The VLDB Journal
DOI 10.1007/s00778-011-0233-y

REGULAR PAPER

# Distributed similarity estimation using derived dimensions

**Konstantinos Georgoulas · Yannis Kotidis**

Received: 8 August 2010 / Revised: 12 March 2011 / Accepted: 6 April 2011
© Springer-Verlag 2011

**Abstract** Computing the similarity between data objects is a fundamental operation for many distributed applications such as those on the World Wide Web, in Peer-to-Peer networks, or even in Sensor Networks. In our work, we provide a framework based on Random Hyperplane Projection (RHP) that permits continuous computation of similarity estimates (using the cosine similarity or the correlation coefficient as the preferred similarity metric) between data descriptions that are streamed from remote sites. These estimates are computed at a monitoring node, without the need for transmitting the actual data values. The original RHP framework is data agnostic and works for arbitrary data sets. However, data in most applications is not uniform. In our work, we first describe the shortcomings of the RHP scheme, in particular, its inefficiency to exploit evident skew in the underlying data distribution and then propose a novel framework that automatically detects correlations and computes an RHP embedding in the Hamming cube tailored to the provided data set using the idea of derived dimensions we first introduce. We further discuss extensions of our framework in order to cope with changes in the data distribution. In such cases, our technique automatically reverts to the basic RHP model for data items that cannot be described accurately through the computed embedding. Our experimental evaluation using several real and synthetic data sets demonstrates that our proposed scheme outperforms the existing RHP algorithm and alternative techniques that have been proposed, providing significantly more accurate similarity computations using the same number of bits.

K. Georgoulas (✉) · Y. Kotidis
Athens University of Economics and Business,
76 Patission Street, Athens, Greece
e-mail: kgeorgou@aueb.gr

Y. Kotidis
e-mail: kotidis@aueb.gr

**Keywords** Locality sensitive hashing · Similarity · Sensor networks

## 1 Introduction

Computing the similarity between data items is a fundamental operation for many applications. Quite often, the data in question are not available at a central location, but is rather dispersed, between sites. For example, pervasive applications are increasingly supported by networked sensory devices that interact with people and themselves in order to provide the desired services and functionality. Because of the unattended nature of many installations and the inexpensive hardware used in the construction of the sensors, nodes often generate imprecise individual readings due to interference or failures [29]. Recent proposals consider mechanisms for cleaning sensor readings by identifying and, possibly, removing outliers [9,13,17,18]. At the core of these techniques lies the need of a network-friendly mechanism for computing the similarity between recent measurements that are streamed from distant nodes. Sensory data collected by nodes needs to be processed and understood in a decentralized manner, so as to avoid depleting the limited resources available at the nodes. A central collection of sensory data is not feasible nor desired, since it results in high energy drain, due to the large number of transmitted messages. Moreover, data in most applications, are continuously collected by the sensor nodes, increasing the need to develop techniques that limit the amount of data transmitted [11,12,17].

The need to perform similarity tests between data is also evident in distributed applications such as those on the World Wide Web (WWW) or in Peer-to-Peer (P2P) networks. Data sharing, management, and query processing in such settings require techniques that reduce the need to transfer

Published online: 22 April 2011

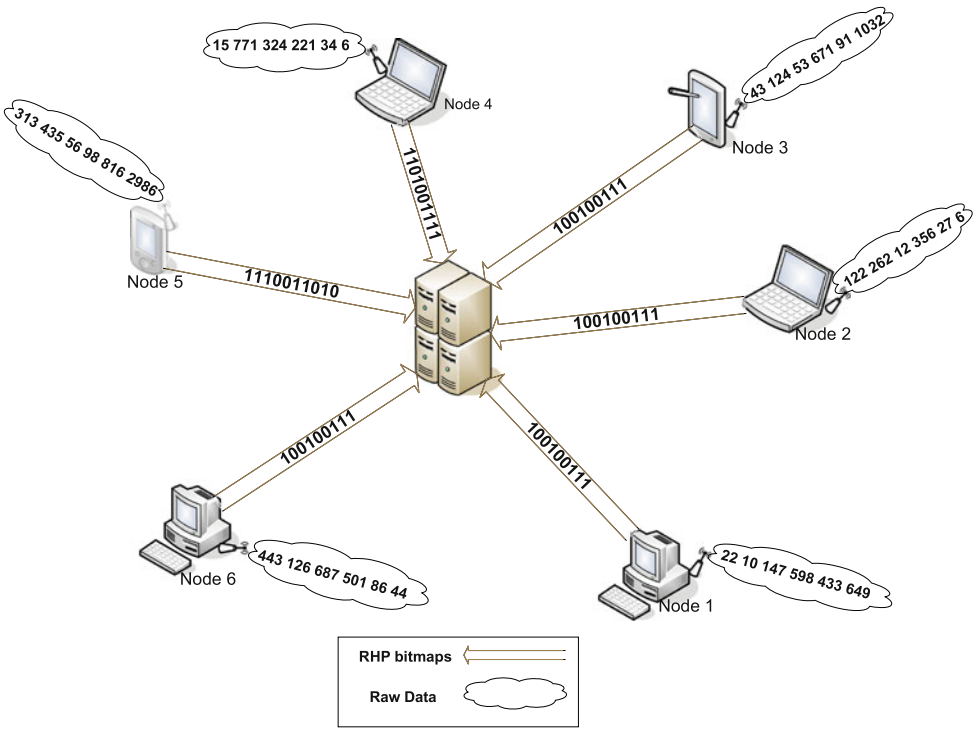



**Fig. 1** An example of a distributed environment, our work could be applied

or duplicate data (or meta-data) among sites. As an example, recent proposals consider peer-to-peer architectures for enabling advanced query processing, when data are horizontally distributed among sites [31,43]. In order to facilitate efficient query routing and processing via an overlay network topology, data clustering techniques are implemented. These techniques often require the evaluation of similarity metrics between data stored at different peers. Likewise, when integrating data stores over the Web, understanding not only schema but also data (dis)-similarity is fundamental for the success of any integration task.

All of the aforementioned applications, while diverse in their assumptions and architecture, share the need of a primitive operation that will allow the assessment of similarity between compact descriptions of data that are streamed from different sites. These descriptions need to be (1) easily computed from the local data attributes, so as to reduce processing cost, and (2) compact in size, so as to permit their network transmission, instead of the original data, for similarity testing. In our work, we assume that transmitting all data to a central site for further processing is not a viable solution. For instance, in P2P or WWW applications, the data may be simply too much to be transported. Similarly, in most sensor network applications, energy and bandwidth restrictions do not permit us to take the data out of the network as this will quickly deplete the batteries of the motes and will result in large number of network collisions [18]. Moreover, in such applications, sensor motes have typically limited amount of memory. This means that the data they acquire lives only for a limited amount of time before being replaced by more recent observations.

A key aspect of our framework is that we regard similarity estimation as a continuous process. This process is highlighted in Fig. 1 where a number of remote sites stream data descriptions to a central (monitoring) node. The monitoring node should be able to estimate the similarity between data streamed by any of these sites by looking at these descriptions only. The data sources are not required to store the original data indefinitely (for instance, in a sensor network application, this data may be discarded rather quickly).[1] The lack of a permanent data storage at the remote sites renders distributed indexing techniques (e.g. [5,28,42,43]) inapplicable to our problem. Moreover, since data reduction is necessary and the original data are never transmitted to the monitoring node, centralized indexing techniques [2,10,26,27,36,40,41,46] are also not appropriate for our setting.

In our work, we provide a framework that permits continuous computation of similarity estimates between data descriptions that are streamed from remote sites. These estimates are computed at the monitoring node, without the need for transmitting the actual data values (as in centralized indexing) nor for further communication with the remote sources (as in the distributed indexing scenario). Our techniques assume a generic description of data as

[1] Depending on the application, the monitoring site may store the data descriptions for further processing, or discard them. However, our techniques are orthogonal to this decision.

multidimensional points and allow the computation of common similarity metrics such as the cosine coefficient and the correlation coefficient. We adopt the Random Hyperplane Projection (RHP) framework [7,23], a novel dimensionality reduction technique, based on Locality Sensitive Hashing (LSH) [7,27] that is used to transform each $d$-dimensional point into a much shorter bitmap of $n$ bits. This encoding is performed independently at each site for its local data. RHP is a powerful technique that trades accuracy for bandwidth, by simply varying the desired level of dimensionality reduction. The loss of accuracy comes from the projection of the original data into a space of lower dimensionality, however, it can be easily controlled by varying the desired length of the bitmaps and through a boosting process that utilizes multiple (shorter) bitmaps [18] for computing the similarity of the data objects.

The main drawback of the RHP mapping is that it assumes a uniform distribution of the data in the $d$-dimensional space. Nevertheless, data in real applications is unlikely to be uniform. As an example, when sensors monitor physical quantities like humidity, pressure, or light, the collected data values are typically skewed, reflecting the conditions in the monitored area. In Web or P2P applications, data on peers is usually clustered into a few thematic areas that reflect the user's interests. As will be demonstrated the uniform assumption of a typical data-agnostic RHP encoding scheme, severely limits its performance. In this work, we propose a dimensionality reduction framework that takes into account skew that is often evident in the data distribution. Our techniques deliberately alter the way data are mapped into a lower-dimensionality space so as to increase the accuracy of the produced bitmaps. Moreover, our algorithms retain the advantages of the basic RHP scheme, in particular, its simplicity in producing the mappings and subsequently computing the similarity of the original data objects based on them. This is necessary in applications where nodes have limited processing capabilities (as in sensor nodes) or when large volumes of data need to be processed (as in Web or P2P applications).

The main advantage of our proposed framework is that it detects and exploits skew or correlations in the underlying data distribution. Similar to RHP, a $n$-dimensional representation of the data are computed in the Hamming cube. However, our framework also computes an extended mapping into $m$-additional dimensions. This mapping is derived based on simple precomputed statistics obtained from a sample of the data and the available embedding on the $n$-dimensional Hamming cube. In this way, our method manages to project the data items into a higher dimensionality space ($n$+$m$ dimensions), while still using $n$ bits for the data encodings. The additional derived $m$ dimensions are utilized when computing the similarity between the data items, resulting in this way in more accurate evaluations. The new framework is termed $RHP(n, m)$ in order to distinguish it from the basic RHP technique.

The contributions of this paper can be summarized as follows:

- We introduce $RHP(n, m)$, a LSH data reduction framework that supports popular similarity measures, namely the cosine coefficient and the correlation coefficient. Unlike the original RHP technique that is oblivious to the underlying data, $RHP(n, m)$ utilizes prior knowledge of the data distribution in order to produce, with the same space, more accurate descriptions and thus, allows a more accurate computation of the similarity based on them.
- We describe a novel process for capturing the distribution of the data and accordingly alter the RHP mappings. This is achieved by computing a few intuitive statistics using a sample obtained not from the data items (as this would negate the benefits of the whole framework) but rather from much shorter RHP encodings. We then utilize these statistics in order to select those RHP dimensions that better describe the underlying data.
- We introduce techniques that detect data items that cannot be accurately described using the available statistics. For such items, our algorithms automatically fall back into using the basic $RHP(n)$ scheme that is oblivious to the data characteristics.
- We present a detailed experimental analysis of our techniques for a variety of real and synthetic data sets. Our results demonstrate that our methods can reliably compute the similarity between data objects and consistently outperform the standard RHP scheme.

This paper proceeds as follows. In Sect. 2, we present an application of our framework for computing outliers in wireless sensor networks. In Sect. 3, we discuss related work. Section 4 presents the basic Random Hyperplane Projection (RHP) technique and discusses its advantages and shortcomings. In Sect. 5, we formally present our framework, while in Sect. 6, we describe interesting extensions to our basic scheme that increase its accuracy and allows it to cope with changes in the underlying data distribution. Section 7 presents our experimental evaluation, while Sect. 8 provides concluding remarks.

## 2 Motivational example

In our recent work, we introduced a distributed framework for computing outliers in wireless sensor networks based on RHP [18]. Our method assumes a clustered network organization depicted in Fig. 2. Regular sensor motes compute RHP encodings from their latest $d$ measurements. These encodings are transmitted to their clusterheads, which can in-turn estimate the similarity among the latest values of any

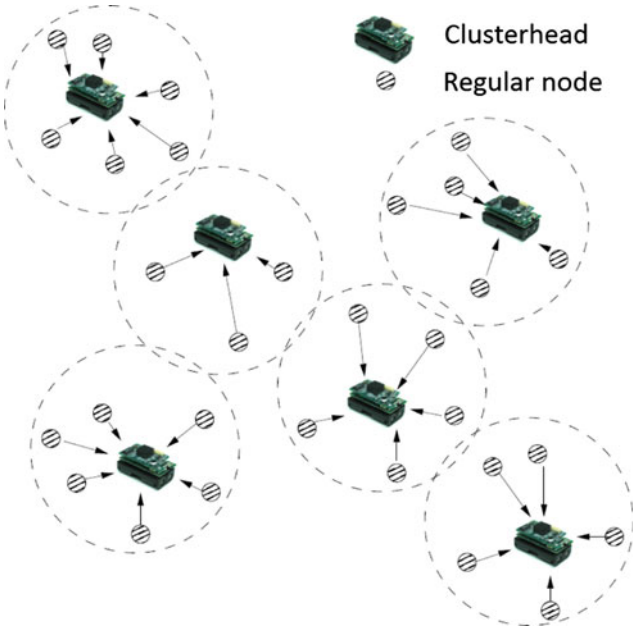



**Fig. 2** Computing outliers [18]: motes transmit RHP bitmaps describing their latest $d$ measurements to their clusterheads. Each clusterhead computes a local list of potential outliers based on pair-wise similarity tests of all motes in its cluster. Local lists are communicated between clusterheads in order to compute a final global list of outliers (not shown in figure)

pair of motes within its cluster by comparing their bitmaps. Based on the performed similarity tests and a desired minimum support specified by the application, each clusterhead is able to compute a local list of potential outliers. These lists are then communicated among the clusterheads in order to compute the final list of outliers that is reported to the application.

This particular application scenario possesses some unique requirements:

- Data are continuously generated at remote locations (sensor motes). This data cannot be stored locally, due to the limited memory capabilities of the sensor nodes [37]. Thus, this data, or an appropriately constructed summary, need to be transmitted to the monitoring site (the clusterhead in this setup).
- A continuous transmission of raw data measurements is not possible nor desired, since (1) the networking capabilities in these kind of ad-hoc networks are rather primitive and (2) network operation is associated with excessive energy drain [37].
- Some form of data reduction is required in order to reduce bandwidth and energy consumption. The data summaries should (1) be of limited size to reduce communication cost and (2) allow us to estimate similarity without the need to access the raw data (that is not available).

Thus, at the core of this application lies the requirement to accurately evaluate the similarity between distant multidimensional data vectors (containing the measurements obtained by the sensors) in a network-friendly manner. The work of [18] discussed a solution using the much sorter *RHP* bitmaps instead. The techniques we present in this paper can be directly applied in this scenario. All is required is to replace the original RHP bitmaps with the proposed $RHP(n, m)$ encodings. This would result in increased accuracy when computing the similarity between sensory data, compared to using *RHP*.

## 3 Related work

Random Hyperplane Projection (RHP), the particular Locality Sensitive Hashing (LSH) scheme that we extend in this work, was initially introduced in [23] to provide solutions to the MAX-CUT problem. Since then, other forms of LSH have been applied in many applications including similarity estimation [7] and clustering [38]. The most common use of LSH is as an indexing structure (typically a main-memory one) intended to support approximate nearest neighbor (NN) queries [2,10,27] or as an index for set value attributes [22]. Most of these techniques assume a centralized computation of such queries where the aim is to utilize the LSH index in order to reduce the number of objects that need to be retrieved for NN retrieval. The work of [25] considers different LSH mappings for distributed NN indexing of high dimensional data.

The aforementioned techniques consider LSH schemes as an indexing structure for $L_p$ norms, while in our work, we extend RHP (a substantially different form of LSH) that is devised for the cosine similarity, a popular yet different error metric. Moreover, the form of LSH considered in these papers is mainly used to create an in-memory "index" that is subsequently probed for nearest neighbor queries. In creating this index, an object needs to be hashed into multiple buckets in order to achieve reasonable accuracy, thus, the size of this index is often larger than the data size.[2] This property renders these particular techniques unsuitable for distributed settings where the volume of transmitted data need to be reduced (as in a sensor network application). Clearly, in a distributed setting it would be cheaper to simply transmit the raw data to the central cite. In this way, similarity could be evaluated with 100% accuracy with smaller communication overhead.

There are recent proposals on LSH indexing [26,36] that try to overcome the space-explosion that the traditional technique requires. For instance, in [36], the authors present

[2] Even if pointers to the data objects are utilized, (1) these pointers still consume space and (2) the actual data objects need also be present during query evaluation.

Multi-probe LSH, a novel extension that carefully devises a probing sequence to look up multiple buckets that have a high probability of containing the nearest neighbors of a query object. As a result they achieve the same performance with the traditional LSH index using fewer number of hash tables. The substantial space reduction achieved makes this technique applicable for NN retrieval in a centralized setting when large volumes of data need to be indexed. In [15], the authors utilize a sample of the data to model Multi-probe LSH and devise an adaptive search strategy that reduces performance variance between different NN queries. A well-known shortcoming of the traditional LSH index is that it requires tuning of (i) the number of hash functions required to build the label of each bucket and (ii) the number of independent buckets to construct. These parameters depend on the number of points in the data set and the value distribution. The authors of [5] discuss a tuning strategy that takes into account data distribution for the parameters of LSH and also present an embedding of the index they propose in a P2P system. The authors of [40,41] introduce locality sensitive B-tree (LSB-tree), an intuitive access method that indexes the hash values of the objects based on their Z-order values. The coordinates of the raw data objects are stored in the leaves of the tree along with these values.

A key difference between all such forms of LSH devised for $L_p$ norms and the RHP framework is that while the LSH index helps eliminate distant points from consideration, in order to actually compute the distances between the data points and locate the nearest neighbors, the actual data vectors (or some locally stored approximation of them) are required. To the best of our knowledge, there is no known formula for deriving the distances of the objects based on their LSH hash values. RHP does not have this limitation. As will be explained, cosine similarity can be estimated using the RHP bitmaps based on their hamming distance. Thus, in our technique (1) we do not transmit nor we maintain the local data points (for instance this is infeasible in the Senor Networks scenario we described in Sect. 2), (2) our focus is on reducing the number of transmitted bits thus some form of data reduction is necessary, and (3) we do not address the problem of NN using $L_p$ norms but rather we aim to compute the similarity between short descriptions of the data that are streamed from remote sites in a continuous manner.

In our work, we investigate techniques that increase the accuracy of RHP applied in similarity estimation in a streaming, distributed setting. The Random Hyperplane Projection scheme [23] that we consider has been originally proposed to compute the cosine coefficient. In [18], it is shown that the correlation coefficient can also be computed using the same family of hash functions. In [22], a different family of hash functions has been proposed in order to compute the Jaccard index. The correlation coefficient and the Jaccard index have been recently considered and evaluated for detecting outliers in sensor networks [13,18,44]. The cosine similarity has been used in diverse applications such as IP traffic monitoring [19] and computing the similarity between documents [33,35]. Our techniques can be used in any of the aforementioned applications for extending the accuracy of the similarity evaluations between data items through their RHP encodings, while limiting the amount of data that needs to be transferred between remote sites.

In our recent work [13,18,34], we have tackled the problem of computing the similarity between readings that are generated by sensor nodes. This particular application posses some unique requirements. First, due to the limited memory capabilities of sensor nodes, in most sensor network applications, data are continuously collected by motes and maintained in memory for a limited amount of time. Thus, permanent distributed storage of the data are not desirable. This requirement renders distributed indexing techniques inapplicable, since data itself is short-lived. Moreover, in such an application, results need to be generated continuously and computed based on recently collected measurements. Furthermore, a central collection of sensor data is not feasible nor desired, since it results in high energy drain, due to the large amounts of transmitted data [18]. Hence, what is required are continuous distributed and in-network approaches that reduce the communication cost and manage to prolong the network lifetime. Our proposed techniques are directly applicable in such settings as they can help reduce the number of transmitted messages while retaining the ability to accurately compute similarity.

There is an abundance of related work for similarity estimation in metric spaces [8], including among others, dimensionality reduction techniques [3,16,32], and indexing structures [6,30,46]. Recently, the authors of [4] proposed a hash-based indexing method for approximate nearest neighbor retrieval that can also be applied to non-metric distance measures as well. Several dimensionality reduction techniques have been proposed recently, both for centralized and distributed, possibly streaming, applications. Examples include Sketches [1,14], Wavelets [21,39], and Histograms [20,24,32]. While some of these techniques can be adapted for the problem we consider, a key difference is that they are typically applied for reducing the frequency distribution over large data domains, such as the set of IP addresses in the Internet (e.g. [19]) or the customers or products in a data warehouse (e.g. [39]). In contrast, in our applications, the dimensionality of the original data space is much lower, as the reduction is performed at a per-item basis. Thus, the embeddings they provide are (for most of them) of the form of multidimensional vectors and require real values for storing the computed coordinates. In contrast, *RHP* projects the data into the Hamming cube, thus, the coordinates of the points in the projected space are 0/1 bit values.

**Table 1** Notation used in this paper

| Symbol | Description |
|---|---|
| $x$,$y$ | Data items described as $d$-dimensional vectors (points) |
| $\theta(x, y)$ | The angle between vectors $x$,$y$ |
| $lsh(x)$ | The bitmap encoding produced after applying RHP to $x$ |
| $n$ | RHP bitmap length |
| $r_i$ | $i$th random $d$-dimensional vector |
| $h_{r_i}(x)$ | Hash function for $r_i$ applied on data item $x$ |
| $D_h(lsh(x), lsh(y))$ | The hamming distance between RHP bitmaps |
| $P_{j\|i}$ | Probability that $h_{r_j}(x) = 1$ when $h_{r_i}(x) = 1$ |
| $P_{j\|\neg i}$ | Probability that $h_{r_j}(x) = 1$ when $h_{r_i}(x) = 0$ |
| $expLsh(x)$ | $(n+m)$-dim vector: $n$ values are obtained from $lsh(x)$, remaining $m$ values are derived using conditional probabilities $P_{j\|i}$, $P_{j\|\neg i}$ |

This allows our techniques to provide substantially higher reduction ratios, as is made evident by our experimental evaluation.

## 4 Preliminaries

### 4.1 The RHP framework

We now present the basic locality sensitive hashing scheme that our framework extends. The notation used in our discussion is summarized in Table 1. The corresponding definitions are presented in appropriate areas of the text.

A *Locality Sensitive Hashing scheme* is defined in [7] as a distribution on a family $F$ of hash functions that operate on a set of data items, such that for two data items $x$, $y$:

$$P_{h\epsilon F}[h(x) = h(y)] = sim(x, y) \tag{1}$$

where $sim(x, y)\epsilon[0, 1]$ is some similarity measure.

In our framework, we utilize a particular form of LSH termed Random Hyperplane Projection (RHP) [7,23]. We assume a collection of data described in the $d$-dimensional space. In RHP, we generate a family of hash functions as follows. We produce a spherically symmetric random vector $r$ of unit length from this $d$ dimensional space. Using $r$, we define a hash function $h_r$ as:

$$h_r(x) = \begin{cases} 1, & \text{if } r \cdot x \geq 0 \\ 0, & \text{if } r \cdot x < 0 \end{cases} \tag{2}$$

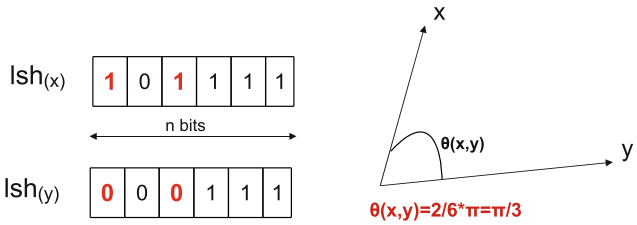



**Fig. 3** Estimating the angle of two data items $x$, $y$ using their $lsh(x)$ bitmaps

i.e. $h_r()$ evaluates to 1 for all data items whose dot product with $r$ is positive and to 0 for the rest of the data. It is easy to see [45] that for any two vectors $x$, $y$

$$P[h_r(x) = h_r(y)] = 1 - \frac{\theta(x, y)}{\pi} \tag{3}$$

If we repeat this process using $n$ random vectors $r_1, \ldots, r_n$, an input data item $x$ is mapped into a bitmap $lsh(x)$ of length $n$. Bit $i$ in this bitmap is the evaluation of $h_{r_i}(x)$.

Let $x$ and $y$ be two input data items and $lsh(x)$, $lsh(y)$ their RHP bitmaps, respectively. Based on Eq. 3 it follows that

$$\frac{\theta(x, y)}{\pi} = \frac{D_h(lsh(x), lsh(y))}{n} \tag{4}$$

$D_h(lsh(x), lsh(y))$ in the above formula denotes the hamming distance of the produced bitmaps. This equation states that the number of bits that differ in the RHP encodings of vectors $x$ and $y$ is proportional to their angle. Solving the formula for $\theta(x, y)$ allows us to estimate the angle between the two vectors from their RHP encodings.

For example, suppose that an application needs to estimate the similarity between two data items $x$, $y$ whose RHP bitmaps are shown in Fig. 3. According Eq. 4, only the bits in which their corresponding bitmaps differ contribute to the estimation of their angle $\theta(x, y)$. Furthermore, the more bits in which the two bitmaps differ the greater the estimated angle $\theta(x, y)$. In this particular example, only two of the six bits in these bitmaps differ and the estimated angle due to the Eq. 4 is $\theta(x, y) = \frac{2}{6} * \pi = \frac{\pi}{3}$.

From the angle computation, one can trivially derive the cosine similarity $cos(\theta(x, y))$ between $x$ and $y$. Moreover, let $E(x)$ denote the mean value of vector $x$. The correlation coefficient $corr(x, y)$ between $x$ and $y$ can then be computed as $corr(x, y) = corr(x - E(X), y - E(y)) = cos(\theta(x - E(x), y - E(y))$ [18]. Thus, using the RHP bitmaps, we can also compute the correlation coefficient of $x$ and $y$. Both these metrics are fundamental in assessing the similarity between data items. For instance, the cosine similarity is used in [19] in evaluating the similarity between network traffic patterns in IP networks. Similarly, the correlation coefficient has been recently used in detecting outliers in measurements obtained from sensor networks [13].

### 4.2 Benefits and shortcomings of RHP

The basic RHP scheme is an intuitive method for reducing the size (dimensionality) of the input data items, while retaining the ability to compute the angle (similarity) between them. Moreover, the RHP scheme works easily in distributed settings. What is required is that all sites (sensor nodes, peers, etc) utilize a common seed value in order to generate locally the same family of random vectors $r_i$. (Thus, there is no need to transfer the random vectors between sites at a pre-processing step.) Then, the $lsh()$ encodings can be constructed independently and communicated as needed in order to compute similarity between data objects stored in remote sites. The benefit of applying RHP is that much fewer bits need to be transferred in such cases. For example, assuming a typical 32 bit internal representation of real values, the reduction ratio $RR$ obtained by using RHP bitmaps of length $n$ instead of the actual data objects is:

$$RR = \frac{\text{size of original data description}}{\text{size of RHP bitmap}} = \frac{32 \times d}{n} \tag{5}$$

Thus, the benefits of RHP increase linearly with the volume of data that needs to be transmitted. Another characteristic that increases the suitability of RHP for restricted environments is that its encodings are computed in a straightforward manner. All that is required is to compute the sign of simple linear equations (dot products). In case of severe memory constraints, a site does not need to store the random vectors $r_i$ locally. Using the common seed, the random vectors can be generated on the fly for the computation of each dot product. Thus, the technique requires $O(d)$ space and $O(n \times d)$ time per item.[3] Both requirements are rather modest. Computing the angle between two data objects from their encodings through Eq. 4 entails the computation of the hamming distance between bitmaps $lsh(x)$, $lsh(y)$, a process that is done efficiently in most platforms by XORing the bitmaps and counting the ones in the result.

The shortcomings of the basic RHP scheme stem from the fact that it requires a family $F$ of random vectors $r_i$ that are uniformly distributed in the $d$-dimensional space. When the distribution of the data objects is not uniform, this results in under-utilizing many members of $F$. Figure 4 provides an intuition of how RHP works in two-dimensions. For the sake of this example, let us assume that all data objects (two-dimensional points) fall in the area (slice) denoted as $D$ in the figure. It is easy to see that for all random vectors $r_i$ that do not belong in one of the two "orthogonal" slices $O_1$ and $O_2$ in the figure, their dot product with $x$ always has the same sign. All such random vectors do not contribute in computing the angle between two objects $x$ and $y$ (from slice $D$), since

[3] If all random vectors are materialized, the space requirements increase to $O(n \times d)$.

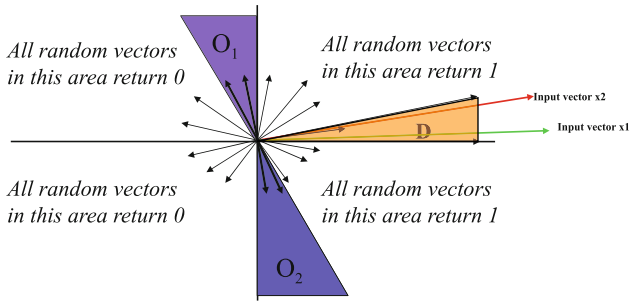



**Fig. 4** Performance of $RHP(n)$ when all data falls into area $D$. Only random vectors in the shaded areas $O_1$ and $O_2$ can help distinguish between different data items

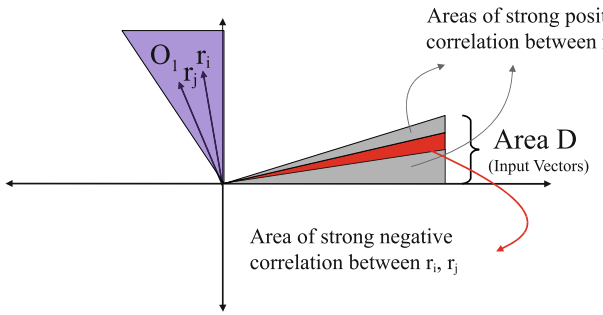



**Fig. 5** Correlations between bit values of random vectors in orthogonal area $O_1$ for input vectors lie inside the area D

the corresponding bits will either be both set (one) or clear (zero).

Only random vectors from slices $O_1$ and $O_2$ may produce different results for $x$ and $y$. When data skew increases, slices $D$, $O_1$, and $O_2$ become thinner and, thus, the percentage of "useful" random vectors $r_i$ decreases proportionally. Similar arguments apply in higher dimensions. This simplified example demonstrates that for data that is not uniformly distributed in $R^d$, often many of the bits used in the $lsh(x)$ encoding cannot contribute toward computing the angle of the vectors. This means that out of the $n$ bits that we transmit, typically only a few of those are helpful in computing the similarity between the data. Unfortunately, without knowing before-hand the values of $x$ and $y$, it is not possible to decide, which of the random vectors are useful and which are not.

Following the same example, we can see another aspect of the original method that can be exploited to achieve better results. As it was described in the previous paragraph, only random vectors from slices $O_1$ and $O_2$ may produce different bit values for $x$ and $y$ and so be helpful in computing the similarity between the data. However, some of these "helpful" bits may not be transmitted because, as will be explained, their values can be estimated by bits contributed by other random vectors of these orthogonal areas.

In the example shown in Fig. 5, we can observe correlations between bit values of random vectors that lie inside

the orthogonal area $O_1$. It is obvious from the figure that for an input vector that lies inside the red subarea of area D, the bit values of $r_j$ and $r_i$ are always different, while for input vectors that lie inside the gray areas of slice D, both $r_j$ and $r_i$ return the same bit value. Thus, these two random vectors' bit values are correlated and more specifically, they are negatively correlated for input vectors in the red area and positively correlated for input vectors in the gray areas. So, depending where most data items are (red or gray areas) given the bit value of one random vector inside area $O_1$, for example bit of $r_i$, we may be able to estimate the bit value of the other random vector $r_j$.

This observation leads to the conclusion that capturing the correlations between the bit values of the random vectors can result in estimation of the similarity between the data with reduced communication cost. This happens because random vectors' bit values, which are strongly correlated with other bits, can be estimated (at the central station) given the bit values of the other (transmitted) random vectors. This observation is key to the framework we present in the next section.

## 5 Our $RHP(n, m)$ scheme

### 5.1 Key ideas

We now discuss our new LSH scheme that alleviates the shortcomings of RHP while it retains its benefits. To distinguish it from our proposed framework, we will refer to the basic RHP process as $RHP(n)$. As already noted, quite often many of the $n$ random vectors employed in $RHP(n)$ do not contribute in the computation of the similarity, as they result in similar bits (one or zero) for many data items and moreover, for random vectors that contribute in the computation of similarity, their bits are often strongly correlated and so some of these bits can be estimated and their transmission may be avoided. A key idea of our framework is to detect and exploit such correlations between the random vectors $r_i$ by considering an extended family of $n$+$m$ random vectors (for $m \geq 0$). As in $RHP(n)$, this family is computed using a common seed value. We will deliberately distinguish two types of random vectors: materialized and derived. Materialized random vectors contribute to the encoding $lsh(x)$ by producing a bit value based on Eq. 2. Derived random vectors $r_j$ are not used in constructing the bitmap. As will be explained, the value of their hash function $h_{r_j}(x)$ is estimated using the $h_{r_i}(x)$ values of the materialized $r_i$s and some precomputed statistics. In total, there are $n$ materialized random vectors and $m$ derived ones. Thus, the $lsh(x)$ encoding of $x$ in our framework will still contain exactly $n$ bits, entailing the same construction and communication overhead as in $RHP(n)$. We will refer to our proposed framework as $RHP(n, m)$.

The difference in the new scheme is in the way we decode the bitmaps for computing the angle between two data items $x$ and $y$. During the decoding process, all $n + m$ random vectors are utilized, resulting in increased accuracy compared with $RHP(n)$ that only utilizes $n$ random vectors. In our framework, each derived random vector $r_j$ is associated with exactly one materialized random vector $r_i$ that we refer to as its "representative". The representative random vector $r_i$ is used in order to compute the probability that the $j$th bit that corresponds to $h_{r_j}(x)$ (which is not available in the $lsh()$ encoding) would be set. Since our main focus is to retain the benefits of the RHP scheme and in particular (i) its simplicity in computing the angle between the data objects and (ii) the small-space requirement of the decoding process, we will utilize two simple statistics for each derived random vector in the decoding step. Let $P_{j|i} = P[h_{r_j}(x) = 1 | h_{r_i}(x) = 1]$ denote the probability that the hash function $h_{r_j}(x)$ evaluates to one when $h_{r_i}(x) = 1$ over all possible data items $x$ in our data set. Similarly, let $P_{j|\neg i} = P[h_{r_j}(x) = 1 | h_{r_i}(x) = 0]$ denote the probability that the hash function $h_{r_j}(x)$ evaluates to one when the hash value for the $i$th random vector is zero. During the decoding process, we utilize these pre-computed probabilities in order to estimate the value of the $j$th bit.

### 5.2 Overview of our framework

Functionality of our scheme consists of two phases, depicted in Figs. 6, 7, respectively: the initialization phase and the normal operation phase.

Suppose that our framework is deployed and works in a distributed environment, where remote sites (sensors, pda's, laptops) collect data and a central site performs similarity testing between data in this network (Fig. 1). During the first phase, all sites transmit $n + m$ bits for each data item that they obtain. These bits are computed by $n + m$ random vectors using the original $RHP(n + m)$ method. After a small period of time, this procedure stops and the central site has obtained a satisfactory amount of $RHP(n + m)$ encodings—a sample of the data. A greedy algorithm that runs at the central site captures the distribution of the data and the correlations between the random vectors, using these $RHP(n + m)$ encodings. The algorithm selects the $n$ more helpful random vectors—we call them materialized—that must be used by the remote sites to compute the bits that will be transmitted during the normal operation phase. For each of the remaining $m$ random vectors—we call them derived—the algorithm selects one materialized random vector as its representative. More precisely, representative of a derived random vector is assigned the materialized random vector that it is more correlated with. Furthermore, the central site computes the conditional probabilities that are necessary for the estimation of the bit values of the derived random vectors according to the bit value of their representatives. Finally, the initialization

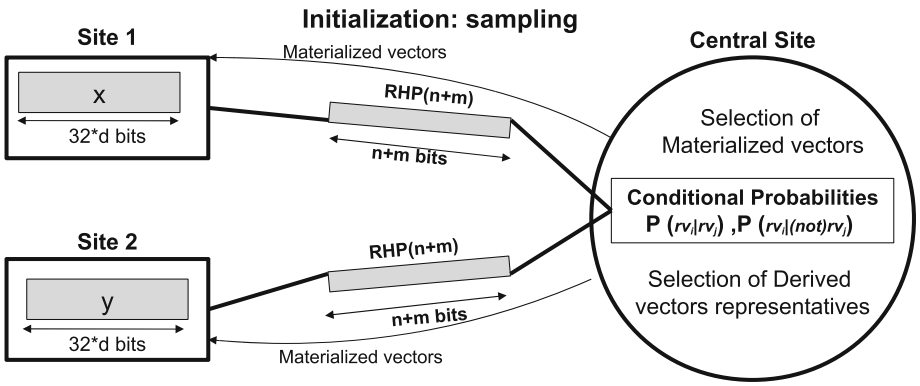



**Fig. 6** Initialization phase of the $RHP(n, m)$ scheme: sampling the data distribution using $RHP(n + m)$ encodings

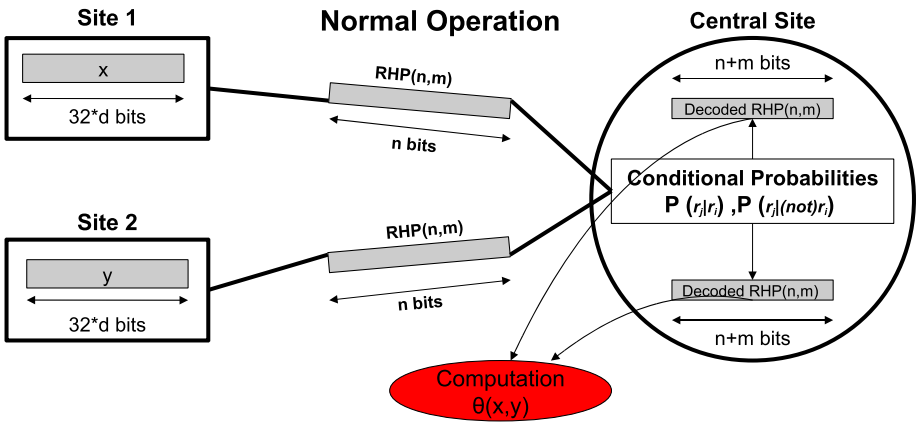



**Fig. 7** Normal Operation phase of the $RHP(n, m)$ scheme: decoding $RHP(n, m)$ bitmaps and estimating the similarity of data items $x$ and $y$

phase terminates with a broadcast message from the central site to the remote nodes that declares which random vectors must get materialized. All is required is a bitmap of length $n + m$, where the value of the $i$th bit denotes the status of the corresponding random vector $r_i$, i.e., if set the random vector is considered materialized, otherwise it is considered derived.

What follows is the normal operation of our framework, in which each remote node computes the $n$ bits that the materialized vectors contribute, creating the $RHP(n, m)$ encoding of each data item, and transmitting it to the central site. When the central sites receives the $RHP(n, m)$ encodings from the remote nodes, it performs the decoding of the bitmaps and estimates the missing bits of the derived random vectors using the conditional probabilities, creating this way a representation of $n+m$ values. The similarity between two data items is computed on this extended representation using an extension of Eq. 4 that we will derive in the following paragraphs.

### 5.3 Sampling the data distribution

In order to compute the required statistics that our method needs, we employ a sampling process in order to obtain a random sample $S$ of the data set. A key point in our work is that we do not sample from the original dataspace but rather from the RHP encodings of the data. In particular, using the common seed value we generate the $RHP(n + m)$ encodings of $S$. In case of data stored in remote sites, this means that only the $lsh(x)$ bitmaps of length $n+m$ each are transmitted toward a central location. Let $|S|$ denote the cardinality of the sampled data, then this process requires transmitting $|S|(n + m)$ bits. The sampled RHP encodings are stored in a two-dimensional $|S| \times (n + m)$ array $sLSH$. The rows of the array correspond to different $lsh(x)$ representations, while its columns to the random vectors used. Thus, $sLSH[i, j]$ denotes the $j$th bit of the lsh bitmap for the $i$th data item.

**Table 2** $sLSH$ array containing sampled $lsh(x_i)$ bitmaps

| | $r_1$ | $r_2$ | $r_3$ | $r_4$ |
|---|---|---|---|---|
| $x_1$ | 1 | 0 | 1 | 1 |
| $x_2$ | 0 | 1 | 1 | 0 |
| $x_3$ | 1 | 0 | 1 | 0 |
| $x_4$ | 0 | 1 | 0 | 0 |
| $x_5$ | 1 | 0 | 1 | 0 |

Table 2 shows the contents of the $sLSH$ array for some sample data. In this particular example, RHP encodings of five data items $(x_1 \ldots x_5)$ are used as a sample of the data and four random vectors $(r_1 \ldots r_4)$ contribute to the construction of these encodings. First line of the array (1011) represents the $RHP(4)$ encoding of data item $x_1$ computed by the use of

random vectors $r_1 \ldots r_4$, while first column of table (10101) represents the bit values that random vector $r_1$ contributes to the encodings of data items $x_1 \ldots x_5$. Taking a closer look at Table 2, all the bit values of column 2, which corresponds to the hash values produced by the second random vector, are opposite to the bit values of column 1, which corresponds to the hash values produced by random vector $r_1$. This observation denotes the strong negative correlation between the behavior of random vectors $r_1$ and $r_2$. Many other correlations in the behavior of the random vectors may be captured in the $sLSH$ array and will be exploited by our techniques discussed next.

An important aspect of the described sampling process is that it does not require the transmission of the original data items but rather their $RHP(n+m)$ encodings. As a consequence, in an application like the one discussed in Sect. 2, we use the basic $RHP(n+m)$ framework for the first few similarity tests, exploiting the gathered bitmaps in order to construct the sample. This results in an overhead of $m$ bits per item compared with $RHP(n)$, while the sample is constructed.

The sampling process avoids the transmission of the original data items and thus, retains the benefits of the RHP scheme, while the sample is constructed. As will be explained in the following sections, the obtained sample is used in order to detect possible correlations among the *RHP* dimensions. For this process, our techniques look at the produced *RHP* bitmaps in order to detect possible correlations, such as those denoted in Fig. 5. Similarly, when many random vectors do not contribute to the computation of the angle between data items (because of data skew), as in Fig. 4, from the obtained sample our techniques will be able to select a much shorter list of "useful" random vectors. In the example of Fig. 4, a single random vector not in area $O_1$ nor in $O_2$ suffices in order to estimate the hash values of all random vectors in these areas. All these correlations are observed and manipulated by our method in the *RHP* space. Thus, there is no loss of information for the techniques described next because of our sampling of the *RHP* vectors instead of the original data.

### 5.4 Estimating the utility of random vectors

We now describe how we utilize the sampled bitmaps in order to select $n$ random vectors to materialize, from the available set of $n+m$ random vectors we used to generate the sample. Intuitively, a random vector is a strong candidate for materialization if

- It is useful for distinguishing between many data items.
- It can be used as a representative in order to accurately predict the behavior of derived random vectors $r_j$ based on the conditional probabilities $P_{j|i}$ and $P_{j|\neg i}$.

Recall (Eq. 4) that a random vector $r_i$ contributes to computing the angle between data items $x$ and $y$, when $h_{r_i}(x) \neq h_{r_i}(y)$. Based on this observation, we compute the *utility* of random vector $r_i$ as

$$utility_i = \sum_{0 \leq x < |S|-1, x+1 \leq y < |S|} |sLSH[x,i] - sLSH[y,i]| \tag{6}$$

Thus, the utility of $r_i$ measures the number of occasions random vector $r_i$ contributes bits that differ over all possible pairs $x$, $y$ in the sample.

For the aforementioned example of the $sLSH$ array (Table 2), the utility of the first random vector $r_1$ is $utility_1 = 2+2+1+1 = 6$. Similarly, we can compute $utility_2 = 2+2+1+1 = 6$, $utility_3 = 1+1+1+1 = 4$, and $utility_4 = 4+0+0+0 = 4$. Sample's size is 5 and so the number of all possible pairs of data items in the sample is $\binom{5}{2} = 10$. As an example, in column 2 of Table 2, which corresponds to random vector $r_2$, first bit forms four pairs with the four next bits, and two of these pairs differ in their values. Second bit of the same column forms three pairs with the next bit values, and there are two pairs that have different values. Third bit value forms two pairs with fourth and fifth bit and only the first of these two pairs differ in value. Finally, fourth and fifth bit of column 2 create a pair of different values, making the sum of the pairs with different bit values 6, which is the utility score of random vectors $r_2$.

In addition to choosing random vectors with high utility scores, we also want to materialize random vectors that can be used to predict the behavior of non-materialized random vectors (Eq. 10). Given two random vectors $r_i$ and $r_j$ the columns $i$ and $j$ of array $sLSH$ depict the behavior of these random vectors (i.e., the values of their respective hash functions) for the sampled data. Let $sLSH[.,i]$ denote the $i$-th column of $sLSH$. Each such column is a bitmap of length $|S|$. A standard way to assess the correlation between the values of these bitmaps is to compute their correlation coefficient $corr_{i,j}$:

$$corr_{i,j} = \frac{cov(sLSH[.,i], sLSH[.,j])}{\sigma_{sLSH[.,i]} \sigma_{sLSH[.,i]}} \tag{7}$$

where $cov()$, $\sigma$ are the covariance and standard deviation functions, respectively. A strong (positive or negative) correlation between columns $sLSH[.,i]$ and $sLSH[.,j]$ indicates that random vector $r_i$ is a good candidate for representing $r_j$ and vise-versa. Thus, we want $corr_{i,j}$ be near +1 or $-1$. On the contrary, when $|corr_{i,j}|$ is close to zero, then there is no evident connection (in the sample) in the behavior of the two random vectors.

**Algorithm 1** GREEDY ALGORITHM.

**Require:** $(n, m, \{r_i | i = 1..(n+m)]\}, T)$
1: {Initially, all $r_i$s are candidates for materialization}
2: *Cand*=$\{r_i | i = [1..(n+m)]\}$
3: *Mat*=$\emptyset$ {Materialized random vectors}
4: *Der*=$\emptyset$ {Derived random vectors}
5: **while** ($|Mat| < n$) AND ($Cand \neq \emptyset$) **do**
6: {Select $r_i$ with highest utility score}
7: $k$=$argmax_{i \in Cand}(utility_i)$
8: $Mat = Mat \cup \{r_k\}$ {Mark $r_k$ as materialized}
9: *Cand*=*Cand*-$\{r_k\}$ {Remove from candidate list}
10: {Remove strongly correlated (to $r_k$) random vectors}
11: **for** $r_i \in Cand$ **do**
12: **if** $|corr_{i,k}| \geq T$ **then**
13: *Cand*=*Cand*-$\{r_i\}$
14: **end if**
15: **end for**
16: **end while**
17: {List remaining $r_i$s}
18: *Der*=$\{r_i | i = [1..(n+m)]\}$ - *Mat*
19: {Assure n materialized vectors have been selected}
20: **while** ($|Mat| < n$) **do**
21: $k$=$argmax_{i \in Der}(utility_i)$
22: $Mat = Mat \cup \{r_k\}$ {Mark $r_k$ as materialized}
23: *Der*=*Der*-$\{r_k\}$
24: **end while**
25: {Remaining $r_i s$ are marked as derived}
26: **for** $r_j \in Der$ **do**
27: $i$=$argmax_{k \in Mat}(|corr_{k,j}|)$
28: $Representative_j = i$ {Mark as representative}
29: **end for**

### 5.5 Choosing among random vectors

Based on these observations, we propose a simple greedy algorithm for selecting $n$ random vectors to be materialized (out of the $n + m$ available choices). The algorithm is presented in Algorithm 1 and proceeds as follows. Initially, all $n + m$ random vectors are candidates for materialization (set *Cand*, Line 2). At each step, the algorithm selects from set $Cand$, the random vector $r_k$ with the highest utility score and places it in the materialized set *Mat* (Lines 6–8). When random vector $r_k$ is selected for materialization, we remove from set *Cand* all random vectors $r_i$ that have a strong correlation (based on parameter $T$) with $r_k$ (Lines 10–16). The intuition is that the hash values for these $r_j$s can be easily estimated using conditional probabilities $P_{j|i}$ and $P_{j|\neg i}$. A typical value of $T$ is 0.95 in our implementation. In Lines 17–24, we check whether $n$ materialized vectors have been selected in set *Mat*. If more vectors are needed, they are chosen from the remaining vectors based on their utility score. At a final phase (Lines 25–29), the algorithm selects the representative of each derived random vector (the remaining $m$ random vectors not in set *Mat* at the end of the selection process) based on the absolute values of the correlation coefficients $corr_{k,j}$ between a materialized random vector $r_k$ and a derived random vector $r_j$. The running time of the algorithm is $O(n \times (n + m))$. We note that in the applications of interest, $n, m$ take small values, typically, less than 100.

After the selection of the representative random vectors, it is straightforward to compute probabilities $P_{j|i}$ and $P_{j|\neg i}$ from the corresponding columns $sLSH[., i]$ and $sLSH[., j]$. Then, array $sLSH$ can be discarded. In the end of this process, we have:

- A set of $n$ random vectors *Mat* denoted as materialized.
- A set of $m$ random vectors *Der* denoted as derived.
- The representative of each derived random vector ($m$ values in total).
- The conditional probabilities ($2 \times m$ values in total) required for estimating the angles between data items, as will be explained.

Using this information (of size $O(n + m)$), any site is able to compute the similarity of two data items from their $RHP(n, m)$ bitmaps.

As an example, assume $n = 2$, $m = 3$ and that the first and fifth random vectors have been selected as materialized. In order to convey the selection and location of the materialized vectors, a bitmap of the form "10001" is transmitted to the remote sites. The bitmap has length equal to $n + m$ and the positions of 1s denote the selection of materialized vectors. This bitmap is transmitted once to the remote sites after the initialization phase.

One may be tempted to discard from the final set of derived random vectors, those vectors that exhibit a low utility score, since they are not expected to contribute much to the computation of similarity. We note that the $RHP(n, m)$ scheme (similar to $RHP(n)$) requires a random collection of random vectors in order for the decoding process (described next) to produce correct, in expectation, estimates for $\theta(x, y)$. The removal of derived random vectors with low utility score will result in a biased estimate, which would often overestimate the true angle. In Sect. 5.8, we elaborate more on this issue.

### 5.6 Similarity estimation in our framework

Our decoding process computes an intermediate extended representation of $n$+$m$ values described as a vector $expLsh(x)$. The coordinates of this vector are computed as follows:

- If the $j$th random vector is materialized, the value used in the $j$th coordinate of $expLsh(x)$ is the $j$th bit that has been computed in $lsh(x)$.
- If the $j$th random vector is derived, then the value of the $j$th coordinate in $expLsh(x)$ is computed from the representative $r_i$ of $r_j$ as $P_{j|i}$, if the $i$th bit in $lsh(x)$ is set or as $P_{j|\neg i}$ otherwise.

**Table 3** Representatives and conditional probabilities for derived random vectors

| Derived $r_j$ | Representative of $r_j$ | $P_{j\|i}$ | $P_{j\|\neg i}$ |
|---|---|---|---|
| $r_2$ | $r_1$ | 0 | 1 |
| $r_3$ | $r_1$ | 1 | 0.50 |
| $r_4$ | $r_1$ | 0.33 | 0 |

Intuitively, the $expLsh(x)$ encoding is an approximation of the $lsh(x)$ bitmap that would have been obtained if all random vectors were materialized (as in $RHP(n+m)$). Since, the values of $h_{r_j}(x)$ are not available for derived random vectors, we use the conditional probabilities $P_{j|i}$ and $P_{j|\neg i}$ and the available bits from their representatives.

The angle between two input data items $x$ and $y$ is computed by manipulating their $expLsh()$ representations. Let $expLsh_i(x)$ denote the $i$th coordinate of vector $expLsh(x)$. Based on its construction, $expLsh_i(x)$ denotes the probability that $h_{r_i}(x)$=1. Thus, the probability that the $i$th hash function $h_{r_i}()$ produces different results for inputs $x$ and $y$ based on their available $lsh(x)$ and $lsh(y)$ encodings is

$$diff_i(lsh(x), lsh(y)) = expLsh_i(x) * (1 - expLsh_i(y)) + (1 - expLsh_i(x)) * expLsh_i(y) \quad (8)$$

where $diff_i(lsh(x), lsh(y)) \in [0, 1]$. Then, the expected hamming distance of the $n$+$m$ hash values for $x$ and $y$ can be estimated as

$$expD_h(lsh(x), lsh(y)) = \sum_{i \in [0, n+m)} diff_i(lsh(x), lsh(y)) \quad (9)$$

Please observe that $expD_h(lsh(x), lsh(y)) = D_h(lsh(x), lsh(y))$ for $m = 0$. Based on Eq. 4, the angle between vectors $x$ and $y$ is computed as

$$\theta(x, y) = \frac{expD_h(lsh(x), lsh(y))}{n+m}\pi \quad (10)$$

The described decoding process requires $O(n+m)$ space and time. In terms of the communication cost, $RHP(n, m)$ produces bitmaps of length $n$, as in $RHP(n)$.

Back to the example of Table 2, recall that there are 4 random vectors depicted with utility scores $utility_1 = utility_2 = 6$ and $utility_3 = utility_4 = 4$. At the first step, the algorithm selects one of $r_1, r_2$ that have the highest utility score among the 4 vectors. Assume that random vector $r_1$ is selected to be materialized. The correlation coefficients between this vector and the remaining three have values $corr_{1,2} = -1$, $corr_{1,3} = 0.61$, and $corr_{1,4} = 0.41$. The final selection of the algorithm will contain one materialized random vector ($r_1$) and three derived random vectors ($r_2, r_3, r_4$) that will use $r_1$ as their representative with the conditional probabilities depicted in Table 3.

Assume that we want to use this scheme in order to compute the angle between two vectors $x$ and $y$ with $lsh(x) = 1$ and $lsh(y) = 0$. Their intermediate $RHP(1, 3)$ encodings, based on Table 3 would be $expLsh(x) = (1, 0, 1, 0.33)$ and $expLsh(y) = (0, 1, 0.5, 0)$. Based on Eq. 10, the angle between the two vectors is computed as $\theta(x, y) = \frac{1+1+0.5+0.33}{4}\pi = 0.7075\pi$.

### 5.7 Parameter tuning

Our techniques require setting of two important parameters: the number of materialized vectors $n$, and the number of derived vectors $m$. In the distributed setting of Fig. 7 that we consider, the value of $n$ is set based on the number of bits we are willing (or can afford) to communicate for each data item (i.e., the desired reduction ratio). We expect this to be an application-dependent selection. For example, in the sensor network application, we have described in Sect. 2, there is a direct connection between the energy consumption, the network lifetime and $n$ [18]. Of course, a larger value of $n$ results in better estimates at the cost of a larger bandwidth consumption and energy drain.

The selection of a proper value for the number of derived vectors $m$ is more involved. In our discussion, so far, we have assumed that the number of materialized vectors $m$ is given as an input to our algorithms. One obvious question is how does one determine the proper value of $m$ for a particular application? An additional challenge that we face is that the data are not centrally available. Of course, one could centrally collect a random data sample from all sites in order to evaluate different values of $m$ and select the best one $m^*$. However, this would negate the key advantage of our techniques: which we never transmit the original data items to the central site.

In order to retain the advantages of our framework regarding the number of bits transmitted, we have devised an automated distributed process for tuning the value of $m$. This process involves the following steps and happens at the beginning of the initialization phase described earlier:

- The central site transmits a common seed, so that all remote sites can produce locally the same family of random vectors $r_i$.
- Each local site $k$ evaluates the performance of $RHP(n, m)$ on a random sample of its data, starting with $m = 0$ and increasing $m$ by one at each iteration. This process stops when the estimation error of $RHP(n, m_k + 1)$ is more that the error of the previous step ($RHP(n, m_k)$). The site transmits back to the central site the value of $m_k$.
- The central site collects the selections $m_k$ of each remote site and selects $m^* = median(m_k)$.

The extra communication overhead this process adds to the initialization phase is very small as each remote site needs to transmit a single value $m_k$. The intuition behind this process is that initially, increasing the value of $m$ from $0, 1, 2, \ldots$ results in better estimates as more dimensions are introduced in the *RHP* mapping. However, since the derived hash values are not computed but they are rather estimated from the statistics we learn, increasing the value of $m$ results in more "noise" in the estimates, as the conditional probabilities $P_{j|i}$ and $P_{j|\neg i}$ will not be able to accurately depict the behavior of the data when mapped to the Hamming cube, if the dimensionality of the latter (which is equal to $n + m_k$) increases. Thus, after a point, we expect performance to get worse with increasing $m$. In our experimental evaluation, we empirically verify this intuition. We note that since the same seed is used for generating the random vectors at the remote sites, there is a common enumeration of the $r_i$ vectors, and thus the values of $m_k$ refer to the same underlying set of possible $r_i$s.

The selection of the median among all values $m_k$ reported by the remote sites works as a safe estimate for a proper value of $m$. Clearly, the proposed method does not provide any type of formal guarantees for $m^*$. However, it has been shown to work well on different data sets we tried, including the real sensory dataset we present in our experimental evaluation.

### 5.8 Why derived vectors are important

Our framework makes a distinction between materialized and derived vectors. The hash values for the selected family of materialized vectors are computed and transmitted to the central site at a per-item basis, while the values for the remaining derived vectors are estimated using the conditional probabilities. This distinction does not mean that derived vectors are of less value. In fact, their contribution to the correct estimation of the similarity between data items is equally important.

For example, for the skewed dataset of Fig. 4, one may be tempted to devise families of random vectors that are tailored to the apparent data skew. In this example, this can be easily achieved by placing $n$ random vectors $r_i$ within slices $O_1$ and $O_2$ and let the remaining $m$ vectors be classified as derived. Obviously, for any pair of data from the highlighted area $D$, the bit values for the derived vectors are the same. In our framework, the computation of similarity based on Eq. 10 can be simplified as

$$\theta(x, y) = \frac{D_h(lsh(x), lsh(y))}{n + m}\pi$$

where $D_h()$ computes the hamming distance on the bitmaps produced by the materialized vectors only.

Let us assume that one decides to consider only materialized vectors for computing the similarity between data. Then, according to the standard *RHP* scheme, the estimation will be

$$\theta(x, y) = \frac{D_h(lsh(x), lsh(y))}{n}\pi$$

Thus, an $RHP(n_{mat} = n)$ scheme that only considers the set of materialized vectors will result in significant overestimation for the computed angle values. This overestimation will be proportional to the number of derived vectors this scheme ignores.

While it is possible in this contrived two-dimensional example, where data are restricted in the highlighted area, to adjust the formula in order to produce correct on expectation results, there is no obvious way to devise a solution in higher dimensions. Notice that in such cases, each pair of $d$-dimensional vectors $x$, $y$ defines a different plane, and therefore, hyper-slices $O_1$, $O_2$ are defined differently for each input pair $x$, $y$ in consideration.

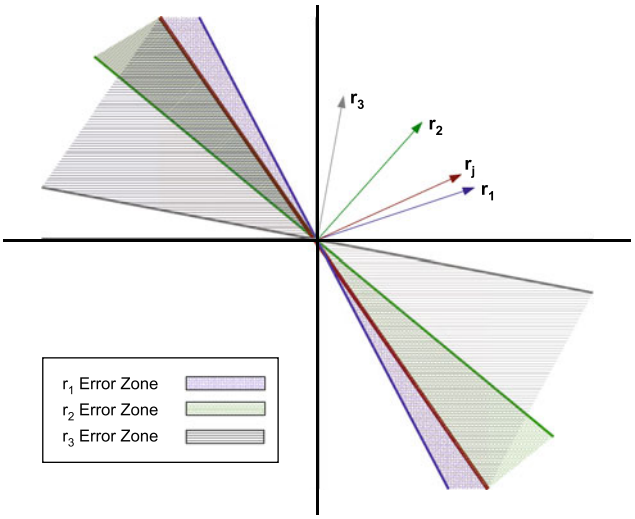



**Fig. 8** Error zones for candidate representatives of $r_j$

### 5.9 Selection of representatives revisited

In our work, we have considered selecting exactly one representative materialized vector for each derived random vector $r_j$. An open question is whether conditioning the hash values of $r_j$ on multiple materialized vectors would lead to better performance? We will try to address this question based on a geometric interpretation of the *RHP* hashing scheme.

Figure 8 presents a 2-dimensional example of four random vectors $r_1$, $r_2$, $r_3$, and $r_j$. Let us assume that $r_j$ is chosen as the derived random vector. For candidate representative vector $r_i$ $(i = 1, 2, 3)$, we denote in the figure the data space in which data points return different bit values than $r_j$ $(h_{r_i}(x) \neq h_{r_j}(x))$ as the error zone of $r_i$. Clearly, random vector $r_1$ is the best selection in this example as it generates the smallest error zone: in this 2-dimensional example, the size of the error

zone of $r_i$ is proportional to the angle between $r_i$ and $r_j$. A yet more important observation is that the remaining random vectors ($r_2$ and $r_3$) are not worth considering (in addition to $r_1$). Including them in the computation of the hash value of $r_j$ does not reduce the resulting error zone in the data space. Equivalently, one can observe that there is not a single data point in this data space where the prediction from $r_2$ or $r_3$ is more accurate than the prediction provided by the single representative $r_1$ for $r_j$. Thus, a single representative suffices in two-dimensions.

From this discussion, one may be tempted to select the closest random vector $r_i$ as the representative of $r_j$. While this works in two dimensions, independently of the data distribution, in higher dimensions the best selection is data-driven. More over, for a data item $x$, the best selection depends not only from $x$ but also on the data points $y$ for which we would like to compute the pair-wise similarity. More precisely, for each data pair $x$,$y$ one should project the random vectors $r_i$ onto the two-dimensional plane defined by $x$ and $y$ and consider the bit values returned. Then, the best selection comes from the materialized random vector $r_i$ which, for the particular dataset, has the most similar behavior to $r_j$ when considering all data pairs. This is in fact what our techniques try to achieve. Since not all data is available, we utilize a small sample of the data and, for each derived random vector $r_j$, we select as its representative the materialized vectors whose hash values are mostly correlated to those of $r_j$.

While this discussion explains the intuition on our decision to utilize a single representative for each derived random vector, in the experimental evaluation, we also provide experiments when multiple representatives are considered in the $RHP(n, m)$ scheme. As will be shown these more complicated models do not seem to provide sizable benefits in the computation of similarity. Moreover, a scheme that uses $k$ representatives $r_{i1},\ldots, r_{ik}$ for $r_j$ needs to compute and maintain conditional probabilities $P[h_{r_j}(x)|h_{r_{i1}}(x)h_{r_{i2}}(x)\ldots h_{r_{ik}}(x)]$ which requires $O(2^k)$ space, since the hash values are in $\{0, 1\}$. Thus, the space requirements increase exponentially with the number of representatives considered.

## 6 Extensions to the $RHP(n, m)$ scheme

### 6.1 Dynamic selection of representatives

In addition to the greedy algorithm that selects representatives for each derived vector, we here propose an alternative technique that selects the representatives dynamically, during the decoding of each $RHP(n, m)$ bitmap. In order to distinguish between the two schemes, we refer to the selection of representatives by the greedy algorithm as *static selection*, while we use the term *dynamic selection of representatives* for the method we describe next. As we will demonstrate in our experimental evaluation, the dynamic selection offers, in most cases, more accurate estimates, because of the larger space of correlations it considers during the decoding phase.

As it was mentioned, the new method affects the decoding phase of our scheme, improving the estimation of bit values of the derived vectors taking into account the values of the materialized vectors of each input vector's encoding. In our original method, the representative of each derived vector is selected once during the initialization phase. The conditional probabilities for estimating the values of each derived vector bit are also computed at the same time by the central site and are then used during the normal operation for similarity computations. The assigned representative for a derived vector can only be changed during the re-sampling phase, in case data distribution change were detected by central cite.[4] These properties of the original method denote the static way in which the technique behaves during computation and usage of the $RHP(n, m)$ encodings.

The new dynamic method that is presented in this section, extends the static scheme by assigning representatives to the derived dimensions at a per-item basis: the actual bit values of the materialized vectors in $lsh(x)$ are taken into consideration in order to select the representatives for the remaining derived dimensions. So, each time the central site obtains an $RHP(n, m)$ encoding (which consists of $n$ bits for the $n$ materialized vectors) of a data item, it further selects for each derived vector, which is the most suitable materialized vector for it, to be its representative. To make this selection, the central site must preserve for each one of the derived vectors, two conditional probabilities for every materialized vector that could be its representative. Thus, taking into consideration the bit value of each materialized vector and all these conditional probabilities, each derived vector's representative is decided based on which materialized bit value achieves the best estimation of its bit value.

During the sampling phase, the central site computes probabilities $P_{j|i}$ and $P_{j|\neg i}$ where $r_i$ is a materialized random vector and $r_j$ is a derived one. Thus, there are a total of $2 \times n \times m$ values computed from the sample. The static selection method only retains the conditional probabilities for $i = representative_j$, i.e. $2 \times m$ values in total. In contrast, the dynamic selection algorithms needs all $2 \times n \times m$ values for its operation.

Recall that $P_{j|i}$ (resp. $P_{j|\neg i}$) denotes the conditional probability that the bit of the $r_j$ derived random vector is set when the bit for materialized $r_i$ is (resp. is not) set. Assume that $P_{j|i}$ has a value near one. This suggest a strong positive correlation between the hash values produced by the two random vectors. Similarly, when $P_{j|i}$ has a value near zero and the bit for $r_i$ is set, then the bit for $r_j$ is, most of the times,

[4] This process is discussed in the next subsection.

**Table 4** $sLSH$ containing sampled $lsh_4(x_i)$

| | $r_1$ | $r_2$ | $r_3$ | $r_4$ |
|---|---|---|---|---|
| $x_1$ | 1 | 1 | 1 | 1 |
| $x_2$ | 0 | 0 | 1 | 0 |
| $x_3$ | 1 | 0 | 1 | 0 |
| $x_4$ | 0 | 1 | 0 | 0 |
| $x_5$ | 1 | 0 | 1 | 0 |

zero. Similar observations hold for $P_{j|\neg i}$: a value near one (resp. zero) results in strong negative (resp. positive) correlations, when the bit for $r_i$ is not set. On the contrary, when these conditional probabilities take values near 0.5 then, the value produced by the $r_i$ vector is not a strong indicator of the expected value of the bit corresponding to the $r_j$ random vector. Based on this observation, a criteria of inconsistency is introduced in order to measure the "goodness" of a candidate representative selection. More formally, for a data item $x$, we compute the inconsistency of the $(r_j,r_i)$ pair as

$$\text{inconsistency}_{j,i}(x) = \begin{cases} P_{j|i} * (1 - P_{j|i}), & \text{if } h_{r_i}(x) = 1 \\ P_{j|\neg i} * (1 - P_{j|\neg i}), & \text{if } h_{r_i}(x) = 0 \end{cases} \tag{11}$$

The lower the inconsistency score is, the better the derived vector's bit value can be estimated. We can easily see that this happens when the conditional probabilities take values near zero or one.

The proposed method for dynamically selecting the representatives exploits the notion of inconsistency and estimates the bit value of a derived vector using as its representative, the vector for which the inconsistency score is lowest. The estimation of the derived vector's bit value is made in the same way as in the static method, using the appropriate conditional probability referring to the specific materialized vector that was selected as representative.

It must be noted that the dynamic selection of representatives method is slightly more complicated than the static one. As already described, it needs space to store $2 \times n \times m$ conditional probabilities (at the central site) compared with the $2 \times m$ values that the static method uses. Furthermore, the new method's computational overhead for decoding a bitmap requires $O(n + n \times m)$ time compared with $O(n + m)$ time for decoding a bitmap using the static selection of representatives. Although the new method is more demanding in computational and storage resources, it is a viable alternative because these overheads refer to the central site, which typically does not lack of resources.

For example, suppose that the central site during the initialization phase obtains the $RHP(n + m)$ encodings shown in Table 4. In order to distinguish the encodings of the discussed RHP schemes, we will use the notation $lsh_k(x)$ to denote the bitmap obtained by $RHP(k)$. In this sample, there are five different encodings $(lsh(x_1), .., lsh(x_5))$, computed with the usage of four random vectors $(r_1, .., r_4)$. Two of these four random vectors will get materialized and the remaining will be derived $(n = 2, m = 2)$. The greedy algorithm, based on the computed utility scores $(utility_1 = 6, utility_2 = 6, utility_3 = 4, utility_4 = 4)$ will select $r_1$ and $r_2$ to be materialized. The static selection method, taking into consideration the absolute values of the correlation coefficient between materialized and derived vectors $(corr_{1,3} = 0.61, corr_{1,4} = 0.41, corr_{2,3} = 0.61, corr_{2,4} = 0.61)$, assigns $r_1$ as representative of $r_3$ and $r_2$ as representative of $r_4$. The conditional probabilities, retained by the static method for this selection of representatives, are depicted in the first two rows of Table 5. So, given a data item's $RHP(n, m)$ encoding, such as the one depicted in Table 6, the static method will estimate the two missing bit values of the derived vectors as is shown Table 7.

**Table 5** Conditional probabilities for derived random vectors. Static method needs values in the first two rows only

| Derived $r_j$ | Materialized $r_i$ | $P_{j\|i}$ | $P_{j\|\neg i}$ |
|---|---|---|---|
| $r_3$ | $r_1$ | 1 | 0.50 |
| $r_4$ | $r_2$ | 0.50 | 0 |
| $r_4$ | $r_1$ | 0.33 | 0 |
| $r_3$ | $r_2$ | 0.50 | 1 |

**Table 6** $lsh_{2,2}(x_{data})$ bitmap prior decoding

| $lsh_{2,2}(x_{data})$ | 0 | 0 |
|---|---|---|

**Table 7** $expLsh(x_{data})$ bitmap after static method decoding

| $expLsh(x_{data})$ | 0 | 0 | 0.50 | 0 |
|---|---|---|---|---|

If we focus on the relationship between $r_3$ and its static representative $r_1$, we can see that (in this sample) when $h_{r_1}(x) = 1$ then $h_{r_3}(x)$ is always one, thus $P_{3|1} = 1$. On the contrary, when $h_{r_1}(x) = 0$, we cannot be sure on the value of the bit for $r_3$ from this representative selection (notice that it is set half of the times, thus $P_{3|\neg 1} = 0.5$). To remedy this problem, in this particular example of Table 6, the dynamic selection method will estimate the bit value of $r_3$ using as representative the $r_2$ vector instead of $r_1$ (selected by the static method). This decision will be made because of the inconsistency scores related to the $r_3$ vector $(inconsistency_{3,2}(x_{data}) = 0 < inconsistency_{3,1}(x_{data}) = 0.25)$. Thus, the dynamic selection method will choose $r_2$ as representative of $r_1$, because the corresponding inconsistency score is lower. So, the bit value of random vector $r_3$ is estimated based on $r_2$ using $P_{3|\neg 2} = 1$. The resulting bitmap, after decoding, is shown in Table 8.

**Table 8** $expLsh(x_{data})$ bitmap after dynamic method decoding

| $expLsh(x_{data})$ | 0 | 0 | 1 | 0 |
|---|---|---|---|---|

### 6.2 Handling evolving data sets

The proposed $RHP(n, m)$ framework makes use of precomputed statistics in order to boost the accuracy of the standard random hyperplane projection scheme. A natural question that arises, is whether our techniques will be able to adapt to (transient or permanent) changes in the characteristics of the data sources. In this section, we introduce techniques that detect data items that cannot be accurately described using the available statistics. In such a case, our algorithms fall back into using the basic $RHP(n)$ scheme that is oblivious to the data characteristics.

In our original $RHP(n, m)$ framework, we compute and communicate $lsh_{n,m}(x)$ bitmaps (of length $n$). Given the common seed value, a local site can further compute the standard $RHP(n)$ encoding $lsh_n(x)$.

Recall that the $i$th bit of $lsh_n(x)$ is set if the dot product of $x$ with random vector $r_i$ is positive, or zero otherwise. Let

$$d_i(x) = \begin{cases} +1, & \text{if } h_{r_i}(x) = r_i \cdot x \geq 0 \\ -1, & \text{if } h_{r_i}(x) = r_i \cdot x < 0 \end{cases} \tag{12}$$

denote the sign of dot product. Obviously $d_i(x) = 1$ if the $i$th bit is set and $-1$ otherwise. It is easy to see that an approximation of the data item $x$ can be obtained by reversing the $RHP$ procedure, i.e., by adding random vectors $r_i$ when $d_i(x) = 1$ and subtracting them otherwise. More formally, we can use $lsh_n(x)$ to obtain an approximation $\hat{x}_n$ of $x$ via the following simple linear transformation

$$\hat{x}_n = \sum_{i=1}^{n} d_i(x) r_i \tag{13}$$

For the case of our $RHP(n, m)$ encoding, the process is more involved. Recall that from $lsh_{n,m}(x)$ (a bitmap of length $n$), we can obtain an intermediate extended representation $expLsh(x)$, i.e., a real vector with $n + m$ coordinates. The hash values of the derived dimensions in the $expLsh(x)$ representation are real numbers in the range $[0, 1]$ computed from the corresponding conditional probabilities. In our technique, we fist round the values of $expLsh(x)$ to its nearest integer in order to obtain a bitmap of length $n + m$ and then compute $\hat{x}_{n,m}$ using Eq. 13 (for $n + m$ vectors), i.e., we add random vector $r_i$ if the $i$th bit in the resulting bitmap is set, or subtract it otherwise.

To summarize, a local site using the common seed provided, may also computes $lsh_n(x)$, in addition to $lsh_{n,m}(x)$. Given that the actual data item $x$ is locally available, we can asses the accuracy of the two $RHP$ variants by comparing their approximations to $x$. In particular, let $\theta(x, \hat{x}_{n,m})$ denote the angle between $x$ and its approximation via our technique and $\theta(x, \hat{x}_n)$ the angle between $x$ and its $RHP(n)$ approximation. If

$$\theta(x, \hat{x}_{n,m}) \leq \theta(x, \hat{x}_n) \tag{14}$$

our technique better approximates the data item $x$, the encoding $lsh_{n,m}(x)$ is transmitted to the central site. Otherwise, we choose to transmit the standard $lsh_n(x)$ embedding, augmented with an additional bit that informs the site, which will decode the embedding that the standard $RHP(n)$ process has been used.

This process requires a small modification in the computation of similarity at the central site. Given two encodings $lsh_{n,m}(x)$, $lsh_{n,m}(y)$ (resp. $lsh_n(x)$, $lsh_n(y)$), we compute the angle of the original data items using Eq. 10 (resp. Eq. 4) as before. In order to compare $lsh_n(x)$, $lsh_{n,m}(y)$ (the symmetric case is similar), we first obtain $expLSH(y)$ as already described, and then we compare the bitmaps via Eq. 10 using only the hash values of the random vectors utilized in $RHP(n)$ and setting $m = 0$.

The advantages of this simple extension are twofold. First, it can easily detect transient readings that differ significantly from the sampled data used in learning the conditional probabilities $P_{j|i}$ and $P_{j|\neg i}$ and fall back into using the basic $RHP(n)$ scheme for them. Second, it can be used to trigger a new sampling process in order to compute a new set of materialized random vectors using Algorithm 1, which will be more helpful in cases that the data distribution has changed dramatically and the current materialized vectors do not contribute enough to the similarity estimation of the data.

We here propose a simple technique that will periodically adapt the computed probabilities to changes in the data distribution. More formally, the new sampling process will be triggered when, for a time window $W$, the number of $RHP(n)$ encodings transmitted to the central site (their number can be easily computed given the extra bit that differentiates them from the $RHP(n, m)$ encodings) is greater than $\alpha\%$ of the encodings the central cite has obtained. When this happens, it indicates that the data distribution has changed and the precomputed statistics, used for the selection of the materialized vectors and the representatives of the derived ones, do not hold anymore. A small value of $\alpha$ results in a more aggressive adaptation to the data characteristics. We note, that during the sampling process, similarity estimation is still possible, as the remote sites simply transmit $RHP(n+m)$ bitmaps in order to obtain the sample. A large value of parameter $\alpha$ will result in less frequent adaptations. However, because of the fall-back mechanism, we described, similarity estimation will still be accurate using the standard $RHP(n)$ method for items that do not pass the test of Eq. 14.

## 7 Experiments

### 7.1 Experimental set up and metrics used

Our experimental evaluation utilizes both real and synthetic datasets. The six real datasets we used are:

- *TEMPERATURE:* This data set contains temperature readings obtained by sensor nodes at the Intel Labs [13]. We used non-overlapping windows of 32 epochs to generate 912 32-dimensional records of sensory measurements.
- *HUMIDITY:* This data set contains humidity measurements obtained by sensor nodes at the Intel Labs [13]. We used non-overlapping windows of 32 epochs to generate 720 32-dimensional records of sensory measurements.
- *NBA:* This data set contains 21,384 17-dimensional records each representing a player's performance per year. Attributes include, minutes played, points, offensive/defensive rebounds, assists, etc.
- *HOUSE:* This data set consists of 100,000 6-dimensional records containing real-estate information from the United States (data available at zillow.com). Features include price, number of rooms, area etc.
- *COREL:* This data set contains 32-dimensional features of color histograms extracted from a Corel image collection.[5] The data set consists of 68,040 records.
- *COVER:* This is a popular real data set used in many articles on information indexing [26].[6] We used the 10 numerical attributes in our study. The dataset consists of 581,012 records.

The first two data sets were selected as representatives of a sensor network application like the one described in our motivational example. (In fact, this particular data sets were also used in [18].) The remaining data sets were selected in order to assess the proposed techniques in a diverse set of data. We also generated synthetic 64-dimensional data sets, in which we control data skew, as will be explained. These data sets are presented in Sect. 7.4.

We have performed several experiments and present, in most figures, the average $L_1$ error in computing the angle between different pairs of records $x, y$. The average $L_1$ error is defined as

$$AverageL_1Err = avg_{x,y}(|\theta(x, y) - \theta(lsh(x), lsh(y))|) \tag{15}$$

where $\theta(x, y)$ denotes the real angle between the data items and $\theta(lsh(x), lsh(y))$, the estimation we obtain from the RHP encodings of the respective method. Please notice that from the $L_1$ error it is trivial to obtain the estimation error in computing the cosine similarity. We also ran experiments using the correlation coefficient, as the similarity metric, however, they are omitted because they were qualitatively similar.

In some of the figures, we further provide experiments computing the average relative error

$$AverageRelErr = avg_{x,y}\frac{|\theta(x, y) - \theta(lsh(x), lsh(y))|}{\theta(x, y)} \tag{16}$$

as well as the sum-squared error

$$SSE = \sum_{x,y}(\theta(x, y) - \theta(lsh(x), lsh(y)))^2 \tag{17}$$

In all runs, we use a 10% sample as an input to Algorithm 1 (with the exception of the much larger HOUSE and COVER data sets for which a 1% sample was used) and for computing the conditional probabilities. We ran all experiments using an Apple MacBook Pro 2.26 GHz laptop with 2 GB of memory.

Our experiments proceed as follows. In Sect. 7.2, we explore the dependency of our technique to the number $m$ of derived vectors used. We also explore alternative techniques that only consider materialized vectors for deciphering the similarity of data from their bitmaps and demonstrate that they cannot provide accurate estimates. In Sect. 7.3, we compare our technique against $RHP(n)$ and also against two popular data reduction methods, using the six real data sets we described. Our results demonstrate that $RHP(n, m)$ is significantly more accurate than the alternative techniques considered, especially when fewer bits (large reduction ratio) are used to describe the data items. In Sect. 7.4, we evaluate our techniques while varying data skew, using synthetically generated data. In Sect. 7.5, we evaluate using more than one representative for each derived vector. We also test the proposed dynamic selection of representatives and demonstrate that it can help produce better estimates, at the cost of increased computational overhead. In Sect. 7.6, we explore the benefits of our techniques in a distributed setting. Finally, in Sect. 7.7, we evaluate the hybrid technique when the data, which we used to train our method, is injected with randomly created items. Our results show that the hybrid method manages to gracefully revert to the $RHP(n)$ scheme for those data items that are not properly captured by the computed conditional probabilities.

[5] Data is available from http://archive.ics.uci.edu/ml/databases/CorelFeatures.

[6] Data is available from http://archive.ics.uci.edu/ml/datasets/Covertype.

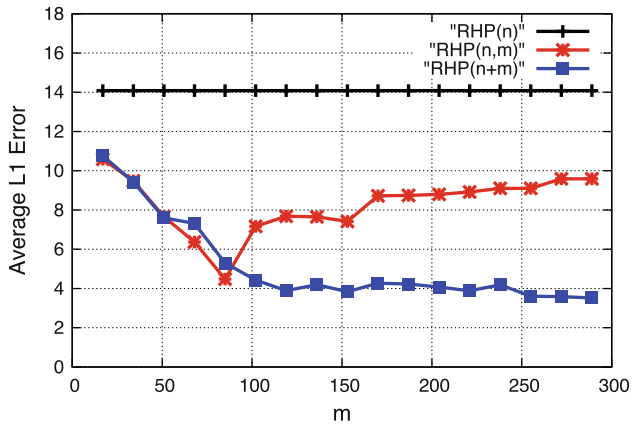



**Fig. 9** Sensitivity to $m$, NBA data set ($n = 34$). $RHP(n)$, $RHP(n, m)$ use 34 bits/data item. $RHP(n + m)$ uses $34 + m$ bits per data item

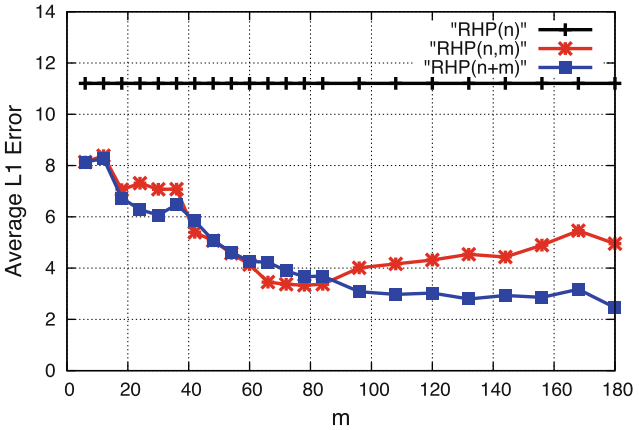



**Fig. 10** Sensitivity to $m$, HOUSE data set ($n = 12$). $RHP(n)$, $RHP(n, m)$ use 12 bits/data item. $RHP(n + m)$ uses $12 + m$ bits per data item

### 7.2 Selecting the number of derived vectors

In the experiments of Figs. 9, 10, we evaluate the performance of our technique when we vary $m$, the number of derived random vectors. For each data set, we used 4950 pairs of objects generated from a random sample of size 100 in order to compute the average errors depicted in the graphs. Parameter $n$ was set to 34 and 12 for the NBA and the HOUSE data set, respectively, so that in both experiments, we achieve a reduction ratio of 16:1 (Eq. 5). We can see that for all depicted values of $m$, the $RHP(n, m)$ framework outperforms $RHP(n)$ (which uses the same space per item) up to a factor of 3. More importantly, the following trend seems to appear. Initially, increasing the value of $m$ from $1, 2, \ldots$ results in better estimates. However, after we exceed a certain threshold, a further increase in $m$ results in worse performance. This happens because the additional derived random vectors we introduce are not strongly correlated to the materialized set of random vectors, and thus, their behavior cannot be accurately described by the conditional probabilities $P_{j|i}$ and $P_{j|\neg i}$. We observed the same trend for different values of $n$ and for the

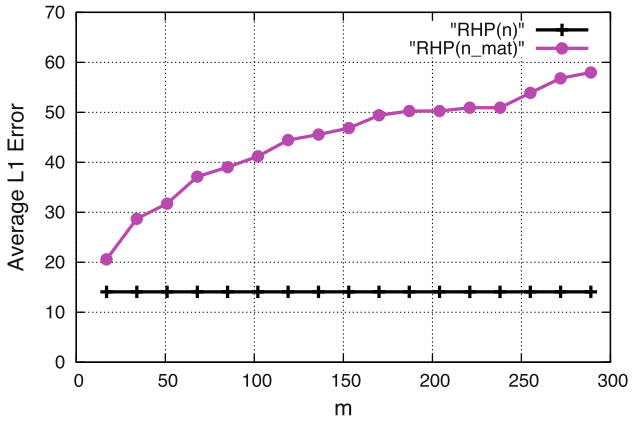



**Fig. 11** RHP with materialized vectors only, NBA Dataset

other data sets. The shape of the curves in these figures suggests that for an application-selected reduction ratio (equivalently a given value of $n$), we can obtain a good selection $m^*$ by performing a quick search over a small range of values for parameter $m$, using the sample. In all experiments reported in the rest of this section, we used this process in the range $m : (0, 8n)$ in order to select the best $m$ in each run.

In Figs. 9, 10, we also depict the performance of $RHP(n + m)$ that transmits $m$ additional bits compared with the other two methods. We can see that up to the value of $m^*$ our proposed method matches (and in several instances it exceeds) the performance of $RHP(n + m)$, indicating that the estimation of the hash values for the $m$ derived random vectors using the conditional probabilities is extremely accurate. We emphasize that for $m^* = 85$ in Fig. 9, our technique has practically the same performance as $RHP(n + m)$ even-though it uses only 34 bits per item compared with 34 + 85=119 bits for the latter. The same trend appears in Fig. 10, where our proposed method is using only 12 bits per data item, while $RHP(n + m)$ needs at least 12 + 84 = 96 bits to achieve the same level of accuracy.

We also ran experiments using only the set of $n$ materialized vectors that our algorithm selects (i.e., without using the remaining derived vectors in the estimation process). Performance for this alternative technique is depicted in Figs. 11, 12. However, as expected (Sect. 5.8), this technique returned average errors up to 5 times higher than $RHP(n)$ and is not included in the remaining graphs.

### 7.3 Comparison against alternative techniques when varying the reduction ratio

We now present a series of results where we compare our technique against $RHP(n)$ and also against two popular data reduction methods. The first, denoted as *Sketches* in the figures utilizes random projections (Sketches [1]) in order to embed the data points into a lower dimensionality space. Sketches are a popular tool utilized in various streaming

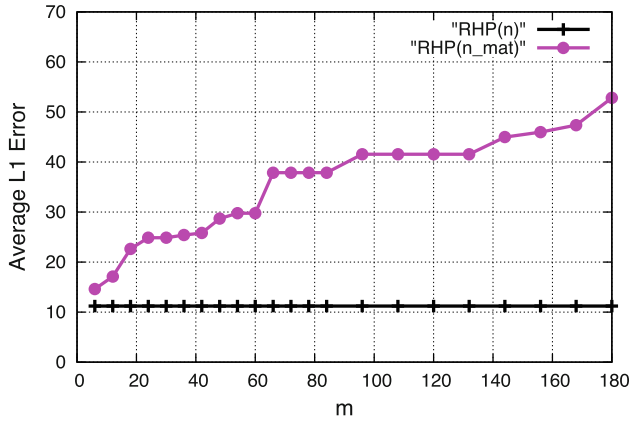



**Fig. 12** RHP with materialized vectors only, HOUSE Dataset

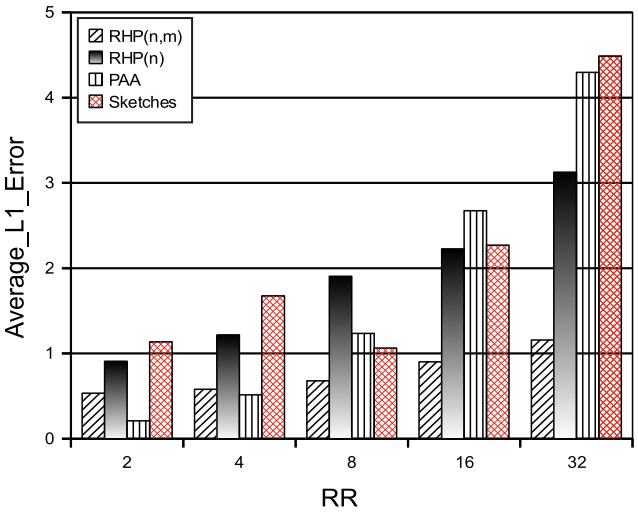



**Fig. 13** TEMPERATURE Dataset, RR≤32

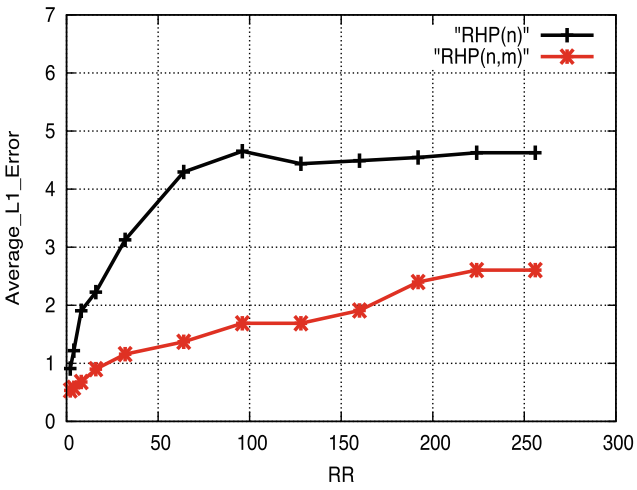



**Fig. 14** TEMPERATURE Dataset, all RRs

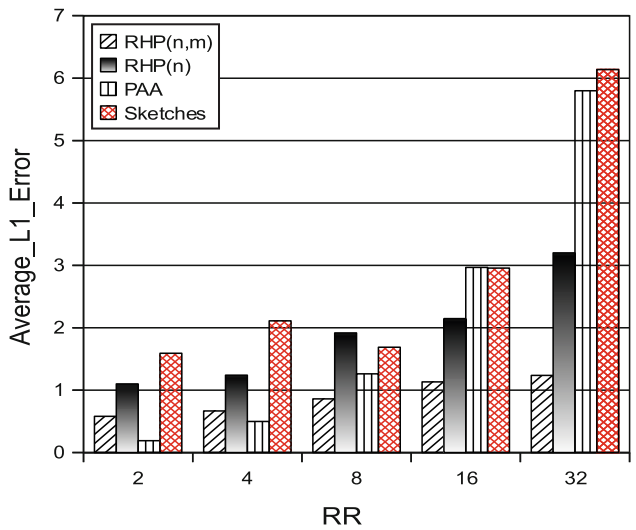



**Fig. 15** HUMIDITY Dataset, RR ≤ 32

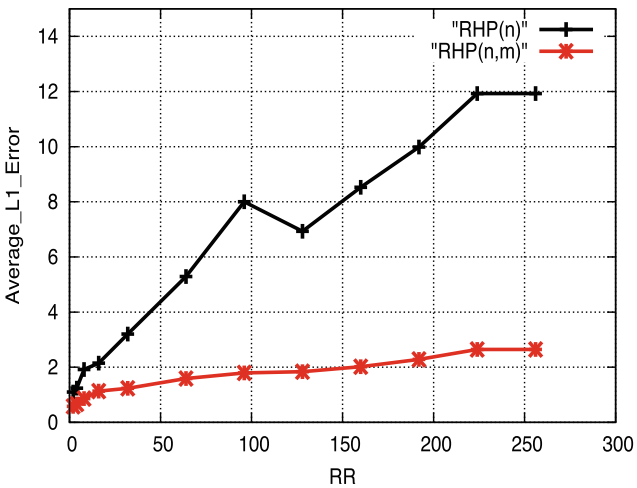



**Fig. 16** HUMIDITY Dataset, all RR values

algorithms [14,21] and have been used in computing the cosine similarity in network monitoring applications [19]. The second technique we consider is the popular Piecewise Aggregate Approximation (PAA) [32], a simple yet effective technique used in time series similarity estimation. As discussed in Sect. 3, a key difference of the *RHP* method and these techniques is that the former projects the data into the Hamming hyper-cube, thus, the coordinates of the points in the projected space are 0/1 bit values. In contrast, both Sketches and PAA (and most other dimensionality reduction methods) embed the $d$-dimensional data into $R^k$, for some value of $k << d$. Still, each of the $k$ coordinates in that space requires floating point arithmetic using (typically) 32 bits. The implication is that $RHP(n)$ and $RHP(n, m)$ are able to obtain much higher data reduction ratios than the alternative techniques.

In Figs. 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, we present graphs showing the accuracy of our $RHP(n, m)$ framework compared with $RHP(n)$ and the aforementioned

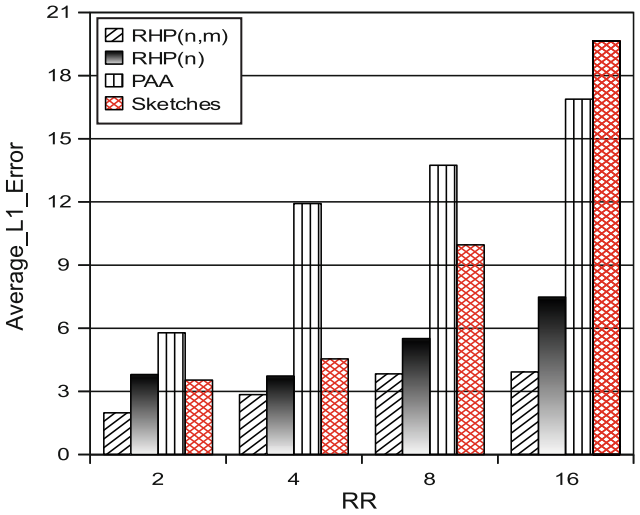



**Fig. 17** NBA Dataset, RR $\leq$ 16

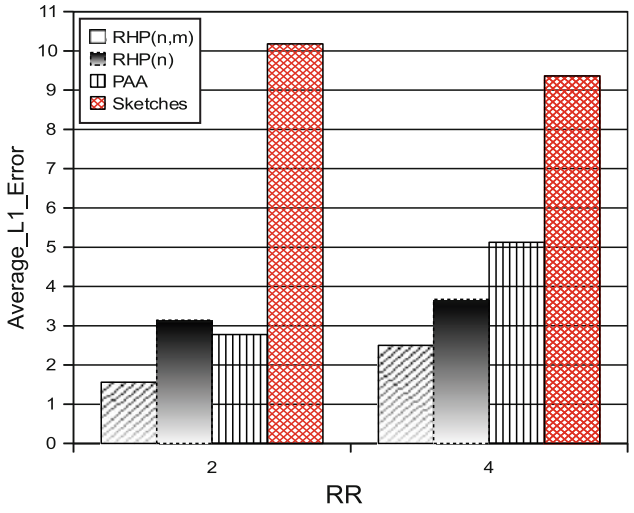



**Fig. 19** HOUSE Dataset, RR $\leq$ 4

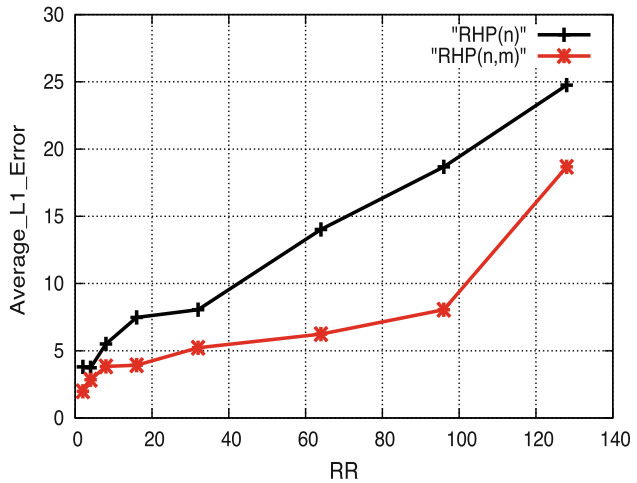



**Fig. 18** NBA Dataset, all RR values

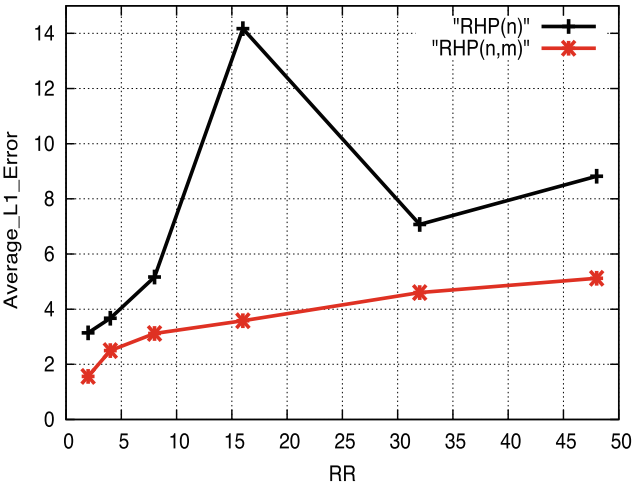



**Fig. 20** HOUSE Dataset, all RR values

techniques for different reduction ratios. The errors have been computed on a random set of 10,000 data pairs (i.e., 20,000 records) sampled from each data set. Since Sketches and PAA can only be used up to a certain reduction ratio value, for clarity, we split the results in two graphs per data set. The first graph compares all four techniques for small values of RR, while the second focuses on the performance of $RHP(n, m)$ and $RHP(n)$ for an extended range of RR values. In most cases reported, the *RHP* techniques manage to produce better similarity estimates (the $y$-axis in all graphs depicts the average $L_1$ error). Moreover, in all cases, our method provides significantly more accurate estimates than *RHP*, reducing the estimation error by up to 70%.

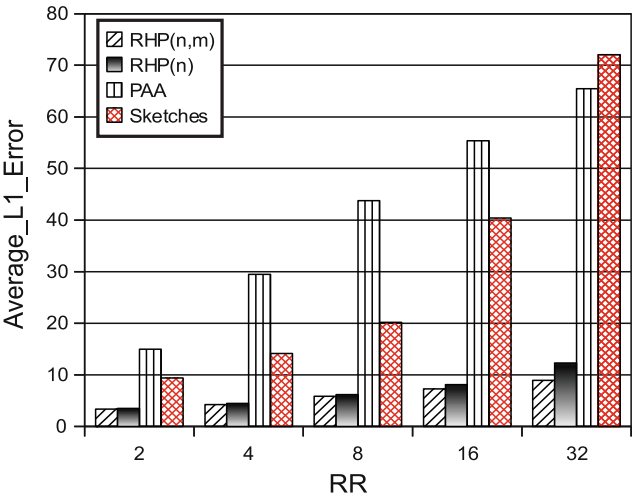



**Fig. 21** COREL Dataset, RR $\leq$ 32

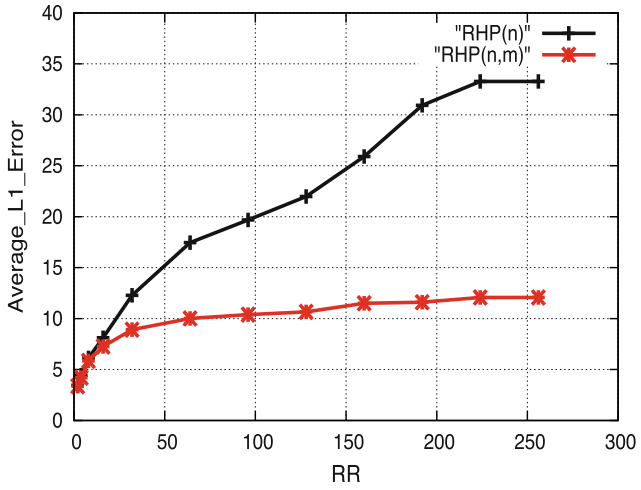



**Fig. 22** COREL Dataset, all RR values

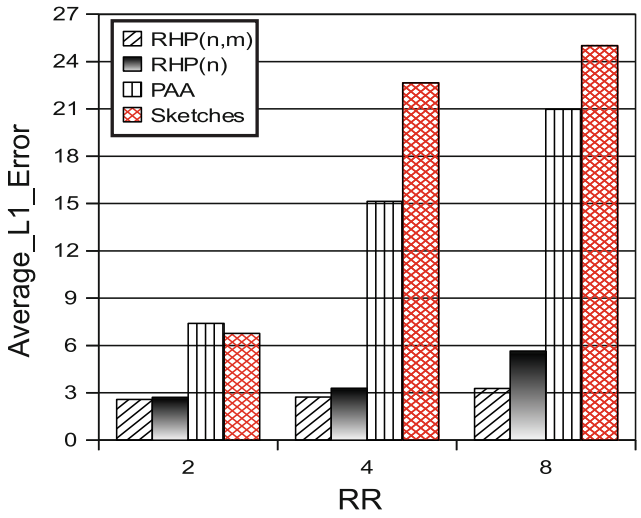



**Fig. 23** COVER Dataset, RR $\leq 8$

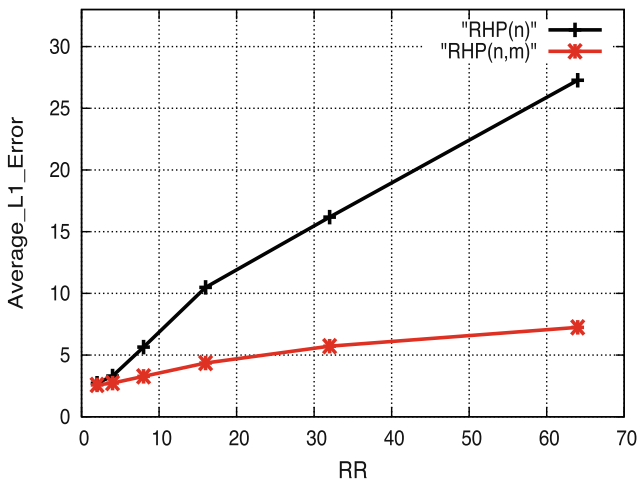



**Fig. 24** COVER Dataset, all RR values

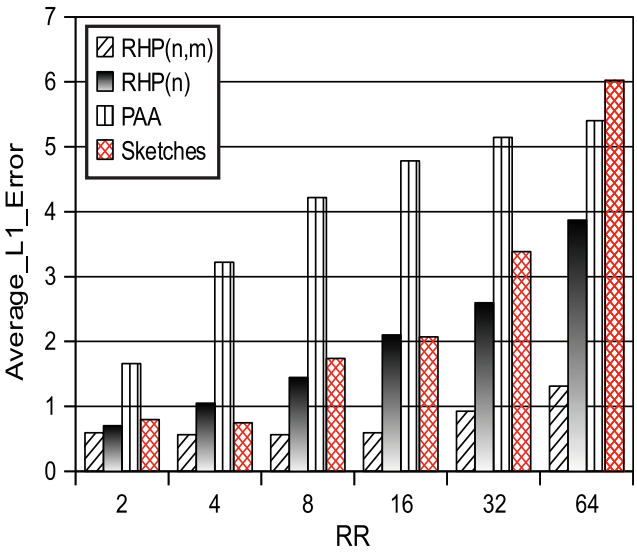



**Fig. 25** One cluster in Dataset

### 7.4 Experiments on synthetic data sets

Additionally to the experiments that were presented in the previous paragraph, we further measured the performance of our technique in synthetic data sets, where we are able to control the skew of the data. The goal of these experiments was to evaluate how our method and its competitors are effected by data skew. For this reason, we used four synthetic data sets each one with different skew. The first data set was created so as all its data items are organized in one cluster. The second data set uses two clusters, the third four clusters and the fourth data set eight clusters. The data clusters were generated with the following process:

- We created one (resp. 2,4,8) random 64-dimensional vector for the first (resp. the second,third and fourth) data set. We call these vectors primary vectors.
- Each data item was generated by randomly selecting a primary vector. We first mapped this vector to the hyperspherical coordinate system and then we rotated it in space by modifying each one of its angular coordinates by an angle smaller than a threshold of 5 degrees. The resulted perturbed vector was then mapped back to the Euclidean space.

In Figs. 25, 26, 27, 28, we depict the performance of the four techniques for the aforementioned data sets. For each data set, we used 10,000 pairs of data items (20,000 records) for the comparisons. Our technique was trained using 1,000 data items different from those used in the evaluation. Again, our method $RHP(n, n)$ outperforms the alternative techniques considered and its advantages are more evident for larger reduction ratios. In the more dispersed data

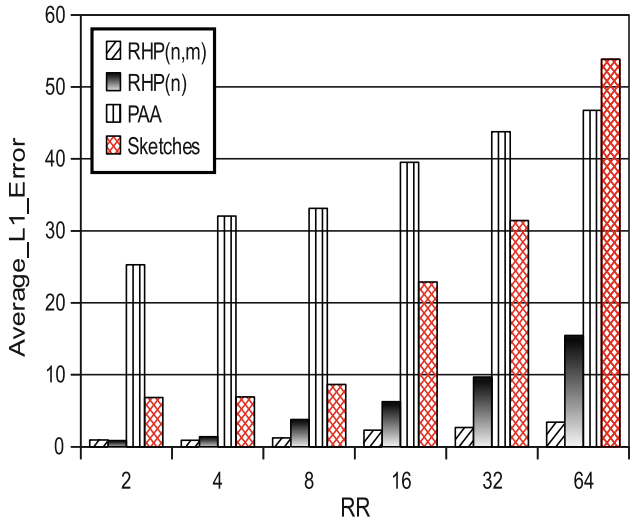



**Fig. 26** Two clusters in Dataset

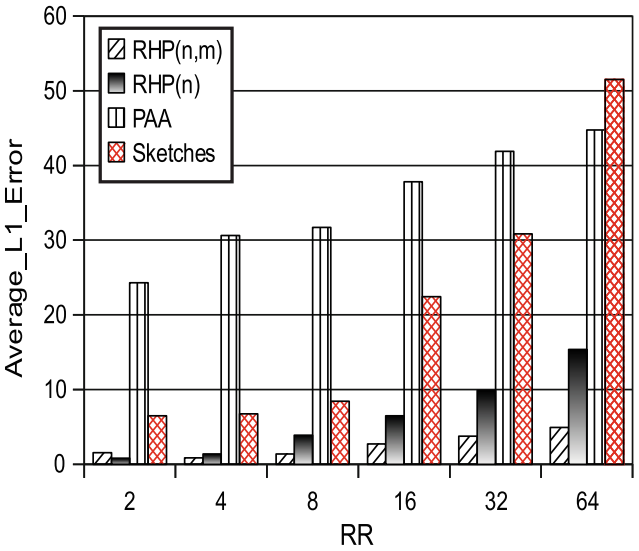



**Fig. 27** Four clusters in Dataset

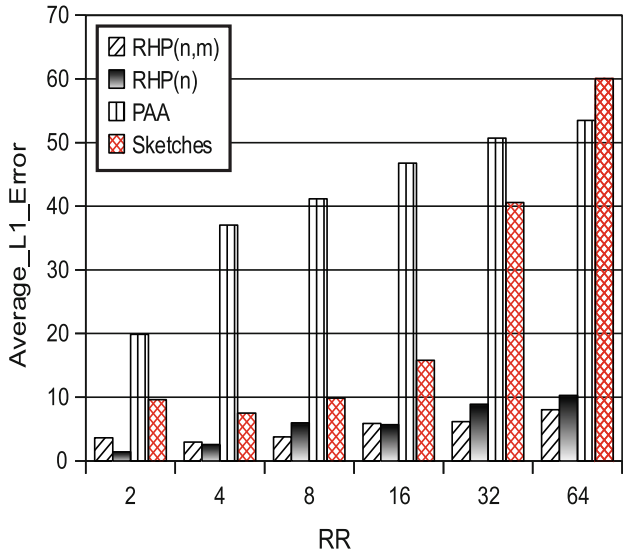



**Fig. 28** Eight clusters in Dataset

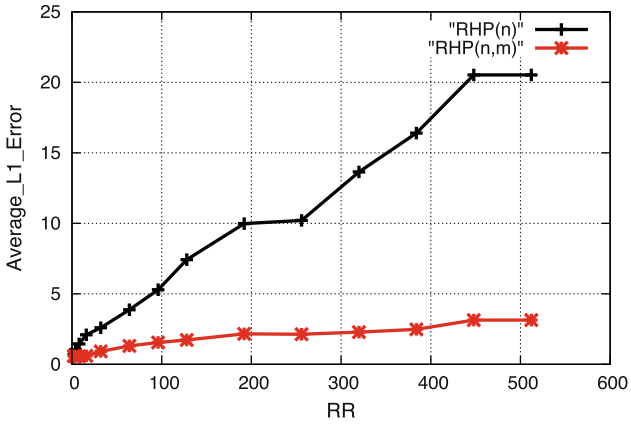



**Fig. 29** All RR values, one cluster in Dataset

sets with four and eight clusters, we can see that $RHP(n)$ is marginally more accurate than $RHP(n, m)$ for small reduction ratios (i.e., when substantially more bits are used to describe the data items). The reason is that the generated clusters when viewed in the hyperspherical coordinates are rather large (with a radius of 10 degrees along each coordinate). Thus, the data sets with four and eight clusters tend to cover a large amount of the data space, reducing the benefits of our technique. Of course, when data are completely random, we do expect $RHP(n)$ to perform best, assuming that enough bits are available (thus, its benefits are more evident for RR = 2 in these data). For values of RR exceeding 4, our technique provides better estimates in all data sets used. As has been explained in Sect. 6.2, using the hybrid method, we can retain the benefits of our techniques for the portion of data that is strongly correlated in space, while we fall back to the basic technique for the rest of the data items. In these experiments, we did not utilize the hybrid method. However, we explore this technique later in Sect. 7.7.

In Fig. 29, we depict the performance of $RHP(n, m)$ and $RHP(n)$ for the data set with one cluster for a larger range of reduction ratio values. As expected our method is substantially more accurate when fewer bits are used to describe the data items, due to the use of the derived dimensions.

### 7.5 Selection of representatives

In Fig. 30, we evaluate the performance of the $RHP(n, m)$ scheme when up to five representatives are considered in estimating the behavior of a derived random vector. The figure shows minor differences between the different variants of the algorithm. Since adding more representatives increases the space requirements of the decoding algorithm (please

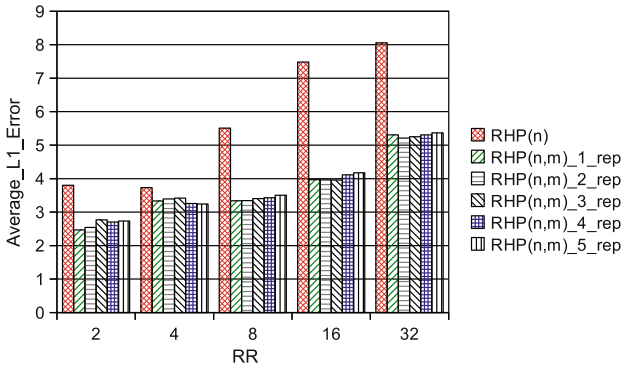



**Fig. 30** Using more than one representative, NBA Dataset

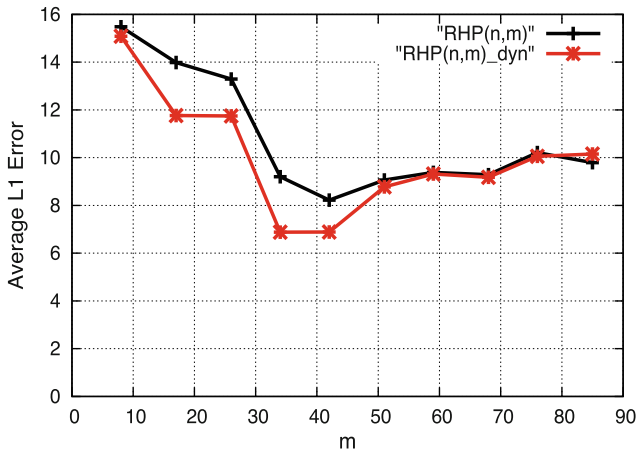



**Fig. 31** Average L1 error for dynamic and static selection of representatives for NBA Dataset

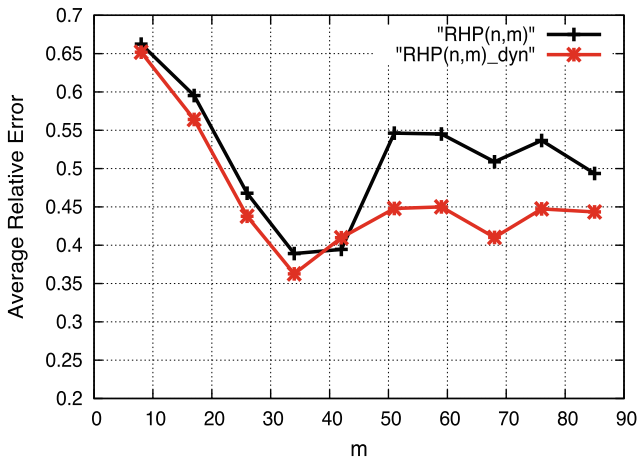



**Fig. 32** Average relative error for dynamic and static selection of representatives for NBA Dataset

see discussion of Sect. 5.9) using exactly one representative seems more appealing.

Another experiment was held in order to evaluate the performance of the dynamic selection of representatives for the derived vectors (as it was described in Sect. 6.1) compared

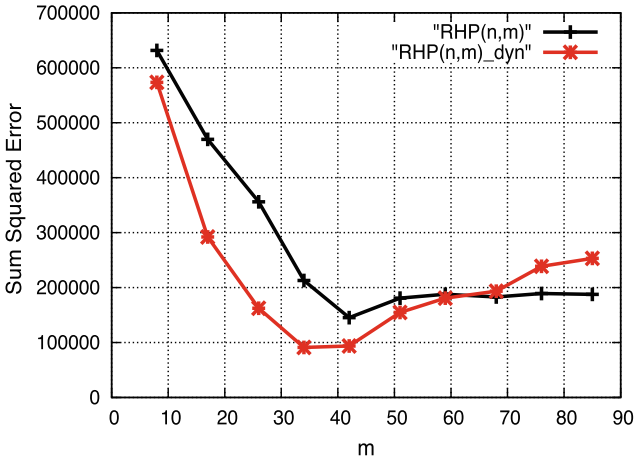



**Fig. 33** SSE for dynamic and static selection of representatives for NBA Dataset

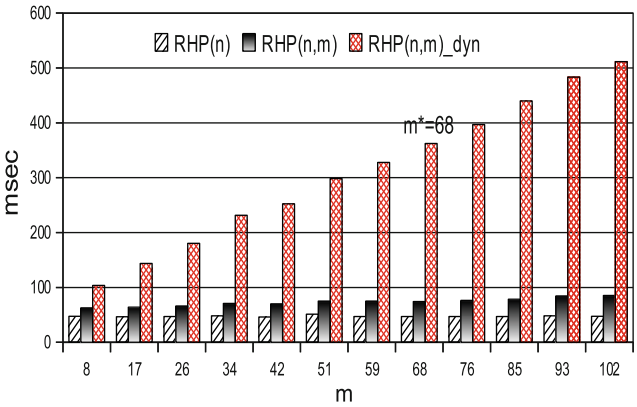



**Fig. 34** Execution time

with the static selection that the greedy algorithm suggests. We used three different metrics to measure the performance of each method on the NBA data set, for $n = 17$ (reduction ratio 32:1). Figures 31, 32 and 33 depict the average L1 error, the average relative error and the sum squared error respectively for both methods (dynamic and static selection of representatives). These graphs suggest that the dynamic selection generally performs better than the static selection and reduces the error up to 25% in case of the average L1 error, up to 18% in case of the average relative error and up to 57% for the sum squared error. These benefits of the dynamic selection are available at an increased computational overhead, compared with the static selection. This is depicted in Fig. 34 were we show the average time in msecs for performing a comparison for the two versions of $RHP(n, m)$ as well as for the original $RHP(n)$ technique varying the number of derived vectors $m$. The static selection has practically the same performance as $RHP(n)$. As expected, the time required to decode an $RHP(n, m)$ bitmap grows linearly with $m$ for the dynamic selection ($n$ is constant in this setting). This graph is reported in order to provide a better intuition on the

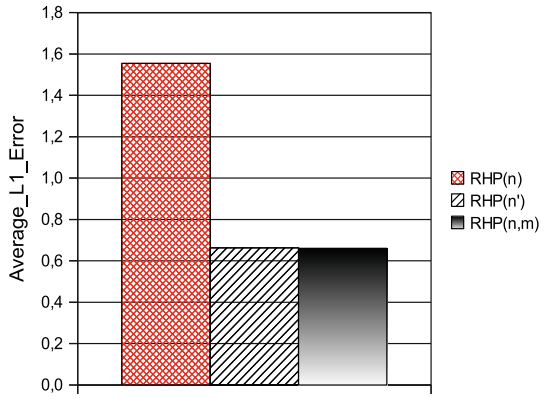



**Fig. 35** Average L1 error

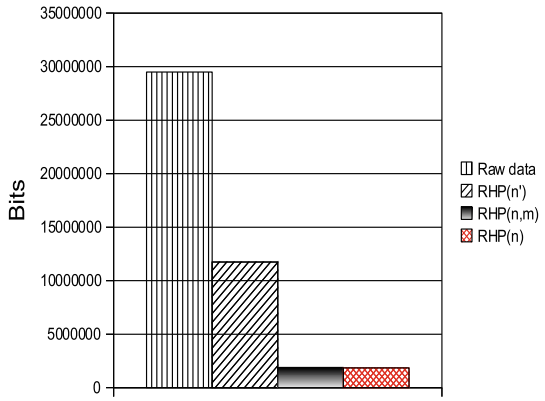



**Fig. 36** Transmitted bits

complexity of the techniques. We also depict the number of derived vectors $m^*$ that our algorithms have selected, which determines their performance for this data set.

### 7.6 Experiments in distributed environments

In order to highlight the benefits of our scheme in a distributed environment, we used the TEMPERATURE data set of [13] in order to simulate a network of 48 sensor nodes that report temperature readings to a base station. The base station uses the most recent observations by each sensor in order to compute similarity between motes (for example to find areas with correlated observations). We ran four different scenarios using this data set. In the first, the sensor nodes transmit their raw measurements, allowing exact computation of similarity at the cost of increased bandwidth consumption. In the second, we used $RHP(n)$ for RR = 16. The third alternative utilizes $RHP(n, m)$, for the same reduction ratio. We further used the distributed process described in Sect. 5.7 in order to derive the value of $m = m^*$. Finally, we also tested $RHP(n')$ for a larger value of $n'$ (smaller reduction ratio), so as to obtain approximately the same average L1 error as our method. In Fig. 35, we report the average L1 error for the different *RHP* schemes. We note that for the same bandwidth consumption, our method reduces the average error by a factor of 2, approximately. In Fig. 36, we report the number of bits transmitted in the network for all four methods. Clearly, transmitting the raw data results in increased bandwidth consumption, which is shown to reduce network lifetime [18]. For the same error as *RHP*, our method reduces bandwidth consumption by almost a factor of 7. Compared with $RHP(n)$ (same reduction ratio), our technique uses marginally more bandwidth (because of the extra communication involved in computing $m^*$ and during the sampling process, which are both included in the figure), however, it is substantially more accurate, as is depicted in Fig. 35.

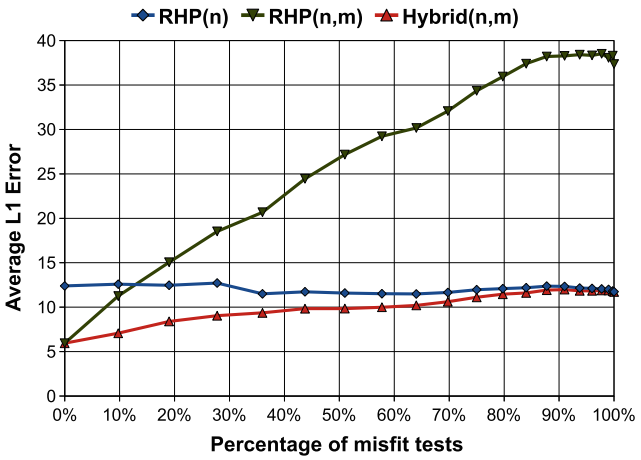



**Fig. 37** Performance when random data points are injected in the NBA data set

### 7.7 Hybrid RHP

We further evaluate the extension to our framework described in Sect. 6.2. We trained our algorithms using a 10% sample of the NBA data set and then used a different selection of 100 data items for computing their similarity. During the evaluation process, we progressively replaced data items with randomly generated data points in $R^{17}$. These random points significantly differ from the sample that we used to train our method and present the worst case example for our technique as they uniformly cover the data space, negating any strong correlations between materialized and derived random vectors. On the contrary, they do not affect the basic $RHP(n)$ scheme that is oblivious to the data in query. The same procedure was repeated for the HOUSE data set using random points from $R^6$. In Figs. 37, 38, we evaluate the performance of (1) our original framework, (2) the classic $RHP(n)$ scheme, (3) the extension described in Sect. 6.2, denoted as $Hybrid(n, m)$ in the graphs, for the NBA and HOUSE data sets, respectively. Reduction ratio was 16:1 in all algorithms. We can see that our technique starts loosing

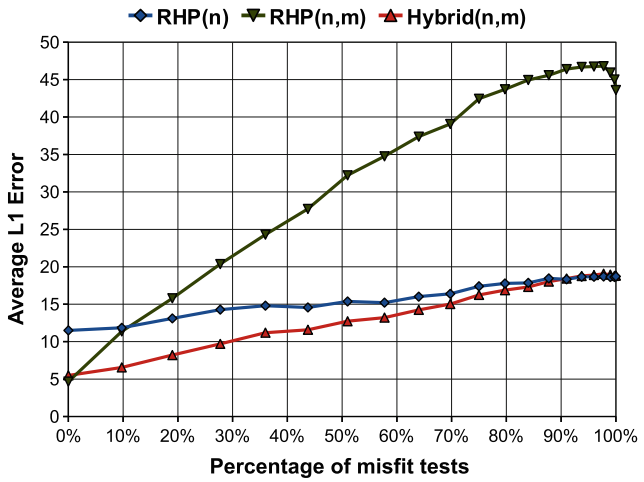



**Fig. 38** Performance when random data points are injected in the HOUSE data set

its accuracy, when more random data items are used in the evaluation process (x-axis in the graphs depicts the percentage of tested pairs for which at least one of the data items was not from the original data set). This is expected, as these random items do not follow the original data distribution. We can see in both data sets that the Hybrid method retains the benefits of our technique and outperforms the $RHP(n)$ scheme in this challenging test. This is because the Hybrid algorithm is far better than $RHP(n)$ for items selected from the original data (as is depicted in Figs. 18 and 20), while performs comparable to $RHP(n)$, when random data points are considered.

## 8 Conclusions

In this paper, we presented a data reduction framework that allows us to perform similarity tests between data descriptions transmitted from remote sites. Our method adopts the RHP framework but also takes into account the data distribution, detects correlations in the underlying data and achieves much better accuracy. We introduced a process for computing statistics from a RHP-projected data sample of higher dimensionality and then utilized these statistics in order to select a subset of possible dimensions on which the data are finally projected. We also discussed extensions to our proposed framework that dynamically alter the mappings between materialized and derived dimensions, in order to provide more accurate estimates. Furthermore, we presented techniques that allow our algorithms to cope with changes in data distribution. A detailed experimental evaluation using several real and synthetic data sets was presented and showed that our method outperforms the original RHP framework and alternative techniques we considered.

## References


1. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. In: STOC (1996)
2. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Commun. ACM **51**(1), 117–122 (2008)
3. Athitsos, V., Alon, J., Sclaroff, S., Kollios, G.: BoostMap: an embedding method for efficient nearest neighbor retrieval. IEEE Trans. Pattern Anal. Mach. Intell. **30**(1), 89–104 (2008)
4. Athitsos, V., Potamias, M., Papapetrou, P., Kollios, G.: Nearest neighbor retrieval using distance-based hashing. In: ICDE, pp. 327–336 (2008)
5. Bawa, M., Condie, T., Ganesan, P.: Lsh Forest: self-tuning indexes for similarity search. In: WWW, pp. 651–660 (2005)
6. Berchtold, S., Böhm, C., Kriegel, H.-P.: The pyramid-technique: towards breaking the curse of dimensionality. In: SIGMOD Conference, pp. 142–153 (1998)
7. Charikar, M.: Similarity estimation techniques from rounding algorithms. In: STOC (2002)
8. Chávez, E., Navarro, G., Baeza-Yates, R.A., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. **33**(3), 273–321 (2001)
9. Chen, J., Kher, S., Somani, A.: Distributed fault detection of wireless sensor networks. In: DIWANS (2006)
10. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: SCG (2004)
11. Deligiannakis, A., Kotidis, Y., Roussopoulos, N.: Compressing historical information in sensor networks. In: ACM SIGMOD (2004)
12. Deligiannakis, A., Kotidis, Y., Roussopoulos, N.: Hierarchical in-network data aggregation with quality guarantees. In: Proceedings of EDBT (2004)
13. Deligiannakis, A., Kotidis, Y., Vassalos, V., Stoumpos, V., Delis, A.: Another outlier bites the dust: computing meaningful aggregates in sensor networks. In: ICDE (2009)
14. Dobra, A., Garofalakis, M.N., Gehrke, J., Rastogi, R.: Multi-query optimization for sketch-based estimation. Inf. Syst. **34**(2), 209–230 (2009)
15. Dong, W., Wang, Z., Josephson, W., Charikar, M., Li, K.: Modeling LSH for performance tuning. In: CIKM, pp. 669–678 (2008)
16. Faloutsos, C. and Lin, K.-I.: FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In: SIGMOD Conference, pp. 163–174 (1995)
17. Giatrakos, N., Kotidis, Y., Deligiannakis, A.: PAO: power-efficient attribution of outliers in wireless sensor networks. In: DMSN (2010)
18. Giatrakos, N., Kotidis, Y., Deligiannakis, A., Vassalos, V., Theodoridis, Y.: TACO: tunable approximate computation of outliers in wireless sensor networks. In: SIGMOD (2010)
19. Gilbert, A., Kotidis, Y., Muthukrishnan, S., Strauss, M.: QuickSAND: quick summary and analysis of network data. Technical report, DIMACS 2001-43, Dec (2001)
20. Gilbert, A.C., Guha, S., Indyk, P., Kotidis, Y., Muthukrishnan, S., Strauss, M.: Fast, small-space algorithms for approximate histogram maintenance. In: STOC, pp. 389–398 (2002)
21. Gilbert, A.C., Kotidis, Y., Muthukrishnan, S., Strauss, M.: One-pass wavelet decompositions of data streams. IEEE Trans. Knowl. Data Eng. **15**(3), 541–554 (2003)
22. Gionis, A., Gunopulos, D., Koudas, N.: Efficient and tunable similar set retrieval. In: SIGMOD (2001)
23. Goemans, M., Williamson, D.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM, **42**(6) (1995)

24. Guha, S., Koudas, N., Shim, K.: Approximation and streaming algorithms for histogram construction problems. ACM Trans. Database Syst. **31**(1), 396–438 (2006)
25. Haghani, P., Michel, S., Aberer, K.: Distributed similarity search in high dimensions using locality sensitive hashing. In: EDBT, pp. 744–755 (2009)
26. Hua, Y., Xiao, B., Feng, D., Yu, B.: Bounded LSH for similarity search in peer-to-peer file systems. In: ICPP, pp. 644–651 (2008)
27. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: STOC (1998)
28. Jagadish, H.V., Ooi, B.C., and Vu, Q.H.: BATON: a balanced tree structure for peer-to-peer networks. In: VLDB, pp. 661–672 (2005)
29. Jeffery, S., Alonso, G., Franklin, M.J., Hong, W., Widom, J.: Declarative support for sensor data cleaning. In: Pervasive (2006)
30. Traina, C.T.A.J.M. Jr.., Faloutsos, C., Seeger, B.: Fast indexing and visualization of metric data sets using slim-trees. IEEE Trans. Knowl. Data Eng. **14**(2), 244–260 (2002)
31. Kalnis, P., Ng, W.S., Ooi, B.C., Tan, K.-L.: Answering similarity queries in peer-to-peer networks. Inf. Syst. **31**(1), 57–72 (2006)
32. Keogh, E.J., Chakrabarti, K., Pazzani, M.J., Mehrotra, S.: Dimensionality reduction for fast similarity search in large time series databases. Knowl. Inf. Syst. **3**(3), 263–286 (2001)
33. Kim, J.W., Candan, K.S.: Skip-and-prune: cosine-based top-k query processing for efficient context-sensitive document retrieval. In SIGMOD Conference, pp. 115–126 (2009)
34. Kotidis, Y., Deligiannakis, A., Stoumpos, V., Vassalos, V., Delis, A.: Robust management of outliers in sensor network aggregate queries. In: MobiDE (2007)
35. Koudas, N., Marathe, A., Srivastava, D.: Propagating updates in SPIDER. In: ICDE, pp. 1146–1153 (2007)
36. Lv, Q., Josephson, W., Wang, Z., Charikar, M., Li, K.: Multi-probe LSH: efficient indexing for high-dimensional similarity search. In: VLDB, pp. 950–961 (2007)
37. Madden S., Franklin M.J., Hellerstein J.M., Hong W.: TAG: a tiny aggregation service for ad hoc sensor networks. In: OSDI Conference (2002)
38. Ravichandran, D., Pantel, P., Hovy, E.: Randomized algorithms and NLP: using locality sensitive hash function for high speed noun clustering. In: ACL (2005)
39. Sacharidis, D., Deligiannakis, A., Sellis, T.K.: Hierarchically compressed wavelet synopses. VLDB J. **18**(1), 203–231 (2009)
40. Tao, Y., Yi, K., Sheng, C., Kalnis, P.: Quality and efficiency in high dimensional nearest neighbor search. In: SIGMOD Conference, pp. 563–576 (2009)
41. Tao, Y., Yi, K., Sheng, C., Kalnis, P.: Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. ACM Trans. Database Syst. **35**(3), (2010)
42. Vlachou, A., Doulkeridis, C., Kotidis, Y.: Peer-to-peer similarity search based on M-tree indexing. In: DASFAA (2), pp. 269–275 (2010)
43. Vlachou, A., Doulkeridis, C., Kotidis, Y., Vazirgiannis, M.: SKYPEER: efficient subspace skyline computation over distributed data. In: Proceedings of ICDE (2007)
44. Xiao, X., Peng, W., Hung, C., Lee, W.: Using sensorranks for in-network detection of faulty readings in wireless sensor networks. In: MobiDE (2007)
45. Xue, G., Jiang, Y., You, Y., Li, M.: A topology-aware hierarchical structured overlay network based on locality sensitive hashing scheme. In: UPGRADE (2007)
46. Yu, C., Ooi, B.C., Tan, K.-L., Jagadish, H.V.: Indexing the Distance: An Efficient Method to KNN Processing. In: VLDB, pp. 421–430 (2001)