QED : An Efficient Framework for Temporal Region Query Processing

Yi-Hong Chu, Kun-Ta Chuang and Ming-Syan Chen

Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, ROC
E-mail: {yihong,doug}@arbor.ee.ntu.edu.tw, mschen@cc.ee.ntu.edu.tw

Abstract. In this paper, we explore a new problem of *"temporal dense region query"* to discover the dense regions in the constrainted time intervals which can be separated or not. A *Querying tEmporal Dense Region framework* (abbreviated as *QED*) proposed to deal with this problem consists of two phases: (1) an offline maintaining phase, to maintain the statistics of data by constructing a number of summarized structures, RF-trees; (2) an online query processing phase, to provide an efficient algorithm to execute queries on the RF-trees. The QED framework has the advantage that by using the summarized structures, RF-trees, the queries can be executed efficiently without accessing the raw data. In addition, a number of RF-trees can be merged with one another efficiently such that the queries will be executed efficiently on the combined RF-tree. As validated by our empirical studies, the QED framework performs very efficiently while producing the results of high quality.
Keywords: Temporal dense region query, dense region query

1 Introduction

Region-oriented queries have been recognized as important operations in many applications for the purpose of locating the regions satisfying certain conditions, e.g., those on density, total area, etc. [1][4][5][6][7]. In this paper, we focus on the query to find all dense regions whose densities are larger than their surrouding regions. In the literature, such queries are usually called as *"dense region queries"*, and the dense region can be identified by examining whether its density exceeds a density threshold. The previous work STING [6] conducted a study on speeding up the process of answering region-oriented queries by constructing an index structure to capture the statistical information of the data. Thus the queries can be executed efficiently by using the index structure without directly accessing the raw data.

However, previous research tends to ignore the time feature of the data. They treat all data as one large segment, and execute queries over the entire database. However, in practice, the characteristic of the data may change over time. It is noted that some dense regions may only exist in certain time intervals but will not be discovered if taking all data records into account since their average densities

may not exceed the density threshold. Therefore, discovering dense regions over different time intervals is crucial for users to get the interesting patterns hidden in data.

To discover such temporal patterns, we explore in this paper a novel problem, named *"temporal dense region query"* to address the dense region discovery in constrainted time intervals. The time constraint will be represented by a continuous time interval such as (6AM˜8AM), or a set of *separated* time intervals, such as (6AM˜8AM) and (9AM˜10AM). The problem of temporal dense region query is to find the dense regions in the queried time intervals. However, it is infeasible for the previous work on dense region queries to efficiently execute the temporal dense region queries. Because the queried time intervals are unknown in advance, the direct extension of their methods would be to delay the construction of index structures until the user queries the dataset, which is, however, inefficient for an interactive query environment. To the best of knowledge, despite of its importance, the problem of temporal dense region query had not been explicitly studied before.

Consequently, we devise in this paper a *"Querying tEmporal Dense region"* framework (abbreviated as *QED*), to execute temporal dense region queries. The QED framework consists of two phases: (1) an offline maintaining phase, to maintain the summarized structures of the evolving data; (2) an online query processing phase, to provide an efficient algorithm to execute queries on the summarized structures. Note that since the query processing is only applied to the summarized structures rather than to the original data points, the QED framework proposed is very efficient in practice. Furthermore, in order to support the diverse kinds of time intervals for dense region discovery, a number of base time slots are provided in advance for users to specify the time periods of interest, where the time slots are derived by segmenting the data into a number of partitions. Thus users will specify the time intervals by a number of time slots which can be separated or not. However the queried time periods are unknown when the data are summarized in the offline maintaining phase. It is challenging to summarize adequate statistics for queries with different time slots to be executed efficiently in the online phase. Therefore a novel summarized structure, referred to as *RF-tree* (standing for *Region Feature tree*), is proposed, and it has the property that a number of RF-trees can be merged with one another efficiently. Explicitly, in the offline phase a RF-tree will be constructed for each time slot, and in the online phase the queries will be executed efficiently on the RF-tree derived by merging the RF-trees with respect to the queried time slots. As validated by our experiments, the QED framework performs very efficiently while producing query results of very high quality.

The rest of this paper is organized as follows. In Section 2, the problem of temporal dense region query is explored. The offline maintaining phase of the QED framework is presented in Section 3, and the online discovering phase is described in Section 4. In Section 5 the empirical studies are conducted to evaluate the performance of QED. This paper concludes with Section 6.

2 Temporal Dense Region Query

2.1 Problem Description

There are two concepts in the temporal dense region query: one is the dense region; the other is the set of time intervals to be queried for dense regions.

Assume that the data records contained in the $d-$dimensional dataset are viewed as points in the d-dimensional data space constructed by the d attributes. In this paper, we use the grid-based approach to discover the dense regions. Initially the data space is partitioned into a number of small and non-overlapping cells which are obtained by partitioning each of the d dimensions into δ equal-length intervals. A cell is called a "*dense cell*" if it contains points exceeding a predetermined density threshold ρ. Thus based on these dense cells, the dense regions will be formed by a set of connected dense cells.

In addition, a number of "*time slots*" are provided in advance for users to specify a variety of time periods of interest. These *time slots* are obtained by segmenting the data points with a time granularity, e.g. week, month, year, etc. Two time slots are called *separated* if there are one or more slots between them. Thus the time period specified in the query will be represented by one or some separated time slots.

Problem Definition: (Temporal Dense Region Query)

Given a set of time slots, and the density threshold ρ, find the dense regions in those data points contained in the queried time slots, where each of the dense regions is formed by a set of connected dense cells with the number of data points in each of them exceeding ρ.

2.2 Overview of the QED Framework

A *"Querying tEmporal Dense region"* framework (abbreviated as *QED)*, is proposed in this paper to deal with the temporal dense region query. The QED framework consists of two phases: (1) an offline maintaining phase, to maintain the statistics of the data; (2) an online query processing phase, to execute temporal dense region queries on the summarized statistics.

Phase 1: Offline maintaining phase

The main task of the offline maintaining phase is to maintain the statistics of the data such that queries can be executed on the summarized information instead of the original data points, thereby enabling the dense region discovery to be very efficient. It is a two-step approach described as follows:

Step1: Partition the data set : The data set is partitioned into *time slots*.

Step2: Construct the RF-tree: For each time slot, a summarized structure, RF-tree is constructed to maintain the statistics of the data points. The RF-tree has the advantage that a number of RF-trees can be merged with one another efficiently. Therefore in the online phase, the overall statistics in the queried time slots can be obtained by combining the corresponding RF-trees, and then queries can be executed on the combined RF-tree.

Phase 2: Online query processing phase

When the user issues the queries, it is the main task of the online processing phase to process these queries. The online query processing phase is also a two-step approach:

Step1: Combine the RF-trees: Those RF-trees in the queried time slots are combined into a RF-tree by an efficient algorithm to describe in Section 4.1.

Step2: Execute the query: The query is executed on the combined RF-tree to discover the dense regions with respect to the density threshold δ specified in the query.

3 Offline Maintaining Phase

In the offline maintaining phase, an RF-tree will be constructed for each time slot to maintain the statistics of the data. In Section 3.1, we will give the definition of the uniform region which is discovered in the RF-tree for summarizing the cells in the data space. The algorithm for constructing the RF-tree will be described in Section 3.2.

3.1 The Definition of the Uniform Region

In the RF-tree, the entire data space is represented by a number of non-overlapped *regions* which are defined as follows:

Definition 1 *(region)*: A **region** in the d-dimensional data space is defined by the intersection of one interval from each of the d attributes, and can be represented by a Disjunctive Normal Form expression $(l_1 \leq A_1 < h_1) \wedge \cdots \wedge (l_d \leq A_d < h_d)$.

Definition 2 *(region feature)*: A **Region Feature** of a region R in the $d-$dimensional data space is defined as a binary: **RF** $=(N_R, N_c)$ where N_R is the number of data points contained in region R, and N_c is the corresponding number of cells.

It is noted that by setting δ to be a larger value the cells in the data space will become smaller. Therefore, if the cells in a region have nearly the same number of data contained in them, the total data points in this region can be viewed as approximately uniformly distributed in it. Such a region is called as *"uniform region"*. In a uniform region, the number of data points contained in each of the cells will be very close to the value N_R/N_c, which is the average number of data points in a cell and is calculated from the RF of this uniform region. Thus, in the RF-tree the uniform region is used to summarize the cells contained in it by taking the average value, N_R/N_c, to approximate the number of data points in each cell.

To identify the uniform regions in the data space, we use an entropy-based technique. The entropy is in essence a measure of the uncertainty of a random variable[2]. When the random variable follows the uniform distribution, we are most uncertain about the outcome and the corresponding entropy is the highest. In light of the entropy principle, a uniform region will typically has a higher entropy than a non-uniform one. In the following, we first define the entropy of a region, and then use it to identify a uniform region. Let the RF of a region be

denoted as (N_R, N_c). Also, $n(c_i)$ denotes the number of data points contained in the cell c_i. Therefore the entropy of the region R, $H(R)$, can be defined as follows:

Definition 3: *(entropy of a region)*

$$H(R) = -\sum_{i=1}^{N_c} \frac{n(c_i)}{N_R} \times \log \frac{n(c_i)}{N_R} \ , \qquad if \ \ n(c_i) \neq 0.$$

Consequently, we can judge whether a region R is a uniform region or not by first calculating the maximal entropy of this region, $H_{\max}(R)$, and then comparing $H(R)$ with $H_{\max}(R)$. Note that the maximum entropy of a random variable is proved to be the value, -log $\frac{1}{|\chi|}$, where $|\chi|$ denotes the number of possible outcomes of this random variable [2]. Analogously, the maximum entropy of a region R is defined as follows:

Definition 4 *(maximum entropy of a region)*: As defined above, N_c is the number of cells contained in the region R. The maximum entropy of the region R is defined as

$$H_{\max}(R) = -\log \frac{1}{N_c}.$$

Then, with a given similarity threshold θ, a uniform region is defined as follows:

Definition 5 *(uniform region)*: A region R is a uniform region if

$$\frac{H(R)}{H_{\max}(R)} \geq \theta.$$

3.2 Algorithm for Constructing the RF-tree

In this section we will introduce the RF-tree constructed for each time slot in the offline maintaining phase. The RF-tree is a hierarchical structure constructed to discover the uniform regions by summarizing the cells in the data space.

The process of constructing the RF-tree is a top-down and recursive approach. Initially, the root of the RF-tree is set as the entire data space. Let the root be at level 1 and its children at level 2, etc. Each node in the RF-tree represents a region in the data space, and the node in level i corresponds to the union of regions of its children at level $i + 1$. Each node in the RF-tree will be examined whether it is a uniform region by applying the Definition 5 with the similarity threshold θ. Thus, if it is examined as a uniform region, it will become a leaf node; otherwise, it will be a nonleaf node and its children will be derived by partitioning the region into a set of subregions, which are obtained by segmenting the interval of each of the dimension of this region into two intervals.

4 Online Query Processing Phase

The main task of the online query processing phase is to execute the query by using the RF-trees. Note that the overall statistics of the data in the queried time

slots for dense region discovery can be derived by combining the RF-trees in the queried time slots. Section 4.1 introduces an efficient algorithm for combining the RF-trees in the queried time slots. Then, the dense region discovery will be executed on the combined RF-tree, which is described in Section 4.2.

4.1 Algorithm for Combining the RF-trees

In the following, algorithm **COMB** (combine RF-trees) outlined below is to combine two RF-trees, and it can be easily extended to deal with more than two RF-trees. Algorithm **COMB** is a top-down approach. Let the roots of the two RF-trees be denoted as r_1, and r_2, and the root of the combined RF-tree be denoted as r_b. Initially in Step 3, r_b is set as the entire data space. In Step 4 and Step 5, the RF (Region Feature) of r_b is set up according to the RF of r_1 and r_2. Then in Step 6, procedure **SC** (Set up Children) is called to set up the children of r_b by taking r_1, r_2, and r_b as its inputs n_1, n_1, and n_b.

Procedure **SC** is to set up the children of input node n_b by taking into consideration of the three cases of n_1, and n_2: (1) n_1 and n_2 are both uniform (Step 3 to Step 5); (2) n_1 and n_2 are both non-uniform (Step 6 to Step 25); (3) only one of n_1 and n_2 is uniform (Step 26 to Step 38).

<**Case 1**>: n_1 and n_2 are both uniform. That is, the data points are uniformly distributed in n_1 and n_2. It will be also uniform if all data points are taken into consideration such that n_b will be a uniform region. Thus, procedure **SC** terminates.

<**Case 2**>: n_1 and n_2 are both non-uniform. Thus, n_1 and n_2 will both have at most 2^d children. In Step 7, the 2^d children of n_b are first generated by partitioning the region of n_b into 2^d subregions. For each child C, in Step 10 and Step 11, the node C_1 and C_2 which are child nodes of n_1 and n_2 with respect to the same region of C are identified. Then, in Step 12 to Step 24, the RF of C will be set up according to the RF of C_1 and C_2. Specifically, in Step 14, only C_1 is identified such that the children of C will be just set up with respect to the corresponding children of C_1 by calling procedure **AC** (Assign Children), and the descendants of C are set up recursively. Otherwise, in Step 24, the children of C will be set up by calling procedure **SC** to recursively combine the identified C_1 and C_2.

<**Case 3**>: only one of n_1 and n_2 is uniform. Without loss of generality, assume that n_1 is non-uniform and n_2 is uniform such that n_1 will have at most 2^d children. In Step 28, the 2^d children of n_b are first generated, and then for each child C, the RF of C will be set up in Step 30 to Step 37. Specifically, in Step 31, for each child C, $C.p$ denotes the average number of data derived from the uniform node n_2. Note that in Step 37 by calling procedure **AC**, the children of C will be just set up with respect to the corresponding children of C_1, and $C.p$ will be further averaged to the children of C if C will have children set up in procedure **AC**.

Note that procedure **SC** is applied recursively to combine the nodes in two RF-trees. It will terminate if Case 1 is encountered as shown in Step 3, or n_1 and n_2 are cells as shown in Step 1.

Algorithm COMB: Combine RF-trees
Input: RF-tree1, RF-tree2
Output: the combined RF-tree

1. // r_1 is the root of the RF-tree1, r_2 is the root of the RF-tree2
2. // r_b is the root of the combined RF-tree
3. r_b = entire data space
4. $r_b.N_R = r_1.N_R + r_2.N_R$
5. $r_b.N_c = r_1.N_c$
6. SC(r_1, r_2, r_b)

Procedure SC: Set up Children
Input: (node n_1, node n_2, node n_b)

1. if (n_1 is a cell & n_2 is a cell)
2. return
3. if (n_1 is uniform & n_2 is uniform) { //Case 1
4. return
5. }
6. else if (n_1 is non-uniform & n_2 is non-uniform) { //Case 2
7. Generate 2^d children of node n_b
8. For each child node C
9. $C.N_c = (n_b.N_c)/2^d$
10. C_1 = the child of n_1 with respect to the same region of C
11. C_2 = the child of n_2 with respect to the same region of C
12. if (C_1 == null && C_2 == null)
13. remove the chid node C from n_b
14. else if (C_1! = null && C_2 == null)
15. $C.N_R = C_1.N_R$
16. $C.p = 0$
17. AC(C_1, C)
18. else if (C_1 == null && C_2! = null)
19. $C.N_R = C_2.N_R$
20. $C.p = 0$
21. AC(C_2, C)
22. else
23. $C.N_R = C_1.N_R + C_2.N_R$
24. SC(C_1, C_2, C)
25. }
26. else { // Case 3
27. // Suppose n_1 is non-uniform, and n_2 is uniform
28. Generate 2^d children of node n_b
29. For each child node C
30. $C.N_c = (n_b.N_c)/2^d$
31. $C.p = (n_2.N_R)/2^d$
32. C_1 = find a child of n_1 with respect to the same region of C
33. if (C_1 == null)
34. $C.N_R = C.p$

```
35.             else if (C_1 ! = null )
36.                 C.N_R = C.p + C_1.N_R
37.                 AC(C_1, C)
38    }
```

Procedure AC: Assign Children

```
Input: (node n_1, node n_b)
1.    if (n_1 has no children)
2.        return
3.    else
4.        For each child node C_1 of n_1
5.            Generate a child C with respect to the same region of C_1
6.            if (n_b.p ! = 0)
7.                C.p = (n_b.p)/2^d
8.            else
9.                C.p = 0
10.           C.N_R = C_1.N_R + C.p
11.           C.N_c = C_1.N_c
12.           AC(C_1, C)
```

4.2 Execute the temporal dense region query

After the combing process, the query will be executed on the combined RF-tree. Initially all leaf nodes in the combined RF-tree are examined to discover the dense cells in the data space, and then the leaf nodes containing dense cells will be put into a queue for further dense region discovery. Note that the leaf nodes will be of two cases : (1) a cell, and (2) a uniform region. For the case (1), if the number of data points it contains exceeds the density threshold ρ, it is identified as a dense cell and is put into the queue. For the case (2), the cells contained in this uniform region will have the same average number of data points, i.e. N_R/N_c, calculated from its RF. Thus these cells will be identified as dense ones if the value of N_R/N_c exceeds the density threshold ρ, and then this uniform region will be put into the queue.

After all leaf nodes are examined, the dense regions can be discovered by grouping the connected ones in the queue. This can be executed by a breadth-first search. Each time we take out a leaf node n_i from the queue, and examine the rest ones whether they are connected to n_i. If no one is identified, output the node n_i to be a dense region. Otherwise, the identified nodes will be first taken out from the queue, and then the rest nodes are recursively examined on whether they are connected to the previous identified nodes. Finally, the leaf nodes connected to n_i will become a dense region.

5 Experiments

5.1 Quality of the RF-tree

To evaluate the quality of the RF-tree, we generate a two dimensional data set of 5000 data points, and δ is set to 16. The density threshold ρ is set to 1.1

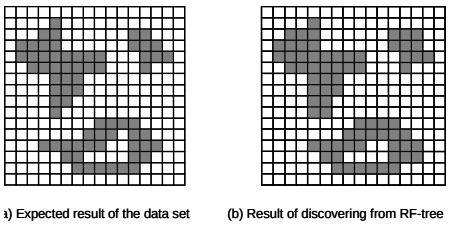


Fig. 1. Illustration of the quality of the RF-tree.

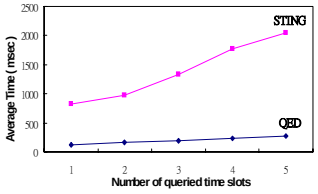
times the expectation of data points in each cell under uniform distribution, i.e. $1.1 \times (5000/16^2)$. The expected result is shown in Figure 1(a). Figure 1(b) is the result of executing the query on the RF-tree constructed by setting θ to 0.9. From Figure 1, it is seen that the RF-tree is able to successfully discover the dense regions and the query result is very close to the expected one.

5.2 Performance of combining RF-trees

In this experiment the scalability of the QED on queries with different number of time slots is studied. One real data set is used, which is the census-income data of 25,0000 data points from the UCI KDD Archive [3] with three attributes, i.e. age, income, weight. The total data points are partitioned into five time slots $W_1 \ldots W_5$ such that there are 50000 data points in each one. We test the performance with queries with the number of queried time slots varying from 1 to 5. There are five queries with one time slot, i.e. W_1, W_2, W_3, W_4, and W_5. There are ten ones with two time slots, i.e., W_1W_2, W_1W_3, W_1W_4,...,etc. For each query, the density threshold ρ is set to 1.1 times of the expectation of data points in each cell under uniform distribution.

Figure 2(a) shows the relative execution time of STING and RF-tree on varied number of time slots. The execution time for queries with the same number of time slots is averaged. In this experiment, STING is extended to deal with the temporal dense region queries with two steps: (1) constructing their proposed index hierarchy on all data in the queried time slots, and (2) using the index hierarchy to answer queries. As shown in Figure 2(a), QED outperforms the extended STING algorithm about by very prominent margin. Note that with the efficient combining procedure for the RF-trees, the execution time of QED only slightly increases when the number of queried time slots increases.

In addition, in Figure 2(b) we use the metric of *recall* and *precision* to evaluate the qualities of query result of the QED framework. *Recall* is defined as the percentage of the expected dense cells identified by QED. *Precision* is defined as the percentage of the dense cells identified in QED truly dense expectedly. *F-score* is defined as the value of (2×recall×precision)/(recall+precision). As shown in Figure 2(b), all of the expected dense cells are discovered by QED because the recall are all ones. Very few cells are falsely identified as dense by



(a) Time complexity

Num of time slots	recall	precision	F-score
1	1	0.9719	0.9845
2	1	0.9655	0.9828
3	1	0.9665	0.9829
4	1	0.9661	0.9827
5	1	0.9746	0.9872

(b) Quality of discovered dense regions

Fig. 2. The performance of querying on the combined RF-tree

QED as expected, and the precisions are very close to 1. Moreover, QED is still robust when the number of queried time slots increases since recall and precison remain very close to 1 as shown in the figure. Therefore, as validated in these experiments QED is effective and efficient for temporal dense region queries.

6 Conclusion

In this paper, the problem of temporal dense region query is explored to discover dense regions in the queried time slots. We also propose the QED framework to execute temporal dense region queries. QED is advantageous in that various queries with different density thresholds and time slots can be efficiently supported by using the concept of time slot and proposed RF-tree. With the merit of the efficiency of combining RF-trees, the QED framework scales well with respect to varied number of time slots. As evaluated by the synthetic and real data, QED is powerful in discovering the dense regions and outperforms prior methods significantly.

Acknowledgements

The work was supported in part by the National Science Council of Taiwan, R.O.C., under Contracts NSC93-2752-E-002-006-PAE.

References

1. D.-S. Cho, B.-H. Hong, and J. Max. Efficient Region Query Processing by Optimal Page Ordering. *In Proc. of ADBIS-DASFAA*, 2000.
2. T. M. Cover and J. A. Thomas. Elements of Information Theory. *Wiley*, 1991.
3. S. Hettich and S. Bay. The UCI KDD archive. *[http://kdd.ics.uci.edu]*, 1999.
4. G. G. Lai, D. Fussel, and D. F. Wong. HV/VH Trees: A New Spatial Data Structure for Fast Region Queries. *In Proc. of Design Automation Conference*, 1993.
5. B. S. Lee, R. R. Snapp, R. Musick, and T. Critchlow. Metadata Models for Ad Hoc Queries on Terabyte-Scale Scientific Simulations. *J. Braz. Comp. Soc.*, 2002.
6. W. Wang, J. Yang, and R. Muntz. STING: A Statistical Information Grid Approach to Spatial Data Mining. *In Proc. of VLDB*, 1997.
7. A.M. Yip, E.H. Wu, M.K. Ng, and T.F. Chan. An Efficient Algorithm for Dense Regions Discovery from Large-Scale Data Streams. *In Proc. of PAKDD*, 2004.