IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 9, Issue 1 (Jan. - Feb. 2013), PP 13-17
www.iosrjournals.org

Mining Precise Top-K Dominating Itemset from Data Streams

G. Sandhya[1], S. Kousalya Devi[2], K. Megala Devi[3], Dr. C. Kumar Charlie Paul[4]
[1](Department of CSE, A. S. L. Pauls College of Engineering and Technology, India)
[2](Department of CSE, A. S. L. Pauls College of Engineering and Technology, India)
[3](Department of ECE, A. S. L. Pauls College of Engineering & Technology, India)
[4](Department of ECE, A. S. L. Pauls College of Engineering & Technology, India)

Abstract: *Top-K dominating query is a preference based query which determines the dominating itemset from the data stream. This query combines the notion of ranking function with the concept of dominance. Data-stream mining has more constraints and requirements. It determines valuable information from a great deal of primitive data. This is adequate for static datasets, where updates are rare. As many modern applications adopt dynamic datasets, they need continuous query processing algorithms to revive the query result. The Sliding Window defines the set of active points which are the most recently arrived points. The existing work combines the ADVANCED ALGORITHM, the APPROXIMATE HOEFFDING BOUND ALGORITHM and the APPROXIMATE MINIMUM SCORE ALGORITHM. This combination guarantees accuracy, provides faster processing and best performance. The enhancement work proposes the TOP-K SCRUTINIZING ALGORITHM for finding dominant itemset from datastream by partitioning the Sliding Window into buckets of equal size. This is an Event processing algorithm which includes event scheduling and rescheduling towards avoiding the examination of points for inclusion in the Top-K. Experimental results using real datasets and synthetic datasets show that the proposed method reduces CPU time and memory overhead. Also, it is quite efficient and achieves high accuracy.*

Keywords- *Data-stream, Itemset, Event Processing Algorithms, Sliding Window, Top-K Dominating Queries.*

I. Introduction

Frequently, two preference based queries are in use. They are the Top-K queries and the Skyline queries. The Top-k queries [8] require a ranking function which assigns a value to each point. The ranking function is user defined. The Skyline queries are scaling invariant which means that if scaling applies to dimension values then, the result remains the same. The Top-K queries maintain the advantages and eliminate the drawbacks of both the Top-K queries and the Skyline queries. At present, two basic Sliding Window types are available. They are Count based Sliding Window and Time based Sliding Window.

In a Count based Sliding Window [1], the number of active points remains even. The running out time of some points will be equal to the entering time of the same amount of other points. In a Time based Sliding Window [1], the number of active points may not remain even. The running out time of a point does not depend on the entering and running out time of other points. A Grid based indexing scheme [9] maintains simplicity during the presentation of the algorithm. This scheme serves the purpose of book keeping, which deletes a running out point and inserts a new point. Subsequently, it updates the scores of the Top-K dominating points. The scheme makes easy computation of the domination score of a point. Regular grid and adaptive grid are the two types of grid available.

The initial work studies the combination of an event processing algorithm and two approximate algorithms which achieves more than 95 percent accuracy. Subsequently, the work proposes an event processing method which partitions the Sliding Window into equal size and thus precise the result having efficiency more than 97 percent. The final work carries a thorough experimental evaluation based on real datasets and synthetic datasets. This provides evidence regarding the CPU time of the proposed algorithm. Remaining of the paper organizes as follows. Section 2 shows the work associated with the problem. Section 3 studies the combination of the event processing algorithm and two approximate algorithms. Section 4 discusses a new technique by partitioning the Sliding Window. Section 5 presents the experimental evaluation of based on some datasets. Section 6 concludes the work. At last, section 7 discusses the future work.

II. Related Work

Preferences involve in disciplines such as Game Theory, Computational Geometry and many others. The batch counting [8] calculates the scores of skyline points in batch instead of iteratively applying separate range queries. This involves a light weight technique [8] to derive the upper bound score of non-leaf entries at low cost and the lazy counting technique to delay the counting of points for the purpose of forming better groups

for batch counting. To be effective, the tree traverses with a carefully designed priority order aiming at minimizing the I/O cost.

A grid-based indexing scheme [7] facilitates an efficient search and update operations, evading costly re-organization costs. Here, the method uses an adaptive grid which separates equally the tuples in each dimension. The schema formulates and tackles the problem of PROBABILISTIC TOP-K DOMINATING (PTD) query [5] in the context of uncertain databases. This can easily apply the pruning conditions once if the lower bound scores are available. The Sliding Window Combinatorial Approximation (SWCA) [5] determines frequent itemsets over sliding windows in a datastream. The Min Top-K algorithm [2] maintains a MINIMAL TOP-K CANDIDATE SET (MTK) which determines the Top-K value and thus eliminates the need for re-computation.

Reverse Top-K queries [4] are essential for the manufacturers to access the potential market and impact of their products based on competition. As stated, [6] discusses the problem of processing Top-K dominating queries on multidimensional data. The TOP-K MONITORING ALGORITHM (TMA) [9] re-calculates the result from scratch. However, the SKY-BAND MONITORING ALGORITHM (SMA) maintains a superset of the current answer, for the purpose of avoiding frequent re-computations. According to [1], the running out time of a point will be equal to the sum of the entering time of a point and the number of active points. Here, the data forms a sequence of points. The entering and running out time instances characterize these points.

III. Combined Algorithm

1. Introduction

Three algorithms can work in combination. They are the ADVANCED ALGORITHM (ADA), APPROXIMATE HOEFFDING BOUND ALGORITHM (AHBA) and APPROXIMATE MINIMUM SCORE ALGORITHM (AMSA). For the purpose of simplicity, the combination of algorithms adopts Count based Sliding Window. Here, if a new point enters the Sliding Window the oldest one will run out and the time progresses. Hence, the running out time of the points will be equal to the sum of the running out time of the points and the number of active points. Here, P is the set of active points, N is the number of active points, $score(P_i)$ is the number of points dominated by P_i, Top-K is the set of K points with the best scores, K is the number of points in the result, *now* is the current time instance, and $score_k$ is the K^{th} best score.

2. Advanced Algorithm

The ADVANCED ALGORITHM [1] uses candidate points and thus reduces the number of exact score computations. The points whose event processing time is in the near future are the candidate points. The algorithm endlessly evaluates the score of these special points. Some points will terminate before processing the event, when the run out time of a point is less than the event processing time. Such an event characterizes as obsolete. Here, if it evicts obsolete events then, it reduces the storage requirements. The algorithm determines a safe interval [1] by using r points and a counter as follows:

$$SI(P_i) = \min\left\{\left[\frac{score_{k-r} - score(p_i)}{2}\right], exp_{r+1} - now\right\} \quad \ldots(1)$$

3. Approximate Algorithms

Two approximate algorithms suitable for ADA are APPROXIMATE HOEFFDING BOUND ALGORITHM and APPROXIMATE MINIMUM SCORE ALGORITHM in which AHBA [1] guarantees accuracy by maintaining samples. If the maximum score of a cell is greater than the difference between the top score and the time threshold also no other cell dominates the same cell with a sample, then AMSA [1] keeps sample for the corresponding cell (Δt is the time threshold and score_k is the top score). Δt initializes to n/1000 and adjusts the values automatically. This controls the error introduced in the score estimation. AMSA uses quick and dirty approximation, thus offers fast solution. If $score_1$ - $score_k$ < $score_k$ – minimum score of a cell, then exclude all points belonging to the corresponding cell. This combination achieves better efficiency, provides faster solution and guarantees accuracy.

IV. Top-K Scrutinizing Algorithm

The TOP-K SCRUTINIZING ALGORITHM (TSA) is an event processing algorithm which partitions the Sliding Window into buckets of equal size and finds dominant itemset from datastream. Table.1 illustrates the pseudocode of TSA. The TSA algorithm considers the entering points and the current time instance. The inputs are the Truck Datasets, Buckets of equal size B, Sliding Window with five Buckets and n Active Points. The algorithm obtains the Top-K dominating points as output. Here, P is the set of active points, N is the number of active points, $score(P_i)$ is the number of points dominated by P_i, Top-k is the set of K points with the

best scores, K is the number of points in the result, *now* is the current time instance, $score_k$ is the K^{th} best score.

Table.1: Top-K Scrutinizing Algorithm.

Algorithm TSA (P_x, *now*)
P_x : the entering points
now : the current time instance
Input : Truck Datasets, Buckets of equal size **B**, Sliding Window **W**= { B_1, B_2, B_3, B_4, B_5}, Active Points **P**= { P_1, P_2, P_3,…, P_n }
Output : Top-k Dominating Points

1. Update index and scores of points in *TOPK*;
2. e_r. *score*=0;
3. e_r. *egt=now*;
4. Call BucketInOut( )
5. P_x← inserted into the window W
6. **if** (entry.P_1 = = exit.P_1 = = B_1 && entry.P_2 = = exit.P_2 = = B_2 && entry.P_3 = =exit.P_3 = = B_3 && entry.P_4 = = exit.P_4 = = B_4 && entry.P_5 = = exit.P_5 = = B_5) **then**
7. Call FindReferencePoint(P_x)
8. P_r←FindReferencePoint(P_x);
9. e_r←event of P_r;
10. *score*(P_x) ← e_r. *score* + *now* - e_r. *egt*-1;
11. **if** (*score*(P_x) ≥ $score_k$) **then**
12. Calculate *score*(P_x) from scratch;
13. **if** (*score*(P_x) ≥ $score_k$) **then**
14. Insert P_x in *TOPK*;
15. **if** (P_x ∉ *TOPK*) **then**
16. ScheduleEvent (*x*, *score*(P_x), *now*);
17. e_i←*EventQueue*.RemoveTop ( );
18. **while**(e_i. *ept* = = now);
19. **if** (number of top-k points ≥ k) **then**
20. *score* ← e_i. *score* + e_i. *ept* - e_i. *egt;*
21. ScheduleEvent (*i*, *score*, *now*);
22. **if** (e_i *is not rescheduled*) **then**
23. Calculate *score*(P_i) from scratch;
24. **if** (*score*(P_i) ≥ $score_k$) **then**
25. Insert P_i in *TOPK*;
26. **else** ScheduleEvent (*i*, *score*(P_i),*now*);
27. e_i←*EventQueue*.RemoveTop ( );
28. **function** ScheduleEvent (*j*, *score*, *now*)
29. exp_1←least run out time of points in *TOPK;*
30. *ept* ←min([$score_k$- *score*)/2]+*now*), exp_1*);*
31. **if** (*ept* ≥ *now*) **then**
32. e_j.*ept*← *ept;*
33. e_j.*egt*←*now*;
34. e_j.*score* ← *score;*
35. EventQueue.Insert(e_j);

The event processing algorithm contains an event processing time, event generation time and score of a point at the event generation time. Each update applies a sequence of operation. Initially, the algorithm updates the bookkeeping structure. Line 1 deletes the running out points and enters the new points. Line 1 also updates the scores of the Top-K dominating points. Lines 2 to 16 process the entering points P_x to determine whether they should be a part of Top-K. Line 4 to 5 calls the procedure BucketInOut( ) which tracks the in-out operation of each bucket when a point P_x enters into the window W. Each time when a bucket in-out operation occurs, the algorithm deletes the earliest transaction of the corresponding bucket which contains the summary of transactions from the current window at each sliding. Hence, there is no need to maintain the whole transactions within the current window in memory all along to support window sliding. The partition of the sliding window is into five buckets of equal size and each bucket corresponds to a set of transactions. This is to reduce the

memory overhead during the storage of transactions. If there are more buckets, then the memory overhead arises during storage of the transactions. Line 6 checks in which bucket the points enter and the oldest points run out. Line 7 calls the procedure FindReferencePoint( ), which tries to locate a point P_r in the bucket such that, it dominates P_x and it is not a part of Top-K. If such a point does not exist or the upper bound of the score is larger than or equal to $score_k$, then line 12 calculates P_x from scratch. If *score*(P_x) is greater than $score_k$, then line 14 inserts P_x in the Top-K.

Otherwise, line 28 to 35 generates an event for P_x using the procedure scheduleEvent( ). This procedure takes three parameters such as the id j of the point, its score and *now*, which is the current time instance. First, it calculates the event processing time. Subsequently, it checks whether time is greater than or equal to *now*. If so, it inserts the event into the priority queue. Observe that, the event processing time is less than *now* as if the parameter score is larger than $score_k$. Thus, line 16 always generates an event, since either the upper bound in line 11 or the exact score in line 13 is less than $score_k$. At last, lines 17 to 27 processes all the events with the event processing time equal to *now*. Lines 20 to 21 tries to re-calculate its e_i.*ept* value by using the upper bound of *score*(P_i) for an event e_i. If the event re-inserts into the event priority queue, then consider the subsequent event. If not, if the upper bound estimation is poor, it is possible the calculated event time to be less than *now*. In such a case, lines 22 to 26 proceeds with the exact score computation of *score*(P_i) and either inserts P_i in the Top-K or re-calculates the event time based on the exact score of P_i. Line 19 controls the run out of the Top-K dominating points. If a Top-K point expires in *now* and no updates in $score_k$ then, no need to try to calculate event times. In this case, it sets the $score_k$ to -1 to forcibly insert other points in Top-K. Afterward, it calculates the score of the point of the initially examined event and inserts the point in Top-K. Eventually, it tries to re-calculate the event time of the remaining events.

V. Experimental Results

All the algorithms implements using C Sharp and conducts experiment on a Pentium at 3.0 GHz with 1 GB of Random Access Memory (RAM). This uses the real datasets and the synthetic datasets, named as the car-race datasets (http://www.amstat.org/publications/jse/datasets/nascarr.dat.txt). The dataset undergoes a multidimensional analysis. The analysis considers the attributes such as the number of drivers, monthly consumer price index, track length, laps completed, numbers of caution flags and lead changes, total prize money, completion time, and coordinates of the track.

Table.2: Performance Analysis.

Number of Iterations	CPU Time in Milliseconds			
	ADA	ADA with AHBA	ADA with AMSA and AHBA	TSA
1	577	390	312	187
2	468	468	390	202
3	436	577	343	265
4	468	405	302	218

Table.2 shows the performance analysis of the algorithms such as ADA, ADA with AHBA, ADA with AMSA and AHBA and TSA. The algorithms have the following CPU time during the first iteration. ADA takes 577 milliseconds; ADA with AHBA takes 390 milliseconds; ADA with AMSA and AHBA takes 312 milliseconds and TSA takes 187 milliseconds. In the second iteration, the algorithms have the following CPU time. ADA takes 468 milliseconds; ADA with AHBA takes 468 milliseconds; ADA with AMSA and AHBA takes 390 milliseconds and TSA takes 202 milliseconds. The third iteration has the following CPU time. ADA takes 436 milliseconds; ADA with AHBA takes 577 milliseconds; ADA with AMSA and AHBA takes 343 milliseconds and TSA takes 265 milliseconds. During the second iteration, the algorithms have the following CPU time. ADA takes 468 milliseconds; ADA with AHBA takes 405 milliseconds; ADA with AMSA and AHBA takes 302 milliseconds and TSA takes 218 milliseconds. The iteration continues till it obtains the Top-K itemset.

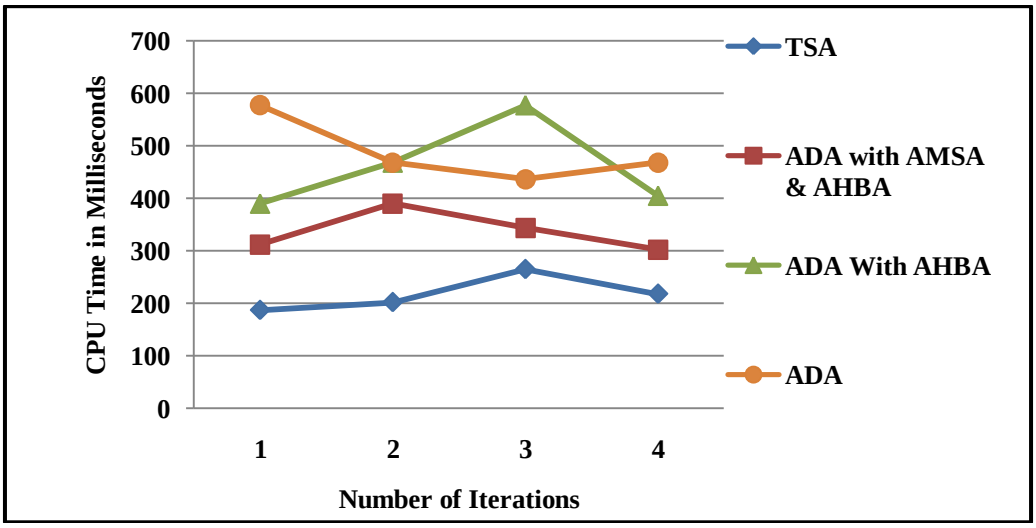

Fig.1: iterations versus CPU time.

Fig.1 plots the CPU time obtained from the previous table for each of the iterations in a graph. The graph takes the number of iterations in x-axis and the CPU time in y-axis. The solid diamond shape represents the CPU time of TSA. The solid square shape represents the CPU time of ADA with AMSA and AHBA. The solid triangle shape represents the CPU time of ADA with AHBA. The solid circle shape represents the CPU time of ADA. From the Table.2 and Fig.1, it is clear that TSA reduces the CPU time.

VI. Conclusion

Top-K dominating query determines the dominating itemset from the data stream by combining the notion of ranking function with the concept of dominance. The combination of ADA with AMSA and AHBA is an event processing algorithm in which ADA shows the best performance, AMSA offers fast solution and AHBA guarantees accuracy. TSA also uses the concepts of Top-K dominating query and retrieves dominant itemset from datastream, by partitioning the Sliding Window into buckets of equal size. It is clear that TSA reduces the memory overhead since the Sliding Windows gets divided into five buckets and maintains the transactions for those five buckets. Finding Top-K itemset using these buckets may also provide higher accuracy. It is also evident that TSA has reduced the CPU time better than the combination of ADA with AMSA and AHBA. Hence, TSA which is an event based algorithm offers significant flexibility, higher accuracy, reduces the memory overhead and the CPU time.

References

Journal Papers:

[1] M. Kontaki, A. N. Papadopoulos and Y. Manolopoulos, Continuous Top-K Dominating Queries, *Proc. IEEE Transactions on Knowledge & Data Engineering*, *Vol. 24*, 2012, 840-853.

Proceedings Papers:

[2] Di Yang, Avani Shastri, Elke A. Rundensteiner and Mathew O. Ward, An Optimal Strategy for Monitoring Top-k Queries in Streaming Windows, *Proc. Int'l Conf. Extending Database Technology: Advances in Database Technology*, 2011.
[3] Kuen-Fang Jea and Chao-Wei Li, A Sliding-Window Based Adaptive Approximating Method to Discover Recent Frequent Itemsets from Data Streams, *Proc. IMECS*, 2010.
[4] A. Vlachou, C. Doulkeridis, Y. Kotidis and K. Nørvag, Reverse Top-k Queries, *Proc. IEEE Int'l Conf. Data Engg.*, 2010.
[5] X. Lian and L. Chen, Top-k Dominating Queries in Uncertain Databases, *Proc. 12th Int'l Conf. Extending Database Technology: Advances in Database Technology*, 2009.
[6] M. L. Yiu and N. Mamoulis, Multidimensional Top-k Dominating Queries, *Proc. Int'l Conf. Very Large Data Bases*, 2009, 1-23.
[7] M. Kontaki, A. N. Papadopoulos, and Y. Manolopoulos, Continuous Top-k Dominating Queries in Subspaces, *Proc. Panhellenic Conf. Informatics*, 2008.
[8] M. L. Yiu and N. Mamoulis, Efficient Processing of Top-k Dominating Queries on Multi-Dimensional Data, *Proc. Int'l Conf. Very Large Data Bases*, 2007, 483-494.
[9] K. Mouratidis, S. Bakiras, and D. Papadias, Continuous Monitoring of Top-k Queries over Sliding Windows, *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2006, 635-646.