [4] Huiping Cao, David W. Cheung, and Nikos Mamoulis. Discovering partial periodic patterns in discrete data sequences. In *Advances in Knowledge Discovery and Data Mining, 8th Pacific-Asia Conference, PAKDD 2004*.
- [5] Richard Cole. Searching and storing similar lists. In *Journal of Algorithms*, 1986.
- [6] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings 1994 ACM SIGMOD Conference, Minneapolis, MN*, pages 419–429, 1994.
- [7] Xianping Ge and Padhraic Smyth. Deformable markov model templates for time-series pattern matching. In *Knowledge Discovery and Data Mining*, pages 81–90, 2000.
- [8] Jiawei Han, Guozhu Dong, and Yiwen Yin. Efficient mining of partial periodic patterns in time series database. In *Proceedings of the 15th International Conference on Data Engineering, 23-26 March 1999, Sydney, Australia*, pages 106–115. IEEE Computer Society, 1999.
- [9] James Kaufman, Jussi Myllymaki, and Jared Jackson. City simulator developed by IBM Almaden. <http://www.alphaworks.ibm.com/tech/citysimulator>.
- [10] Eamonn Keogh and M. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, editors, *Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 239–241, New York City, NY, 1998. ACM Press.
- [11] Eamonn J. Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD Conference*, 2001.
- [12] Seok-Lyonh Lee, Seok-Ju Chun, Deok-Hwan Kim, Ju-Hong Lee, and Chin-Wan Chung. Similarity search for multidimensional data sequences. In *ICDE*, pages 599–608, 2000.
- [13] Hagit Shatkay and Stanley B. Zdonik. Approximate queries and representations for large data sequences. In *ICDE*, pages 536–545, 1996.
- [14] Jianyong Wang and Jiawei Han. Bide: Efficient mining of frequent closed sequences. In *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA*.
- [15] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. *Proceedings of the 29th International Conference on Very Large Databases(VLDB)*, pages 358–369, 2002.