DOI:10.1145/3441689

Scalable Signal Reconstruction for a Broad Range of Applications

By Abolfazl Asudeh, Jees Augustine, Saravanan Thirumuruganathan, Azade Nazi, Nan Zhang, Gautam Das, and Divesh Srivastava

Abstract

Signal reconstruction problem (SRP) is an important optimization problem where the objective is to identify a solution to an underdetermined system of linear equations that is closest to a given prior. It has a substantial number of applications in diverse areas, such as network traffic engineering, medical image reconstruction, acoustics, astronomy, and many more. Unfortunately, most of the common approaches for solving SRP do not scale to large problem sizes. We propose a novel and scalable algorithm for solving this critical problem. Specifically, we make four major contributions. First, we propose a dual formulation of the problem and develop the DIRECT algorithm that is significantly more efficient than the state of the art. Second, we show how adapting database techniques developed for scalable similarity joins provides a substantial speedup over DIRECT. Third, we describe several practical techniques that allow our algorithm to scale—on a single machine—to settings that are orders of magnitude larger than previously studied. Finally, we use the database techniques of materialization and reuse to extend our result to dynamic settings where the input to the SRP changes. Extensive experiments on real-world and synthetic data confirm the efficiency, effectiveness, and scalability of our proposal.

1. INTRODUCTION

The database community has been at the forefront of grappling with challenges of big data and has developed numerous techniques for the scalable processing and analysis of massive datasets. These techniques often originate from solving core data management challenges but then find their way into effectively addressing the needs of big data analytics. We study how database techniques can benefit *large-scale signal reconstruction*,[13] which is of interest to research communities as diverse as computer networks,[15] medical imaging,[7] etc. We demonstrate that the scalability of existing solutions can be significantly improved using ideas originally developed for similarity joins[5] and selectivity estimation for set similarity queries.[3]

Signal reconstruction problem (SRP): The essence of SRP is to solve a linear system of the form $AX = b$, where X is a high-dimensional unknown *signal* (represented by an m-d vector in $\mathbb{R}^m$), b is a low-dimensional projection of X that can be observed in practice (represented by an n-d vector in $\mathbb{R}^n$ with $n \ll m$), and A is an $n \times m$ matrix that captures the linear relationship between X and b. There are many real-world applications that follow the SRP model (see Section 2.1). High-dimensional signals such as environmental temperature can only be observed through low-dimensional observations, such as readings captured by a small number of temperature sensors. End-to-end network traffic, another high-dimensional signal, is often monitored through low-dimensional readings such as traffic volume on routers in the backbone or edge networks. In these applications, the laws of physics or the topology of computer networks reveal the value of A, and our objective is to reconstruct the high-dimensional signal X from the observation b based on the knowledge of A.

As $n \ll m$, the linear system is underdetermined. That is, for a given A and b, there are an infinite number of feasible solutions (of X) that satisfy $AX = b$. In order to identify the best reconstruction of the signal, it is customary to define and optimize for a *loss function* that measures the distance between the reconstructed X and a prior understanding of certain properties of X. For instance, one's prior belief of X can be specified as an m-d vector X' and define the loss function as the ℓ_2-norm of $X - X'$, that is, $\|X - X'\|_2$. In other cases, when prior knowledge indicates that X is sparse, one can define the loss function as the ℓ_2-norm of X, aiming to minimize the number of nonzero elements in the reconstructed signal. For the purpose of this paper, we consider the ℓ_2-based loss function of $\|X - X'\|_2$, which has been adopted in many application-oriented studies such as Grangeat and Amans[7] and Zhang et al.[15]

Running example of SRP: SRP has a broad range of applications. For the ease of exposition, we use as a running example based on network tomography (Section 2.1), where the objective is to compute the pairwise end-to-end traffic in IP networks. Pairwise traffic measures the volume of traffic between all pairs of source-destination nodes in an IP network and has numerous uses such as capacity planning, traffic engineering, and detecting traffic anomalies. Informally, consider an IP network where various sources and destinations send different amounts of traffic to each other. The network administrator is aware of the network topology and the routing table (from which we can construct matrix A). In addition, the administrator can observe the traffic passing through each link in the backbone network (observation b). The goal is to find the amount of traffic flow between all source-destination pairs (signal X). Note that one cannot directly measure the raw traffic between all source-destination pairs due to challenges in

The original version of this paper was entitled "Leveraging Similarity Joins for Signal Reconstruction" and was published in *PVLDB 10*, 11 (2018), 1276–1288.

instrumentation and storage—see Zhang et al.[15] for a technical discussion. In almost all real-world IP networks, the number of source-destination pairs is significantly larger than the number of links, leading to an underdetermined linear system. To reconstruct the pairwise traffic, the network community introduced various traffic models, for example, the gravity model,[15] as the prior for X', and used the ℓ_2-distance between X and the prior as the loss function. Note that in reconstructing the pairwise distances, efficiency is a concern front and center, especially given the rise of software designed networks (SDNs) that feature much larger sizes and much more frequent topological changes, pushing further the scalability requirements of signal reconstruction algorithms.

Research gap: Because of the importance of SRP, there has been extensive work from multiple communities on finding efficient solutions. To solve the problem efficiently, methods explored in the recent literature include statistical likelihood-based iterative algorithms based on expectation-maximization, as well as the use of linear algebraic techniques such as computing the pseudoinverse of A[13] or performing singular value decomposition (SVD) on A, and iterative algorithms for solving the linear system.[13] Yet even these approaches cannot scale to fully meet the requirements in practice, especially in settings such as traffic reconstruction in large-scale IP networks—which call for a more scalable solution.

Our approach: In this paper, we consider a special case of SRP where A, X, and b are nonnegative with A being a *sparse binary matrix*. Such a setting finds its applications in many domains, as explained in Section 2.1. We present an exact algorithm (Direct) based on the transformation of the problem into its Lagrangian dual representation. Direct already outperforms commonly used approaches for SRP, as it avoids expensive linear algebraic operations required by the previous solutions and scales up to medium-size settings. Next, we investigate whether our approach can be sped up even further, by replacing exact computations with approximation techniques. After a careful investigation of Direct, it turns out that the computational bottleneck is a special case of matrix multiplication involving a sparse binary matrix with its transpose. We use the observation that a small number of cells in the result matrix of the bottleneck operation take the bulk of the values and propose a threshold-based algorithm for approximating it. Specifically, we reduce the problem to computing the dot product of two vectors if and only if their similarity is above a user-provided threshold. Our key idea here is to leverage various database techniques to speed up the multiplication operation. We propose a hybrid algorithm based on a number of techniques originally proposed for computing similarity joins and selectivity estimation of set similarity queries, resulting in significant speedup, enabling our proposal to scale to large-scale settings.

We push the boundaries to very large systems (VLS) with sizes in the order of a million equations with a billion unknowns. We identify that the barrier to this extension is the output size of the multiplication of A by its transpose, that is, AA^T, as it is simply too large to be kept in memory. We conducted careful theoretical analyses and experimental evaluation on the number of nonzero elements in this matrix that confirm the matrix is sparse in practice. We then leverage this sparsity to efficiently solve very large systems of equations. Finally, we consider the scenario where the input to our problem changes dynamically. We pay attention to the observation that the underlying structure of the system A does not change frequently. Vector b, on the other hand, may change often. We utilize the database technique of materialization and reuse a carefully constructed *signature matrix* for dynamic settings.

2. PROBLEM FORMULATION

We consider a special class of SRP that has a number of applications in network traffic engineering, tomographic image reconstruction, and many others. We are given a system of linear equations $AX = b$ where

- $A \in \{1, 0\}^{n\times m}$ is a sparse binary matrix $n \ll m$.
- $X \in \mathbb{R}^m$ is the "signal" to be reconstructed and is a vector of unknown values.
- $b \in \mathbb{R}^n$ is the vector of observations.

Each row in the matrix A corresponds to an equation with each column corresponding to an unknown variable. When the number of equations (n) is much smaller than the number of unknowns (m), the system of linear equations is said to be underdetermined and does not have a unique solution. The solution space can be represented as a hyperplane in an $m' \in [2, m]$ dimensional vector space.[a] Because SRP does not have a unique solution, one must have auxiliary criteria to choose the best solution from the set of (possibly infinite) valid solutions. A common approach in SRP is to provide a prior X' and the objective is to pick the solution X that is closest to X'. We study the problem where the objective is to find the point satisfying $AX = b$ that minimizes the ℓ_2-distance from a prior point X'. Formally, the problem is defined as:

$$\begin{aligned} \min \; & \|X - X'\|_2 \\ \text{s.t. } & AX = b \end{aligned} \tag{1}$$

[a] We assume that the problem has at least one solution.

Figure 1. Visualizing the problem.

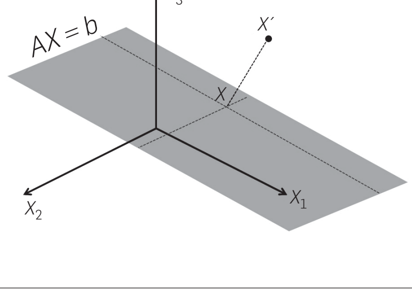


Figure 1 provides a visualization of the problem in three dimensions. The gray plane is the solution space with the prior marked as a point X'. The intersection of the perpendicular line to the plane that passes though X' is the point that minimizes $\|X - X'\|_2$.

We observe that SRP is a special case of quadratic programming where (a) the constraints are only in the form of equality, (b) matrix A is sparse, and (c) matrix A is binary (and hence unweighted). By leveraging these characteristics, we seek to design more efficient solutions compared with the baselines that are designed for general cases. In Section 3, we use the dual representative of the problem to propose an efficient exact algorithm. In Section 4, we show how leveraging similarity joins techniques help in achieving significant speedup without sacrificing much accuracy.

2.1. Applications of SRP

SRP covers a broad range of real-world problems that use signal reconstruction. In practice, it is popular to observe low-dimensional projections in the form of (unweighted) aggregates of a high-dimensional signal vector. For example, in general network flow applications (such as road traffic estimation[16]), the value on each edge is the summation of the flow values that includes this edge as part of the path between them. Of course, a requirement to our problem is an "expert-provided" prior template, such as *gravity model*[15] for the network flow problems. Another major application domain for SRP problem over aggregates is image reconstruction, where observations are unweighted projections of unknowns. Image reconstruction has broad applications ranging from medical imaging[7] to astronomy[14] and physics.[10] Some of the other applications of SRP, in general, include radar data reconstruction[9] and transmission electron microscopy,[8] to name a few. To showcase some applications in more detail, we sketch a few examples in the context of network flow problems and image reconstruction in the following.

Network tomography. Traffic matrix computation *(the running example)*: Consider an IP network with n traffic links and m source-destination traffic flows (SD flow) between the ingress and egress points, where $n \ll m$. The ingress/egress points can be points of presence (PoPs) or routers or even IP prefixes depending on the level of granularity required. The network has a routing policy and prescribes a path for each of the SD flows that can be captured in a $\#links(n) \times \#flows(m)$ binary matrix A, where the entry $A[i, j] = 1$ if the link i is used to route the traffic of the jth SD flow. The matrix A is sparse and "fat" with more SD flows (columns) than number of links (rows). Note that, one cannot directly measure each of the SD flows on a link owing to efficiency reasons. However, one can easily measure the total volume of the network traffic that passes through a given link using network protocols such as SNMP. Thus, the load on each link i becomes the observed vector b. To obtain a prior X', one can use any traffic model such as the popular and intuitive *gravity model*.[15] It assumes independence between source and destination and states that traffic between any given source s and destination d is proportional to the product of network traffic entering at s and that exiting at d.

Traffic analysis attack in P2P networks: In traffic analysis attack, the information leak on traffic data is exploited to expose the user traffic pattern in P2P networks. Here, we propose the following traffic analysis attack that can be modeled to our problem: consider an adversary who monitors the link level traffics in a P2P network. Applying SRP, one can directly identify the volume of traffic between any pair of users in a P2P network.

Image reconstruction. Image reconstruction[7] has a wide range of applications in different fields such as medical imaging,[7] and physics.[10] Given a set of (usually 2D) projection of a (usually 3D) image, the objective is to reconstruct it. The reconstruction is usually done with the help of some prior knowledge. For example, knowing that the 2D projections are taken from a human face, one may use a template 3D face photo and, among all possible 3D reconstructions from the 2D images, find the one that is the closest to the template, making the image reconstruction more effective.

CT scan: A popular application of SRP is tomographic reconstruction, which is a multidimensional linear inverse problem with wide range of applications in medical imaging[7] such as CT scans (computed tomography). A CT scan takes multiple 2D projections (vector b) through X-rays from different angles (matrix A) and the objective is to reconstruct the 3D image from the projections. Many 3D images may produce the same projections necessitating the use of priors to choose an appropriate reconstruction.

Radio astronomy: In astronomy, SRP has application for reconstructing interferometric images where the astrophysical signals are probed through Fourier measurements. The objective is to reconstruct the images from the observations—forming an SRP scenario. Also, the specific prior information about the signals plays an important role in reconstruction, as mentioned in Wiaux et al.[14]

3. EXACT SOLUTION FOR SOLVING SRP

We begin by describing two representative approaches for solving SRP from prior research and highlight their shortcomings. We then propose a dual representation of the problem that can be solved exactly in an efficient manner and already outperforms the baselines. This alternate formulation allows one to leverage various database techniques for speeding it up.

3.1. Lagrangian formulation of SRP

We leverage the Lagrangian dual form of SRP as a special case of quadratic programming and design an efficient exact solution for it. For SRP as specified in Equation 1, $f(X) = \frac{1}{2}X^T X - X'^T X$ and $g(X) = AX$.[b] Thus, our problem can be rewritten as:

$$L(X, \lambda) = \frac{1}{2}X^T X - X'^T X + \lambda^T (AX - b) \qquad (2)$$

[b] Note that $\min \frac{1}{2}X^T X - X'^T X$ is the same as $\min \|X - X'\|_2$.

[c] Because, looking at Figure 1, Equation 1 has a single optimal point, Equation 2 has one stationary point that happens to be the saddle point.

Next, we find the stationary point[c] of Equation 2 in the general form by taking the derivatives with regard to X and λ, and setting them to zero, we get:

$$X = X' - A^T(AA^T)^{-1}(AX' - b) \tag{3}$$

Solving SRP in dual form. The stationary point of Equation 2 is the optimal solution for our problem (Equation 1). In contrast to prior work, we solve the SRP problem by directly solving Equation 3. We make two observations. First, the matrix $AA^T \in \mathbb{Z}^{n\times n}$ always has an inverse as it is full rank. From Figure 1, one can note that the problem has a unique solution that minimizes the distance from the prior. It means that AA^T is full rank, because otherwise the problem was not feasible and would not have a solution. Second, Equation 3 does have a matrix inverse operator that is expensive to compute. However, one can avoid taking the inverse of AA^T by computing ξ in Equation 4 and replacing $(AA^T)^{-1}(AX' - b)$ by it in Equation 3.

$$(AA^T)\xi = AX' - b \tag{4}$$

Algorithm 1 provides the pseudocode for Direct.

Algorithm 1 Direct
Input: A, b, and X'
Output: X

```
1: $t = AA^T$
2: $t_2 = AX' - b$
3: Solve system of linear equations: $t\,\xi = t_2$
4: $X = X' - A^T\xi$
5: return $X$
```

Performance analysis of Direct. Let us now investigate the performance of our algorithm. Recall that A is a fat matrix with $n \ll m$, whereas X and X' are m-dimensional vectors, and b is a n-dimensional vector. Line 1 of Algorithm 1 takes $O(n^2m)$, whereas Line 2 takes $O(nm)$. Line 3 involves solving a system of linear equations. A naive way would be to compute the inverse of t that can take as much as $O(n^3)$. However, by observing that t is sparse, one can use approaches such as Gauss-Jordan elimination or other iterative methods that are practically much faster for sparse matrices. Finally, the computation of Line 4 is in $O(nm)$. Looking at Direct holistically, one can notice that its computational bottleneck is Line 1, thereby making the overall complexity to be $O(n^2m)$.

An additional approach to speedup Direct is to observe that matrix A is sparse and thereby to store it in a manner that allows efficient matrix multiplication. Because A is binary (and hence unweighted), a natural representation is to store only the indices of nonzero values. Figures 2a and 2b show the nonsparse and sparse representation of a matrix A. Note that AA^T is symmetric as $t[i, j]$ and $t[j, i]$ are obtained by the dot product of rows i and j of A. Let l be the number of nonzero elements in each row. Because A is sparse, $l \ll m$, one can design a natural matrix multiplication algorithm with time complexity of $O(nml)$ that is orders of magnitude faster than algorithm such as Strassen algorithm.

Figure 2. Illustration of the sparse representation of *A*. (a) Nonsparse representation and (b) sparse representation.

0	0	0	1	0	0	0	1	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1	0	1
0	1	0	0	0	0	1	0	0	0

(a)

$\langle 3, 7\rangle$
$\langle 2\rangle$
$\langle 5, 7, 9\rangle$
$\langle 1, 6\rangle$

(b)

4. TRADING OFF ACCURACY WITH EFFICIENCY

In many applications of SRP, m is often in $O(n^2)$, thereby making the computational complexity of Direct to be $O(n^4)$. The key bottleneck is the computation of AA^T. On the other hand, for large problem instances, the user may accept trading off accuracy with efficiency and prefer a close-to-exact solution that is computed quickly, rather than the expensive exact solution. Our objective is to speed up Direct by computing the bottleneck step, that is, computing AA^T, approximately. We show how to leverage a threshold-based approach by only computing the values of matrix AA^T that are larger than a certain threshold. We describe the connection between this problem variant and similarity joins and propose a hybrid method by adopting two classical algorithms designed for similarity estimation, which results in an efficient solution for computing AA^T.

4.1. Bounding values in matrix AA^T

We begin by showing that one can efficiently compute the bound for each cell value in matrix AA^T. Figure 3 shows a sparse matrix A with 183 rows and 495 columns, in which the

Figure 3. An example of the binary sparse matrix $A_{183\times495}$.

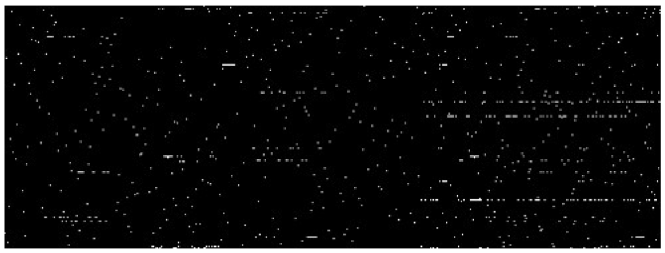
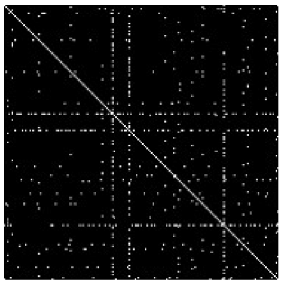


Figure 4. The nonzero elements in AA^T for the example of Figure 3.



nonzero elements are highlighted in white. Figure 4 shows the nonzero elements in matrix AA^T. We can notice that AA^T is square and also sparse due to the fact that every element of AA^T is the dot product of two sparse vectors (two rows of matrix A). Furthermore, one can also observe a more subtle phenomenon that we state in Theorem 1, which could be used to design an efficient algorithm.

THEOREM 1. *Given a sparse binary matrix A, considering the elements on the diagonal of* AA^T, *that is,* $t[i, i], \forall 0 \leq i < n$:

- $t[i, i] = |A[i]|$, *where* $|A[i]|$ *is the number of nonzero elements in row* $A[i]$.
- $t[i, i]$ *is an upper bound for the elements in the row* $t[i]$ *and the column* $t[, i]$; *formally,* $\forall 0 \leq j < n$: $t[i, j] \leq t[i, i]$ *and* $t[i, j] \leq t[j, j]$.

The proof can be found in Asudeh et al.[2]

Consider two representations of AA^T of the example matrix given in Figure 3. Figure 4 shows all the nonzero elements of AA^T, whereas Figure 5 shows a magnitude-weighted variant wherein cells with larger values are plotted in brighter colors. Figure 5 visually shows that the elements on the diagonal are brighter than the ones in the same row and column as predicted by Theorem 1. One may notice that most of the nonzero elements of AA^T (in Figure 4) are small values (in Figure 5). Although there are a reasonable number of nonzero elements, the number of elements with higher magnitude is often much smaller. Next, we use this insight along with Theorem 1 for speeding up DIRECT.

4.2. Threshold-based computation of AA^T

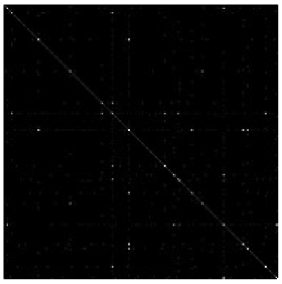
By developing a bound on the cell values in AA^T, we can see that a small number of elements in AA^T take the bulk of the value. This is the key in designing a threshold-based algorithm for computing AA^T wherein we only compute values of AA^T that are above a certain threshold. Specifically, we use the elements on the diagonal as an upper bound and only compute the elements for which this upper bound is larger than a user-specified threshold. Note that, if the threshold is equal to 1, the algorithm will compute the values of all elements. However, the user-specified threshold allows additional opportunities for efficiency.

Algorithm 2 provides the pseudocode for the threshold-based multiplication of sparse binary matrix A with its transpose. This algorithm depends on the existence of an oracle called SIM that given two rows $A[i]$ and $A[j]$, and the threshold τ, returns the dot product of $A[i]$ and $A[j]$ if the result is not less than τ.

Algorithm 2 Approx AA^T
Input: Sparse matrix A, Threshold τ
Output: t

```
1: $\mathcal{F} = \{\}$
2: for $i = 0$ to $n - 1$ do
3:     $t[i, i] = |A[i]|$
4:     if $|A[i]| \geq \tau$ then add $i$ to $\mathcal{F}$
5: end for
6: for every pair $i, j \in \mathcal{F}$ do
7:     $t[i, j] = t[j, i] = \text{SIM}(A[i], A[j], \tau)$
8: end for
9: return $t$
```

Figure 5. Magnitude of weights in AA^T for the example of Figure 3.



4.3. Leveraging similarity joins for Oracle SIM

The database community has extensively studied mechanisms for computing set similarity for applications such as data cleaning[5] where the objective is to efficiently identify the set of tuples that are "close enough" on multiple attributes. We next describe how to implement the oracle SIM by leveraging prior research on computing set similarity. Especially, we propose a hybrid method that combines the threshold-based similarity joins with the sketch-based methods to resolve their shortcomings.

Oracle SIM through set similarity. Given two rows $A[i]$ and $A[j]$, and the threshold τ, SIM should find the dot product of $A[i]$ and $A[j]$ if it is not less than τ. We can make an interesting connection between SIM and set similarity problems as follows. Let every column in matrix A be an object o in a universe $\mathcal{U}$ of m elements. Every row $A[i]$ represents a set U_i in $\mathcal{U}$, where $\forall o_j \in \mathcal{U}$, $o_j \in U_i$ *iff* $A[i, j] = 1$. Equivalently, each row corresponds to a set U_i that stores the indices of the nonzero columns similar to Figure 2b. Using this transformation, we can see that our objective is to compute $|U_i \cap U_j|$ for all pairs of sets U_i and U_j where $|U_i \cap U_j| \geq \tau$. Note that we represent $|U_i \cap U_j|$ by $\cap_{i,j}$ and $|U_i \cup U_j|$ by $\cup_{i,j}$, respectively.

Due to its widespread importance, different versions of this problem have been extensively studied in the DB community. We consider one exact approach and two approximate approaches based on threshold-based algorithms[5] and sketch-based methods.[3,6] We then compare and contrast the two approximate approaches, describe the scenarios when they provide better performance, and propose a hybrid algorithm based on these scenarios.

Exact approach: set intersection. One can see that when $\tau = 1$, the problem boils down to computing AA^T exactly. This in turn boils down to computing the intersection between two sets as efficiently as possible. The sparse representation of the matrix often provides the nonzero columns in an ordered manner. The simplest approaches for finding the intersection of ordered sets is to perform a linear merge by scanning both the lists in parallel and leveraging the ordered nature similar to the merge step of merge sort. One can also speedup this approach by using sophisticated approaches such as binary search on one of the lists or using sophisticated data structures such as treaps or skip lists. Each of these approaches allows one to "skip" some elements of a set when necessary.

Approximate approach: threshold-based algorithms. Threshold-based algorithms, such as Chaudhuri et al.,[5] identify the pair of sets such that their similarity is more than a given threshold. This has a number of applications such as data cleaning, deduplication, collaborative filtering, and product recommendation in advertisement where the objective is to quickly identify the pairs that are highly similar. The key idea is that if the intersection of two sets is large, the intersection of small subsets of them is nonzero.[5] More precisely, for two sets U_i and U_j with size h, if $\cap_{i,j} \geq \tau$, any subset $U_i' \subset U_i$ and $U_j' \subset U_j$ of size $h - \tau + 1$ will overlap; that is, $|U_i' \cap U_j'| > 0$. Using this idea, while considering an ordering of the objects, the algorithm first finds the set of candidate pairs that overlap in a subset of size $h - \tau + 1$. In the second step, the algorithm verifies the pairs, by removing the false positives.

One can see that the effectiveness of this method highly depends on the value of τ and, considering the target application, it works well for the cases where τ is large. For example, consider a case where $h = 100$. When $\tau = 99$ (i.e., 99% similarity), the first filtering step needs to compare the subsets of size 2 and is efficient, whereas if $\tau = 10$, the filtering step needs to compare the subset pairs of size 91, which is close to the entire set. The latter case is quite possible in our problem. To understand it better, let us consider matrix A in Figure 3, while setting τ equal to 5 in Algorithm 2. Even though the size of many of the rows is close to the threshold, there are rows $A[i]$ where $|A[i]|$ is significantly larger than it. For example, for two rows $A[i]$ and $A[j]$ where $|A[i]| \geq 50$ and $|A[j]| \geq 50$, to satisfy the condition that the dot product should not be less than τ, the filtering step needs to compare the subsets of size ≥ 44, which is close to the exact comparison of $A[i]$ and $A[j]$.

Approximate approach: sketch-based algorithms. Sketch-based methods such as Beyer et al.[3] and Cohen and Kaplan[6] use a precomputed synopsis such as a minhash for answering different set aggregates such as Jaccard similarity. The main idea behind the minhashing-[4]based algorithms is as follows: consider a hash (ordering) of the elements in $\mathcal{U}$. For each set U_i, let $h_{\min}(U_i)$ be the element $o \in U_i$ that has the minimum hash value. Two sets U_i and U_j have the same minhash, when the element with the smallest hash value belongs to their intersection. Hence, it is easy to see that the probability that $h_{\min}(U_i) = h_{\min}(U_j)$ is equal to $\frac{\cap_{i,j}}{\cup_{i,j}}$, that is, Jaccard similarity of U_i and U_j. Bottom-k sketch,[6] a variant of minhashing, picks the hash of the k elements in U_i with the smallest hash value, as its signature. The Jaccard similarity of two sets U_i and U_j is estimated as $\frac{k_\cap(i,j)}{k}$, where $k_\cap(i,j)$ is $|h_k(U_i) \cap h_k(U_j)|$. Beyer et al.[3] use the bottom-k sketch for estimating the union and intersection of the sets. Let $h_{i,j}[k]$ be the hash value of the kth smallest hash value in $h_k(U_i) \cup h_k(U_j)$. The idea is that the larger the size of a set is, the smaller the expected value of the kth element in hash is. Using the results of Beyer et al.,[3] $\frac{m(k-1)}{h_{i,j}[k]}$ is an unbiased estimator for $\cup_{i,j}$. Hence, the estimation for $\cap_{i,j}$ is as provided in Equation 5.

$$E\left[\cap_{i,j}\right] = \frac{k_\cap(i,j)}{k}\frac{m(k-1)}{h_{i,j}[k]} \tag{5}$$

Estimating $\cup_{i,j}$ with Equation 5 performs well when $\cup_{i,j} \gg 1$,[3] that is, the larger sets. Hence, we combine the threshold-based and sketch-based algorithms to design the oracle SIM, as a hybrid method that, based on the sizes of the rows $A[i]$ and $A[j]$, adopts the threshold-based computation with sketch-based estimation for computing the dot product of $A[i]$ and $A[j]$. We consider $\log(m)$ as the threshold to decide which strategy to adopt. Considering the effectiveness of threshold-based approaches when U_i and U_j are small and, as a result, the two sets need a large overlap to have the intersection larger than τ, if $|U_i|$ and $|U_j|$ are less than $\log(m)$, we choose the threshold-based intersection computation. However, if the size of U_i or U_j is more, then we use the bottom-k sketch, while considering k to be $\log(m)$. For each element $o_j \in \mathbb{U}$, we set $h(o_j) = j$. Hence, for each vector U_i, the index of the first $\log(m)$ elements in it is its bottom-k sketch. Using this strategy, Algorithm 3 shows the pseudocode of the oracle SIM.

Given two sets U_i and U_j (corresponding to the rows $A[i]$ and $A[j]$) together with the threshold τ, the algorithm aims to compute the value of $\cap_{i,j}$, if it is larger than τ. Combining the two aforementioned methods, if $|U_i|$ and $|U_j|$ are more than a value α, the algorithm uses sampling to estimate $\cap_{i,j}$; otherwise, it applies the threshold-based method to compute it. During the sampling, rather than sampling from $\mathcal{U}$, the algorithm samples from U_i to reduce the underestimation of probability. In this case, in order to compute $\cap_{i,j}$, the algorithm, for each sample, picks a random object from U_i and checks its existence in U_j. It is easy to see it is an unbiased estimator for $\cap_{i,j}$, where its expected value is $\cap_{i,j}$. If $|U_i|$ or $|U_j|$ is less than α, the algorithm applies threshold-based strategy for computing $\cap_{i,j}$. As discussed earlier in this subsection, in order for $\cap_{i,j}$ to be more than τ, the subsets of size $\cap_{i,j} - \tau + 1$ should intersect. Hence, the algorithm first applies the threshold filtering, and only if the two subsets intersect, it continues with computing $\cap_{i,j}$.

Algorithm 3 SIM
Input: The sets U_i and U_j, Threshold τ
Output: c

```
1: if $|U_i| \geq \log(m)$ and $|U_j| \geq \log(m)$ then
2:     $h_i$ = the first $k$ elements in $U_i$
3:     $h_j$ = the first $k$ elements in $U_j$
4:     $k_\cap(i,j) = |h_i \cap h_j|$
5:     $h_{i,j}[k]$ = the first $k$ elements in $h_i \cup h_j$
6:     $c = \frac{k_\cap(i,j)}{k}\frac{m(k-1)}{h_{i,j}[k]}$
7: else
8:     $c = 0$
9:     if $|U_i| > |U_j|$ then swap $U_i$ and $U_j$
```

```
10:     $\beta = |U_i| - \tau$
11:     for $k = 0$ to $\beta$ do: if $U_i[k] \in U_j$ then $c = c + 1$
12:     if $c = 0$ then return 0
13:     for $k = \beta$ to $|U_i| - 1$ do: if $U_i[k] \in U_j$ then $c = c + 1$
14:   end if
15:   return $c$
```

5. SCALING TO VERY LARGE SETTINGS

So far, we considered the scenario where n is not a large number. Recall that n is the size of the low-dimensional projection of the unknown variables. We relax this assumption and extend Direct for handling the cases where n is very large (and still $n \ll m$). For example, n can be in the order of a million, whereas m is in the order of a billion. A key aspect of Direct is that it leverages the sparse representation of the matrix (as against its complete dense representation) for speedup. However, when n is very large, even fitting the sparse representation of A into the memory may not be possible. Even if there is only one nonzero value in *every column*, we need $O(m)$ storage for the matrix.

Interestingly, the similarity joins-based techniques proposed in Section 4 do not require to completely materialize even sparse representation of A for estimating AA^T. Also, there are many scenarios where the user is interested in knowing the values of a subset of components of the reconstructed signals such as those corresponding to the largest values of the reconstructed signal. We now show how to adapt our algorithms to handle these scenarios.

Consider Algorithm 1 where the critical step is the first line. Algorithm 3 applies bottom-k sketch for the sets whose size is more than log m. Thus, choosing the signature size in the bottom-k sketch to be in $O(\log m)$, Algorithm 3 needs at most $O(\log m)$ elements from *each row*. As a result, Line 1 of Direct needs a representation of size $O(n \log m)$ of A. For instance, in our example of $n = 10^6$ and $m = 10^{12}$, the size of the representative of A is only in the order of 1 million rows by 40 columns.

A key assumption for scaling our results to very large settings is that $t = AA^T$ is sparse in practice. In Asudeh et al.,[1] we theoretically study the sparsity of t. Specifically, we provide a lower bound and an upper bound on how sparse t can be. Using an adversarial example, we show the existence of cases for which the matrix is not sparse. Still, as we shall illustrate in Section 7, AA^T is sparse in practice. It even becomes significantly more sparse after applying thresholding. Therefore, we only store the nonzero values of matrix t, rather than the complete n by n matrix. Line 2 of Algorithm 1 is the multiplication of matrix A with X' whose dimensions are m by 1 followed by subtracting the n-dimensional result vector from the vector b. For this line, for each row of A, we use a sample of size $O(\log m)$ for the nonzero elements of the row, while using the values of X' as the sampling distribution. The result is a representation of size $O(n \log m)$ of A. Also, rather than loading the complete vector X' to the memory, in an iterative manner, we bring loadable buckets of it to the memory, update the calculation for that bucket, and move to the next one. In Line 4, t is the nonzero elements of AA^T and t' is an n by 1 vector, and finding the n by 1 vector ξ is doable, using methods such as Gauss-Jordan. Finally, we only limit the calculations to the variables of interest, or even if the computation of all variables is required, in an iterative manner, we move a loadable bucket of them to the memory, compute their values, and move to the next bucket.

6. DYNAMIC SIGNAL RECONSTRUCTION

So far, we focused on the static variant of the SRP, where the objective is to find the point in the answer space that minimizes the distance to the prior X'. We next investigate a practical scenario where the input to SRP changes. The naive solution is to invoke our algorithms from scratch whenever the input changes. We observed that in many instances of SRP, not all the inputs of a signal reconstruction change. Hence, it is possible to *materialize* some results from the previous iterations and *reuse* them to compute the solution for the current iteration.

Consider our running example of SRP where the objective is to compute the end-to-end traffic between the source-destination pairs in an IP network. Although the actual traffic between the pairs may change quickly, the underlying network topology changes infrequently. Hence, the binary matrix A is also unchanged. The changes in the traffic affect the observation vector b and possibly the prior point X'. Reconstructing the signal X every few minutes based on current observations when there is no change in network topology is an extremely important scenario in traffic engineering.

Recall that computing AA^T is the performance bottleneck of Direct. Interestingly, because the underlying topology of the graph does not change, the computation of AA^T can be considered as an amortized preprocessing step that can be materialized and reused for dynamic changes. Also remember that Line 3 of Algorithm 1 uses Gaussian elimination for finding ξ in $(AA^T)\xi = AX' - b$. We observed that this is the second performance bottleneck after the computation of AA^T. We propose a novel approach[1] to speedup this computation by materializing (and maintaining) an $n \times (n + 1)$ signature **matrix** S that enables the computation of ξ in $O(n^2)$, instead of $O(n^3)$ for the recomputation.[d]

Constructing the signature matrix. For ease of explanation, let $R = AA^T$ and $t = AX' - b$. Now the objective is to find ξ in:

$$R\,\xi = t$$

Note that in the above equation, R is fixed as AA^T does not change. At a high level, in order to generate the signature matrix, we apply Gaussian elimination for a general form of t and maintain the delayed operation in the signature matrix. Later on, upon the arrival of an update, we use the signature matrix to updates on t and compute ξ accordingly. We would like to highlight the similarity of our signature matrix with LU decomposition, where the matrix AA^T is decomposed into two matrices L and U.[12] Compared to our proposal,[1] the update using the L and U matrices as

[d] We note that one can materialize the inverse matrix $(AA^T)^{-1}$ (or $A^T(AA^T)^{-1}$) as the signature. This however would require more storage and would not give computational advantage compared to our proposal.[1]

signature needs solving of two (albeit specialized) systems of linear equations of time complexity $O(n^2)$ for computing ξ. We conducted experiment on validating the effectiveness of our methods and the results are described in Section 7.

7. EXPERIMENTAL EVALUATION

7.1. Experimental setup

Hardware and platform. All of our small-scale experiments were performed on a Macintosh machine with a 2.6 GHz CPU and 8GB memory. The algorithms were implemented using Python 2.7 and MATLAB. For very large setting experiments, we used a 4.0 GHz, 64GB server that runs on Ubuntu 18.04 and the code was rewritten in C++ for scalability and efficiency.

Datasets. We conducted extensive experiments to demonstrate the efficacy of our algorithms over graphs with diverse values for a number of nodes, edges, and source-destination pairs. Recall that given a communication network, the size of the routing matrix A is parameterized by the number of edges and number of source-destination pairs—and not by the number of nodes and edges. We used different datasets with different scales for the experiments. We outline a subset of those datasets in Figure 6. Please refer to Asudeh et al.[1] for the complete list. For small- and medium-size datasets (N_1 in Figure 6), used for comparing against the prior work, we use the synthetic datasets. Our large datasets are real datasets that were derived from a p2p dataset from SNAP repository of Stanford University.[e] Each of the derived large datasets is a subgraph of the overall p2p graph and was obtained by Forest Fire model. For very large (VLS) datasets, we used the complete Gnutella dataset, as well as a popular location-based social networking platform, Brightkite.

Once we sample the network and obtain a connected graph, we consider all possible source destination pairs to be the individual flows. For each source-destination pair, we calculated the shortest path between them and used *Pareto* traffic generation model for generating the flow values. The *prior* point for the experiments (X') was obtained as a function of gravity model from Zhang et al.[15]

7.2. Experimental results

We report a representative subset of our experiment results here. Please refer to Asudeh et al.[1,2] for the complete results.

[e] snap.stanford.edu/data/p2p-Gnutella04.html.

Figure 6. Dataset characteristics.

Network	#Nodes	#Edges	#SD pairs
N_1	274	281	827
p2p-3	1438	7081	2M
VLS2	10879	44944	32M
VLS3	8298	104469	32M
VLS4	108300	191886	64M
VLS5	36692	372612	128M
VLS6	58228	43310 6	0.5B

First, as shown in Figure 7, Direct significantly outperforms the baselines QP and WLSE[15] on the small dataset N_1. In addition to comparing with these two baselines, for N_1, we also used compressive sensing for estimating flow values, which took more than 23 s, even for our smallest setting. We next evaluate the exact version of Direct and its approximate counterpart (using Algorithm 2) that leverages techniques from similarity joins to speed up the computation. We use Direct-e to refer to the exact version of Direct and Direct-a for its approximate version. We also evaluate the performance of our algorithms to two different threshold values of (m/1000) and (m/100), where m is the number of source-destination pairs. Choosing an appropriate threshold is often domain specific with larger thresholds providing better speedups. We compare the performance of the algorithms Direct-e and Direct-a through execution time and accuracy.

p2p-3 (2M source-destination pairs). This network has 2M source-destination pairs with 7081 edges sampled from the *SNAP p2p* dataset. Figure 8 shows that Direct-e takes much as 1500 s to compute the exact solution. This is often prohibitive and simply unacceptable for many traffic engineering tasks. However, our approximate algorithms can provide the result in as little as 35 s. This is a significant reduction in execution time with a speedup of as much as 97% over the running time of Direct-e. We would like to mention that our experiments[2] demonstrate negligible

Figure 7. Direct v.s. baselines in N_1: $n = 281$ and $m = 827$.

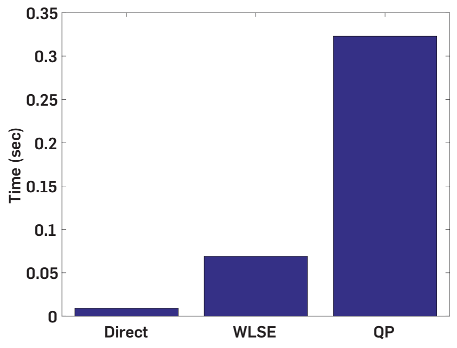
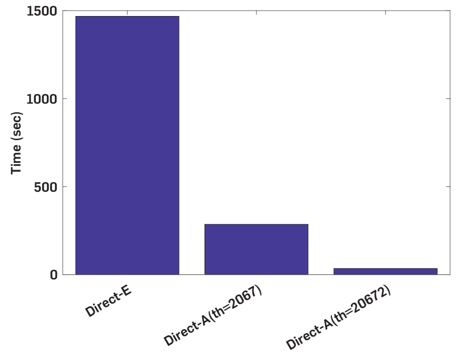


Figure 8. Execution time of Direct-e, Direct-a($\tau = 2067$), and Direct-a ($\tau = 20672$) in p2p-3.



approximation errors, even for threshold value of $(m/100)$, which is tolerable for many tasks in network traffic engineering such as routing optimization.[15]

Sparsity and thresholding results of AA^T. We chose VLS2 settings to demonstrate the effectiveness of thresholding, the lower and upper bounds provided by theory for the settings, and an overall reduction in nonzero elements by a suitable threshold. The results are provided in Figure 9. We also included the theoretical lower bound and upper bound in the figure. The number of nonzero values in $t = AA^T$ for this setting is $97M$, which is about 4.85% of the total cells. However, with a modest threshold, $\tau = 2$, this number quickly dropped to 0.003%, which highlights the effectiveness of thresholding.

Scalability results. In this experiment, we show the scalability of our final algorithm. To do so, we compare the performance of DIRECT-A across different input scales of $n \times m$, which confirm the scalability of DIRECT-A for the very large settings through experiments on VLS2 to VLS6. Figure 10 presents the results from scalability experiments for very large settings with varied values of n and m. Note that all the experiments are run on a single machine. Still, even for the very large setting of $.5M \times .5B$, the algorithm finished in a reasonable time of less than 17 minutes.

Figure 9. AA^T **sparsity on** ***VLS2*****.**

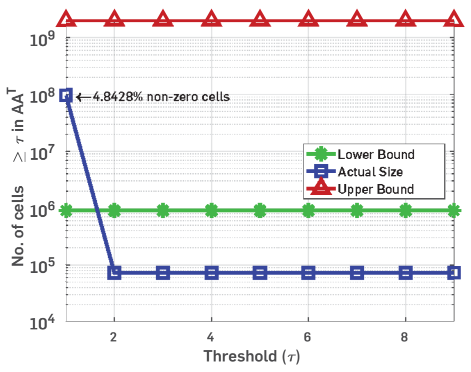


Figure 10. Scalability on *n* and *m*.

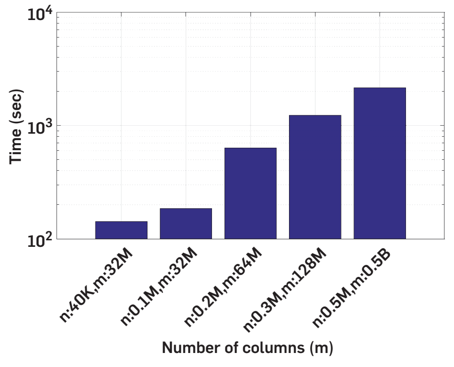
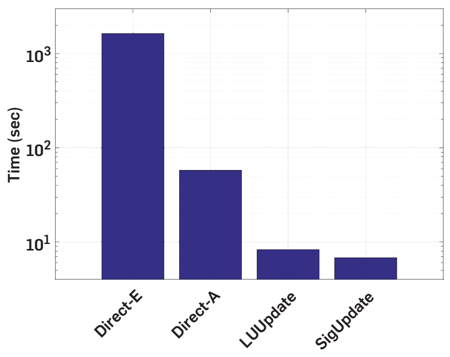


Figure 11. Dynamic-update performance on network *p2p*-3.



Dynamic signal reconstruction results. Our last set of experiments is for handling the dynamic updates. The results for dynamic scenario for $p2p$-3 are given in Figure 11. For expounding the effects of our dynamic approach through signature matrix, we also considered adopting LU decomposition for signature matrix (LUUpdate). As is evident, both LUUpdate and SigUpdate perform well and SigUpdate slightly outperforms the other. This is because LUUpdate requires solving two systems of linear equations.

8. RELATED WORK

Linear algebraic techniques for solving SRP: There has been extensive work on solving the system of linear equations using diverse techniques such as computing the pseudoinverse of A[13] or performing singular value decomposition (SVD) on A, and iterative algorithms for solving the linear system.[13] However, none of these methods scale for large-scale signal reconstruction problems. A key bottleneck in these approaches is the computation of the pseudoinverse for matrix A. Any matrix B such that $ABA = A$ is defined as a pseudoinverse for A. It is possible to identify "the infinitely many possible generalized inverses,"[13] each with its own advantages and disadvantages. Moore-Penrose Pseudoinverse (MPP)[11] is one of the most well-known and widely used pseudoinverse. MPP is the pseudoinverse that has the smallest Frobenius norm, minimizes the least-square fit in overdetermined systems, and finds the shortest solution in the underdetermined ones. However, none of the pseudoinverse definitions suits our purpose of finding the solution X that minimizes the ℓ_2-distance from a prior. Furthermore, computing pseudoinverses is often done by SVD that is computationally very expensive.

9. CONCLUSION

In this paper, we investigated how a wide ranging problem of large-scale signal reconstruction can benefit from techniques developed by the database community. Efficiently solving SRP has a number of applications in diverse domains such as network traffic engineering, astronomy, and medical imaging. We propose an algorithm DIRECT based on the Lagrangian dual form of SRP. We identify a number of computational bottlenecks in DIRECT and evaluate the use of database techniques such as sampling and similarity joins for speeding them up without much loss in accuracy. Our experiments on networks that are orders of magnitude larger than prior work show the potential of our approach.

ACKNOWLEDGMENTS

This work was supported in part by AT&T, the National Science Foundation under grants 1343976, 1443858, 1624074, and 1760059, and the Army Research Office under grant W911NF-15-1-0020.

References

1. Asudeh, A., Augustine, J., Nazi, A., Thirumuruganathan, S., Zhang, N., Das, G., Srivastava, D. Scalable algorithms for signal reconstruction by leveraging similarity joins. *VLDB J. 29*, 2 (2020), 681–707.
2. Asudeh, A., Nazi, A., Augustine, J., Thirumuruganathan, S., Zhang, N., Das, G., Srivastava, D. Leveraging similarity joins for signal reconstruction. *PVLDB 10*, 11 (2018), 1276–1288.
3. Beyer, K., Gemulla, R., Haas, P.J., Reinwald, B., Sismanis, Y. Distinct-value synopses for multiset operations. *Commun. ACM 10*, 52 (2009), 87–95.
4. Broder, A.Z. On the resemblance and containment of documents. In *SEQUENCES* (1997), IEEE, 21–29.
5. Chaudhuri, S., Ganti, V., Kaushik, R. A primitive operator for similarity joins in data cleaning. In *ICDE* (2006). IEEE.
6. Cohen, E., Kaplan, H. Tighter estimation using bottom k sketches. *PVLDB 1*, 1 (2008), 213–224.
7. Grangeat, P., Amans, J.-L. *Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, Vol. 4. Springer Science & Business Media, Springer Netherlands, 1996.
8. Kalinin, S.V., Strelcov, E., Belianinov, A., Somnath, S., Vasudevan, R.K., Lingerfelt, E.J., Archibald, R.K., Chen, C., Proksch, R., Laanait, N., et al. Big, deep, and smart data in scanning probe microscopy. *ACS Nano 10*, 10 (2016), 9068–9086.
9. Liu, Z., Shi, Z., Jiang, M., Zhang, J., Chen, L., Zhang, T., Liu, G. Using MC algorithm to implement 3d image reconstruction for yunnan weather radar data. *J. Comput. Commun.* 05, 5 (2017), 50–61.
10. Massey, R., Rhodes, J., Ellis, R., Scoville, N., Leauthaud, A., Finoguenov, A., Capak, P., Bacon, D., Aussel, H., Kneib, J.-P., et al. Dark matter maps reveal cosmic scaffolding. *arXiv preprint astro-ph/0701594* (2007).
11. Penrose, R. A generalized inverse for matrices. In *Mathematical Proceedings of the Cambridge Philosophical Society*, Volume 51 (1955), 406–413.
12. Trefethen, L.N., Bau III, , D. Technical report. *Numerical Linear Algebra*. Philadelphia: Society for Industrial and Applied Mathematics. ISBN 978–0–89871–361–9, 1997.
13. Vogel, C.R. *Computational methods for inverse problems*. SIAM, 2002.
14. Wiaux, Y., Jacques, L., Puy, G., Scaife, A.M., Vandergheynst, P. Compressed sensing imaging techniques for radio interferometry. *Monthly Notices of the Royal Astronomical Society 3*, 395 (2009), 1733–1742.
15. Zhang, Y., Roughan, M., Duffield, N., Greenberg, A. Fast accurate computation of large-scale IP traffic matrices from link loads. In *SIGMETRICS*, Volume 31, 2003.
16. Zhu, Y., Li, Z., Zhu, H., Li, M., Zhang, Q. A compressive sensing approach to urban traffic estimation with prob vehicles. *IEEE Trans. Mobile Comput. 11*, 12 (2012), 2289–2302.

Abolfazl Asudeh (asudeh@uic.edu), University of Illinois at Chicago.

Jees Augustine and **Gautam Das** (({jees.augustine@mavs, gdas@cse}.uta.edu)), University of Texas at Arlington.

Saravanan Thirumuruganathan (sthirumuruganathan@hbku.edu.qa), QCRI, HBKU.

Azade Nazi (azade.nazi@google.com), Google Brain.

Nan Zhang (nzhang@american.edu), Kogod School of Business, American University.

Divesh Srivastava (divesh@research.att.com), AT&T Labs-Research.