Spatially-Decaying Aggregation Over a Network

Edith Cohen [*] Haim Kaplan [†]

June 9, 2006

Abstract

Data items are often associated with a location in which they are present or collected, and their relevance or influence decays with their distance. Aggregate values over such data thus depend on the observing location, where the weight given to each item depends on its distance from that location. We term such aggregation *spatially-decaying*.

Spatially-decaying aggregation has numerous applications: Individual sensor nodes collect readings of an environmental parameter such as contamination level or parking spot availability; the nodes then communicate to integrate their readings so that each location obtains contamination level or parking availability in its neighborhood. Nodes in a p2p network could use a summary of content and properties of nodes in their neighborhood in order to guide search. In graphical databases such as Web hyperlink structure, properties such as subject of pages that can reach or be reached from a page using link traversals provide information on the page.

We formalize the notion of spatially-decaying aggregation and develop efficient algorithms for fundamental aggregation functions, including sums and averages, random sampling, heavy hitters, quantiles, and L_p norms.

1 Introduction

In many applications, data items are associated with locations on some network. Data present at one location is relevant to other locations, yet, this relevance or influence decreases with the distance between the locations. Thus each location views and aggregates the data through a different distribution, where the *weight* given to each item decreases with its distance. This dependence on the distance is quantified by a *decay function*, which is a non-increasing function. Some natural decay functions are threshold functions (BALL_r), where items in the r-neighborhood have uniform weights (and other items have 0 weight), Exponential decay, where the weight decreases exponentially with the distance, and Polynomial decay, where the weight decreases polynomially with the distance. While global aggregates assigns equal weight to all items, and result in a single global value, *spatially decaying* aggregates depend on the decay function used and on the "observing" location (the location with respect to which the aggregation is performed).

We consider several fundamental aggregation functions: With *spatially-decaying sum*, each node obtains an estimate on the *sum* of the weight times value product over all items. Figure 1 shows an example of a network with values at each node and the respective decaying sums under the BALL_2 decay function. A related aggregate, derivable from the sum, is the *spatially-decaying average* (a weighted average of values of items). Spatially-decaying random samples are defined with respect to the weight distribution (for example, for a BALL_r decay function a node obtains a uniform random sample from its r-neighborhood). Using

[*] AT&T Research Labs, 180 Park Ave. Florham Park, NJ, USA. Email: edith@research.att.com.

[†] School of Computer Science, Tel-Aviv University, Tel Aviv, Israel. Email: haimk@cs.tau.ac.il. Research supported in part by the German Israeli Foundation (GIF) grant no. 2051-1156-6/2002.

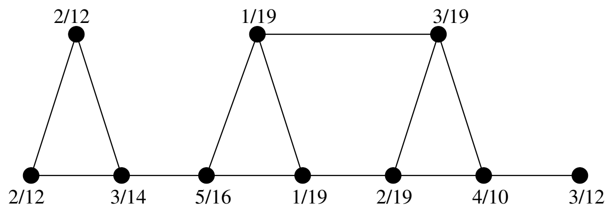


Figure 1: A network where each node is labeled by the value of the item it carries and the sum of values in its 2-neighborhood (BALL_2 decaying sum).

random sampling as a building block we obtain algorithms for the spatially-decaying variants of other basic aggregation problems such as approximate frequency counts [35] and quantiles [37, 36]. It is also interesting to compute aggregates over the set of *distinct elements* (set of distinct values over items with nonzero weight): We consider the counting, multiplicity (frequency) of a random distinct element, random sampling aggregates, and approximate *spatially-decaying L_p norms* for $p \in [1, 2]$ (of vectors where items constitute spatially-decaying updates to certain coordinates [15, 12]).

Our model and algorithms support a wide range of applications, that include distributed or centralized aggregation, one-time computation or periodic to support continuous queries. The desired result of the aggregation process can be a single numeric result at each node or a *summary* that enables a node to respond locally to aggregation queries for different decay functions or aggregate functions. Our algorithms can be modified to perform aggregations over the set of reverse nearest neighbors (RNN). Such aggregates are of interest for decision support and had been studied on stored data sets and on data streams [30, 31]. In these applications, some locations function as "servers" to all other locations, where each location is served by its closest server. Thus, the set of RNNs of a node is its potential set of clients if it becomes a server. Statistics (such as averages, quantiles, random selections, or variance) over these sets for different points can support decisions whether to turn a point into a server.

Applications Our work is motivated by several application areas (both distributed and centralized):

- **Sensor networks:** Sensor networks consist of many individual sensor nodes, each collecting data on its environment, for example, temperature, density of some contaminant, identities of passing vehicles, or availability of a parking spot (see, e.g.[14, 34, 13]). Each reading by one sensor is relevant to other locations, but the relevance, for example of an available parking space, decreases with its distance from the destination [19, 20, 13]. Spatially-decaying aggregation provides each location with information on their larger neighborhood such as parking availability, average or median temperature, the number of distinct vehicles passing through, or the median speed of a vehicle. RNN aggregates can be used to decide when a certain node should start or stop providing some service to its neighboring nodes. Since power is a limiting factor in sensor nets, it is important that aggregation is performed efficiently [33, 38]. Since node distribution can be highly irregular [20, 13], it is important to handle general (or arbitrary planar) topologies.

- **p2p networks:** p2p networks consist of nodes connected via an overlay network. An important functionality they provide is search, where nodes initiate queries for data that can be present at other nodes. Some popular p2p systems (such as the Gnutella protocol [10]) deploy flood searching, where the query is propagated for a fixed number of hops, thus to a neighborhood that includes all nodes that lie within several hops away. Aggregates over the neighborhood that can be used to tune the efficiency of the search process include the number of hosts, the total number of (distinct or not) items offered, the median load or bandwidth of a node, the total number of distinct hosts issuing queries or the total number of queries seen by neighboring nodes. The search process can be improved with each node obtaining a summary of the types of content present in its neighborhood and in the neighborhood of its immediate neighbors [11].

- **Graphical databases:** Many databases have natural graph form with data present at the nodes. On hyperlink graphs of Web pages, the context or importance of a Web page is indicated by aggregate information on pages it links to. For XML documents representing bibliographic data, a context for a new article or book can be deduced by articles authored by collaborators of its author. For road maps, it can be useful to obtain for each location the number of restaurants or gas stations in its proximity. For images, spatially-decaying aggregates can assist in feature extraction.

Challenges: The algorithmic challenge in efficient spatially-decaying aggregations is that aggregate values are location-dependent. Each data item influences many locations, and it influences these locations to different extents. The naive approach is to collect, for each node, values of all items that influence the aggregate (have a nonzero weight) and then compute (from scratch for each node) the aggregate value. This scales poorly, as it can result in a quadratic running time in a centralized setting, and in quadratic communication in a distributed setting (for flooding all values to all locations in which they are considered). It is not hard to see that for many aggregates and decay functions, such quadratic bounds are necessary for obtaining *exact* aggregate values. We thus focus on obtaining *approximate* values. Our approximations are within a small $(1 + \epsilon)$ relative error with very high confidence which suffices for many applications. We develop novel summarization techniques and obtain algorithms that use per-node communication that depends *poly-logarithmically* (rather than linearly) on the size of the *influencing* data. Our centralized algorithms run in close to linear time in the size of the data base.

Related work: The spatial decay model generalizes (from an algorithmic perspective but not so much from an application perspective) the sliding-window model [12, 22] for massive data streams [2] and more generally, the computation of time-decaying aggregates on massive data streams [9]. In particular, the threshold (BALL_r) decay function generalizes sliding windows. There is a straightforward reduction of a spatially decaying aggregate query on a path network to a continuous query (of the same aggregate and under the same decay function) of a time-decaying aggregate (e.g., over a sliding window). Therefore, some lower bounds from the sliding window model carry over to our model. The spatial decay setting, however, is considerably more complex and existing techniques for sliding windows and time-decaying aggregates (such as Exponential Histograms [12] for sums and deterministic summaries [23, 32] for quantiles) do not seem to extend to the general spatial setting. For example, as Exponentially decaying sum, which is trivial to maintain efficiently as a time-decay function [27, 17] is not known to be any easier to track than other decay functions in the spatial-decay model. The spatial decay model also generalizes computation of non-decaying aggregates in a network (e.g. [38]).

The spatial decay model is conceptually related to the spatial gossip model of Kempe, Kleinberg, and Demers [29, 28]. The main difference is that in spatial decay, communication is performed along the edges of an *arbitrary graph* and the cost we consider is per-node communication cost. In spatial gossip protocols, communication is performed by contacting *random* nodes with likelihood that depends on distance, and cost is measured by the time it takes information to spread. In many applications, including sensor networks and graphical databases, node and link placement are given and can be rather irregular, thus, it is important to handle general topologies.

Basic SQL type aggregates over neighborhoods in sensor nets had been recently studied [20, 13]. Deshpande et al [13] considered computing aggregates with respect to a fixed partition, which is less natural for some applications (for example, parking availability within some walking distance of destination is more relevant than availability in some region that contains the destination (eg "Tribecca") and excludes closer spots that are technically outside the region.); the time-decay equivalent of fixed partitions is using fixed (rather than sliding) windows.

Ganeriwal et al [19] did not consider the algorithmic efficiency of performing average aggregates, but proposed Voronoi diagrams as a way of assigning a weight to each sensor reading for a per-area weighted average aggregation, an issue that is orthogonal to ours.

Our algorithms use techniques we developed in [6] to efficiently sum values over neighborhoods, but several key new ideas and techniques were necessary in order to handle different aggregate functions, general decay functions, and to perform the aggregations in a distributed setting.

Our Contributions: We formalize and motivate the notion of *spatially-decaying* aggregation and develop algorithmic techniques to efficiently approximate fundamental aggregates. The aggregates we consider include basic SQL aggregates and had been previously studied over data sets, data streams, or time-decaying aggregation over data streams, but our spatial model is more complex and general and necessitated developing and applying a new set of techniques. Our algorithms are nearly optimal (are within polylogarithmic factors from the lower bounds) and vastly improve over naive approaches.

Outline We organize our presentation as follows. In section 2 we introduce the spatial decay model in its general setting, define the main problems we study and some natural decay functions. Section 3 is concerned with computing decaying sums for any decay function. Section 4 focuses on computing MV/D lists, a basic technique used for computing decaying sums and other aggregate functions. In Section 5 we develop algorithms for random sampling, Section 6 is concerned with approximate L_p norms, and Section 7 with counting distinct values. Section 8 deals with the important case of the Euclidean metric. In Section 9 we present lower bounds based on the relations between spatial decay and sliding windows, in Section 10 we develop a technique based on Exponential Histograms [12] for computing spatially-decaying sums on grids, and in Section 11 we explore a unified model which captures both spatial and time decay. Finally, Section 12 contains an experimental study which compares and evaluates different techniques.

2 Preliminaries

We model the network as a graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ is the set of nodes, and there is an edge between two nodes if and only if the two nodes can communicate. We denote the number of edges by m. Edges can have nonnegative lengths associated with them, which are interpreted as distances. We consider both directed and undirected graphs (symmetric or asymmetric distances). We denote by $\text{DIST}(v_i, v_j)$ the distance between two nodes v_i and v_j with respect to the shortest-path metric on the edge lengths. Nodes in the network have a set of data items $\{(f_i, \ell_i)|i \in I\}$, where for each item $i \in I$, $f_i \geq 0$ is the value of the item (which can generally be a vector or non-numeric entities) and $\ell_i \in V$ is the location of the item.

A *decay function* is a non-increasing function $g(x) \geq 0$ defined for $x \geq 0$. The decay function determines the *weight* of a remote item as a function of its distance. The weight of an item $i \in I$ as viewed by a node $u \in V$ is $w_{u,g}(i) = g(\text{DIST}(u, \ell_i))$. An *aggregate function* is a function $h()$ defined on a multiset of a value-weight pairs. Given a network G with data items $\{(f_i, \ell_i)|i \in I\}$, a decay function $g()$, and a node u, the spatially-decaying $h()$-aggregate at the node u is the value of h on the multiset $\bigcup_{i \in I}\{(f_i, w_{u,g}(i))\}$. Our definitions and results can be generalized to the case where items have arbitrary "initial" weights, w_i^0, and the weight of the item at distance ℓ_i is $w_{u,g}(i) = w_i^0 g((\text{DIST}(u, \ell_i)))$. For simplicity of presentation we subsequently only treat the case where "initial" item weights are uniform.

We denote the value of an aggregate H according to decay function $g()$ at location u by $H_g(u)$. Our goal is to obtain (estimates) of the decaying aggregate value $H_g(u)$ at all locations $u \in V$. We seek algorithms that minimize the amount of communication between nodes, as well as the computation and storage required at each node. The output of the algorithms we develop here is a *summary* at each node u from which an approximate value of the aggregate $H_g(u)$ can be computed for **any** decay function $g()$ (which is stronger and more general than just obtaining values for a particular aggregate and a specific decay function.)

2.1 Aggregate functions

Sum and count The *spatially-decaying sum* is such that the function h sums up the value components $f_i \geq 0$ of the pairs in the multiset, each attenuated by the corresponding weight, that is, the spatially-decaying sum at node u is

$$S_g(u) = \sum_{i \in I} f_i w_{u,g}(i) \; .$$

In the special case where the values f_i are binary, we refer to this aggregate as the *spatially-decaying count*. Our algorithms produce a summary at each node, that allows it to obtain with very high confidence, for *any* decay function $g()$, $(1 \pm \epsilon)$-approximate estimates, $S'_g(u)$, where $\epsilon > 0$ is fixed. That is, $S'_g(u)$ are such that $\frac{|S'_g(u) - S_g(u)|}{S_g(u)} \leq \epsilon$ for every $u \in V$.

Average A related aggregate is the *spatially-decaying average*, defined as

$$A_g(u) = \frac{\sum_{i \in I} f_i w_{u,g}(i)}{\sum_{i \in I} w_{u,g}(i)} \; .$$

The numerator is the spatially decaying sum and the denominator is the spatially-decaying count with respect to the data items $\{(1, \ell_i) \mid i \in I\}$. Thus, an approximate decaying average can be obtained from corresponding approximate decaying sum and approximate decaying count.

Random sampling: For a node u and a decay function $g()$, $R_g(u)$ is a random variable that returns an item $i \in I$ with probability proportional to its weight, that is, the probability that i is drawn is equal to

$$\frac{w_{u,g}(i)}{\sum_{i \in I} w_{u,g}(i)} \; .$$

This can be generalized to weighted sampling with respect to some item weights $f_i \geq 0$, where the probability that i is drawn is

$$\frac{f_i w_{u,g}(i)}{\sum_{i \in I} (f_i w_{u,g}(i))} \; .$$

Each application of our algorithm results in a summary at each node u from which we obtain an item drawn according to $R_g(u)$ for any decay function $g()$. (We allow dependencies between items drawn for different decay functions and different nodes).

Quantiles: For some $0 \leq p \leq 1$ and any decay function $g()$, u can obtain an item with value that is with confidence $1 - \delta$ a $[p \pm \epsilon]$-quantile of the distribution of values over weighted items. Using an existing folklore technique, an approximate quantile with confidence $1 - \delta$ can be obtained by taking the p quantile of $O(\epsilon^{-2} \ln \delta^{-1})$ independent random samples.

Frequency counts (heavy hitters): The *spatially-decaying frequency counts (heavy hitters)* is to find all elements such that the total relative weight of their occurrences is at least $p + \epsilon$. If an element is reported then its total weight is at least $p - \epsilon$. Approximate heavy hitters can be computed from independent random samples [35].

L_p **norms** ($p \in [1, 2]$)**:** Each item $i \in I$ is an update of a coordinate of a d-dimensional vector and is specified by a triplet (c_i, a_i, ℓ_i). The value $c_i \in [d] = \{0, \dots, d-1\}$ specifies the coordinate which this item updates. The value $a_i \in \{0, \dots, M\}$ is the amount by which we increment the target coordinate, and $\ell_i \in V$ is the item's location. The d-dimensional vector $\mathbf{V}(g, u)$ associated with location $u \in V$ is then defined by the coordinate values

$$\mathbf{V}(g, u)_j = \sum_{i|c_i=j} w_{u,g}(i) a_i \ .$$

For a fixed $p \in [1, 2]$, we show how to obtain, using $o(d)$ communication per node, at each location u, a summary of size $o(d)$, such that for any decay function $g()$, the node can obtain an approximate value of $||\mathbf{V}(g, u)||_p$ (the L_p norm of $\mathbf{V}(g, u)$). One ingredient in our solution is a technique by Indyk [26] for computing sketches of vectors which preserve approximate L_p differences (for $p \in [1, 2]$). The problem Indyk considered (for non-decaying data streams) was introduced in [15]. For the problem of computing approximate L_p norms in the sliding window model, Datar et al [12] developed an algorithm which combines Indyk's vector sketches with their Exponential Histogram (EH) technique. Their solution, however, does not seem to generalize to the spatial setting.

Distinct values: For any node u and decay function $g()$, $D_g(u) = |\{f_i \mid w_{u,g}(i) > 0\}|$ is the number of distinct values over items that have a positive weight. We show how each node u can obtain a summary such that for any $g()$ it can obtain an approximate value of $D_g(u)$. This problem is equivalent to its restriction to BALL$_r$ decay functions, that is, for any given $r \geq 0$, obtain an approximate number of distinct values present in the r-neighborhood. The techniques extend to the weighted version of the problem where we are interested in the sum, over applicable distinct elements, of the values of these elements.

We are also able to obtain a *random* distinct element, that is, a distinct element drawn uniformly at random, and (approximate) multiplicity of a random distinct element.

Another family of important aggregates is the (approximate) spatially-decaying variance and moments. In [8] we reduced this problem to computing (approximate) spatially-decaying sums and medians, which are studied here.

2.2 Decay functions

Our results hold for general decay functions. We list families of decay functions that are of particular interest: *Ball decay functions (*BALL$_r$*)* are parameterized by the radius r, and have $g(x) = 1$ for $x \leq r$ and $g(x) = 0$ otherwise. That is, all data values within a distance of r have equal importance and all other data values have 0 weight. *Exponential decay (*EXPD$_\lambda$*)* is such that for a parameter $\lambda > 0$, $g(x) = \exp(-\lambda x)$. With EXPD, the relative significance of each value decreases exponentially with its distance. Exponential decay is commonly used in practice for time-decay. One reason for its common use is that it can be maintained easily as a time-decay function, using a single register (It is not clear, however, if it is simpler to compute than other decay functions in the spatial setting on general networks). *Polynomial decay* (POLYD$_\alpha$) is such that for a parameter α, $g(x) = 1/x^\alpha$. Polynomial decay is often a natural choice when a smooth decay is desired and when Exponential decay is too drastic. In many natural graphs (like d-dimensional grids), the neighborhood size increases polynomially with the distance. For such graphs, EXPD would suppress longer horizons. Many natural effects (for example, electro-magnetic radiation) have polynomial decrease with distance.

2.3 Model assumptions

Spatially-decaying aggregation has applications in a centralized setting, where performance is measured by running time and storage, and distributed settings, where we consider communication and per-node storage. In a distributed setting we require some mechanism that allows nodes to broadcast along a shortest path tree. In some settings the complete topology or at least some routing table can be obtained at each node (see [38]). In other more dynamic settings (such as p2p networks) each node is only aware of its neighbors.

The distance metric can be the shortest path metric defined by the edge-lengths or the Euclidean metric depending on the application. Edge-length metrics may capture number of network hops (with uniform edge lengths) or propagation time (where the length of each edge is the sum of the latency and the processing time at the end nodes). Most of our results are for the shortest path metric. Section 8 considers the Euclidean metric.

3 Decaying Sum

We will make use of data structures that we term *Neighborhood summaries* (NH-summaries). These data structures are maintained in each node of the network and support *neighborhood-sum queries*: For any given radius $r \geq 0$, the NH-summary provides node u with $(1+\epsilon)$ approximation of $S_{\text{BALL}_r}(u)$: the total sum of values of items that are within distance r from u. Also of interest are *d-limited NH-summaries*, which support neighborhood-sum queries over r-neighborhoods for $r \leq d$.

NH-summaries generalize *window-summaries* on data streams which provide, for each time window t, the sum of values of items observed in the last t time units. (Approximate) window summaries can be obtained using *Exponential Histograms* (EH) introduced by Datar et al [12]. They use $O(\epsilon^{-1}\log^2 W)$ memory bits for obtaining $(1+\epsilon)$-approximate estimates for windows up to size W.

We next argue that NH-summaries are general enough to support the computation of spatially-decaying sums under any decay function. The following lemma generalizes a lemma in [9] that shows that in the context of time decay, window-summaries can be used to compute time-decaying sums under general decay functions.

Lemma 3.1 *NH-summaries allow us to compute spatially-decaying sums under any decay function. If the summaries are approximate then so are the sums obtained.*

Proof. Consider some node u, and assume the items are numbered in increasing order of their distance to u. For convenience, assume that distances are integers. Let $\text{DIST}(u, \ell_i) = d_i$. Let $\Delta_j = g(j) - g(j+1)$. Since $g()$ is non-increasing, $\Delta_j \geq 0$ for all j. (w.l.o.g. we assume that $g(x) \to 0$ as x increases. Thus, $g(x) = \sum_{k \geq x} \Delta_k$. Otherwise, the problem is composed of a non-decaying problem and a decaying sum problem where the decay function goes to 0.) The decaying sum for u can be written as:

$$\begin{aligned} S_g(u) &= \sum_{i \in I} f_i g(d_i) = \sum_i f_i \sum_{k \geq d_i} \Delta_k = \sum_i \sum_{k \geq d_i} f_i \Delta_k = \\ &= \sum_k \Delta_k \sum_{i | d_i \leq k} f_i = \sum_k \Delta_k S_{\text{BALL}_k}(u) \,. \end{aligned}$$

Thus, $S_g(u)$ is expressed as a linear combination of $S_{\text{BALL}_r}(u)$ decaying sums. □

Define a *d-limited decay function* to be any decay function g such that $g(d) = 0$ (and thus $g(x) = 0$ for all $x \geq d$). Lemma 3.1 can be extended to using d-limited NH-summaries to express spatially-decaying sums for all *d-limited decay functions*.

When computing the sum, the number of summands can be reduced: When the approximate summary provide the same estimate $\hat{S}_{\text{BALL}_r}(u)$ of $S_{\text{BALL}_r}(u)$ for a range of values $a \leq r \leq b$, we can "replace" the sub-sum $= \sum_{k=a}^{b} \Delta_k \hat{S}_{\text{BALL}_k}(u)$ with the single term $(g(a) - g(b+1))\hat{S}_{\text{BALL}_a}(u)$.

The data structure that we use for our NH-summaries are *Min-value/Distance lists* (MV/D lists). MV/D lists were introduced in [6] in the context of a fast estimation algorithm for neighborhood sizes of nodes in a graph. MV/D lists are defined with respect to numerical values called *ranks*, that we associate with each data item. For data item i we use a rank drawn independently at random from a distribution. The *MV/D list* of a node u is a summary that allows us to retrieve, for any distance d, the minimum rank associated with an item that lies within distance d from u.

In principle, many distributions can be used to draw the ranks, but a particularly convenient distribution is the Exponential distribution with parameter f_i. (Items with $f_i \equiv 0$ get ranks $r_i = +\infty$.) To intuitively see why the Exponential distribution works well, observe that an Exponential distribution with parameter f_i is equivalent to the minimum of f_i Exponential distributions with parameter 1. Therefore, the rank of an item with weight f_i "acts" like the minimum of the ranks of f_i items of value 1.

The following lemma was established in [6].

Lemma 3.2 *MV/D lists with ranks drawn independently at random can be used as NH-summaries to answer NH queries. When using k (independent) lists, the probability of relative error larger than ϵ is $\exp(-\Omega(\epsilon^2 k))$. Therefore, for confidence $1-\delta$ and relative error at most ϵ we need $O(\epsilon^{-2} \ln \delta^{-1})$ lists. For a fixed ϵ, the probability that relative error exceeds ϵ can be made an arbitrarily low constant using a constant number of MV/D lists and polynomially-low using a logarithmic number of lists.)*

With Exponentially-distributed ranks, when we have k independent min-ranks, the estimator that gives us the properties in Lemma 3.2 is $k-1$ divided by the sum of the k min ranks. This is a well known unbiased estimator on the parameter of an Exponential distribution given k samples from the distribution.

Lemma 3.2 combined with Lemma 3.1 reduce the problem of obtaining approximate decaying sums to computing MV/D lists. The communication and storage costs of producing and representing the NH-summary are determined by the product of the number of MV/D lists ($O(\epsilon^{-2} \ln \delta^{-1})$), the size of the bit representation of the ranks ($O(\ln \epsilon^{-1} \ln \ln n)$), and the number of items or the communication in producing a single MV/D list.

4 MV/D Lists

As already mentioned, MV/D lists are defined with respect to random ranks associated with the data items. We denote by r_i the rank associated with item $i \in I$. An MV/D list has the form

$$((0 = d_0, r_0), (d_1, r_1), (d_2, r_2), (d_3, r_3), \ldots)$$

where $d_j < d_{j+1}$ are increasing and $r_j > r_{j+1}$ are decreasing. We refer to each distance-rank pair as an *element* of the list. The MV/D list has the property that for all j, the minimum rank found within distance $d \in [d_j, d_{j+1})$ from u is r_j. Clearly the *MV/D list* of a node u allows to retrieve, for any distance d, the minimum rank associated with an item that lies within distance d from u. (This can be done in time logarithmic in the length of the list by binary search.)

For the sake of simplicity, we treat in the sequel the case where there is at most one item in each node and focus on the decaying count problem, where $f_i \in \{0, 1\}$ for every $i \in I$. We will outline how bounds extend to the more general case.

We discuss the size of the representation of the ranks that is needed. It is easy to see that $O(\log \epsilon^{-1})$ significant bits are sufficient, the representation of the exponent requires $O(\log \log n)$ for n items with binary values (For non-binary values, n is replaced by n^* defined in the sequel).

The following lemma bounds the expected length of the MV/D list. The essence of the proof is that when ranks are drawn independently at random in a way that with high probability they are all distinct, they are not correlated with distances from any particular node u (The smallest rank is equally likely to be at any distance from u, and so is the second smallest rank, etc.). The MV/D list of u contains the prefix minima of this random permutation. For more details see [6].

Lemma 4.1 *When ranks are drawn independently at random, from say an exponential or a uniform distribution, then the expected* length *of the respective MV/D lists (in terms of number of elements in the list) is* $O(\log n)$ *where* n *is the number of nodes (holding nonzero items).*

Lemma 4.1 holds for binary item values and at most one such item per node. To extend this lemma to the general case (of multiple nonnegative-real-valued items in each node), define n^* to be the ratio of the sum of values of all items $\sum_i f_i$ to the smallest value of a nonzero item. Lemma 4.1 holds for the general case if you replace the $O(\log n)$ in the statement of the lemma by $O(\log n^*)$.[1] Note that if the number of items is bounded by a polynomial in n and each value is integral and bounded by a polynomial in n, then n^* is polynomial in n and $\log n^* = O(\log n)$. Moreover, if we restrict our attention to d-limited decay functions then n, and n^* in the derived bounds can be replaced by the respective quantities, n_d or n_d^* which are the number, and ratio of sum to smallest values, in a d-neighborhood, respectively.

Note that d-limited decay functions do not include smooth functions like polynomial or exponential decay, for such functions, all items in the system can influence any location. But when we can restrict ourselves to d-limited decay functions, the saving can be considerable. For example for bounded degree networks the size of an r-neighborhood is bounded by a function of r and the communication is thus polynomial in r (polylogarithmic in the size of the r-neighborhood and independent of the total number of nodes).

4.1 Computing MV/D lists

We start by reviewing a main-memory algorithm to compute MV/D lists [6] that was used for estimating neighborhood sizes. We then address interesting and subtle issues regarding the implementation of the technique of [6] in a distributed setting. For example we relax the rigid order in which nodes announce their ranks in [6] and show that it costs only a logarithmic factor in time (or communication in our distributed setting). For each node u we denote by $r(u)$ the rank of the item present at u. If u has no item associated with it, then $r(u)$ is infinity.[2]

The centralized algorithm of Cohen [6] for computing MV/D lists works as follows. All nodes of the graph (with finite ranks) are sorted according to their ranks. Then, from each node, in increasing order of ranks, we perform a *truncated BFS* pass (for uniform edge lengths) or a *truncated Dijkstra shortest-paths* pass (for non-uniform edge lengths). We refer to a truncated pass starting at a node u as an *announcement* made by u.

[1] In this case item i draws its rank from an exponential distribution with parameter f_i. This is equivalent to drawing f_i times from an exponential distribution with parameter 1 and taking the minimum. So think about the $\sum_i f_i$ draws from the exponential distribution with parameter 1 made by all nodes together. Order them by distance from u, group them into groups of size of $f_{min} = \min\{f_i \mid i \in I\}$ and take the minimum in each group. You get a random permutation of length n^*. The expected number of prefix minima of this random permutation is $\log n^*$. The expected number of items on the MVD list of u is at most $2 \log n^*$ since each group corresponding to a prefix minima may be split among the draws in two different nodes and thereby contribute one more number to the MV/D list of u.

[2] In the more general case where multiple items may present at u then $r(u)$ would be the minimum among the ranks of the different items present at u.

Assume edge weights are all uniform so each announcement is a truncated BFS. The truncated BFS pass proceeds very much like a complete BFS pass initiated at u with the following difference. When the pass determines for a node u' that its distance from the announcing node u is $\text{DIST}(u, u')$ then we perform one of the following two cases.

1. If node u' has not received earlier an announcement from a node w ($r(w) < r(u)$) such that $\text{DIST}(w, u') \leq \text{DIST}(u, u')$ (i.e. all previous announcements were by nodes further from u' than u) then the MV/D list of u' is updated to include the entry $(\text{DIST}(u, u'), r(u))$ and the pass proceeds through the outgoing edges of u'.

2. If node u' has already received an announcement from a node closer than u then the pass is halted from u' on (but may proceed from other nodes of the same distance, if the MV/D list of those nodes was augmented).

Each announcement uses at most $O(m)$ time, but if truncated then it is potentially much faster. It is shown in [6] that when nodes make announcements in increasing order of their ranks, the (centralized version of the) entire computation takes $O(m \log n)$ expected time when edge weights are uniform. This bound stems from the following considerations.

1. The number of announcements propagated by a certain node is exactly the size of its MV/D list. To see this, observe that since smaller ranks are announced before higher ranks, when an announcement reached a node and provoked an update of its list, this update can not be overridden by future announcements.

2. The work performed per announcement is proportional to the number of edges incident to the nodes reached by the announcement.

3. When the ranks are drawn independently at random from say an exponential distribution, then the expected size of the MV/D list is $O(\log n)$ by Lemma 4.1.

When edge weights are non-uniform we use Dijkstra's shortest-path algorithm as the announcing process in an analogous way. That is when we process an announcement from u and relax an edge (v, w) we insert w into the heap only if the distance of the path from u to w through v is shorter than the distance of nodes of smaller rank to w. This way, the number of times a node gets into the heap equals to the length of its MV/D list. If we implement Dijkstra's algorithm using Fibonacci heaps [18] then the running time is $O(m \log n + n \log^2 n)$. The additional $O(n \log^2 n)$ term accounts for the delete-min operations: Each node gets into the heap once per item in its MV/D list and the corresponding delete-min operations takes $O(\log n)$ time.

We will next consider the computation of the MV/D lists in a distributed network, where nodes communicate with each other along the edges of the network. Our main building block remains the "announcement process" where a node announces its rank to other nodes, but two important issues arise. First we need to consider the communication involved in making a single announcement. Second we need to consider the order in which announcements are made, since in a distributed setting it is undesirable to sort and synchronize so nodes are invoked in the order of increasing ranks.

We first consider the distribution of a single announcement in the network. When the topology of the network is known at all nodes, each node can compute shortest path trees and use these trees to make sure each announcement follows the shortest path tree of its origin. This way an announcement would require number of messages that is linear in the number of tree edges incident to visited nodes.

When the topology is not available to the nodes, but distances correspond to propagation delay then each announcement can be performed via flooding from its source. In this case, a node can deduce its distance from the source from the time in which it first gets the announcement. Therefore the number of messages would be proportional to the number of edges incident to the nodes which the announcement reaches.

If topology is not known and distances do not necessarily correspond to propagation delay, the nodes can perform a distributed BFS or shortest path computations [1]. BFS can be performed in $O(D^{1+\epsilon})$ time and $O(m^{1+\epsilon})$ messages, where D is the diameter of the network and a time unit is the time required for a message to pass through an edge. In our setting, D can be replaced by the diameter of the truncated network and m can be replaced by the number of edges that are incident to visited nodes. Shortest paths can be computed in $O(n^{1+\epsilon} \log W)$ time and $O(m^{1+\epsilon} \log W)$ messages, where W is the maximum edge length assuming integral lengths. As for BFS, we can take n, and m in these bounds to be the respective quantities for the truncated search.

4.2 Order of announcements

Announcing in increasing rank order makes it possible to charge each message to an item in the MV/D list of the propagating node. We can carry out the announcements in a different order while maintaining at each node a suitable MV/D list for the announcements seen thus far. However in this case deletions can occur from an MV/D list. Deletions occur when a node v appears on the MV/D list of u, but another node v' that is at least as close to u as v and has smaller rank than v makes an announcement later than v. When this happens u has to remove v from its MV/D list and insert v' instead. Since u propagated the rank of v to its neighbors upon receiving it we can no longer charge each message to an item in the MV/D list of u (that ended up without v).

As in the case when no deletions could occur, each time node u inserts a new pair into its MV/D list it has to distribute the corresponding announcement further. An announcement that does not affect the MV/D list of u is truncated by u. Note also that the content of the MV/D list at any point depends only on the set of nodes that made announcements so far (not on their order), and the expected length of the list is logarithmic in the number of nodes that have previously announced. Hence, the expected amount of storage at each node is logarithmic regardless of the order in which announcements are made.

The following lemma proves that if nodes make their announcements in a *random order* (independent of the ranks and inter-node distances), we get only a logarithmic increase in the number of messages. Random order is relatively easy to achieve in a distributed system. The proof bounds the expected number of insertions to the MV/D list of any node.

Lemma 4.2 *When nodes announce their ranks in a random order, the expected number of announcements that a node sends is $O(\log^2 n)$, where n is the number of nodes with value* 1.

Proof. Consider an arbitrary node u and let v be the node of the kth smallest distance from u (assume that by arbitrary tie breaking all distances are distinct). We compute the probability, over the random selection of ranks and announcement order, that v gets on the MV/D list of u when v makes an announcement. Node v gets on the MV/D list of u if and only if no node that is closer to u than v has both rank smaller than v and makes an announcement before the node v.

For all i ($i = 0, \ldots, k-1$), the probability that exactly i of the $k-1$ nodes that are closer to u than v have ranks smaller than the rank of v, is equal to $1/k$ (think of the ranks of these k nodes as a random permutation, then v is as likely to be in any position in the permutation). Suppose that i of the nodes closer to u than v have ranks smaller than the rank of v. Since the order of announcements (induced on these the $i+1$ nodes with ranks equal or smaller than v's) is a random permutation, the probability that none of these nodes make its announcement before v does, is exactly $1/(i+1)$.

Combining the above, we obtain that the probability that v, the kth closest node to u appears on the MV/D list of u is equal to

$$\frac{1}{k}\sum_{j=1}^{k}\frac{1}{j} = \frac{H_k}{k} ,$$

where H_k is the k'th Harmonic number.

Clearly, the sum over all nodes of the probability that the node appears on the MV/D list of u bounds that expected number of nodes that get on the MV/D list of u during the process. That is, the expected size of the list is

$$\sum_{k=1}^{n}\frac{H_k}{k} = O(\log^2 n) .$$

To see the last claim observe that

$$\sum_{k=1}^{n}\frac{H_k}{k} \approx \sum_{k=1}^{n}(\log k)/k \le \sum_{\ell=1}^{\log n}\ell \sum_{j=2^{\ell-1}}^{2^\ell} 1/j \approx \sum_{\ell=1}^{\log n}\ell \approx \log^2 n/2 .$$

(Since the inner sum is always a constant with value approximately 1.)

Node u sends the announcement that it gets only if this announcement gets on its MV/D list when it arrives. So we obtain that the expected number of announcements that u sends is $O(\log^2 n)$. $\square$

As for Lemma 4.1, Lemma 4.2 holds for binary values. To reformulate it for more general values we have to replace $O(\log^2 n)$ by $O(\log^2 n^*)$. (Recall that n^* is the ratio between $\sum_i f_i$ and $\min\{f_i \mid i \in I\}$.) For d-limited MV/D lists we can replace the $O(\log^2 n^*)$ by $O(\log^2 n^*_d)$.

We next show that an *arbitrary* order of announcements can require much more communication.[3] Consider a path network with nodes $v_0, v_1, \ldots, v_{n-1}$ appearing on the path from left to right. Suppose that the nodes make their announcements in the order $v_0, v_1, \ldots$. When the node v_i makes its announcement, it is the closest node that made an announcement thus far to nodes $v_{i+1}, \ldots, v_{n-1}$. Thus (regardless of what the rank of v_i is), the announcement must reach all nodes v_j with $j > i$. Therefore, this order causes $\Omega(n)$ messages to go through most nodes. Observe that even though communication is large, the expected size of the MV/D list at any given point in time is logarithmic (the analysis holds since ranks are independent of location).

When announcements are made in arbitrary order but there are only $D \ll n$ distinct distances, we can derive a tighter bound. Clearly there cannot be more than one element of each particular distance at any particular time on a single MV/D list. Since the ranks are independent of the order of announcements, it is easy to see that the expected total number of elements of each particular distance that appear on the list at any point during the computation is $O(\log n)$. The number of different distances is small if the diameter is small or if distances are discretized into a small number of buckets.[4] Table 1 summarizes the storage (in terms of number of elements on the MV/D list) and communication (in terms of number of announcements reaching a node) using different orders of announcements. Experimental results on comparing the different orders of announcements are given in Section 12.

[3]Since ranks are random any order which is fixed before the ranks are drawn would define a random permutation of the ranks. Hence by arbitrary order we refer to an order that may depend on the location.

[4]If we are interested in polynomial decay, then distances can be discretized into a logarithmic number of buckets while introducing only a small relative error (all distances that are within a factor of $(1+\epsilon)$ of each other can be considered equal).

Announcement order	storage	communication
Ordered	$O(\log n)$	$O(\log n)$
Random	$O(\log n)$	$O(\log^2 n)$
Arbitrary	$O(\log n)$	$O(n)$ or $O(D \log n)$

Table 1: The per-node storage and communication bounds for an MV/D list computation. The communication is the number of announcements reaching a node (Each announcement message consists of distance and rank and is typically of logarithmic size.) The storage is in terms of number of elements (where each element is a distance-rank pair, and possibly item identifier). When nodes contain multiple items with arbitrary values, the n in these bounds should be replaced by n^*. If we restrict our applications to d-limited decay functions, then n and n^* can be replaced by n_d and n_d^*, respectively. We do not count here messages that may be required to compute shortest path trees.

4.3 MV/D lists extensions

k-min MV/D lists In some settings one would use the k smallest ranks in a single rank assignment rather than the smallest rank in k different assignments [6]. It is easy to extend the notion of MV/D lists to this setting and to extend our analysis. We can expect a k-fold increase in storage (for the size of the lists) and in communication but the basics of the truncated BFS or truncated shortest-path computation and the effects of the announcement orders are the same as for the minimum rank computation.

Hashed rank values Hashed values instead of random ranks are used for aggregations over distinct items. Specifically, work on distinct counting used min-wise independent permutations [4, 25] or the location of the least significant 1 bit in a hashed value [16]. It is easy to see that MV/D lists can be constructed for these values as well and will allow us to represent the same aggregates over neighborhoods or windows. Since rank values do not need to be random for the truncated search process to be correct, it also works correctly for these ranks. The analysis of the announcement orders extends using the properties of the min-hash values.

MV/D lists over Reverse Nearest Neighbor (RNN) sets MV/D lists are defined over neighborhoods. Suppose now that there is a subset of nodes that are "servers." Each node is interested in obtaining aggregates over the set of items for which it would be the closest server if it is nominated to be a server. For this purpose we use a modified announcement process: each item is aware of the distance of its closest server and the announcement process stops at that distance. It is not hard to see that the resulting MV/D lists at each node are obtained with respect to the RNN set of the node. If the MV/D list computations in our algorithms are replaced by RNN MV/D list computations, then the respective resulting aggregation is performed over the RNN sets.

5 Random Sampling

We start with the simpler problem of computing, for each node, a summary that would allow it to obtain, for each r, an item selected uniformly at random from its r-neighborhood. (Dependencies between selection for different r and different locations are allowed). This can be performed using a single MV/D lists computation: Each item computes a random rank for itself, and an MV/D list is computed for each node. For each $r \geq 0$, all items in the r-neighborhood are equally likely to have the minimum rank. Thus, the minimum ranked item constitutes a uniform random sample from the r-neighborhood. More generally, a random sample of size k can be obtained either (for sampling with returns) by performing the above k times (using k different MV/D lists), or more efficiently, (and for sampling without returns), by computing *k-min MV/D lists* which provide for each r the k-smallest ranked items in the r-neighborhood. It is not hard to modify our 1-min MV/D list algorithms to compute a k-min MV/D list while incurring (a necessary) k-fold increase in storage and communication. With k-min MV/D lists, an item is placed on the list and propagated further if and only if there are at most $k-1$ other items which are at least as close and have smaller ranks. If we are interested in performing weighted sampling according to some values f_i, we use weighted ranks in the MV/D list computation (with n^* replacing n in the bounds).

We next address the problem of obtaining samples with respect to an *arbitrary* decay function $g()$, where the probability of an item i to be drawn at location u is proportional to its weight $w_{u,g}(i)$.

Let i_r be a uniform random sample from the r-neighborhood of u. As argued above, (dependent) samples i_r for all $r \geq 0$ can be obtained using a single MV/D lists computation. Let N_r be the number of items in the r-neighborhood of u, and $N \equiv N_{r_{\max}}$ be the number of items in the system. (We argue below that we can use estimates of these values instead of the exact values.) For notational convenience we assume integral distances and define $g(r) \equiv 0$ for $r > r_{\max}$. We now consider a node u and a decay function $g()$.

Lemma 5.1 *The following process draws the item $i \in I$ with probability $w_{u,g}(i)/\sum_{i' \in I} w_{u,g}(i')$:*

- *Draw $0 \leq r \leq r_{\max}$ randomly according to $(p_0, \ldots, p_{r_{\max}})$, where*

$$p_r = \frac{(g(r)-g(r+1))N_r}{\sum_{j\geq 0}(g(j)-g(j+1))N_j} = \frac{(g(r)-g(r+1))N_r}{\sum_{i\in I} w_{u,g}(i)}$$

  (Thus, r is selected with probability proportional to $(g(r)-g(r+1))N_r$.)

- *Return the sample i_r.*

Proof. This is a 2-stage sampling process, where in the first stage an "event" $j \in 0, \ldots, r_{\max}$ is drawn according to the probability distribution $(p_0, \ldots, p_{r_{\max}})$. In the second stage, an item is drawn uniformly at random from the items in the j-neighborhood. That is item i is drawn with probability $q_{ji} = 1/N_j$ if $\text{DIST}(u, \ell_i) \leq j$ and with probability $q_{ji} = 0$ otherwise. This 2-stage sampling process is equivalent to drawing according to the probability distribution where item i is drawn with probability equal to $\sum_{j=0}^{r_{\max}} p_j q_{ji}$. Substituting according to the definitions of p_j and q_{ji} we obtain that item i is drawn with probability

$$\begin{aligned}\sum_{j=0}^{r_{\max}} p_j q_{ji} &= \sum_{j=\text{DIST}(u,\ell_i)}^{r_{\max}} p_j q_{ji} \\ &= \frac{\sum_{j=\text{DIST}(u,\ell_i)}^{r_{\max}}(g(j)-g(j+1))}{\sum_{i'\in I} w_{u,g}(i')} \\ &= \frac{g(\text{DIST}(u,\ell_i))}{\sum_{i'\in I} w_{u,g}(i')} = \frac{w_{u,g}(i)}{\sum_{i'\in I} w_{u,g}(i')}\end{aligned}$$

□

The nodes use estimates $\hat{N}_r$ instead of the (unavailable) exact values N_r. We can obtain estimates such that the relative error is small with very high probability, that is, $|\hat{N}_r - N_r|/N_r \leq \epsilon$ for $r \geq 0$. When a node uses estimates we have $q_{ji} = 1/N_j$ but $\hat{p}_j = \frac{(g(j)-g(j+1))\hat{N}_j}{\sum_{j'\geq 0}(g(j')-g(j'+1))\hat{N}_{j'}}$. Both the numerator and denominator of $\hat{p}_j$ are within relative error ϵ from the respective quantities for p_j, thus for $\epsilon < 1/3$ we have $|p_j - \hat{p}_j|/p_j \leq 3\epsilon$. Therefore,

$$\frac{\left|\sum_{j=0}^{r_{\max}} p_j q_{ji} - \sum_{j=0}^{r_{\max}} \hat{p}_j q_{ji}\right|}{\sum_{j=0}^{r_{\max}} p_j q_{ji}} \leq 3\epsilon \; .$$

Thus, the probability that an item i is drawn, is in the range $(1 \pm 3\epsilon)\frac{w_{u,g}(i)}{\sum_{i' \in I} w_{u,g}(i')}$. This approximation is sufficient for most applications of random sampling, including approximate median. We remark that the above derivation easily extends to weighted sampling where node u wants to draw item i with probability proportional to $w_{u,g}(i) f_i$. In this case we define p_i instead of using the number of items in a neighborhood N_r, using the sum of weights f_i over items in the r-neighborhood. Furthermore, within an r-neighborhood we perform a weighted sampling according to the values f_i of the items in this neighborhood.

6 Approximate L_p Norms (for $p \in [1, 2]$)

The data stream version of the problem [12] is as follows: There is an underlying d dimensional vector. Each item i consists of a pair (c_i, a_i), where $c_i \in [d]$ is a coordinate and $a_i \in \{0, \ldots, M\}$ is an increment to the c_ith dimension of the underlying vector. Every window W represents a vector, defined by applying all the items in the window to the **0** vector. The $L_p(W)$ norm of that vector is given by

$$L_p(W) = \left(\sum_{j \in [d]} |s_j|^p \right)^{1/p} ,$$

where s_j is the sum of the increment values of all items (c_i, a_i) where $c_i = j$, obtained in the last W time units. The goal is to efficiently maintain an approximate value of $L_p(W)$ using storage that is significantly smaller than the dimension d. Datar et al [12] showed that any constant relative error and confidence can be obtained using $O(\log N(\log N + \log M) + \log M \log d)$ storage (where N is the maximum window size we would like to support).

We recall the definition of the problem in the spatial setting that was given in Section 2.1. Each item $i \in I$ is an update of a coordinate of a d-dimensional vector and is specified by a triplet (c_i, a_i, ℓ_i). The value $c_i \in [d] = \{0, \ldots, d-1\}$ specifies the coordinate which this item updates. The value $a_i \in \{0, \ldots, M\}$ is the amount by which we increment the target coordinate, and $\ell_i \in V$ is the item's location. The d-dimensional vector $\mathbf{V}(g, u)$ associated with location $u \in V$ and a decay function g, is then defined by the coordinate values

$$\mathbf{V}(g, u)_j = \sum_{i | c_i = j} w_{u,g}(i) a_i \ .$$

Our techniques combined with Indyk's allow to obtain, using $o(d)$ communication per node, at each location u, a summary of size $o(d)$, such that for any decay function $g()$, the node can obtain an approximate value of $||\mathbf{V}(g, u)||_p$ (the L_p norm of $\mathbf{V}(g, u)$).

Observe that for $p = 1$, all coordinates can be aggregated together and the problem reduces to the decaying sum problem. For $p > 1$, clearly, an approximate value of each coordinate of the vector $\mathbf{V}(g, u)$ can be obtained by computing a spatially decaying sum over the update values associated with that coordinate. An approximate value of the L_p norm can be obtained from these estimates. This requires, however, applying the decaying sum computation d times, which imposes a factor d increase in communication. The goal again is to significantly reduce the dependence on d.

We briefly describe Indyk's sketching technique and then explain how to apply it in our setting. The simplified sketching algorithm works as follows. For some fixed value $L = o(d)$, which depends on the desired accuracy and confidence, the algorithm uses dL independent random variables X_i^k (where $i \in [d]$ and $k \in \{1, \ldots, L\}$) which are drawn from some distribution that depends on p (the parameter of the norm). Each node has to have access to these numbers or be able to generate them. For details of how this common knowledge could be obtained see [26].

For each $k \in \{1, \ldots, L\}$, each node v then computes locally from its items a value

$$f(u)_k = \sum_{i|\ell_i = v} a_i X^k_{c_i} \ .$$

We then perform L decaying sum computations using the values $f(u)_k$ ($k = 1, \ldots, L$). The estimated norm at each node can then be computed from these sums according to Indyk's method (the computation depends on the norm parameter p).

7 Distinct Elements

The problem of distinct elements is to (approximately) count the number of distinct values occurring in neighborhoods of each node. This can be performed using an NH-summary computation based on a variant of MV/D lists. The key point here is to draw the ranks such that all items with identical values obtain the same rank, while other crucial properties of the ranks are preserved. That is ranks of distinct values should be independent and identically distributed. A node can obtain such a rank by applying a random hash function (same function across all nodes) to the item's identifier. If we know the distribution of the hashed values, we can map them to pseudo random samples from a desired distribution (say the Exponentially distribution). As with random ranks, we can use several MV/D list computations to obtain an NH-summary that can estimate the number of distinct values in each neighborhood with high probability. (This probability is now over the choice of the hash function.) If we used Exponential pseudo random samples, we can apply the same estimator we used for Exponential random ranks to determine the number of distinct item.

This also applies to the weighted version of the problem where we estimate the weight of distinct items. The analysis is essentially as in [6] if we assume availability of hash functions with perfect random properties. Fortunately, however, the much weaker randomness assumptions of min-wise independence typically suffice [4, 25]. (Counting distinct elements in data stream using min-wise hash functions was studied, eg, by [21, 3].) We also note that the Flajolet Martin [16] distinct counting approach can be used in this framework: we chose the rank of an item to be the location of the least significant 1 bit (technically, with this definition we do max-rank). We then apply the Flajolet-Martin estimator to estimate the count.

To obtain a *random distinct value* we can simply take the value that had the minimum hashed rank. We can also obtain the *multiplicity of a random distinct value*, as follows. We first discuss small multiplicities. Multiplicities up to M can be obtained exactly with M factor on communication: when computing the MV/D lists, we continue to propagate all items with rank equal to the current minimum rank value (at each particular distance). To do so, we may need to record up to M identifiers at each node, and pay an additional factor of M on communication. For high multiplicities, we can reduce the overhead to $O(\log M)$ factor and obtain approximate multiplicities as follows. Consider each item as having a (hashed by value) rank and an additional random rank. For each minimum 1st rank item we consider the second rank and propagate only if smaller. These second ranks allow us to estimate the multiplicity of that value (we need several such 2nd ranks for better accuracy.). Since we don't know the multiplicity of the random element in advance, we can apply the two approaches simultaneously and stop the exact counting when we reach some cutoff value of M.

8 Euclidean MV/D Lists

Up till now, we considered only edge-length metrics on the underlying network. We now turn our attention to spatially-decaying aggregation with respect to the L_2 metric on the Euclidean plane, both when the data is centrally processed and in a distributed setting. MV/D list computation is the basic ingredient in our aggregation algorithms; substituting it with a *Euclidean MV/D list computation* yields algorithms for the Euclidean version of these aggregates.

In the Euclidean MV/D list problem, item locations are points in the Euclidean plane. Items have ranks and each point in the plane has an MV/D list defined according to the Euclidean metric. The goal is for each node to obtain the MV/D list for its location or more generally, to construct a data structure that would allow us to efficiently obtain the MV/D list of *any* query point.

8.1 Centralized setting

The centralized version of the problem amounts to computing and performing point location in a particular collection of Voronoi regions (taken from different diagrams) as stated in the following lemma. The proof of this lemma is immediate from the definitions.

Lemma 8.1 *Let items be indexed in increasing rank order $p_1, \ldots, p_n$. Let c_i be the Voronoi region of the location of item p_i in the Voronoi diagram defined by the locations of $\{p_1, \ldots, p_i\}$. The MV/D list of a query point consists of all i such that the point resides in the region c_i.*

Moreover, it is not hard to see that the expected size of the MV/D list associated with a point is logarithmic (the arguments used in [6] for edge metrics carry over).

Our centralized Euclidean MV/D list computation can be stated in terms of constructing and querying *incremental Voronoi diagram* under a random order of point insertions. The goal is to find the closest neighbor for a query point not only in the final diagram, but with respect to all prefixes of the insertion order. A paper by Guibas et al [24] constructs an appropriate data structure for this task. The size of the data structure is linear, it can be built in $O(n \log n)$ expected time, and the expected query time is $O(\log^2 n)$.

8.2 Distributed setting

In the distributed setting, data items reside at nodes connected by an underlying network. Generally, however, the Euclidean metric may not be representable as an edge metric on the network (unless the network is a full mesh or very degenerate like a path network that lies on a line). We construct an example that shows that a Euclidean MV/D list computation may necessitate quadratic communication: Consider a U-shaped path network, where the nodes coordinates in the Euclidean plane are $(0, 2i)$ and $(1, 2i)$ (for $i = 0, \ldots, n$); the network edges are $((0,0),(1,0))$, $((0,2i),(0,2(i+1)))$, and $((1,2i),(1,2(i+1)))$ (for $0 \leq i < n$). Items are present at nodes $(0, 2i)$ (for $0 \leq i \leq n$). The closest item to a node $(1, 2i)$ is the one residing on node $(0, 2i)$, thus, this particular item is present on the MV/D list of $(1, 2i)$ and must be communicated through all nodes $(0, 2j)$ and $(1, 2j)$ for $j \leq i$.

It turns out that for any $\ell \geq 0$, all announcements from nodes $(0, 2i)$ for $i \geq \ell$ must flow through the nodes $(0, 2\ell)$ and $(1, 2\ell)$. We thus obtain that during an MV/D list computation, an average $\Omega(n)$ announcement messages must traverse a node.

This bad example exploits the mismatch between the network topology and the Euclidean metric: even though the size of the MV/D list of each node is logarithmic, nodes must pass on information through the network that is not relevant to their own MV/D list. On some networks, in particular on grids, it is possible to efficiently compute Euclidean MV/D lists distributively, with nearly linear total communication.

Grid networks

Consider a grid network in the plane with n nodes (i, j), where $1 \leq i \leq \sqrt{n}$, and $1 \leq j \leq \sqrt{n}$. We assume that each node v_i holds a binary data item, and denote by $p_1, \ldots, p_n$ the data items when sorted by their corresponding ranks. We associate the grid square (*cell*) defined by the points $(i \pm 1/2, j \pm 1/2)$ with the node (i, j). Each node builds a data structure that allows it to produce the MV/D list of any query point in its cell.

Consider the Voronoi regions $\{c_1, \ldots, c_n\}$ as in Lemma 8.1. That is region c_i is the Voronoi region of p_i in the Voronoi diagram defined by the locations of $\{p_1, \ldots, p_i\}$. In order to determine the Euclidean MV/D list for all query points that lie within its grid cell, it is sufficient (and necessary) that a node obtains the list of items i for which the region c_i intersects its cell. For that purpose we define the following announcing process.

When a node makes an announcement, a receiving node checks if it "updates" the current list of any point in its cell, if it does, it propagates the message to all its neighbors.

Since each region is connected, the set of grid cells which it intersects is connected, and therefore this propagation scheme guarantees that each announcement reaches all cells which intersect the corresponding region.

We now analyze the communication out of each node. In the following Theorem we establish that the *average* number of regions that intersect a particular grid cell is logarithmic. It then follows that according to the propagation scheme defined above each node delivers a logarithmic number of messages on average.

Theorem 1 *The average, over grid cells, of the number of regions (out of $c_1, \ldots, c_n$) that intersects the grid cell is logarithmic.*

Note, however, that there can be cells that intersect a linear number of regions, and communication for the respective nodes is linear. The proof of Theorem 1 follows from the following two lemma.

Lemma 8.2 *The number of grid cells that are intersected by a bounded contiguous region is bounded by a constant times the sum of the circumference and area of the region (measured by grid units and squared grid units respectively).*

Proof. We separately account for grid cells that are fully or partially contained in the region. The total number of grid cells that are fully contained in the region is bounded by the area of the region. Any cell which is partially contained in the region must intersect the circumference of the region. Consider a point on the circumference and the grid cell it lies in. The closest point on the circumference that does not belong to the grid cell or any (of its 8) neighboring cells must be of distance at least 1 grid unit away, and thus at least 1 grid unit length away along the circumference. Therefore, the total number of grid cells touched by the circumference is at most a constant times its length. □

Lemma 8.3 *Consider a Voronoi diagram defined by a set of points on the grid. Consider a region in that diagram. (We assume wlog that regions are finite, as we limit them by the area of the grid.) The ratio of circumference to area of the region (measured in grid units and squared grid units, respectively) is at most 4.*

Proof. Consider a region c that is closest to a point p (see Figure 2). Consider now a triangulation of c formed by connecting p to each of the vertices $v_0, v_1, \ldots, v_k$ of c (presented in clockwise order). The area of c is the sum, over $i = 0, \ldots, k$, of the areas of the triangles $v_i p v_{i+1}$ (indices are modulo k). The circumference of c is the sum of the lengths of the segments between v_i and v_{i+1}.

Let h_i be the height of the triangle $v_i p v_{i+1}$ (The distance between p and the line defined by v_i and v_{i+1}.) The area of the triangle is $h_i \overline{v_i, v_{i+1}}/2$, where $\overline{v_i, v_{i+1}}$ is the length of the segment connecting v_i and v_{i+1}.

By definition of Voronoi regions, the line defined by v_i and v_{i+1} is of equal distance from two grid points, one on each side of the line. Since the distance between any two grid points is at least one grid unit, the length of h_i is at least half a grid unit.

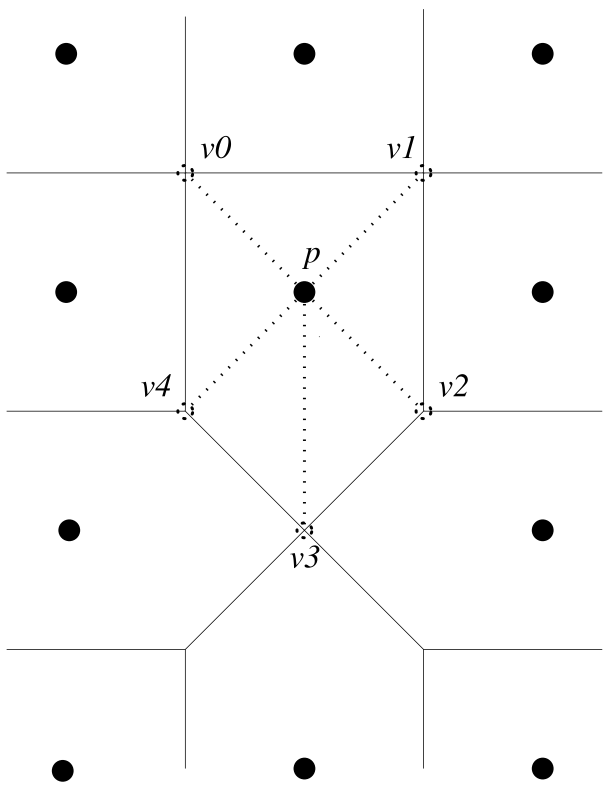


Figure 2: Voronoi diagram defined by a subset of points in a grid. A voronoi region for a point p and the respective triangulation as in Lemma 8.3

We obtain that the ratio of the circumference of c to its area is

$$\frac{\sum_{i=0}^{k} \overline{v_i, v_{i+1}}}{(1/2) \sum_{i=0}^{k} h_i \overline{v_i, v_{i+1}}} \leq \frac{\sum_{i=0}^{k} \overline{v_i, v_{i+1}}}{(1/4) \sum_{i=0}^{k} \overline{v_i, v_{i+1}}} \leq 4 \, .$$

(We substitute $h_i \geq 1/2$ to obtain the last inequality.) Note that the ratio of 4 is tight: If all grid points are used then each Voronoi region is a grid-unit square, with circumference of 4 and area of 1. □

An immediate corollary of Lemma 8.2 and Lemma 8.3 bounds the number of cells intersecting a region by the area of the region.

Corollary 8.4 *The number of grid cells that are intersected by a Voronoi region defined by points on a grid is bounded by a constant times the area of the region (measured by squared grid units).*

The corollary, combined with the following lemma, concludes the proof of Theorem 1.

Lemma 8.5 *The expected total area of the regions $\{c_1, \ldots, c_n\}$ is $O(n \log n)$ grid squares. Furthermore, the expected number of edges defining a region is constant.*

Proof. We first show that the expected area of c_k is n/k. Since the ranks are random any one of the items $p_1, \ldots, p_k$ is as likely to be the kth one. The total area of the regions of the Voronoi diagram defined by $p_1, \ldots, p_k$ is n since they form a partition of the whole grid which consists of n cells. Thus, the expected area of a region selected uniformly at random is n/k. Using the above, we obtain that the expected combined area of the regions is $n \sum_{i=1}^{n} 1/i = O(n \log n)$. Also note that the total number of edges in the diagram is linear, and thus, the expected number of edges defining the cell c_k is constant. □

For random order of announcements, each vertex of the grid propagates the message announced by an item p_k to its neighbors if the Voronoi region of p_k in the Voronoi diagram of points p_j, $j < k$, that announced before p_k intersects its cell. Let c_k denote now the Voronoi region of p_k in the diagram define by the **subset of** $p_1, \ldots, p_k$ that announced before p_k.

Fix a point w in the subset of the plane covered by the grid. An argument as in the proof of Lemma 4.2 shows that the number of regions containing w is $O(\log^2 n)$. Indeed, c_k contains w if no point p_j, $j < k$ is closer to w than p_k and makes its announcement before p_k. We order the points according to their distance from w. Since ranks are independent of the distances the probability that i among $p_1, \ldots, p_{k-1}$ precede p_k in the order of the distance from w is $\frac{1}{k}$ for every $i = 0, \ldots, k-1$. Assuming indeed i points p_j, $j < k$, precede p_k in this order the probability that none makes its announcement before p_k is $\frac{1}{i+1}$. We can now complete this argument as in the proof of Lemma 4.2.

The fact that an arbitrary point w intersect $O(\log^2 n)$ regions on average, implies that the sum of the areas of all regions is $O(n \log^2 n)$. This observation combined with Corollary 8.4 implies that the average number of announcements that a node would deliver under a random order of announcements is $O(\log^2 n)$.

9 Lower Bounds

Consider a problem instance of computing some time-decaying aggregate over N time units where item of value f_i is observed at time $t_i = i$. Also consider a directed path network of size N, where each edge is of unit length and item of value f_i is present at node $\ell_i = i$. Suppose that the same spatial-aggregate function and the same decay function apply in both settings (decay over j elapsed time units is equal to decay over distance j). The aggregate value at node i in the spatial-decay setting is equal to the value at time i in the time-decay setting. Then, we can obtain an algorithm for the time-decay problem from an algorithm to the spatial-decay setting and vice versa. The size of a summary that is propagated from time

i to time $i+1$ in order to make sure we can know the value of the aggregate in later times corresponds to the communication along the edge $(i, i+1)$ (and the storage at node i) and vice versa. Thus, known lower bounds on the space required to maintain time-decaying aggregates together with the reduction described above, imply lower bounds on the number of bits that have to be sent along edges when performing the respective spatially-decaying aggregation on path networks.

We list some examples of such lower bounds. Datar et. al. [12] showed that $\Omega(\frac{1}{\epsilon}\log^2 N)$ bits are needed to maintain ($1 \pm \epsilon$ approximate) time-decaying sums over a binary stream where $g()$ is a sliding window of size N. Their construction in fact can be adjusted to show that $\Omega(\frac{1}{\epsilon}\log^2 N)$ has to be kept during a fraction of the entire time. This implies that when computing spatially-decaying sums for $g() \equiv \text{BALL}_N$ at least $\Omega(\frac{1}{\epsilon}\log^2 N)$ bits must traverse a fraction of the edges. Similarly, a lower bound of $\Omega(\log R)$ bits on per-edge communication for computing (approximate) polynomial and exponential spatially-decaying sums over R items (of value at most polynomial in R) follows from the corresponding lower bounds for polynomial or exponential time-decaying sums that were provided in [9]. Linear ($\Omega(N)$ bits) lower bound on the per-edge communication needed to obtain the minimum value of an item in the N-neighborhood of each node follows from the linear lower bounds on space in the sliding-window model [12]. Note that in contrast to the sliding window model computing the global (non-decaying) minimum over a data stream is an easy problem, and similarly, the global minimum over a network can be computed using logarithmic per-node communication.

10 Exponential Histograms for Spatial Decay on Grid networks

We consider a different algorithm for BALL_r spatially-decaying sum computation. The algorithm applies Exponential-Histograms (EH) which is a data structure that was developed in [12] for sliding window time-decay (which can be viewed as the 1-dimensional version of the problem we consider here).

The extension of the EH technique seems to be specific to grid networks, we obtain approximate sums (*for a specific value of* r) on fixed-dimensional grid networks under the L_∞ metric.[5] We assume that the grid is undirected (distances are symmetric). The approach easily extends to directed grids, where all "parallel" edges are directed the same way.

One limitation of the d-dimensional (for $d \geq 2$) EH solution compared to the 1-dimensional EHs [12] and compared to our general solution in Section 3 is that each "run" applies only to a *specific* value of r. EH in 1-dimension, and our general solution provide approximate $\text{BALL}_{r'}$ decayed sums for all $r' \leq r$. The advantages of this algorithm over the general method discussed earlier are its tighter dependence on ϵ of $O(\frac{1}{\epsilon}\log^2 r)$ and its approximation guarantees (EH is *guaranteed* to give $(1+\epsilon)$-approximate answers whereas the MV/D lists based technique has confidence bounds.)

The EH data structure processes a stream of values and can provide $(1 \pm \epsilon)$ estimates for the sum of the W recent values. In order to answer queries for any $W \leq N$ the EH needs $O(\frac{1}{\epsilon}\log^2 N)$ space assuming values are polynomial in N.

We state the algorithm for 2-dimensional grids, and sketch the fairly straightforward generalization to higher dimensions. Our algorithm uses the EH data structure as a black box and computes an approximation of $S_{2,r}(i,j)$, the sum of the values of the nodes $(i-x, j-y)$, where $0 \leq x, y \leq r$. The $S_{\text{BALL}_r}(i,j)$ approximations for undirected grids can then be obtained by performing this operation symmetrically on the other quadrants and summing the results.[6] The algorithm is as follows.

[5] We remark that the L_∞ metric does not strictly fall in our model, since this metric is not an edge-length metric on the grid, whereas the solution for general spatial decay in Section 3 was for edge-length metrics. Note however that L_∞ on the grid can be treated as an edge-length metric if we add the diagonals and make the lengths of all edges the same. Communication across diagonals can then be emulated by transferring the message on 2 corresponding edges.

[6] In a d-dimensional grid for each node $(i_1, i_2, \ldots, i_d)$ we obtain an approximate sum of all values present at nodes $(k_1, \ldots, k_d)$,

- For each i, propagate an EH for sliding window of size r through $(i, 1), (i, 2), (i, 3), \ldots$. As a result, each node (i, j) will have an approximate value of $S_{1,r}(i, j)$, the sum of values of the nodes $(i, j - r), (i, j - r + 1), \ldots, (i, j)$.

- For each j propagate an EH for sliding window of size r of the nodes $(1, j), (2, j), (3, j), \ldots$ using the value $S_{1,r}(i, j)$ for each node (i, j). As a result, each node (i, j) will have an approximate value for $S_{2,r}(i, j)$, the sum of values of the nodes $(i - x, j - y)$, where $0 \le x, y \le r$.

The EH method requires $O(\frac{1}{\epsilon} \log^2 N)$ bits for maintaining a sliding window of length N (assuming the values are of polynomial size in N) [12]. Our algorithm for d-dimensional grids thus utilizes $O(d\frac{1}{\epsilon} \log^2 r)$ bits of communication per node (for each of the 2^d quadrants).

We sketch how the EH based algorithms can be extended to the L_1 metric. Under the L_1 metric, the r-neighborhood of a grid node u is a diamond shaped region centered at u. We can treat the set of nodes in this diamond shaped region as two squared regions each in a different virtual grid. One virtual grid consists of nodes with even sum of coordinates and the other consists of nodes with an odd sum of coordinates. The edges of the virtual grid are diagonals of cells on our original grid that connect between consecutive points of the virtual grids. The original grid can emulate communication on each virtual grid with constant factor overhead, basically, communication with a neighbor is replaced by communication within a 2-neighborhood.

11 Decay Across Space and Time

We extend our model to aggregation over items that arrive in different locations *and at different times*. We use f_i, t_i and $\ell_i \in V$ to denote the value, time, and location of the ith item. There are two different decay functions: g_D captures decay as a function of distance and g_T captures decay as a function of elapsed time. The weight of an item i with $t_i \le T$, at time T as viewed from location u is $g_D(\text{DIST}(u, \ell_i))g_T(T - t_i)$. The decaying sum at location u at time T is

$$S_{g_D,g_T}(u) = \sum_{i \in I | t_i \le T} f_i g_D(\text{DIST}(u, \ell_i)) g_T(T - t_i) \ .$$

We measure performance by both communication *per time unit* and the amount of storage per node, so that at any time and at any location we can obtain an approximate value of the respective aggregate.

We allow for a tolerance of Δ time units in the interpretation of time stamps. This is necessary, since otherwise every single item that arrives at a distinct time must be broadcasted to the whole system, since it constitutes the most recent arrival. With Δ tolerance, we can use plain spatial aggregations for all items arriving within Δ time units (as they are treated as having the same time stamp). For efficiency, we would further like the order of announcements within each Δ period to be unrelated to location, which can be achieved by adding random delays to item arrival times.

We next consider storage. Storing a spatial summary for each Δ period can be space consuming. To support queries of decaying sum for all (time and distance) decay functions, we should be able to obtain, for each t' and d', approximate sum of values of all items that occurred within the last t' Δ-time-periods and within distance d'. Unfortunately, the size of such a "2-d NH-summary" in general may have to be linear in (the smaller of) the number of distinct distances from our location and the number of distinct time units as the following example shows.

Consider a path network with nodes $v_0, v_1, \ldots, v_{n-1}$ and n items (of value either 0 or 1) where item i occurs at node v_i at time t_i, for $0 \le i \le n - 1$. The times t_i are increasing such that $t_i + \Delta < t_{i+1}$

where $i_j - r \le k_j \le i_j$ for $j \in \{1, \ldots, d\}$.

(items that are further from v_0 occur later in time). Consider the information that needs to be stored at v_0 at some time T (that is later than all item arrival times). Each item has a unique time-distance neighborhood in which it lies: the item (v_i, t_i) is the only item within the $(T - t_i)$-recent time window and i-neighborhood of v_0. Thus, in order to be able to obtain estimates of $\text{BALL}_{r,t}$ for all possible values of r and t, we may need storage that is linear in (the smaller of) the number of distinct distances from our location and the number of distinct time units.

Storage requirements can be reduced with some restrictions: Observe that for each set of items occurring at the same distance (possibly at different times), we can maintain a single time-window summary (using Exponential Histogram for example). Thus, if distances are discretized into L meaningful levels, it suffices to maintain L time-window summaries. A second observation is that if we restrict ourselves to a fixed spatial decay function at each node then it suffices that each node maintains a single time-window summary. The information incorporated in the summary in each time unit t is a single numeric value: an approximate spatially decaying sum

$$S_{g_D,t}(u) = \sum_{i|t_i=t} f_i g_D(\text{DIST}(u, \ell_i)) ,$$

obtained using the NH-summary (MV/D lists) compiled for the current Δ period.

12 Experimental Evaluation

For the simulation part of our evaluation we used terrestrial air temperature and precipitation data. The data was provided for grid points in half-degree spacings on the terrestrial grid. For each grid point it included the mean annual temperature and precipitation (over the time period 1950 to 1999) [5]. We treated the grid as a network and used the L_1 distance on degrees. Overall, our grid included 85,794 points.

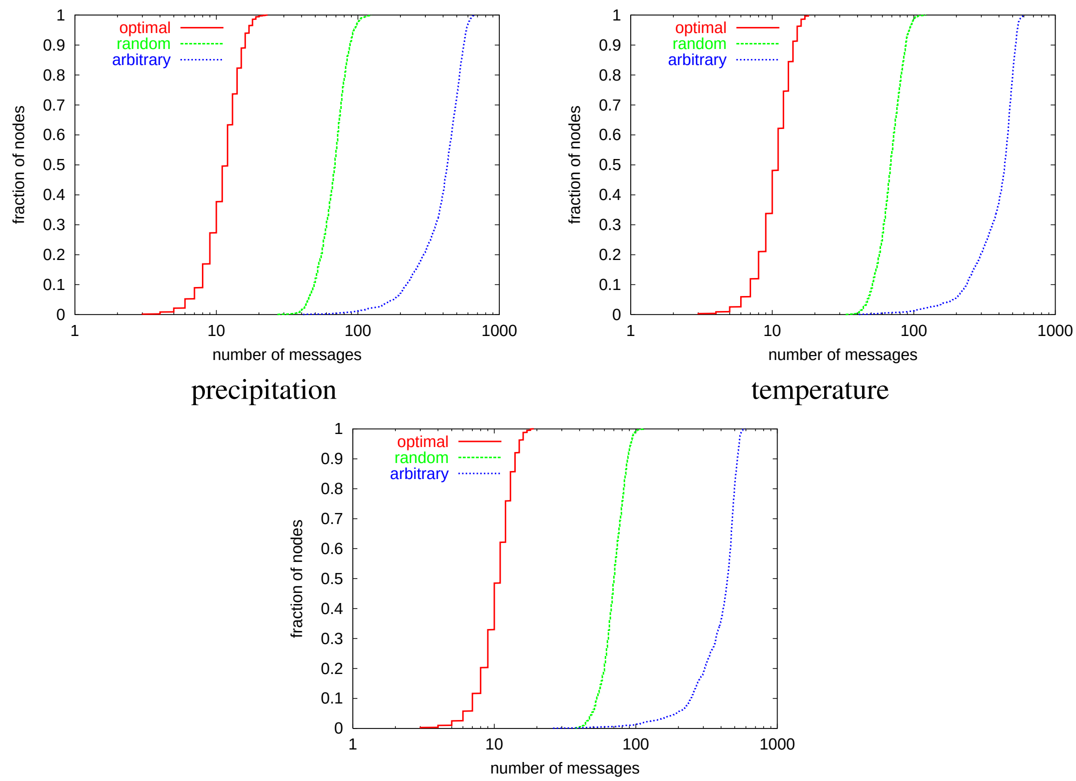
We counted the number of messages per node (with each message containing a rank and the location of the announcing node) for computing a single (∞-limited) MV/D list and the size of this list, for the three orders of announcements considered in Section 4: random order of announcements, arbitrary order (where the announcements follow a lexicographic order on the longitude, latitude coordinates), and optimal order (increasing rank). When multiple lists are used, the total communication is proportional to the number of lists. We looked at the size of the lists and number of messages for precipitation, temperature, and size values. Recall that although the communication depends on the announcement order, the size and content of the MV/D lists depend only on the ranks.

Table 2 lists the average number of messages per node under each order. Figure 3 shows the cumulative distributions. On this 86K node network, random order required about 6-7 times more messages than optimal and arbitrary order about 40 times more messages than optimal. This suggests that a combined random-minimum approach that biases earlier announcements to lower ranks can improve on the performance of random order. Theory provides precise expressions for the distribution of the size of the MV/D lists and for the accuracy of estimates using a certain number of MV/D lists when items' values are binary (this is the case when counting or computing sizes [6, 7]). For arbitrary values we provided in Section 4 a worst-case bound in terms of n^*, which depends on the ratio of the weights of the largest to smallest values. Interestingly, even though there is large variance in precipitation values, the size of the corresponding MV/D lists was only slightly larger than with binary values (11.6 versus 10.7).

We next compare our methods, that produce estimates based on MV/D lists, to the naive method, which produces exact values. The naive algorithm sends every data point to all nodes that it influences. Thus, the communication and storage at each node are proportional to the number of data points that influence it. The influence sets depends on the decay function. For general decay functions it involves 85793 data points from all peer nodes. For d-limited decay functions, the influence set contains only items in the d-neighborhood. Our method produces estimates with quality that increases with the number k of MV/D lists.

	optimal	random	arbitrary
precipitation	11.6	69	409
temperature	10.7	69	404
size	10.7	71	408

Table 2: Average number of messages to obtain an MV/D list under optimal, arbitrary, and random order of announcements, for precipitation levels, temperature, and number of sensors.



precipitation temperature

size

Figure 3: Cumulative number of messages per node needed for computing an MV/D list for precipitation, size, and temperature.

The expected size of each list is logarithmic in the influence set. The NH-summary computed from the k lists has size which is at most the sum of sizes of the k lists. It can provide estimated aggregate values for *any* decay function. The communication depends on the order of announcements, and as shown in Section 3, is logarithmic for optimal order and log-squared for random order. Thus, asymptotically there is an exponential gap in communication and storage between the naive exact method and our MV/D list based approximation.

We drew ranks from the Exponential distribution with parameter λ, where λ was the value (for sizes values were binary; for temperatures we added a value of 60 to obtain nonnegative values). The NH-summary contained a list of distance intervals, and for each distance, we used the unbiased estimate $(k-1)/\sum_{i=1}^{k} r_i$, where r_i was the minimum rank according to MV/D list i for the corresponding distance. Figure 4 shows the distribution of the ratio of the estimate value to the exact aggregate values. The approximate aggregate values were produced using an NH-summary that constituted of 10, 20, 50, or 100 MV/D lists. The figure shows, in agreement with analysis, that the error decreases with the number of lists, and that its distribution is the same for different $\textsc{Ball}_r$ functions (it does not depend on neighborhood size). The aggregate for the inverse-squared decay function $1/(1+\textsc{dist})^2$ was calculated by applying Lemma 3.1 to the NH-summary. This estimate is more accurate than over neighborhoods since it combines estimates over different neighborhoods (estimation errors for different neighborhood are only partially dependent). Figure 5 shows the number of messages per node for d-limited aggregation as a function of d. For the naive method and for the approximate method using $k = 1, 20, 100$ lists. The communication required with the naive exact computation grows much faster than with the approximate methods. Thus, exact computation is favorable for smaller d or when the number of MV/D lists needed to reach the precision that an application requires makes the communication cost comparable to that of the exact computation. The average number of influencing nodes (and per-node average communication cost for the naive exact computation) is 56.1 for 2.5 degree neighborhood, 192.3 for 5 degree neighborhood, and 662.9 for 10 degree neighborhood. For general decay functions (such as inverse-square of the distance), the influence set of each point contains all other (85793) nodes. The figure shows that when using 20 or fewer MV/D lists, d which is at least 10 degrees, and random order of announcements, the approximate method uses fewer messages than the naive method; with optimal order of announcements, the estimation method uses fewer messages even with d starting at 5-degrees. For smooth decay functions, the estimation method even with hundreds of MV/D lists is considerably more efficient than the naive method.

13 Conclusion and Open Problems

We introduced a model for spatially decaying aggregation, which is motivated by emerging applications including p2p and sensor networks where data is associated with its location and its relevance decays with distance. We developed basic techniques and efficient algorithms for some fundamental aggregate functions.

An interesting question is whether and when the problem of producing general summaries (that is, applicable to any decay function) is harder than tailored solutions to specific functions: for time-decay on streams, it is known that the decaying sum problem for particular decay functions, such as Exponential and Polynomial decay, can be tracked more efficiently than general decay [9]. We noted that restricting the decay function seems to allow for better storage bounds for combined spatial and time decay (see Section 11) and for EHs on grids (see Section 10). Another natural set of open problems is to close gaps between our upper bounds and lower bounds carried over from the time-decay model.

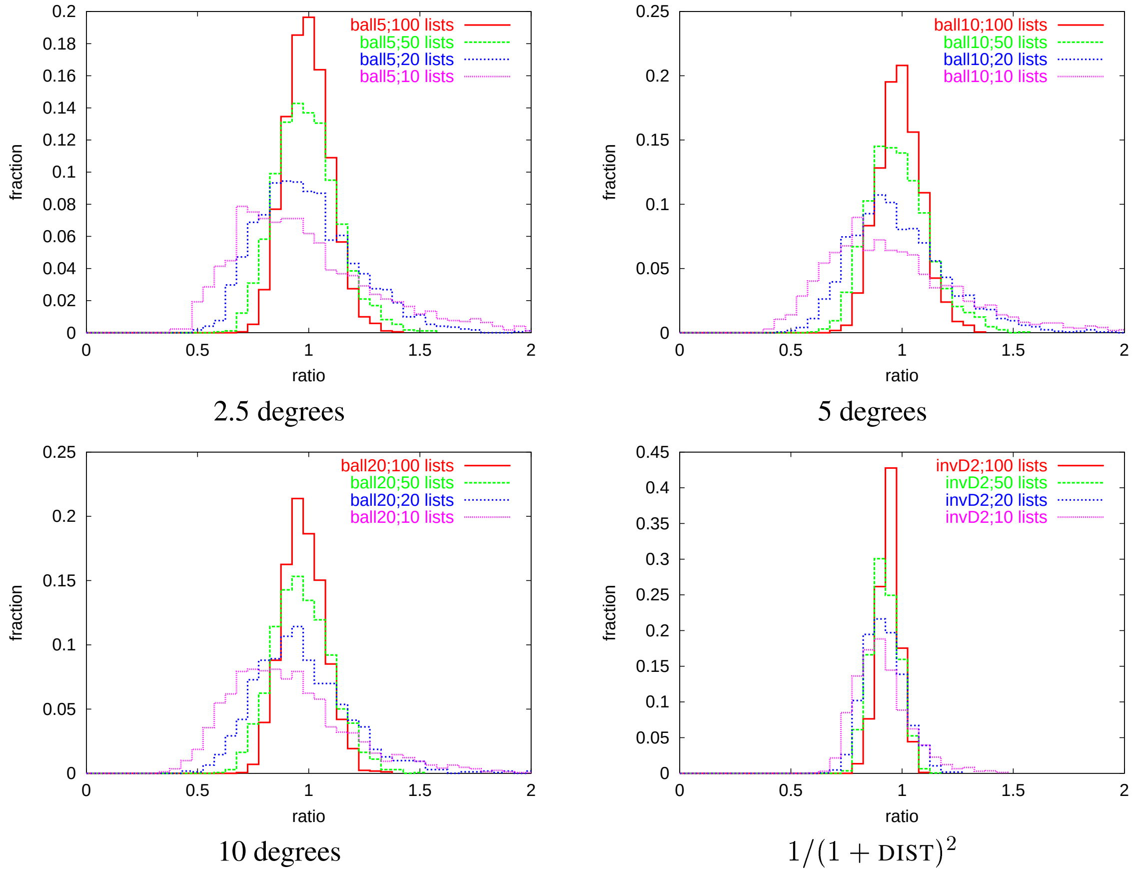
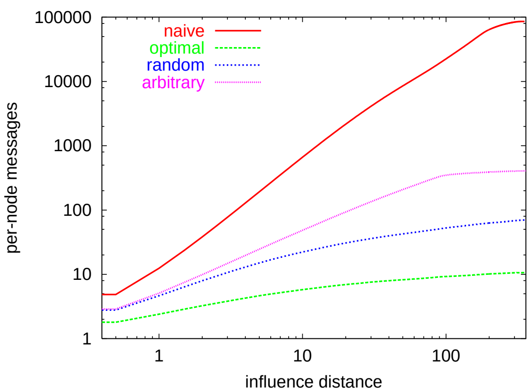
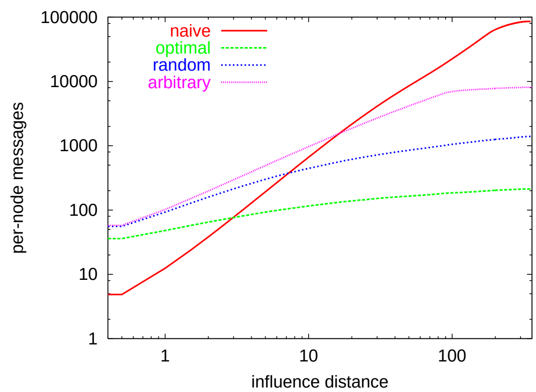


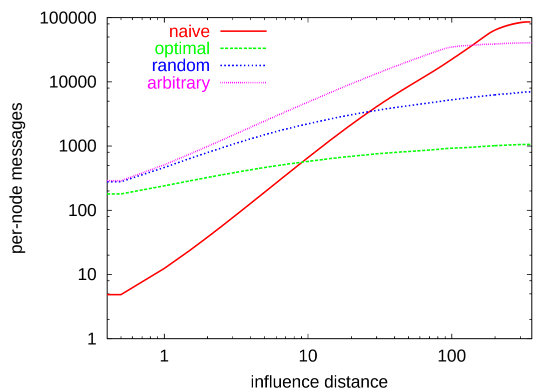
Figure 4: Histogram of the distribution of the ratio of the estimate we obtain to the exact quantity for precipitation levels for $\textsc{Ball}_r$ decay on 2.5, 5, and 10 degree neighborhoods (denoted in the figures captions by “ball5”, “ball10”, and “ball20”, respectively) and for $1/(1 + \textsc{dist})^2$ decay (denoted by “invD2”).



1 list



20 lists



100 lists

Figure 5: Number of messages as a function of the distance of influence under each method: the exact naive method and NH-summaries generated using 1, 20, or 100 MV/D lists generated with optimal, random, or arbitrary order of announcements.

References

[1] B. Awerbuch. Distributed shortest paths algorithms. In *Proc. 21st Annual ACM Symposium on Theory of Computing*, pages 490–500. ACM, 1989.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. of the 2002 ACM Symp. on Principles of Database Systems (PODS 2002)*. ACM, 2002.

[3] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *RANDOM*. ACM, 2002.

[4] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

[5] Terrestrial air temperature and precipitation: monthly and annual climatologies (version 3.02), 2001. `http://climate.udel.edu/~climate`.

[6] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. System Sci.*, 55:441–453, 1997.

[7] E. Cohen. Structure prediction and computation of sparse matrix products. *J. Combinatorial Optimization*, 2:307–332, 1999.

[8] E. Cohen and H. Kaplan. Efficient estimation algorithms for neighborhood variance and other moments. In *Proc. 15th ACM-SIAM Symposium on Discrete Algorithms*. ACM-SIAM, 2004.

[9] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *Proc. of the 2003 ACM Symp. on Principles of Database Systems (PODS 2003)*. ACM, 2003.

[10] Open Source Community. Gnutella. In *http://gnutella.wego.com/*, 2001.

[11] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *ICDE*, 2002.

[12] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002.

[13] A. Deshpande, P. Gibbons, S. Nath, and S. Seshan. Cache-and-query for wide area sensor databases. In *Proceedings of the ACM SIGMOD*, 2003.

[14] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2001.

[15] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L1-difference algorithm for massive data streams. In *Proc. 39th IEEE Annual Symposium on Foundations of Computer Science*, pages 501–511. IEEE, 1999.

[16] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. System Sci.*, 31:182–209, 1985.

[17] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transaction on Networking*, 1(4), 1993.

[18] M. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. Assoc. Comput. Mach.*, 34(3):596–615, 1987.

[19] S. Ganeriwal, Han C. C., and M. B. Srivastava. Spatial average of a continous physical process in sensor networks (poster). In *Proceedings of the 1st ACM conference on embedded networked sensor systems (Sensys)*, 2003.

[20] D. Ganesan, S. Ratnasamy, H. Wang, and D. Estrin. Coping with irregular spatio-temporal sampling in sensor netowrks. In *Proceedings of the ACM SIGCOMM HOTNETS-II workshop*, 2003.

[21] P. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures*. ACM-SIGMOD, 2001.

[22] P. B. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *Proc. of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 63–72. ACM, 2002.

[23] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of the ACM SIGMOD*, 2001.

[24] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[25] P. Indyk. A small approximately min-wise independent family of hash functions. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*. ACM-SIAM, 1999.

[26] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proc. 41st IEEE Annual Symposium on Foundations of Computer Science*, pages 189–197. IEEE, 2001.

[27] V. Jacobson. Congestion avoidance and control. In *Proceedings of the ACM SIGCOMM'88 Conference*, August 1988.

[28] J. Kleinberg and D. Kempe. Protocols and impossibility results for gossip-based communication mechanisms. In *Proc. 43rd IEEE Annual Symposium on Foundations of Computer Science*, pages 471–480. IEEE, 2002.

[29] J. Kleinberg, D. Kempe, and A. J. Demers. Spatial gossip and resource location protocols. In *Proc. 33rd Annual ACM Symposium on Theory of Computing*, pages 163–172. ACM, 2001.

[30] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbors. In *Proc. of the 2000 ACM SIGMOD conference on Management of Data*, pages 201–212. ACM, 2000.

[31] F. Korn, S. Muthukrishnan, and D. Srivastava. Reverse nearest neighbor aggregates over data streams. In *Proc. of the 28th international conference on Very Large Databases (VLDB 2002)*. ACM, 2002.

[32] X. Lin, H. Lu, J. Xu, and J. X. Yu. Continously maintaining quantile sumaries of the most recent n elements over a data stream. In *Proceedings of the 20th International conference on data engineering (ICDE)*. IEEE, 2004.

[33] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (ODSI)*. USENIX Association, 2002.

[34] A. Mainwaring, J. Polastre, R Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International workshop on wireless sensor networks and applications*, 2002.

[35] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc. of the 28th international conference on Very Large Databases (VLDB 2002)*, pages 346–357. ACM, 2002.

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

[37] M. R. Paterson. Progress in selection. Dept. of Computer Science, University of Warwick, Coventry, UK, 1997.

[38] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *Proc. of First IEEE international Workshop on Sensor Network Protocols and Applications*, 2003.