Reverse Nearest Neighbor Aggregates Over Data Streams

Flip Korn
AT&T Labs–Research
flip@research.att.com

S. Muthukrishnan
AT&T Labs–Research
muthu@research.att.com

Divesh Srivastava
AT&T Labs–Research
divesh@research.att.com

Abstract

Reverse Nearest Neighbor (RNN) queries have been studied for finite, stored data sets and are of interest for decision support. However, in many applications such as fixed wireless telephony access and sensor-based highway traffic monitoring, the data arrives in a stream and cannot be stored. Exploratory analysis on this data stream can be formalized naturally using the notion of RNN aggregates (RNNAs), which involve the computation of some aggregate (such as `COUNT` or `MAX DISTANCE`) over the set of reverse nearest neighbor "clients" associated with each "server".

In this paper, we introduce and investigate the problem of computing three types of RNNA queries over data streams of "client" locations: (i) Max-RNNA: given K servers, return the maximum RNNA over all clients to their closest servers; (ii) List-RNNA: given K servers, return a list of RNNAs over all clients to each of the K servers; and (iii) Opt-RNNA: find a subset of at most K servers for which their RNNAs are below a given threshold. While exact computation of these queries is not possible in the data stream model, we present efficient algorithms to approximately answer these RNNA queries over data streams with error guarantees. We provide analytical proofs of constant factor approximations for many RNNA queries, and complement our analyses with experimental evidence of the accuracy of our techniques.

Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

1 Introduction

Reverse nearest neighbor (RNN) queries have been studied recently for finite, stored data sets (see, e.g., [18, 28, 26]), and provide a natural way of identifying the "influence" of a query point on the database, for example, based on the geographical proximity of a new store outlet to its potential customers, or the vector-space similarity of a new service offering to its client profiles. Intuitively, the reverse nearest neighbor set of a "server" query point q is the set of "client" points in the database for which q is the nearest server[1], for example, the set of customers for which the new store outlet is the closest store, or the set of users whose profiles are more similar to the new service offering than to any other service. Often, what is of interest is not the exact RNN set, but *aggregates* on this set, for example, the number of RNN clients of the new service offering, or the median distance between the new store outlet and its RNN customers. These aggregates are very useful for decision support, and we refer to them as *reverse nearest neighbor aggregates* (or RNNAs).

In many applications, the client data arrives in streams, and decision support tools for these applications need to be able to compute answers to *ad hoc* aggregate (in particular, RNNA) queries in an online fashion. We briefly discuss two such applications next.

Fixed Wireless Telephony Access: In this application, fixed wireless base stations are installed at street intersections in towns, or along highways, to enable residential telephony customers to directly connect to the long distance network. Each base station has a region of coverage, and the system

[1] [18] refers to this as the bichromatic case.

is typically overengineered so that overlaps between coverage regions of base stations are large. Call initiations and terminations in this application arrive in *streams*, and are handled by the closest available base station, which can determine the location of the originating call.

A decision support system for this fixed wireless telephony access application may want to continuously determine the worst-case "signal strength" for the base stations (based on the maximum distance between the base station and a customer), and the "load" on the base stations (based on the number of calls handled by the base station). It may also need to optimize the performance of the system of base stations by dynamically turning them on and off.

These problems are naturally expressed using the notion of reverse nearest neighbor aggregates. The fixed wireless base stations in this application are analogous to "servers", while call initiation and termination requests are analogous to "clients". Since each call is handled by the closest available base station, the number of calls handled by the base station is a reverse nearest neighbor aggregate (the `RNN COUNT`), as is the worst-case signal strength (the `MAX` over all the `RNN MAXDIST`s). One may wish to ask "what-if" queries: for a specified subset of K base stations, and the given call stream, what would be the load or worst-case signal strength at each of these base stations? One optimization problem involves the identification of K base stations such that, if all other base stations were unavailable, some RNNA (such as `RNN COUNT` or `RNN MAXDIST`) is minimized.

Highway Traffic Monitoring: In this application, thousands of sensors are deployed on highways [19]. They detect vehicles that pass over them, and provide estimates of vehicle speed and length, which can be used to infer aggregate flow and volume information on a stretch of the highway. User queries in this application arrive in *streams*, and request for traffic conditions at a particular location on the highway, receiving periodic frequent updates based on the data monitored by the closest available sensor, until the query is explicitly terminated by the user.

A decision support system for this traffic monitoring application may want to continuously determine the "accuracy" of the information provided (based on the distance between a user query location from the sensor location), and the "load" on the system (based on the number of user queries for each sensor). It may also want to identify which sensors to keep idle and which to keep active, under natural accuracy or load constraints.

Again, these problems are very naturally expressed using the notion of reverse nearest neighbor aggregates. The sensors in this application are analogous to "servers", while user queries are analogous to "clients". The number of user queries associated with a sensor is its `RNN-COUNT`, and the worst-case accuracy of answers provided to user queries is the maximum over all the sensor `RNN-MAXDIST`s.

1.1 Contributions and Overview

In this paper, we introduce and investigate the problem of *RNNA computation* over data streams consisting of "client" locations, for a static set of available "servers", in particular: (i) Max-RNNA: given K servers, return the maximum RNNA over all clients to any of the servers; (ii) List-RNNA: given K servers, return the RNNA over all clients to each of the K servers; and (iii) Opt-RNNA: find a set of at most K servers for which their RNNAs are below a given threshold. We make the following technical contributions:

- While exact computation of these RNNAs is not possible in the data stream model, we present efficient algorithms to approximately answer them over data streams with worst-case error guarantees.
  - For `Max-RNN-COUNT`, we present a 3-approximation; when clients are inserted but not deleted, we provide a $(1+\epsilon)$-approximation. For `Max-RNN-MAXDIST`, we present a $(1+\epsilon)$-approximation.
  - For `List-RNN-COUNT` (and `List-RNN-MAXDIST`), we obtain lower- and upper-bound estimates for each server as functions of the true counts (resp., true `MAXDIST`s) for the server and its adjacent servers.
  - For `Opt-RNN-COUNT`, we obtain an 8-approximation; for `Opt-RNN-MAXDIST`, we present a $(1+\epsilon)$-approximation.

  For all of these, the space used by our techniques is near-linear in the number of available servers. These results use various count partitioning and space partitioning techniques.
- We complement our analyses of RNNA computation with experimental results, using a variety of real and synthetic data sets, to better understand the behavior of the various count partitioning and space partitioning techniques in practice (Section 5).

The outline of this paper is as follows. We discuss related work in Section 2, formally define the problems in Section 3, describe the algorithmic solutions in Section 4, present our experimental results

in Section 5, and conclude with a summary of our contributions and directions for future work in Section 6.

2 Related Work

Our paper is the first work to study the problem of computing reverse nearest neighbor aggregates over data streams. Related work can be broadly classified into two areas: algorithms over data streams, and algorithms for computing reverse nearest neighbors over a conventional database system. We discuss these in turn.

Data stream algorithms have been of much recent interest, both in the theory as well as in the database communities. The first results were the results of Munro and Patterson [22], who studied the space requirements of selection and sorting as a function of the number of passes over the data; data stream algorithms are one pass algorithms. The data steram model was formalized by Henzinger et al. [14] and Gilbert et al. [9].

Alsabti et al. [1], Manku et al. [20, 21], and Greenwald and Khanna [12] considered the problem of computing the approximate median and other quantiles in a single pass over a data set. (We make use of their algorithm in this paper.) More recently, Gilbert et al. [11] considered the problem of computing approximate online quantiles with probabilistic guarantees over a data stream of insertions and deletions. Gilbert et al. [10] have considered the problem of histogram construction over a data stream. (There is a vast literature on the problem of efficient and accurate histogram construction, not on data streams, which we don't mention here.)

Recent works by Gehrke et al. [8] and Gilbert et al. [9] have looked at maintaining summary structures for maintaining approximate aggregates over a data stream. RNNAs differ from these aggregates (including the correlated aggregates of [8]) in that the association of "tuples" (clients, in our case) with "groups" (servers, in our case) is not fixed, and the same "tuple" can be associated with different groups at different times. This makes it non-trivial to extend the results of [8, 9] for our problem.

There has been recent work on mining data streams, such as the construction of decision trees [7, 6], association rule mining [15], and similarity matching [4]. The clustering algorithms literature is extensive, but only recently have streaming algorithms been studied [3, 13]. They study the k-median clustering problem, which is similar to our problem. However, they study the version with only clients, and their results do not work for the server-client instance we have under data streams.

In general, few results are known in the model of dynamic maintenance, where stream data is discarded as well. Notable exceptions include L_p-norms [16], Hamming norms [4], quantiles [11], and results on maintaining stream statistics over sliding windows [5].

The study of reverse nearest neighbors (RNN) in databases was initiated by Korn and Muthukrishnan [18]. Follow-up work includes the proposal of more efficient access methods for indexing reverse nearest neighbors over finite data sets [28, 26]. However, the problem of efficiently computing reverse nearest neighbor aggregates (RNNAs) has not been studied before, in particular, under space-constrained or data stream models.

3 Preliminaries

We will begin by formalizing the setting. We have a collection of *servers*; server i is located at l_i. We refer to these servers as being *available*; a subset of the available servers may be deemed *active* as specified in the queries, on a query-by-query basis. Let n denote the total number of available servers.

Over time, clients arrive and eventually depart. Each client j has its location L_j associated with it.

For each server i, its *reverse nearest neighbors* is the set of all clients that have i as their nearest neighbor server.[2] The distance function between servers and clients depends on the application. In our case, Euclidean distance seems most appropriate since the servers and clients are spatially located.

The set of all reverse nearest neighbors of a server and their distances from the server are important attributes. While previous work has focused on finding reverse nearest neighbors *sets*, the focus here is on maintaining various statistics on them as the clients arrive and depart. We focus on two instances of these aggregates:

1. `RNN-COUNT`(i) is the number of clients currently in the system for which i is the nearest neighbor. This may be thought of as the "load" on server i since, in our motivating applications, all clients in the reverse nearest neighbor of i are "assigned" to the facility at i.

2. `RNN-MAXDIST`(i) is the largest distance to a client that has i as its nearest neighbor. This may be thought of as the "quality" of assigning clients to that server. For example, in the fixed wireless case, this may correspond to the quality of the connection and in the sensor-based traffic monitoring application, this corresponds to the accuracy of the information.

[2] See [18] for more intuition about reverse nearest neighbors.

If we can store the set of all clients in the system at any moment, these aggregates can be computed and maintained using the R-tree based methods in [18, 28, 26]. In practice, the stream of clients is too large to be stored and indexed in memory. More formally, we work in the data stream model of computation [14], wherein space used is much smaller than the number of clients in the stream. Then one cannot obtain exact RNNA values. Therefore, the focus is on maintaining them approximately.

In addition, RNNAs are often issued in an *ad hoc* fashion as a means of understanding the general distribution of clients with respect to active servers. Exploratory data analysis involves choosing a set of active servers and determining the load and accuracy distribution, or finding active servers in order to optimize various load and accuracy constraints. We formalize these problems as follows.

1. Max-RNNA: given K active servers specified in the query, return the maximum RNNA over all clients to their closest active servers;
2. List-RNNA: given K active servers specified in the query, return a list of the RNNA over all clients to each of the K active servers; and
3. Opt-RNNA: choose a set of at most K servers from the available ones to be active, such that the RNNAs of the active servers are below a given threshold.

All three problems above have instances corresponding to the use of `RNN-COUNT`(i) or `RNN-MAXDIST`(i) as the RNNA. Note that in this context, `RNN-COUNT`(i) and `RNN-MAXDIST`(i) are defined with respect to the active servers specified in the query, and not the available servers in the system. The first problem above focuses on finding the worst-case load or quality over a chosen set of active servers. The second problem focuses on obtaining the list of maximum load or worst case quality for each. The third problem focuses on the task of choosing a set of servers to be active based on the current load and quality distribution on the available servers.

In what follows, we provide novel solutions to the problem of maintaining RNNAs as well as supporting the three queries above. Our solutions will use space near-linear in the number of available servers n in the system at any time, which is quite realistic. Our algorithms have *a priori* worst-case approximation guarantees. We employ the following two approaches:

- **Count partitioning**: bucketing the distribution of clients and histogramming based on the counts of clients;
- **Space partitioning**: bucketing the underlying space and histogramming clients based on their locations.

Both space partitioning and count partitioning are done in a variety of ways depending on the constraints of the problem, and whether one seeks provable results or heuristic improvements. In addition, we apply a greedy strategy (with and without backtracking) to obtain our approximation results for these problems.

4 Our Algorithms

In this section we describe algorithms for answering the three problems described in Section 3. We will assume that servers are on a straight line. This is a convenient abstraction of placement of servers along a highway, and will apply for distances along the highway as well. We discuss extensions of our results to the more general case of clients being distributed on the plane in Section 4.3. For simplicity of exposition, we focus on clients being arranged on the same line as the servers. We will first focus on the `RNN-COUNT`(i) instance followed by `RNN-MAXDIST`(i). Table 1 summarizes our results from this section.

4.1 `RNN-COUNT`(i) Instance

We maintain a data structure on the available servers that can be updated fast with client insertions and deletions.

A trivial algorithm is as follows. For each pair of servers (i, j) where $i < j$, we maintain CL_{ij} which is the number of clients with $L_k \in [l_i, \frac{l_i+l_j}{2})$, and CR_{ij} which is the number of clients with $L_k \in (\frac{l_i+l_j}{2}, l_j]$, for each client k; see Figure 1. These counts can be easily maintained when clients arrive and depart. Consider estimating `RNN-COUNT`(i). Let $a_{<i}$ be the closest active server to its left and $a_{>i}$ the closest active server to its right. Then, the answer to the query `RNN-COUNT`(i) is simply $CL_{ia_{>i}} + CR_{ia_{<i}}$. This is the optimum answer.

However, this solution has two drawbacks. First, if we have n servers in all, our data structure requires $O(n^2)$ space (since we do not know *a priori* which of them will be deemed active in any query), which can be prohibitive. Second, each arrival or departure of a client requires $O(n^2)$ updates of counts in the worst case, which can be computationally prohibitive.

In what follows, we will maintain space near-linear in n, and answer the three types of queries above. The aggregate we determine will not be exact but approximate with guaranteed accuracy, as we shall prove.

Our Data Structure: We maintain a simple data structure: $C(i)$ which is the number of clients be-

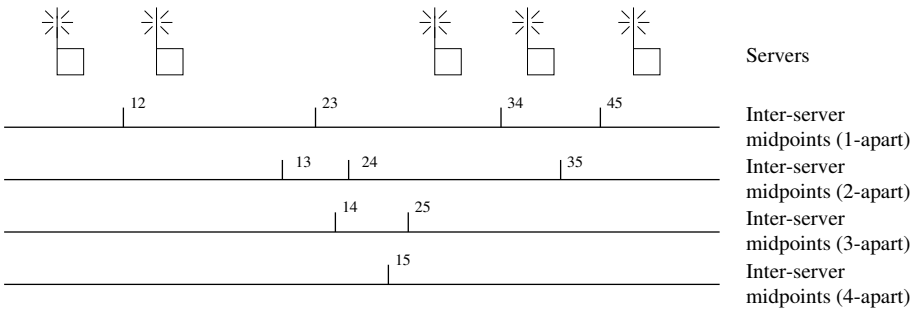


Figure 1: Inter-server midpoints for Computing RNN-COUNTs

tween servers i and $(i+1)$ at any time; $C(0)$ denotes the number of clients to the left of server 1. Thus, the total space is $O(n)$. When client j arrives, we find i such that $l_i \leq L_j \leq l_{i+1}$ and increment $C(i)$; when client j departs, $C(i)$ is decremented. It takes $O(\log n)$ time to find the i value.

Answering Queries: Now we will show how to answer each of the three query types. Say we are presented with the Max-RNNA$(s_1, \ldots, s_K)$ query. Here, $s_1, \ldots, s_K$ are the K servers designated to be active and the goal is to return $\max_i$ RNN-COUNT(s_i). Say the available servers are sorted left-to-right in the order $l_1 \leq \cdots \leq l_n$. The overall steps of the algorithm are as follows:

1. Determine all j such that $l_j \in [l_{s_i}, \frac{l_{s_i}+l_{s_{i+1}}}{2})$ and aggregate their $C(j)$'s. Denoted this by RS_i.
2. Find all j's such that $l_j \in [\frac{l_{s_{i-1}}+l_{s_i}}{2}, l_{s_i}]$ and aggregate $C(j-1)$'s. Denote this by LS_i.
3. Calculate $M_i = LS_i + RS_i$ and output $M = \max_i M_i$.

Theorem 4.1 *Let M^* be the exact answer to query Max-RNNA$(s_1, \ldots, s_K)$ and M be the estimate returned by the algorithm above. We have $M^* \leq M \leq 3M^*$; hence, the algorithm provides a 3-approximation in the worst case.*

Proof: Consider s_{i-1}, s_i and s_{i+1}. Let $J_>$ be the largest j such that $l_j \in [l_{s_i}, \frac{l_{s_i}+l_{s_{i+1}}}{2})$, and $J_<$ to be the smallest j such that $l_j \in (\frac{l_{s_{i-1}}+l_{s_i}}{2}, l_{s_i}]$. Part of $C(J_>)$ belongs to RNN-COUNT(s_i) and the remainder to RNN-COUNT(s_{i+1}); likewise, part of $C(J_< - 1)$ belongs to RNN-COUNT(s_i) and the remainder to RNN-COUNT(s_{i-1}). Therefore, we have $M^* \leq C(J_< - 1) + \cdots + C(J_>) = M$. Also, in each case, the remainder is at most M^*. Hence, $M = C(j_< - 1) + \cdots + C(j_>) \leq 3M^*$, giving the theorem. ∎

Next we consider the List-RNNA$(s_1, s_2, \cdots, s_K)$ query. As before, we determine M_i and output M_i for each s_i in the list. While we cannot give an absolute error guarantee on each M_i, we can provide an approximation guarantee based on the "local" neighborhood of each s_i. In particular,

Theorem 4.2 *Let M_i be the estimate returned by our algorithm for List-RNNA$(s_1, s_2, \cdots, s_K)$ and M_i^* is the corresponding exact answer. We have $M_i^* \leq M_i \leq (M_{i-1}^* + M_i^* + M_{i+1}^*)$.*

The proof is similar to that of Theorem 4.1 and omitted.

Finally, we turn to the Opt-RNNA problem. Given parameter C, our problem is to determine a subset of available servers to be active so that (a) $\max_i$ RNN-COUNT$(s_i) \leq C$ over all active servers s_i in our solution and (b) the number of active servers K is minimized. We will assume that a solution satisfying (a) exists and find the minimum such K; it is easy to modify the algorithm below to detect if (a) cannot be satisfied.

Our algorithm is greedy and proceeds as follows. Assume as before that the available servers are sorted left-to-right, and recall that our data structure maintains $C(i)$, the number of clients located in $[l_i, l_{i+1})$. After having chosen $\kappa < K$ active servers, we have a divider d_κ at one of the server locations l_j, and all active servers have been picked to the left of d_κ. (At the beginning, let $d_0 = l_1$.) Find the smallest server $j^* > j$ such that $\sum_{k=j}^{k=j^*} C(k) > 2C$. There exists at least one server in the interval $[j, j^*]$ that is part of the optimal solution; otherwise, it will follow that more than C clients in the range $[l_j, l_{j^*}]$ will have either the active server immediately to the left of j or the one immediately to right of j^* in the optimal solution as their nearest active server, which violates optimality. We will designate the rightmost server in $[j, j^*)$ as the $(\kappa+1)$th active server, and set $d_{\kappa+1}$ to be l_{j^*} and continue the algorithm.

Theorem 4.3 *Say there is a solution to the Opt-RNNA(C) problem with $\max_i$ RNN-COUNT$(s_i) \leq C$ over all active servers s_i and there are at most K^* active servers. Our algorithm finds a solution of at*

most K^* active servers such that at most $8C$ clients have one of the active servers in the solution as their active nearest neighbor. The running time of this algorithm is $O(K^* \log n)$ where n is the number of available servers.

Proof: Consider j and j^* as above. Let α be the κth active server, β the $(\kappa+1)$th active server (that is, the active server immediately to the left of j^*), and γ the $(\kappa+2)$th active server. We have $\sum_{k=j}^{k=\beta} C(k) < 2C$ by definition of j^*, and likewise $\sum_{k=j^*}^{k=\gamma} C(k) < 2C$. Also, $C(i) \leq 2C$ for any i (in particular, $i \in \{\alpha, \beta, \gamma\}$), since otherwise no solution exists in which each active server s_i has RNN-COUNT(s_i) at most C. We have RNN-COUNT$(\beta) \leq C(\alpha) + \sum_{k=j}^{k=\beta} C(k) + C(\beta) + \sum_{k=j^*}^{k=\gamma} C(k)$ since β may the closest active server for all the clients in the region (l_α, l_γ) in the worst case placement of the clients and servers. Therefore, RNN-COUNT$(\beta) \leq 8C$. The fact that we do not designate more than K^* servers as being active follows since for any region $(d_\kappa, d_{\kappa+1})$, an optimal solution must designate at least one active server, as we do likewise. If we maintain the $C(i)$'s in an array with prefix sums, each active server can be determined using the greedy algorithm above in $O(\log n)$ time. ∎

Note that the algorithm presented above is extremely fast, and gives a reasonably good approximate solution. We can use this as a subroutine to solve the "dual" problem of given a hard upper bound of K on the number of servers to be used, the goal is to minimize the maximum RNN-COUNT(s_i) over all active servers s_i. We can then choose different values of C and run the subroutine above until we get a solution that uses no more than K active servers; this will be a factor 8 approximation to the "dual" problem.

Improved Results for Insert-Only Clients: Consider the case when clients may only arrive (and not depart). This will correspond to the arrival of continuous queries issued by users in the sensor-based traffic application that never get terminated. Our data structures are based on count partitioning.

Say the available servers are sorted left-to-right. We maintain a summary data structure on the servers as clients stream in. Between servers i and $(i+1)$, we will always maintain $C(i)$, the number of clients between servers i and $(i+1)$ at any time. This is the data structure we maintained earlier. In addition, we maintain *ℓ-quantiles*. Greenwald and Khanna show how to maintain an $O(\frac{\log C(i)}{\epsilon})$-space data structure for finding ϵ-approximate quantiles [12]. Using their method, we can identify $c_i^1, \ldots, c_i^\ell$ such that the number of clients lying in $[l_i, L_{c_i^k}]$ is within $(1 \pm \epsilon) k C(i)/\ell$, for any k $(1 \leq k \leq \ell)$ and for ϵ a function of $1/\ell$.

We now describe how queries are answered. We determine the largest j such that $l_j \in [l_{s_i}, \frac{l_{s_i}+l_{s_{i+1}}}{2}]$, and the smallest k such that $l_k \in [\frac{l_{s_{i-1}}+l_{s_i}}{2}, l_{s_i}]$. We also determine largest β such that $L_{c_j^\beta} \in [l_{s_i}, \frac{l_{s_i}+l_{s_{i+1}}}{2}]$, and the smallest γ such that $L_{c_{k-1}^\gamma} \in [\frac{l_{s_{i-1}}+l_{s_i}}{2}, l_{s_i}]$. We return

$$((1/\ell)-\gamma+1)C(k-1)/\ell + \left(\sum_{\alpha \in [\gamma,\beta]} C(\alpha)\right) + (\beta+1)C(j)/\ell$$

as M_i. As an example of query, consider Max-RNNA$(s_1, \ldots, s_K)$. In this case, $\max_i M_i$ is returned as the answer.

Theorem 4.4 *Say M^* is the exact answer to query Max-RNNA$(s_1, \ldots, s_K)$ and M is the estimate returned by the algorithm above. We have $M^*/(1+\delta) \leq M \leq (1+\delta)M^*$, for user-specified constant fraction δ. Hence, the algorithm provides a factor $1+\delta$ approximation in the worst case. The space used is $O((n \log N)/\delta)$ where N is the total number of clients in the system at any time.*

Proof: We have

$$M^* \geq M_i^* \geq \beta C(j)/((1+\epsilon)\ell) + \left(\sum_{\alpha \in [\gamma,\beta]} C(\alpha)\right)$$
$$+((1/\ell)-\gamma+1)C(k-1)/((1+\epsilon)\ell)$$

for all i, because of the $(1+\epsilon)$-approximation of quantiles (and hence the desired counts) at the left and right ends of the interval, as well as the one extra quantile added to the left and the right in the estimation of M_i. Hence, $M^* \geq \max_i M_i/(1+\epsilon) = M/(1+\epsilon)$ giving one side of the inequality we wish to prove.

From our estimation M_i and the inequality above, we have $M_i \leq M_i^*(1+\epsilon) + C(j)/\ell + C(k-1)/\ell$ Also, notice that $M^* \geq C(m)/2$ for any m since at least half of $C(m)$ is closer to the available server to the left or the right. Hence, $M_i \leq M_i^*(1+\epsilon+4/\ell)$ and $M = \max_i M_i \leq M^*(1+\epsilon+4/\ell)$. We can choose ϵ, ℓ such that $M \leq (1+\delta)M^*$ for any constant δ.

The total space used is $O(\sum_i \frac{\log C(i)}{\delta}) = O(\frac{n \log N}{\delta})$, where we have used δ as a pessimistic estimate over ϵ and ℓ that governs the space complexity of the quantile finding algorithm. ∎

Similarly, we can obtain improved results for the List-RNNA and Opt-RNNA queries, as well with the RNN-COUNT(i) instance when clients are only inserted. For the general case when clients may be

deleted as well, the count partitioning data structure above is still useful, but we do not know how to maintain it (in particular the quantiles) in the presence of deletions. Hence, we would not obtain the $(1+\epsilon)$-approximation we have here and have to settle for the 3-approximation we outlined earlier.

4.2 RNN-MAXDIST(i) Instance

In the previous section, we gave small constant approximations for the RNNAs involving counts. In this section, we will present similar algorithms for RNNAs involving distances. Now we need alternate summary structures. In fact, we will use histograms based on space partitioning rather than count partitioning we employed above.

Our Data Structure: As before, say the servers are sorted left-to-right in the order $l_1 \leq \cdots \leq l_n$. We also let U denote the domain size, that is, $U = [\min\{L_j, l_i\}, \max\{L_j, l_i\}]$. We maintain a summary data structure on the available servers as clients stream by. Between servers i and $(i+1)$, we put *dividers* at distances $(1+\epsilon)^j$ from l_i with $\epsilon > 0$, for each possible j; we denote them g_i^j. Likewise, we put dividers at distances $(1+\epsilon)^j$ from l_{i+1} for each possible j and denote them h_{i+1}^j. For convenience, we will let there be dividers at l_i and l_{i+1} as well (each as g's and as h's). There are $O(\log_{1+\epsilon}(l_{i+1} - l_i))$ dividers in all. We overload the notation g_i^k, h_i^k to denote the dividers as well as their positions. We maintain the number of clients that fall between g_i^k and g_i^{k+1} which will be denoted $\#g_i^k$ (likewise for the h's but the count will be denoted $\#h_{i+1}^k$ for appropriate k). We maintain the structure above between every two neighboring servers (not necessarily active) because any subset of servers may be deemed active at any time.

When a client j arrives (or departs), we first determine i such that $l_i \leq L_j < l_{i+1}$. Then we determine g_i^k such that $g_i^k \leq L_j < g_i^{k+1}$. We update $\#g_i^k$ to be $\#g_i^k + 1$ (or $\#g_i^k - 1$ respectively); we do likewise with h_i^k's and update $\#h_{i+1}^k$ for appropriate k's. This takes $O(\log U)$ time to find the g_i^k or h_i^k, and $O(1)$ updates to the data structure.

Answering Queries: Consider the Max-RNNA$(s_1, \ldots, s_K)$ query. Here, $s_1, \ldots, s_K$ are the K servers designated to be active and the goal is to return $\max_i$ RNN-MAXDIST(s_i). Say the servers are sorted left-to-right. The overall steps of the algorithm are as follows:

- Determine the largest g_a^b with nonzero $\#g_a^b$ such that $g_a^b \in [l_{s_i}, \frac{l_{s_i}+l_{s_{i+1}}}{2}]$. We denote $|g_a^b - l_i|$ by RD_i.
- Find the smallest h_e^f with nonzero $\#h_e^f$ such that $h_e^f \in [\frac{l_{s_{i-1}}+l_{s_i}}{2}, l_{s_i}]$. We denote $|h_e^f - l_i|$ by LD_i.
- Calculate $D = \max_i \max\{RD_i, LD_i\}$ and output D.

Theorem 4.5 *Let D^* be the exact answer to query Max-RNNA$(s_1, \ldots, s_K)$ and D be the estimate returned by the algorithm above. We have $D^*/(1+\epsilon) \leq D \leq D^*$; hence, the algorithm provides a $(1+\epsilon)$-approximation in the worst case. The total space used is $O(\sum_i \log_{1+\epsilon}(l_{s_{i+1}} - l_{s_i})) = O(n \log U)$.*

Proof: Consider our algorithm above. Fix active server s_i for discussion. Define $D_i =$ RNN-MAXDIST(s_i). Say there is a client j at position $l_{s_i} + D_i$ (the argument with the client at $l_{s_i} - D_i$ is similar), and no client exists to the right of l_{s_i} that has i as its active nearest neighbor. Clearly then $l_{s_i} + D_i \leq \frac{l_{s_i}+l_{s_{i+1}}}{2}$. Also, there is an (a,b) such that $g_a^b \leq l_{s_i} + D_i < g_a^{b+1}$, $\#g_a^b$ is nonzero and all $g_a^{b+1} \leq g_e^f \leq \frac{l_{s_i}+l_{s_{i+1}}}{2}$ will have $\#g_e^f = 0$ (otherwise, D_i will not be the optimal answer).

Two cases ensue: either $\frac{l_{s_i}+l_{s_{i+1}}}{2} \in [g_a^b, g_a^{b+1}]$ or not. For now, assume the latter. Two subcases ensue. Consider the subcase when our algorithm determines $D = g_a^b - l_{s_i}$ to be answer to the query. Clearly, $g_a^b \leq l_{s_i} + D_i$ implies $D \leq D_i$. Also, $l_{s_i} + D_i \leq g_a^{b+1}$ implies

$$D^* \leq l_a + (1+\epsilon)(g_a^b - l_a) - l_{s_i}$$

since $(g_a^{b+1} - l_a) \leq (1+\epsilon)(g_a^b - l_a)$. It follows that $D_i \leq (1+\epsilon)g_a^b - \epsilon l_a - l_{s_i}$. Since $l_a \geq l_{s_i}$, we have $D_i \leq (1+\epsilon)g_a^b - \epsilon l_{s_i} - l_{s_i} = (1+\epsilon)(g_a^b - l_{s_i}) = (1+\epsilon)D$, proving the lemma in this case. The other subcase is when $D = l_{s_i} - h_e^f \geq g_a^b - l_{s_i}$. It follows that $D \geq D_i/(1+\epsilon)$ using the argument above, and trivially, $D \leq D_i$ as well. That completes the proof of the lemma for the latter case.

For the former case, the same argument holds except that we need to use D^* rather than D_i. This is because when $\frac{l_{s_i}+l_{s_{i+1}}}{2} \in [g_a^b, g_a^{b+1}]$, the nonzero number clients given by $\#g_a^b$ may be the left or to the right of $\frac{l_{s_i}+l_{s_{i+1}}}{2}$ and we do not maintain any information about this outcome in our summary data structure. The outcome affects D_i or D_{i+1} *significantly*, but not $\max\{D_i, D_{i+1}\}$; this helps make the proof complete. Details of this proof will be included in the full version of this paper. ∎

We have used space partitioning, that is, bucketing clients depending on their locations to achieve a provably good approximation above. In particular, we used *exponential* sized buckets since each bucket is $(1+\epsilon)$ times wider than its predecessor.

Problem	*Technique*	*Space Bounds*	*Worst-Case Approximation*
Max-RNN-Count$(s_1, \ldots, s_K)$	Counts	$O(n)$	3
Max-RNN-Count$(s_1, \ldots, s_K)$ (no client deletions)	Counts and quantiles on clients	$O(n \log_{1+\epsilon} N)$	$(1+\epsilon)$
Max-RNN-Maxdist$(s_1, \ldots, s_K)$	Exponential histogram	$O(n \log_{1+\epsilon} U)$	$(1+\epsilon)$
List-RNN-Count$(s_1, \ldots, s_K)$	Counts	$O(n)$	$M_i^* \leq M_i \leq (M_{i-1}^* + M_i^* + M_{i+1}^*)$
List-RNN-Maxdist$(s_1, \ldots, s_K)$	Exponential histogram	$O(n \log_{1+\epsilon} U)$	$D_i^*/(1+\epsilon) \leq D_i \leq \max\{D_{i-1}^*, D_i^*, D_{i+1}^*\}$
Opt-RNNA(C)	Counts	$O(n)$	8
Opt-RNNA(D)	Exponential histogram	$O(n \log_{1+\epsilon} U)$	$(1+\epsilon)$

Table 1: Summary of results.

Now consider the List-RNNA$(s_1, s_2, \cdots, s_K)$ query. The goal is to output $D_i^* =$ RNN-MAXDIST(s_i) for $i = 1, \ldots, K$. As before, we determine $D_i = \max\{RD_i, LD_i\}$ and output D_i for each s_i in the list. While we cannot give an absolute error guarantee on each D_i, we can provide an approximation guarantee based on the "local" neighborhood of each s_i as in the case of RNN-COUNT(s_i) queries. In particular,

Theorem 4.6 *Say D_i^* is the exact answer to query* RNN-MAXDIST(s_i)*, and D_i is the estimate returned by our algorithm for List-RNNA$(s_1, s_2, \cdots, s_K)$. We have $D_i^*/(1+\epsilon) \leq D_i \leq \max\{D_{i-1}^*, D_i^*, D_{i+1}^*\}$.*

The proof is similar to that of Theorem 4.5 and omitted.

Finally, we turn to Opt-RNNA with RNN-MAXDIST(s_i). Formally, we are given a parameter D and our problem is to designate a subset of available servers $\{s_1, \ldots, s_K\}$ as being active so that (a) RNN-MAXDIST$(s_i) \leq D$ over all active servers s_i, and (b) the number of active servers is minimized. We use the same summary structure as above.

The algorithm is greedy but with *limited backtracking*. At any point, we will have a divider α_κ and $\kappa < K$ servers have been made active to the left of α_κ; all clients to the left of α_κ are at a distance of at most $D(1+\epsilon)$ from their closest active servers. We consider the divider $g_a^b > \alpha_\kappa$ such that a is the largest server with $l_a \leq \alpha_\kappa$ and $\#g_a^b > 0$. We have two possibilities. If there exists a server c such that $l_c - g_a^b \leq D(1+\epsilon)$, then we pick the largest such c as the $(\kappa+1)$th active server; we will then find g_e^f such that $g_e^f - l_c \leq D(1+\epsilon)$ and the largest such divider g_e^f will be $\alpha_{\kappa+1}$; the construction continues as before. This is the greedy case. Else, we check if $l_a - g_a^b \leq D(1+\epsilon)$ in which case we make a the $(\kappa+1)$th active server and find g_e^f such that $g_e^f - l_a \leq D(1+\epsilon)$ and the largest such divider g_e^f will be $\alpha_{\kappa+1}$; the construction continues as before. This is the backtracking case. Finally, if neither case holds, then there are no solutions with parameter D.

Theorem 4.7 *Say there is a solution to Opt-RNNA(D) problem with at most K^* active servers. Our algorithm above determines a set of at most K^* active servers such that any client is distance at most $D(1+\epsilon)$, for some constant fraction ϵ. The algorithm takes time $O(K^* \log n)$.* ∎

We defer the proof of the theorem above to the full version of this paper. The crux of the argument is that after we determine the κth active server, the RNN-MAXDIST of it to the right will be larger than that for the leftmost κ active servers in any optimal solution; this holds for any κ including finally K^* which gives the theorem. Again, we can use the subroutine above to solve the "dual" version of the problem where we are allowed a hard upper bound of K and the goal is to minimize the maximum RNN-MAXDIST.

4.3 Extensions

Notice that so far we have assumed that the clients are on the same line (say the x-axis) as the servers. Our results apply to the more general case when the clients are distributed spatially in the plane. Let us focus on the Euclidean distance being the choice of distance between the servers and clients (any L_p distance will also satisfy the following discussion). Answering the three RNNA query types with RNN-COUNT is straightforward because we need to only consider the projection of each client (x, y) onto the x-axis. It is easy to observe that with this modification, all our bounds apply. Next consider the RNN-MAXDIST instance. Now we keep geometric histograms on the intersection of the region between $x = l_i$ and $x = l_{i+1}$ and that of an annulus of radius between $(1+\epsilon)^j$ and $(1+\epsilon)^{j+1}$ with center at l_i. With this modification, our algorithms and analyses for the case of clients being on a line can be adapted

to obtain similar bounds as when they are spatially distributed in the plane.

5 Experiments

We ran an extensive set of experiments on both real and synthetic data to evaluate the "accuracy" of the approximation bounds in practice, as a function of a variety of parameters. We use the average ratio of the exact and estimated answers to evaluate List-RNNA query error, and the worst-case ratio to evaluate that of Max-RNNA and Opt-RNNA queries. In particular, we asked the following questions:

- How close are the *a posteriori* error ratios to the *a priori* bounds?
- How do these errors behave as a function of the number of clients; the number of active servers; the value of ϵ (for `RNN-MAXDIST`); and the optimization constraint?

5.1 Experimental Setup

Data: We used the following data sets: CALIFORNIA, latitudes of 63K buildings in California; MULTIFRAC$_{p,n}$, a binomial multifractal with parameters (p, n), where p determines the 'bias' (for example, when $p = 0.8$ the data follows the so-called '80-20 law'); and UNIF, a uniform distribution. The values in these data streams were normalized to lie in the unit line segment, and they were permuted to arrive in random order.
Queries: We tried all 3×3 combinations of data drawn disjointly from the data sets above for the server and client locations; that is, each data set was partitioned into one subset of servers and one of clients. There were 500 available servers in all experiments, with varying numbers of clients.

5.2 Accuracy for List-RNNA Queries

To gain an understanding of the *a posteriori* error compared to the *a priori* bound for List-RNN-Count, we computed the error ratio $\frac{\hat{C}_i}{C_i}$, where C_i and $\hat{C}_i$ are the exact and approximate `RNN-COUNT`(s_i)'s associated with the s_ith server in the list of K active servers, respectively. We took the average over all s_i having $C_i > 0$, and then averaged over ten randomly chosen configurations of K active servers. Figure 2(a) graphs the error ratio as a function of the number of clients, with K fixed at 200; server and client locations were disjointly drawn from CALIFORNIA. The curve represents the *a posteriori* error. Note that the 3-approximation bound from 4.1 does not pertain to *all* C_i, only to $\max_i C_i$. Nonetheless, the average ratio is still well below 3, at around 1.8. Figure 2(b) presents a graph where we held the number of clients constant at $N = 62$K and increased the number of active servers K. The error appears to increase proportionally with K.

We performed a similar set of experiments for the `RNN-MAXDIST` instance. Figure 3(a) presents graphs of the error ratio as a function of the number of clients, with $K = 200$ and $\epsilon = 1.2$; server and client locations were again drawn from CALIFORNIA. The two curves represent *a priori* and *a posteriori* errors, respectively. The average error factor is computed as $\frac{1}{n}\sum_i \frac{D_i}{\hat{D}_i}$; it is bounded by $(1+\epsilon)$, as illustrated by the curve for the *a priori* bound. As the graphs show, the approximation factor is typically much less than $(1+\epsilon)$. Whereas the bound allows for 20% error, on average the error was below 5%.

We ran experiments to gain a better understanding of how the average error behaves as a function of the tightness of the approximation bound ϵ. Figure 3(b) shows the average approximation factor increasing proportionally with increasing ϵ, but at only one-fifth the rate.

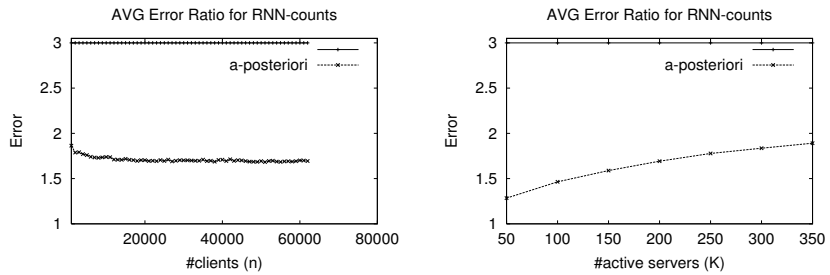
Finally, we considered the effect of the number of active servers on the error. Figure 3(c) presents the plots. The *a posteriori* error appears to increase with increasing K, but levels off.

5.3 Accuracy for Max-RNNA Queries

It was observed above that the *a posteriori* error is significantly lower than the *a priori* bound when averaged over the active servers; this metric is useful to evaluate List-RNNA queries. For Max-RNNA queries, it is more appropriate to evaluate the active server whose RNNA has the largest error. We ran the following experiments to get a sense of how accurate the 3-approximation bound is for the `RNN-COUNT` instance of this query type. We determined the largest exact count $\max_i C_i$ and largest approximate count $\max_j \hat{C}_j$ (over active servers), and plotted the ratio $\frac{\max_j \hat{C}_j}{\max_i C_i}$. Note that this ratio is at least as large as $\frac{\hat{C}_k}{\max_i C_k}$ for which the 3-approximation bound holds.

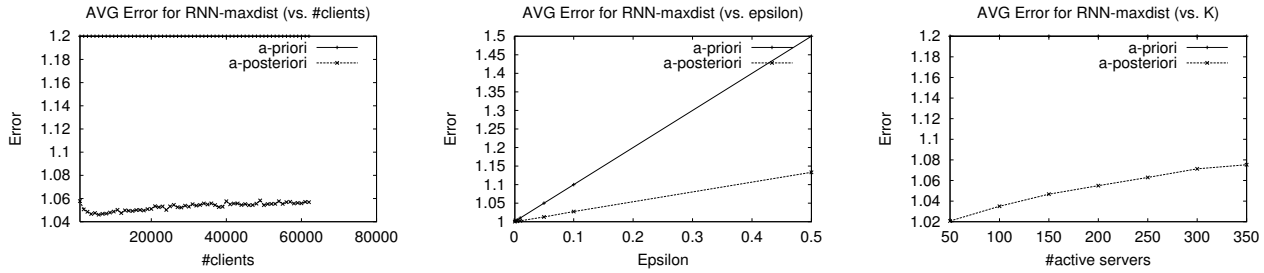
The ratio for servers and clients from CALIFORNIA was between 1.8 and 2.1, as shown in Figure 4(a); the error increases with K in Figure 4(b). Thus, the $\frac{\hat{C}_k}{\max_k C_k}$ error, which is no larger than this, often falls well below the 3-approximation.

We ran a similar set of experiments for the `RNN-MAXDIST` instance. In Figure 5(a), the maximum error ratio $\frac{\max_i D_i}{\max_j \hat{D}_j}$ does not appear to be affected by N; here $K = 200$. In Figure 5(b), the error does not appear to be affected by K; here $N = 62$K.



(a) increasing N (b) increasing K

Figure 2: Average error of List-RNN-Count $\text{AVG}_i(\frac{\hat{C}_i}{C_i})$ as a function of (a) the number of clients (with $K = 200$); and (b) active servers (with $N = 62\text{K}$) over real data (from CALIFORNIA).



(a) increasing N (b) increasing ϵ (c) increasing K

Figure 3: Average error of List-RNN-Maxdist $\text{AVG}_i(\frac{D_i}{D_i})$ as a function of (a) the number of clients (with $K = 200$, $\epsilon = 1.2$); (b) threshold (with $N = 62\text{K}$, $K = 200$); and (c) the number of active servers (with $N = 62\text{K}$, $\epsilon = 1.2$) over real data (from CALIFORNIA).

5.4 Accuracy for Opt-RNNA Queries

We ran the Opt-RNNA query for the `RNN-COUNT` instance, which has an *a priori* bound of $8C$. Let C_i be the exact `RNN-COUNT`(s_i) and C be the optimization constraint. We measured the maximum error ratio $\max_i \frac{C_i}{C}$. Figure 6 presents graphs for this ratio as a function of the number of clients; this ratio is guaranteed to be at most 8. The constraint specified was $C = 500$. For the CALIFORNIA data, the maximum ratio was never greater than 4 and was usually closer to 3. Our experiments did not yield much insight into the relationship between error and C.

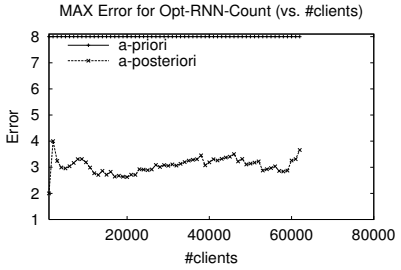
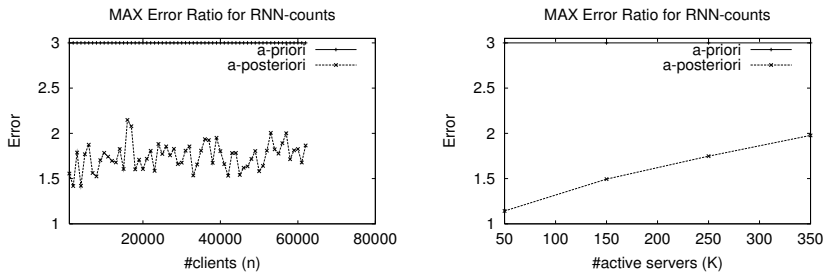


Figure 6: Maximum error for Opt-RNN-Count ($\max_i \frac{C_i}{C}$) as a function of the number of clients (using CALIFORNIA), with $C = 500$.

6 Conclusions and Future Work

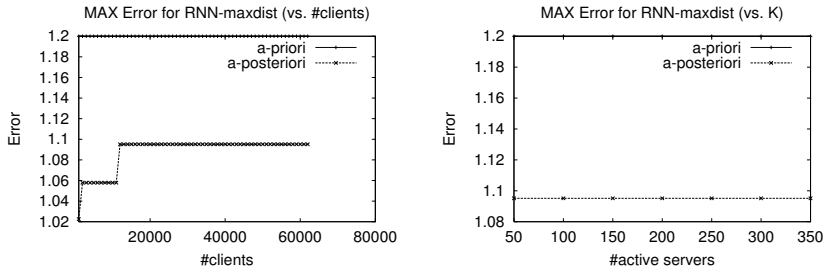
Reverse nearest neighbor aggregates (RNNAs) are of natural interest in decision support systems for applications that compute proximity, based on geographical distance or vector-space similarity, between "servers" and "clients". Applications for RNNAs range from the classical (such as facility location) to the emerging (such as fixed wireless telephony access and sensor-based traffic monitoring). Increasingly, and especially in emerging applications, server and client data arrives in streams, and decision support tools need to be able to compute answers to queries in an online fashion.

In this paper, we introduced and investigated the problem of RNNA computation over a data stream of client arrivals and departures, with a static set of available servers. In particular, we studied three problems (Max-RNNA, List-RNNA



(a) increasing N (b) increasing K

Figure 4: Maximum error for Max-RNN-Count ($\frac{\max_i \hat{C}_i}{\max_j C_j}$) as a function of the number of (a) clients (with $K = 200$); and (b) active servers (with $N = 62$K) over real data (from CALIFORNIA).



(a) (b)

Figure 5: Maximum error of Max-RNN-Maxdist ($\frac{\max_i D_i}{\max_j \hat{D}_j}$) as a function of the number of (a) clients (with $K = 200$); and (b) active servers (with $N = 62$K) over real data (from CALIFORNIA).

and Opt-RNNA) for two aggregates (COUNT and MAXDIST), in the data stream model. While exact computation of these RNNAs is impossible, we presented efficient algorithms to approximately answer them with worst-case error guarantees, using space near-linear in the number of available servers, based on various count partitioning and space partitioning techniques. Our experimental results, based on real and synthetic datasets, provide evidence for the accuracy of our techniques.

There are many promising directions of future work. How can one extend our techniques to deal with *servers* in two and three dimensions? Our space partitioning and count partitioning techniques can be extended quite naturally, but new analyses are needed for provably accuracy guarantees. This may be critical for some applications. As servers become dynamic in RNNA applications, one would need to deal with the case where the set of available servers changes dynamically as well. Our space and count partitioning methods can be extended quite naturally by combining and refining partitions appropriately; again, new analyses are needed. Within decision support systems and databases, there are technically challenging problems in integrating such sophisticated aggregate queries with traditional query languages, and query evaluation engines.

References

[1] K. Alsabti, S. Ranka, and V. Singh. A one-pass algorithm for accurately estimating quantiles for disk-resident data. In *Proceedings of the International Conference on Very Large Databases*, pages 346–355, 1997.

[2] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *2nd International Conference on Mobile Data Management*, Hong Kong, Jan. 2001.

[3] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *ACM STOC*, 1997.

[4] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms. In *Intl. Conf. on Very Large Databases (VLDB)*, Hong Kong, China, Aug. 2002.

[5] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *ACM-SIAM Symposium on Discrete Algorithms (SODA'02)*, San Francisco, CA, Jan. 2002.

[6] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2000.

[7] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-Y. Loh. BOAT-optimistic decision tree construction. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 169–180, 1999.

[8] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2001.

[9] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of the International Conference on Very Large Databases*, Rome, Italy, Sept. 2001.

[10] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, M. Strauss. Fast, small space algorithms for approximate histogram maintenance. In *ACM STOC*, Montreal, Quebec, May 2002.

[11] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. How to summarize the universe: dynamic maintenance of quantiles In *Intl. Conf. on Very Large Databases (VLDB)*, Hong Kong, China, Aug. 2002.

[12] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 58–66, Santa Barbara, CA, May 2001.

[13] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *FOCS*, Nov. 2000.

[14] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. Technical Report 1998-011, Digital Equipment Corporation, Systems Research Center, 1998.

[15] C. Hidber. Online association rule mining. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1999.

[16] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *FOCS*, pages 189–197, 2000.

[17] J. Kahn, R. Katz, and K. Pister. Mobile networking for smart dust. In *MOBICOM*, Seattle, WA, Aug. 1999.

[18] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbors. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 201–212, Dallas, TX, May 2000.

[19] S. Madden and M. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proceedings of the IEEE International Conference on Data Engineering*, 2002.

[20] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 426–435, 1998.

[21] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 251–262, 1999.

[22] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, pages 315–323, 1980.

[23] G. Pottie and W. Kaiser. Wireless sensor networks. *Communications of the ACM*, 43(5):51–58, May 2000.

[24] P. Saffo. Sensors: The next wave of innovation. *Communications of the ACM*, 40(2):92–97, Aug. 1997.

[25] M. Schroeder. *Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise*. W.H. Freeman and Company, New York, NY, 1991.

[26] I. Stanoi, M. Riedewald, D. Agrawal, and A. E. Abbadi. Discovery of influence sets in frequently updated databases. In *Proceedings of the International Conference on Very Large Databases*, Rome, Italy, Sept. 2001.

[27] M. Weiser. Some computer science problems in ubiquitous computing. *Communications of the ACM*, July 1993.

[28] C. Yang and K.-I. Lin. An index structure for efficient reverse nearest neighbor queries. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 485–492, Heidelberg, Germany, 2001.