Relaxed Queries over Data Streams[*]

Barbara Catania, Giovanna Guerrini, Maria Teresa Pinto, and Paola Podestà

Università di Genova, Italy

Abstract. Relaxation skyline queries have been proposed, in the relational context, as a solution to the so-called empty answer problem. Given a query composed of selection and join operations, a relaxation skyline query relies on the usage of a relaxation function (usually, a numeric function) to quantify the distance of each tuple (pair of tuples in case of join) from the specified conditions and uses a skyline-based semantics to compute the answer. Though the empty answer problem is extremely relevant also in a streaming context, where users may not be acquainted with the actual data arriving on the stream, it has been largely neglected. Specifically, no solutions have been proposed so far for skyline-based relaxation over data streams. In this paper, we define relaxation skyline queries for window-based join over data streams, propose one processing algorithm and present a preliminary experimental evaluation of the designed technique.

1 Introduction

The last decade has been characterized by the raise of data intensive applications, with new data querying needs, and novel processing environments. Data integration applications, Web services, sensor networks, data stream management systems, P2P, cloud computing, and hosting are only few examples of these emerging technologies. The novel processing requirements related to these new contexts made traditional query processing approaches unsatisfactory and required the development of specific advanced query processing techniques.

One specific issue for such advanced techniques concerns *approximation*. Indeed, data characteristics (e.g., heterogeneity, incompleteness, and uncertainty), resource limitations, huge data volumes, and volatility, typical of emerging applications and technologies, suggest it may be preferred to relax the query definition, using Query Relaxation techniques, or to generate an approximate result set, with quality guarantees, using Approximate Query Processing techniques, instead of getting an unsatisfactory answer. An answer can be unsatisfactory because either the user has to wait too long for getting the result, or the answer is empty, or the answer contains too many tuples.

In this paper we are interested in Query Relaxation techniques for solving the empty answer problem. Two different approaches have been provided to address this issue: the first approach relies on techniques for rewriting the query using weaker conditions, in order to get a larger answer set [8, 10, 15]; the second approach exploits quantitative or qualitative preferences in order to relax the query

[*] Extended Abstract

and returning the best results, leading to the definition of top-k [7] and skyline queries [3]. Skyline queries rely on qualitative preferences and determine best results in terms of a partial relation among items, defined as a dominance relation with respect to a set of given attributes (representing the user preference), by returning those items that are not dominated by any other item (*skyline items*).[1] With respect to the second group of approaches, *relaxation skyline* (r-skyline) queries have been proposed in the relational context with the aim of using some system-defined preferences for relaxing selection and join conditions and avoid the empty answer problem [9]. The basic idea of r-skyline queries is to use a relaxing function (usually, a numeric function) to quantify the distance of each tuple (pair of tuples in case of join) from the specified conditions and to rely on a skyline-based semantics to compute the results. The relaxed evaluation of the query thus provides a non-empty answer while being close to the original query formulated by the user. Relaxation skyline queries have been proposed for stored relational data [9] and a similar approach has been provided for sensor networks [11].

While the empty answer problem has been deeply investigated for stored data, few proposals exist for data stream management. In this context, as discussed in [5], several approximation approaches have been proposed to deal with limited or constrained resource availability during data stream processing. However, only few approximation techniques finalized at improving the quality of result, either in terms of completeness or accuracy, have been defined. Such techniques would however be very useful in the streaming context, where the limited knowledge of the users about the actual data arriving on the stream may often lead to the execution of queries returning an unsatisfactory answer, e.g., queries that return an empty result over several windows.

This paper addresses this problem by presenting some preliminary results of an on-going research concerning the definition of relaxation queries and related processing techniques for data streams. In particular, the contribution of the paper concerns the definition of r-skyline queries for window-based join over data streams and a related processing algorithm. The proposed algorithm has been obtained by integrating the approaches presented in [12], for processing skyline queries over a single data stream, and in [9], for processing r-skylines for stored relational data, and extending them to deal with r-skylines over selection and window-based join queries. A preliminary experimental evaluation of the designed technique is also presented, showing the impact of relaxation on performance.

The paper is organized as follows. Section 2 introduces basic concepts on data streams and presents r-skyline queries. A processing algorithm for r-skylines is then presented in Section 3 while preliminary experimental results are reported in Section 4. Some concluding remarks about the obtained results and next steps we plan to follow in our research are then provided in Section 5.

[1] Given a set of points, each corresponding to a list of values for the relevant attributes, a point A dominates a point B if it is better in at least one dimension and equal or better in all the others, with respect to some ordering [3].

2 Relaxation skyline queries for data streams

A data stream is a continuous, unbounded, and potentially infinite sequence of data (tuples, in this paper). In a data stream, each item is associated with a timestamp, either assigned by the source dataset or by the system at arrival time. Queries over data streams can be either one-time, if they are evaluated once on a given subset of data, or continuous, if they are continuously evaluated as soon as new data arrive. According to STREAM [1], continuous queries can be evaluated over data streams and time-varying relations. A time-varying relation is a function that maps each time value to a set of tuples, representing the relation state (i.e., a classical relation) at a certain time instant. Continuous queries are evaluated at each time instant on the relation states and on the subsets of the data streams available at that instant. Window operators are applied on streams in order to compute, at each time instant, a subset of the items arrived so far. Windows can be either time-based, if all the tuples arrived in the last k time units are retained, or count-based, in case the last k tuples are retained.

Example 1. Consider an e-commerce application, dealing with two data streams, `Order` and `Delivery`, containing tuples concerning customer orders (orderID, customer, cost, dateOrder) and product delivery (orderID, clerk, dateOrder). We are interested in determining, in a continuous way, information about orders, whose cost is equal to 500 Euro, related to deliveries performed in the last 3 minutes. The corresponding STREAM's CQL [2] query is:

```
SELECT o.orderID, o.customer
FROM Order o [unbounded], Delivery d [RANGE 3 Minutes]
WHERE o.cost = 500 and o.orderID = d.orderID
```

In the query, `[unbounded]` and `[RANGE 3 Minutes]` are two window operators. At each time instant, the first one returns the set of tuples arrived from the beginning of the stream up to now; the second one returns the set of tuples arrived in the last 3 minutes. $\diamondsuit$

The concept of relaxation skyline (r-skyline), first introduced in [9] for stored data, extends the concept of skyline query [3] to deal with derived attributes, each representing the distance of the considered tuple (pair of tuples, in case of join) to a condition contained in the query. In the following, we consider queries corresponding to a binary join over two data streams, followed by a number of selections. Distances between tuples and query conditions can be computed using a relaxation function defined as follows.

Definition 1 *Let S and T be two data streams. Let $C_1 = S.A_i\ \theta\ v$ and $C_2 = S.A_i\ \theta\ T.A_j$, where θ is a comparison operator, A_i is an attribute of S, A_j is an attribute of T, and v is an allowed value for attribute A_i. Let s be a tuple in S and t a tuple in T. Function RELAX over (s, C_1) returns 0 if $s.A_i\ \theta\ v$, $|s.A_i - v|$ otherwise. Function RELAX over (s, t, C_2) returns 0 if $s.A_i\ \theta\ t.A_j$, $|s.A_i - t.A_j|$ otherwise. In the following, $RELAX(s, t, Q)$, $s \in S$, $t \in T$ denotes the values of function RELAX computed for each condition in Q with respect to tuples s and t.* $\square$

tuple	orderID	customer	cost	dateOrder	τ
o$_1$	0001	James	600	07/11/2010	1
o$_2$	0002	Thomas	150	07/11/2010	2
o$_3$	0003	Thomas	300	07/11/2010	3
o$_4$	0004	Nick	650	07/11/2010	4
o$_5$	0005	Nick	500	09/11/2010	5
o$_6$	0006	Nick	100	09/11/2010	6
o$_7$	0007	James	300	10/11/2010	7
...	...	...	...	...	...

tuple	orderID	clerk	dataOrder	τ
d_1	0001	Simon	08/11/2010	3
d_2	0002	Jack	13/11/2010	4
d_3	0003	Jack	13/11/2010	5
d_4	0004	Simon	14/11/2010	6
d_5	0005	Simon	14/11/2010	7
d_6	0006	Simon	14/11/2010	8
d_7	0007	Jack	15/11/2010	9
...	...	...	...	...

Table 1. Data streams `Order` (left) and `Delivery` (right)

The following definition adapts the definition of dominance and relaxation skyline given in [9] to data streams.

Definition 2 *Let Q be a query, S and T be two data streams, s_1 and s_2 be tuples of S, t_1 and t_2 be tuples of T, w_1 and w_2 be two window operators. The pair of tuples $\langle s1, t1\rangle$ dominates ($\preceq$) the pair $\langle s2, t2\rangle$ if: (i) relaxed values in $RELAX(s_1, t_1, Q)$ are lower than or equal to all the corresponding relaxed values in $RELAX(s_2, t_2, Q)$; (ii) at least one relaxed value in $RELAX(s_1, t_1, Q)$ is lower than the corresponding relaxed value in $RELAX(s_2, t_2, Q)$.*

The r-skyline of S and T with respect to Q, w_1 and w_2, denoted as $rs(S, T, Q, w_1, w_2)$, is the relation that at time τ contains the tuples in $(S[w_1] \times T[w_2])(\tau)$ that are not dominated by any tuple in $(S[w_1] \times T[w_2])(\tau)$. □

Example 2. Consider Example 1 and the relations in Table 1, representing a portion of data streams `Order` and `Delivery`. Timestamps τ are progressive integer numbers, corresponding to the arrival minute. When applying the relaxation function $RELAX$ to condition $C \equiv$ `Orders.cost` = `500` and tuples o_1, o_2, o_3, the following values are obtained: $RELAX(o_1,C_1) = 100, RELAX(o_2,C_1) = 350, RELAX(o_3,C_1) = 200$. Now consider the query of Example 1. Relaxing only condition C (since the join condition relies on order identifiers, which cannot be approximated in a relevant way), r-skyline computation is performed as follows:

- at times $\tau = 1$, $\tau = 2$, the result set is empty (no join);
- at $\tau = 3$, join execution returns just one pair $\langle o_1, d_1\rangle$, which is in turn returned as result of the r-skyline at time 3;
- at $\tau = 4$, the matching pairs are $\langle o_1, d_1\rangle$ and $\langle o_2, d_2\rangle$; based on Definition 2, $\langle o_1, d_1\rangle \preceq \langle o_2, d_2\rangle$ and $\langle o_1, d_1\rangle$ is returned as result of the r-skyline at time 4;
- at $\tau = 5$, the matching pairs are $\langle o_2, d_2\rangle$ and $\langle o_3, d_3\rangle$, tuple d_1 has already expired; based on Definition 2, $\langle o_3, d_3\rangle \preceq \langle o_2, d_2\rangle$ and $\langle o_3, d_3\rangle$ is returned as result of the r-skyline at time 5. ◊

3 Relaxation skyline processing over window-based joins

In this section, we describe a processing algorithm (denoted by NRJL - Non Relaxed Join Lazy) for r-skyline queries over data streams. The proposed algorithm is obtained by merging the Pruning Join algorithm presented in [9] for

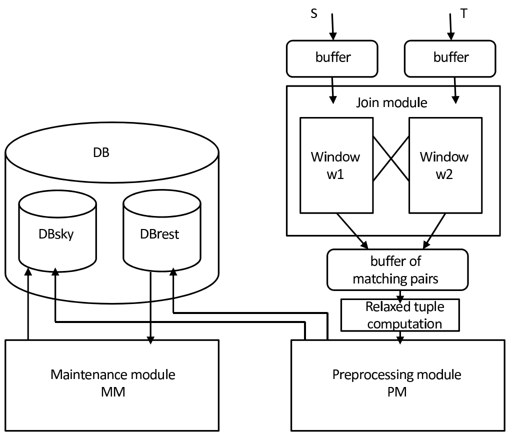

Fig. 1. NRJL Reference architecture

relational data with the Lazy method presented in [12], for skyline computation over data streams. In the following, we assume that all conditions in the query are relaxed except join conditions. This approach is useful, for instance, when join conditions rely on automatically generated semantic-less identifiers.

Fig. 1 shows the architecture at the basis of our technique. To achieve independence on the tuple input rate, arriving tuples are placed in an input buffer (an alternative solution would be that of discarding some input tuples, relying on some load shedding approach [13], or summarizing arrived tuples through synopses [4]). The architecture can be divided into two parts. The first part executes the window-based join of the input streams, S and T, considering two arbitrary windows, w_1 and w_2, one for each stream. Any join algorithm can be employed. In the following, we assume that *symmetric hash join* [14] is employed. This algorithm keeps a hash table for each data stream, indexing the tuples in the current window. Each time a new tuple arrives, it is added to the corresponding hash table and *probed* against the other hash table. Pairs of tuples generated by the join module are stored in a buffer, in ascending order of their arrival time. Arrival and expire times of pairs of matching tuples depend on the arrival and expire time of the composing tuples. More precisely, if the join algorithm generates the pair $\langle s,t\rangle$ and a^s, a^t, w_1, w_2 are the arrival times and windows of S and T, respectively, then the arrival time of the pair is $a = \max(a^s, a^t)$ while the expiration time of the pair is $e = \min(a^s + w_1, a^t + w_2)$.

After computing the join, the second part of the architecture deals with r-skyline computations over window-based join results. Skyline computation is realized through two main modules: a *pre-processing module* (PM) and a *maintenance module* (MM). Module PM is activated as soon as a new pair of tuples $\langle s,t\rangle$ is returned by the join module. It performs two main tasks: (i) relaxing selection conditions by generating one d-dimensional tuple u, corresponding to RELAX(s,t,Q) (see Definition 1); (ii) storing u inside a database DB, partitioned into DB_{sky} and DB_{rest}, which store tuples that are and are not in the current skyline, respectively. These last tuples are maintained since they may

Query	W	400	800	1600	3200	6400
Q_2	**uniform**	2	2	3	3	7
Q_2	**anticorrelated**	2	7	11	11	10
Q_4	**uniform**	7	7	12	10	8
Q_4	**anticorrelated**	16	19	21	21	22

Table 2. DB_{sky} average sizes corresponding to varying window sizes

become skyline points after some skyline point expires (see [12] for details). More precisely, if u is dominated by tuples in DB_{sky}, u is inserted in DB_{rest}, otherwise tuples dominated by u in DB_{sky} are dropped and u is inserted in DB_{sky} as a new skyline point.

Module MM handles expiring tuples. It is invoked each time a skyline tuple u in DB_{sky} expires. As first step, MM drops u from DB_{sky}. At this point, some tuples in DB_{rest} may become new skyline tuples. To perform such computations, MM, as a second step, drops all expired tuples from DB_{rest} and determines the tuples in DB_{rest} dominated by u. For each of such tuples r', it checks whether other tuples in DB_{sky} exist dominating r'. r' is added to set U only if no other tuple in DB_{sky} dominates it. Finally, the skyline of set U is computed, to find the tuples to be moved from DB_{rest} to DB_{sky}.

In order to efficiently performs search operations against repositories, we assume relaxed tuples inside DB_{sky} and DB_{rest} are indexed using R-trees [6].

4 Preliminary experimental results

The goal of the experimental evaluation is to investigate the impact of relaxation on performance. To this aim, we consider: (i) processing time of individual tuples; (ii) amortized processing time, computed as the sum of processing times of individual tuples, divided by the number of processed tuples.

In the experimental evaluation, data streams consists of tuples with three attributes. Each attribute has the interval $[0, 1]$ as domain. Experiments have been executed on queries composed by a join and several selections (equality checks with 0). We suppose that the time-based window is recomputed upon each new tuple arrival. Windows of size 400, 800, 1600, 3200, 6400 seconds have been considered. Due to the presence of buffers, the experimental evaluation does not depend on the tuple arrival frequency; however, we considered a tuple arrival every 5 seconds. Assuming an alternate arrival between the two streams, this means that each stream produces a tuple every 10 seconds.[2] Finally, we assumed that each stream contained 30 windows, resulting in a minimal stream size of 1200 tuples and a maximal stream size of 19200 tuples.

Input data have been generated through an automatic generator, following the techniques proposed in [3], using two distinct distributions: *uniform* and

[2] We remark that, due to the join computation, the tuple arrival frequency for modules PM and MM will be much higher and, every second, it will correspond to the average number of matching pairs (about 20 in the performed experiments).

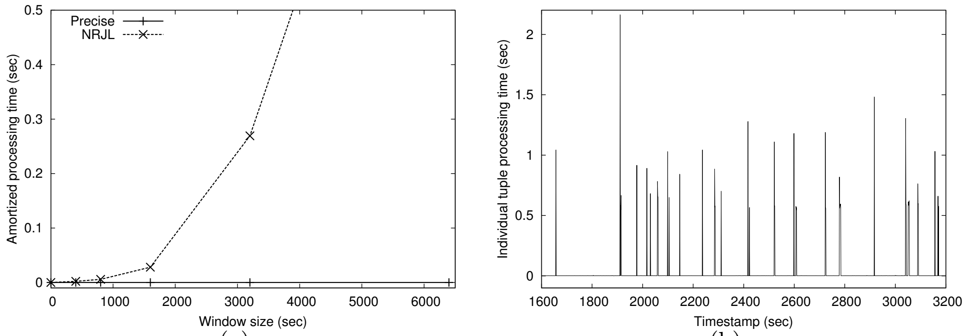

Fig. 2. (a) Amortized cost at varying window size for uniform distribution; (b) Individual tuple processing time for uniform distribution

anticorrelated. Skyline size and thus of processing time [12] are known to be badly influenced by anticorrelated distributions. In our context, since the skyline is computed on join result, an anticorrelated distribution could be imposed on join result, and not only on the inputs. This experiment is left as future work.

Two query types have been considered: *Q4*, containing one selection condition for each attribute and all them are relaxed (for a total of four relaxed conditions), and *Q2*, containing one selection condition for each attribute and only one of them is relaxed for each stream (for a total of two relaxed conditions).

Exp. 1. The average size of the r-skyline in each time instant, corresponding to the average size of DB_{sky}, is determined. The size grows in the number of relaxed conditions: each query type corresponds to a skyline computation on tuples of a given dimension and the skyline size is known to grow in the dimension of the input tuples. Table 2 shows the average dimension of DB_{sky} depending on data distribution and window size. The skyline points for the uniform distribution are always fewer than or equal to those for the anticorrelated distribution.

Exp. 2. Amortized time of NRJL is compared with that of the precise (i.e., without condition relaxation) query. Precise queries are executed as follows: as soon as a new tuple r arrives, we check whether r meets the selection conditions stated for the stream. If all such conditions are met, the symmetric hash join is executed, obtaining the tuples matching r. Fig. 2(a) shows the variation in the average amortized time (in seconds) for queries executed over the various window sizes, for the uniform distribution (results for anticorrelated distribution are similar). The precise query processing time is considerably lower than the time required for relaxed execution. In the precise processing, indeed, no auxiliary data structures, need to be maintained and no relaxation is applied.

Exp. 3. The processing time for each single tuple, starting from the time instant at which the join window becomes full, is analyzed. Fig. 2(b), represents the average individual tuple processing time for NRJL. All processing times are not null, even if, due to a scale problem, they look as they were. Peaks represent the activations of the MM module. Similar results were obtained for the anticorrelated distribution.

ISBN: 978-88-96477-23-6, Copyright (c) 2012 - Edizioni Libreria Progetto and the authors

5 Discussion and conclusion

An ongoing work concerning the processing of relaxed queries over data streams has been presented. The concept of relaxation skyline has been extended to data streams and a processing algorithm has been described. Preliminary experimental results show that, as expected, relaxing queries penalizes performance. Performance can be improved in at least two ways, currently under investigation. One approach consists in designing more efficient algorithms for relaxation skyline computation. Algorithms anticipating some skyline computation during window-based join execution, or reducing the size of the maintained state information, should be investigated. The second direction is to adopt an adaptive processing approach to switch from precise queries to skyline-based ones as soon as, based on some dynamically monitored QoD parameters, the system understands that this is needed for improving result quality. A switch from a skyline-based computation to a precise one will occur as soon as parameters indicate that a precise computation can generate a satisfactory result.

References

1. A. Arasu et al. STREAM: The Stanford Stream Data Manager. *IEEE Data Eng. Bull.*, 26(1):19–26, 2003.
2. S. Babu and J. Widom. Continuous Queries over Data Stream. *SIGMOD Record*, 30(3):109–120, 2001.
3. S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *ICDE*, pages 421–430, 2001.
4. F. Buccafurri and G. Lax. Approximating Sliding Windows by Cyclic Tree-like Histograms for Efficient Range Queries. *Data Knowl. Eng.*, 69(9):979–997, 2010.
5. B. Catania and G. Guerrini. Approximate Queries with Adaptive Processing. In B. Catania and L. Jain, editors, *Advanced Query Processing - Issues and Trends.* Springer, 2012.
6. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD Conference*, pages 47–57, 1984.
7. I. F. Ilyas, G. Beskales, and M. A. Soliman. A Survey of Top-k Query Processing Techniques in Relational Database Systems. *ACM Comput. Surv.*, 40(4), 2008.
8. A. Kadlag et al. Supporting Exploratory Queries in Databases. In *DASFAA*, pages 594–605, 2004.
9. N. Koudas et al. Relaxing Join and Selection Queries. In *VLDB*, 2006.
10. C. Mishra and N. Koudas. Interactive Query Refinement. In *EDBT*, 2009.
11. L. Pan, J. Luo, and J. Li. Probing Queries in Wireless Sensor Networks. In *ICDCS*, pages 546–553, 2008.
12. Y. Tao and D. Papadias. Maintaining Sliding Window Skylines on Data Streams. *IEEE Trans. Knowl. Data Eng.*, 18(2):377–391, 2006.
13. N. Tatbul et al. Load Shedding in a Data Stream Manager. In *VLDB*, pages 309–320, 2003.
14. A. N. Wilschut and P. M. G. Apers. Dataflow Query Execution in a Parallel Main-Memory Environment. In *PDIS*, pages 68–77, 1991.
15. X. Zhou et al. Query Relaxation using Malleable Schemas. In *SIGMOD Conference*, pages 545–556, 2007.