Leveraging Similarity Joins for Signal Reconstruction

Abolfazl Asudeh[†*], Azade Nazi[††*], Jees Augustine[‡], Saravanan Thirumuruganathan[‡†], Nan Zhang[†‡], Gautam Das[‡], Divesh Srivastava[‡‡]
[†]University of Michigan; [††]Microsoft Research; [‡]University of Texas at Arlington; [‡†]QCRI, HBKU; [†‡]Pennsylvania State University; [‡‡]AT&T Labs-Research
[†]asudeh@umich.edu, [††]azade.nazi@microsoft.com, [‡]{jees.augustine@mavs, gdas@cse}.uta.edu, [‡†]sthirumuruganathan@hbku.edu.qa, [†‡]nan@ist.psu.edu, [‡‡]divesh@research.att.com

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 including network traffic engineering, medical image reconstruction, acoustics, astronomy and many more. Most common approaches for SRP do not scale to large problem sizes. In this paper, we propose a dual formulation of this problem and show how adapting database techniques developed for scalable similarity joins provides a significant speedup. Extensive experiments on real-world and synthetic data show that our approach produces a significant speedup of up to 20x over competing approaches.

PVLDB Reference Format:
Abolfazl Asudeh, Azade Nazi, Jees Augustine, Saravanan Thirumuruganathan, Nan Zhang, Gautam Das, and Divesh Srivastava. Leveraging Similarity Joins for Signal Reconstruction. *PVLDB*, 11 (10): 1276-1288, 2018.
DOI: https://doi.org/10.14778/3231751.3231752

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 ways into effectively addressing the needs of big data analytics. For example, the efficiency of machine learning was boosted by database techniques such as materialization [42], join optimization [27], query rewriting for efficiency [3], query progress estimation [25], federated databases [31], etc. This paper studies how database techniques can benefit another foundational problem in big data analytics, *large-scale signal reconstruction* [41], which is of significant interest to research communities such as computer networks [43], medical imaging [21, 24], astronomy [11], acoustics [26], etc. We demonstrate that the scalability of existing solutions can be significantly improved using ideas originally developed for similarity joins [9] and selectivity estimation for set similarity queries [1, 22].

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 a $n \times m$ matrix that captures the linear relationship between X and b. There are many real-world applications that follow the SRP model: For example, high-dimensional signals like environmental temperature can only be observed through low-dimensional observations, like readings captured by a small number of temperature sensors. Similarly, 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.

Since $n \ll m$, the linear system is underdetermined. That is, for a given A and b, there are infinite number of feasible solutions (of X) that satisfy $AX = b$ [23, 41]. 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 example, one can represent one's prior belief of X as an m-d vector X', and define the loss function as the ℓ_2-norm of $X - X'$, i.e., $\|X - X'\|_2$. In other cases, when prior knowledge indicates that X is sparse, one can define the loss function as the ℓ_0-norm of X, aiming to minimize the number of non-zero 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 [11, 21, 43].

Running Example of SRP: While SRP has a broad range of applications, for the ease of discussion, it is important to have a running example of SRP on a domain-specific application. What we use as a running example of SRP throughout the paper is the computation of 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 amount 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 adminis-

*This work was done while the author was a PhD student at the University of Texas at Arlington.

Proceedings of the VLDB Endowment, Vol. 11, No. 10

DOI: https://doi.org/10.14778/3231751.3231752

trator can observe the traffic passing through each link in the backbone network (observation b). The objective 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 instrumentation and storage - see [43, 44] 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, e.g., the gravity model [43], 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) [35] which 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. Traditionally, solving the ℓ_2 loss function was considered to be more efficient than the ℓ_0 version, given the existence of polynomial-time quadratic programming solutions for ℓ_2 compared with the NP-hardness of ℓ_0-optimization [17]. Hence, significant effort was spent on solving the ℓ_0 problem more efficiently. The landmark result of compressive sensing [6], for example, proves that the ℓ_0 loss function can be optimized with efficient ℓ_1-minimization when the signal is very sparse.

Different from compressive sensing, we focus on the ℓ_2 problem here because, even though quadratic programming has polynomial-time solutions (e.g., the ellipsoid method [4]), it is still very expensive in practice when the size of A is large [4]. To address the efficiency issue for the ℓ_2 problem, methods explored in the recent literature include statistical likelihood based iterative algorithms based on expectation-maximization [7], as well as the use of linear algebraic techniques such as computing the pseudoinverse of A [41], performing Singular Value Decomposition (SVD) on A [11], and iterative algorithms for solving the linear system [41]. Yet even these approaches cannot scale to fully meet the requirements in practice, especially the traffic reconstruction needs of large-scale IP networks - which call for a more scalable solution [44].

Our Approach: In this paper, we consider a special case of SRP where A, X, b are non-negative with A being a *sparse binary matrix*. Such a setting finds its applications in many domains such as computer networks [43], medical imaging [21,24], astronomy [11], acoustics [26], etc., including our running example of computation of pairwise end-to-end traffic in IP networks.

Our proposed solution starts with an exact algorithm based on the transformation of the problem into its Lagrangian dual representation. As we shall show in § 6, our algorithm DIRECT, which directly computes X through the dual representation, already outperforms commonly used approaches for SRP, as it avoids expensive linear algebraic operations required by the previous solutions. Next, we investigate whether our approach can be sped up even further, by replacing exact computations with approximation techniques. This can be useful in applications where the user is willing to trade accuracy for efficiency. We carefully investigate the computational bottlenecks of DIRECT and find it to be a special case of matrix multiplication involving a sparse binary matrix with its transpose. We start by investigating a seeming straightforward sampling strategy for approximately computing this matrix multiplication, but encounter a negative result. Then, 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 in solving SRP in comparison with the exact solution.

Experimental Summary: We conduct extensive experiments on both real-world and synthetic datasets with a special emphasis on traffic matrix computation. We compare our method against a number of commonly used approaches such as an efficient quadratic programming based solver, a two stage approximate approach first proposed in [43] and one based on compressive sensing. Our experimental results show that our exact algorithm significantly outperforms the baselines by as much as 20x on large networks. Furthermore, our threshold based approximation approaches inspired from similarity joins provide even more speedup over DIRECT without resulting in any significant increase in reconstruction error.

Summary of Contributions:

- In this paper, we investigate an important optimization problem of Signal Reconstruction Problem (SRP) that has diverse applications. By using techniques that were originally pioneered for databases, we dramatically improve the scale of problems that could be solved.
- We formulate SRP as a Quadratic Programming problem and derive its Lagrangian dual form and propose an exact algorithm DIRECT to solve the dual problem. Our algorithm DIRECT already outperforms commonly used approaches for SRP.
- We identify the computation bottleneck in DIRECT and propose a threshold-based algorithm for approximating it. We propose a hybrid algorithm that combines two algorithms that were designed for efficiently computing set similarity.
- We conduct comprehensive set of experiments on both real and synthetic datasets that confirm the efficiency and effectiveness of our approach.

Paper Organization: We provide the necessary background to SRP and formally define it in § 2. In § 3, we describe the exact algorithm DIRECT for solving SRP. In § 4, we show how to apply approximation using techniques from databases to significantly speed up the computation. In § 5, we discuss how our approach can be easily adapted to identify the top-K components of the reconstructed signal. § 6 describes our comprehensive set of experiments followed by related work in § 7 with § 8 providing the conclusion.

2. PROBLEM FORMULATION

As mentioned in Section 1, 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

- $A \in \mathbb{R}^{n \times m}$ is a sparse binary matrix $n \ll m$. In many applications m is $O(n^2)$.
- $X \in \mathbb{R}^m$ 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 under-determined and does not have a unique solution. The solution space can be repre-

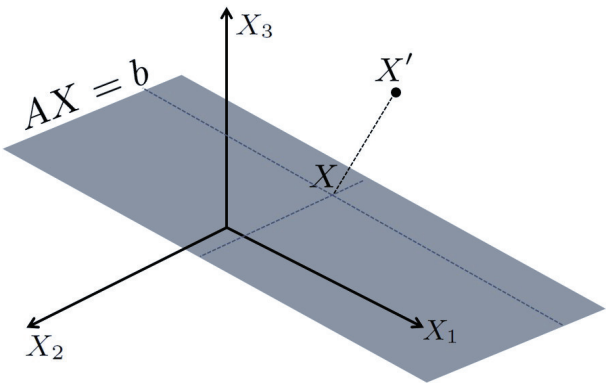


Figure 1: Visualizing the problem

sented as a hyperplane in a $m' \in [2, m]$ dimensional vector space[1]. Since SRP does not have a unique solution, one must have an 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 on the $AX = b$ that minimizes the ℓ_2 distance from a prior point X'. Formally the problem is defined as:

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

Figure 1 provides an example visualization of the problem in 3 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 $\|X - X'\|_2$.

Traffic Matrix Computation. Consider an IP network with n links where the network traffic is the result of m source-destination traffic flows (SD flow) between the ingress/egress points where $n \ll m$. Depending on the granularity required, the ingress/egress points can be PoPs (points of presence) or routers or even IP prefixes. The network has a routing policy that prescribes a path for each possible SD flow that can be specified by a binary matrix A with dimensions $\#links \times \#flows$ where the entry $A[i, j] = 1$ if the link i is used to route the traffic of the j-th SD flow. The matrix A is sparse and "fat" with more SD flows than number of links. Note that due to efficiency reasons, one cannot directly measure all the SD flows. However, one can easily measure the network traffic that passes through a given link using network protocols such as SNMP. 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* [43]. It assumes independence between source and destination and states that traffic between a given source s and destination d is proportional to the product of network traffic entering at s and that exiting at d.

In this paper, we pay attention to the fact 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 un-weighted). By leveraging these characteristics, we seek to design more efficient solutions compared with the baselines that are designed for general cases. Especially, in § 3, after studying the existing work, we use the dual representative of the problem to propose an efficient exact algorithm. Later in § 4, we show how leveraging similarity join techniques help in achieving significant speed up without sacrificing much accuracy.

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

3. EXACT SOLUTION FOR SOLVING SRP

In this section, 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 has a number of appealing properties that allows one to leverage various database techniques for speeding it up.

3.1 Baseline Approaches

A common approach to solve SRP is to observe that Equation 1 is a quadratic programming problem and can be solved using an existing solver. Furthermore, it can be shown that applying Singular Value Decomposition (SVD) on the matrix A is an efficient way to find the solution X that minimizes the distance to X' [44]. If this approach results in a negative value for some component of X, they are set to 0 and an Iterative Proportional Fitting (IPF) technique is used to obtain a non-negative X that still satisfies all the constraints. We refer to this baseline approach as `QP` in our paper. An alternate and more efficient approach proposed in [43] solves SRP in two phases. In the first phase, it obtains an initial solution (that might not be feasible) through the link load information (b) and the routing policy A. In [43], the authors used the gravity model based approach to quickly compute one possible solution. This solution is then refined by applying quadratic programming based optimization to find a feasible solution that minimizes the distance to the initial solution. Intuitively, this approach orthogonally projects the solution obtained by the gravity model into the constraint space such that it satisfies all the constraints in A. Furthermore, one can also choose among different solutions by applying a weighted least squares based approach that gives different weights to different components of X. We refer to this algorithm as `WLSE`. We provide the implementation of these two baselines in § 6 (Figures 8 and 9). While both `QP` and `WLSE` are extensively used in prior work, they are often computationally expensive. This is due to the fact that solving these algorithms requires using linear algebraic procedures for SVD or Pseudo inverse that can be expensive to compute for large matrices.

3.2 Lagrangian Formulation of SRP

In this subsection, we leverage the Lagrangian dual form [28] of SRP as a special case of quadratic programming, and design an efficient exact solution for it.

Lagrangian Dual Form. A general optimization problem in the form of

$$\begin{aligned} \min \ & f(X) \\ \text{s.t. } & g(X) = b \end{aligned} \tag{2}$$

can be rewritten in the Lagrangian dual form as:

$$L(X, \lambda) = f(X) + \lambda^T (g(X) - b) \tag{3}$$

where λ, a vector of size n (n is the number constraints in g), is the set of new variables introduced in the Lagrangian expression. Therefore, $L(X, \lambda)$ contains $n+m$ variables. The stationary points (the points where partial derivatives are zero) of the Lagrangian expression L can be used for finding the optimal solution for the original problem specified in Equation 2.

Lagrangian Dual of SRP. For SRP as specified in Equation 1, $f(X) = \frac{1}{2} X^T X - X'^T X$ and $g(X) = AX$.[2] Thus, our prob-

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

lem can be re-written as:

$$L(X,\lambda) = \frac{1}{2}X^T X - X'^T X + \lambda^T(AX - b) \quad (4)$$

Next, we find the stationary point [3] of Equation 4 in the general form by taking the derivatives with regard to X and λ and setting them to zero, as following:

$$\begin{aligned}\frac{\partial L(X,\lambda)}{\partial X} =& X^T - X'^T + \lambda^T A = 0\\ \Rightarrow& X = X' - A^T\lambda\end{aligned} \quad (5)$$

$$\begin{aligned}\frac{\partial L(X,\lambda)}{\partial \lambda} =& AX - b = 0\\ \Rightarrow& A(X' - A^T\lambda) = b\\ \Rightarrow& AA^T\lambda = AX' - b\\ \Rightarrow& \lambda = (AA^T)^{-1}(AX' - b)\end{aligned} \quad (6)$$

From Equations 5 and 6:

$$X = X' - A^T(AA^T)^{-1}(AX' - b) \quad (7)$$

Solving SRP in Dual Form. The stationary point of Equation 4 is the optimal solution for our problem (Equation 1). In contrast to prior work, we solve the SRP problem by directly solving Equation 7. We make two observations. First, the matrix $AA^T \in \mathbb{R}^{n\times n}$ 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 7 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 8, and replacing $(AA^T)^{-1}(AX' - b)$ by it in Equation 7.

$$(AA^T)\xi = AX' - b \quad (8)$$

Algorithm 1 provides the pseudocode for DIRECT. We also provide its Matlab implementation in § 6 (Figure 7).

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 $n \ll m$ while X and X' are m-dimensional vectors, and b is a n-dimensional vector. Line 1 of Algorithm 1 takes $O(n^2m)$ while 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 $O(n^2m)$.

[3]Since, looking at Figure 1, Equation 1 has a single optimal point, Equation 4 has one stationary point which happens to be the saddle point.

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)

Figure 2: Illustration of the sparse representation of A. (a) Non sparse representation, (b) Sparse representation

In addition to the theoretical analysis, in the experiment section, we empirically study the performance of DIRECT, where, as we shall show, directly computing the value of X significantly outperforms the baselines of solving the quadratic programming in the general form or using the SVD-based approach.

3.3 Speeding Up DIRECT by Sparse Matrix Representation

An alternate approach to speedup DIRECT is to observe that matrix A is sparse and thereby store it in a manner that allows efficient matrix multiplication. Since A is binary (and hence unweighted), a natural representation is to store only the indices of non-zero values. Figures 2a and 2b show the sparse and non-sparse representation of a matrix A.

Note that AA^T is symmetric since $t[i,j]$ and $t[j,i]$ are obtained by the dot product of rows i and j of A. Algorithm 2 shows an efficient way to exactly compute $t = AA^T$ by using the sparse representation that maintains the index of non-zero elements of each row. We can see that Algorithm 2 has a time complexity of $O(nml)$, where l is the upper bound on the number of non-empty elements in each row. Since A is sparse, $l \ll m$ and hence Algorithm 2 is orders of magnitude faster than a traditional matrix-multiplication algorithm such as Strassen algorithm.

Algorithm 2 ExactAA^T
Input: Sparse matrix A
Output: t

1: **for** $i = 0$ to $n-1$ **do**
2: **for** $j = i$ to $n-1$ **do**
3: $t[i,j] = 0$, $k_1 = 0$, $k_2 = 0$
4: **while** $k_1 < |A[i]|$ **and** $k_2 < |A[j]|$ **do**
5: **if** $A[i,k_1] < A[j,k_2]$ **then** $k_1 = k_1 + 1$
6: **else if** $A[i,k_1] > A[j,k_2]$ **then** $k_2 = k_2 + 1$
7: **else** $t[i,j] = t[i,j] + 1$, $k_1 = k_1 + 1$, $k_2 = k_2 + 1$
8: **end while**
9: $t[j,i] = t[i,j]$
10: **end for**
11: **end for**
12: **return** t

4. TRADING OFF ACCURACY WITH EFFICIENCY

While Algorithm 2 is quite efficient for small to medium problem instances, it is not scalable for large problem instances. This is due to the fact that 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. In this section, our objective is to speed up DIRECT by computing the bottle-neck step, i.e., computing AA^T, approximately. We start the section by studying a seemingly straightfor-

ward sampling-based approach for this purpose, but encounter a negative result. Next, 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 Sampling and its Negative Result

In this subsection, we study sampling for approximately computing AA^T. It provides a dramatic speedup albeit at the cost of accuracy. Later on in this subsection, we discuss how the underestimations and overestimations introduced by sampling make it inappropriate for our problem.

Each element $t[i,j]$ in AA^T is the dot product of the two rows $A[i]$ and $A[j]$:

$$t[i,j] = \sum_{k=0}^{m-1} A[i,k] \times A^T[k,j] \tag{9}$$

Instead of performing the exact computation of $t[i,j]$, one can apply sampling to estimate the value. Theorem 1 shows that one can obtain an unbiased estimation of $t[i,j]$ by $A[i,k] \times B[k,j]$ where k is an index drawn uniformly at random in the range of $[0,m)$.

THEOREM 1. *Given two vectors α and β with the same size m, and s samples (S) of integers generated uniformly at random in the range of $[0,m)$, an unbiased estimation for $\alpha \cdot \beta$ is:*

$$\frac{m}{s} \sum_{k=0}^{s-1} \alpha[S[k]] \times \beta[S[k]] \tag{10}$$

PROOF. The proof is straightforward, following the definition $\alpha \cdot \beta = \sum_{k=0}^{m-1} \alpha[k] \times \beta[k]$. Let $\times_{\alpha\beta}$ be a vector of size m in which every element $\times_{\alpha\beta}[k] = \alpha[k] \times \beta[k]$. Then, $\alpha \cdot \beta$ can be rewritten as $\sum_{k=0}^{m-1} \times_{\alpha\beta}[k]$. Also, let $S[i]$ be a uniform random sample between 1 and m. Since $S[i]$ is sampled from the uniform distribution, $\times_{\alpha\beta}[S[i]]$ is an unbiased estimator for the average of $\times_{\alpha\beta}$, i.e., $\frac{1}{m}\sum_{k=0}^{m-1} \times_{\alpha\beta}[k]$. As a result, for a set of samples S:

$$E\Big[\sum_{i=0}^{s-1} \times_{\alpha\beta}[S[i]]\Big] = \frac{s}{m} \sum_{k=0}^{m-1} \times_{\alpha\beta}[k] = \frac{s}{m}\alpha \cdot \beta$$

Hence, $m/s \sum_{i=0}^{s-1} \times_{\alpha\beta}[S[i]]$ is an unbiased estimator for $\alpha \cdot \beta$. □

One can use Equation 10 to estimate the cell values $t[i,j]$ as $A[i] \cdot A[j]$ in $O(s)$. Since $s \ll m$, this approach is much faster than the exact computation of AA^T. Despite the speedup and the unbiased nature, this has a subtle problem that makes it inapplicable in practice. Recall that A is a sparse matrix, i.e., most of the its elements are zero. For the ease of explanation, let us assume that the probability of an element being non-zero is p (since A is sparse $p \to 0$). $t[i,j]$ is estimated more than zero, if there exist an index k in the random samples such that both $A[i,k]$ and $A[j,k]$ are non-zero. For a random index k, the probability that either $A[i,k]$ or $A[j,k]$ is zero is $1-p^2$. Thus, since the samples are drawn uniformly at random, the probability of $t[i,j]$ being estimated as non-zero is $1-(1-p^2)^s$. To have a better understanding of this probability, let us consider an example in which $m = 1000$, 1% of the elements of each row are non-zero, and 20 samples are generated. Thus, assuming that $p = 0.01$, the probability of element $t[i,j]$ being estimated as non-zero is $1-(1-10^{-4})^{20} < 0.002$, i.e., $t[i,j]$ is estimated as 0 with probability higher than 0.998. This problem escalates when wanting to use it in Line 3 of Algorithm 1. Underestimating the values of AA^T, in many cases in practice, makes the determinant of AA^T to be zero (if all the cells in a row or column of AA^T are estimated as zero, the determinant of it will be zero), which makes all values of $(AA^T)^{-1}$ in Equation 7 to be ∞! On the other hand, even in the lucky cases that the algorithm catches the places where both $A[i,k]$ and $A^T[k,j]$ are non-zero, the scaling factor in Line 10 results in overestimating the cell value. In fact, the sampling approach does not consider the special properties of A and the fact that it is multiplied to its transpose. Next, we study some important properties of the problem that results in improving the efficiency of DIRECT.

4.2 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 non-zero elements are highlighted in white. Figure 4 shows the non-zero 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 2 that could used to design an efficient algorithm to improve Algorithm 2.

THEOREM 2. *Given a sparse binary matrix A, considering the elements on the diagonal of AA^T, i.e., $t[i,i]$, $\forall 0 \le i < n$:*

- *$t[i,i] = |A[i]|$, where $|A[i]|$ is the number of non-zero 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, $\forall 0 \le j < n : t[i,j] \le t[i,i]$ and $t[i,j] \le t[j,j]$.*

PROOF. The proof lies on the fact that every element $t[i,i]$ is the dot product of the row $A[i]$ to itself. The dot product of a row $A[i]$ to itself is

$$\begin{aligned} t[i,i] &= A[i].A[i] \\ &= \sum_{k=0}^{m-1} A[i,k]A[i,k] \\ &= \sum_{\forall k \mid A[i,k]\neq 0} 1 \\ &= |A[i]| \end{aligned} \tag{11}$$

Moreover, since each row of A represents the coefficients of an equation, we assume there is at least one non-zero element in that row, i.e., $|A[i]| > 0$.

Now consider two rows $A[i]$ and $A[j]$ where $i \neq j$:

$$\begin{aligned} t[i,j] &= A[i].A[j] \\ &= \sum_{k=0}^{m-1} A[i,k]A[j,k] \\ &= \sum_{\forall k \mid A[i,k]\neq 0 \text{ and } A[j,k]\neq 0} 1 \\ &= |A[i] \wedge A[j]| \end{aligned} \tag{12}$$

Note that the operator $\wedge$ in Equation 12 denotes the binary *and* operation.

For any the binary vector $A[i]$, $|A[i] \wedge A[j]| \le |A[i]|$. Thus, $t[i,j] \le t[i,i]$. Similarly, $t[i,j] \le t[j,j]$. □

Consider two representations of AA^T of the example matrix given in Figure 3. Figure 4 shows all the non-zero elements of

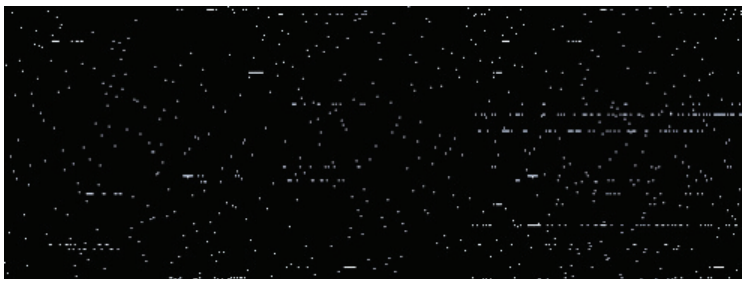


Figure 3: An example of the binary sparse matrix $A_{183\times495}$

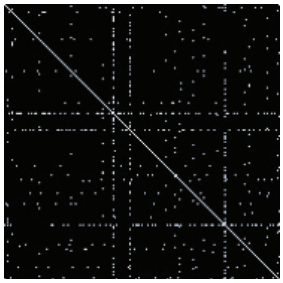


Figure 4: The non-zero elements in AA^T for the example of Figure 3

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

AA^T while 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 2. Furthermore, one may notice that most of the non-zero elements of AA^T (in Figure 4) are small values (in Figure 5). This shows that while there are a reasonable number of non-zero elements, the number of elements with higher magnitude is often much smaller. Next, we use this insight along with Theorem 2 for speeding up DIRECT.

4.3 Threshold Based Computation Of AA^T

In the previous subsection, we discussed the bound on the cell values in AA^T and showed 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 3 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 3 ApproxAA^T
Input: Sparse matrix A, Threshold τ
Output: t

```
1: F = {}
2: for i = 0 to n − 1 do
3:     t[i, i] = |A[i]|
4:     if |A[i]| ≥ τ then add i to F
5: end for
6: for every pair i, j ∈ F do
7:     t[i, j] = t[j, i] = SIM(A[i], A[j], τ)
8: end for
9: return t
```

4.4 Leveraging Similarity Joins for Oracle SIM

The database community has extensively studied mechanisms for computing set similarity for applications such as data cleaning [9] where the objective is to efficiently identify the set of tuples that are "close enough" on multiple attributes. In this subsection, we 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. It is possible to make an interesting connection between SIM and sets 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 non-zero 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. In this paper, we consider one exact approach and two approximate approaches based on threshold-based algorithms [9] and sketch-based methods [1, 10, 12, 22]. 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 non-zero columns in an ordered manner. Finding the intersection of ordered sets has been extensively studied [13, 39]. The simplest approaches 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, skip-lists. Each of these approaches allow one to "skip" some elements of a set when necessary.

Approximate Approach : Threshold based Algorithms. Threshold-based algorithms, such as [9] 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 non zero [9]. More precisely, for two sets U_i and U_j with size h, if $\cap_{i,j} \geq \tau$, any subsets $U_i' \subset U_i$ and $U_j' \subset U_j$ of size $h - \tau + 1$ will overlap; i.e., $|U_i' \cap U_j'| > 0$. Using this idea, while considering an ordering of the objects, the algorithm first finds the set of candidate pairs that overlap in a sub-

set of $h-\tau+1$. In the second step, the algorithm verifies the pairs, by removing the false positives.

One can see the effectiveness of this method highly depends on the value of τ and, considering the target application, it works well for the cases that 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 90, which is close to the entire set. The later 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 3. 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 dot product be not 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 [1, 10, 12, 22] use precomputed synopsis such as a minhash for answering different set aggregates such as Jaccard similarity. The main idea behind the min-hashing [5] based algorithms is as following: 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 min-hash, 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 $\frac{\cap_{i,j}}{\cup_{i,j}}$, i.e., Jaccard similarity of U_i and U_j. Bottom-k sketch [10], a variant of min-hashing 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 $|h_k(U_i) \cap h_k(U_j)|$. Bayer et al. [1] 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 k-th 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 k-th element in hash is. Using the results of [1], $\frac{m(k-1)}{h_{i,j}[k]}$ is an unbiased estimator for $\cup_{i,j}$. Hence the estimation for $\cap_{i,j}$ is as provided in Equation 13.

$$E[\cap_{i,j}] = \frac{k_\cap(i,j)}{k} \frac{m(k-1)}{h_{i,j}[k]} \tag{13}$$

Estimating $\cup_{i,j}$ with Equation 13, performs well when $\cup_{i,j} \gg 1$ [1], i.e., 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 are its bottom-k sketch. Using this strategy, Algorithm 4 shows the pseudo code of the oracle SIM.

Given two given sets U_i and U_j (corresponding for 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 check 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 algorithms 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 4 SIM
Input: the sets U_i and U_j, Threshold τ
Output: c

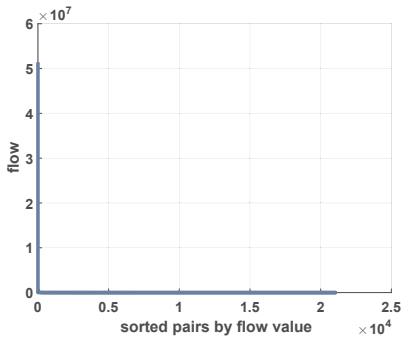
if $|U_i| \geq \log(m)$ and $|U_j| \geq \log(m)$ **then**
  `// apply bottom-k sketch based estimation`
  h_i = the first k elements in U_i
  h_j = the first k elements in U_j
  $k_\cap(i,j) = |h_i \cap h_j|$
  $h_{i,j}[k]$ = the first k elements in $h_i \cup h_j$
  $c = \frac{k_\cap(i,j)}{k} \frac{m(k-1)}{h_{i,j}[k]}$
else
  `// apply threshold-based estimation`
  $c = 0$
  if $|U_i| > |U_j|$ **then** swap U_i and U_j
  $\beta = |U_i| - \tau$
  for $k = 0$ to β **do**
    if $U_i[k] \in U_j$ **then** $c = c + 1$
  end for
  if $c = 0$ **then return** 0
  for $k = \beta$ to $|U_i| - 1$ **do**
    if $U_i[k] \in U_j$ **then** $c = c + 1$
  end for
end if
return c

Performance Analysis. Algorithm 3 has a time complexity of $O(n + \mu^2 \min(l, \log(m)))$, where $\mu = |\{A[i] | |A[i]| \geq \tau\}|$.

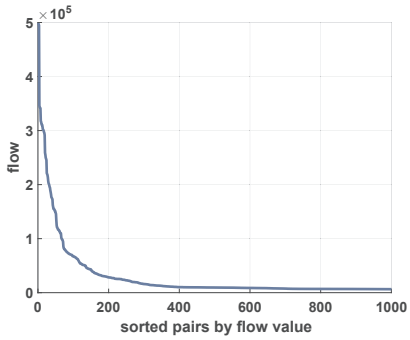
5. DISCUSSIONS

5.1 Identifying Top-k Components of Reconstructed Signal

A natural extension to the signal reconstruction problem is to identify the top-k components of the reconstructed signal. Of course, a naive approach would use DIRECT to compute the signal X and simply pick the top-k values. While top-k has been studied extensively in the DB community, there has been a paucity of work in the top-k signal reconstruction problem. There are a number of applications for quickly identifying the top-k results. Often, network traffic follows a pareto distribution with a small percent of source-destination pairs accounting for a large amount of flow between them. In contrast, the traffic between most of the source-destination pairs are quite small. Figure 6 shows the traffic flows in a network with 1,421 edges and 21,058 source-destination pairs. We can see that the "knee point" is around top 2%; i.e., 98% of the pairs have a small flow value. This observation could be used in the traffic matrix computation example to quickly identify the source-destination pairs with largest traffic. Note that this is different from the traditional framework of Heavy hitters where the objective is to identify the heavy hitters from a stream. In contrast, in order to identify the top-k source-destination pairs, one has to solve the SRP.



(a) flow of all pairs



(b) flow of top 1000 pairs

Figure 6: Illustration of the flow passing through a network N_3 in Table 1

One can adapt Algorithm DIRECT to compute top-k components of reconstructed signal. We achieve this by leveraging the existence of the prior X'. The intuition is that if the distribution of the values does follow a heavy tail distribution and that the prior is close to the answer space (which is often the case in practice), the top-k values of X should be within the top-αk of the prior with high probability where $\alpha > 1$. Therefore, rather than computing the value of all the variables, we only compute the values of the variables that are in the top-αk of the prior. Algorithm 5 shows the adaptation of DIRECT for computing the top-k variables. Applying Algorithm 3, the complexity of Algorithm 5 is $O(k + n^{2.807} + \mu^2 \min(l, \log(m)))$. Even though the worst case complexity involves $O(n^{2.807})$, it is often much smaller due to the sparse nature of matrix $t = AA^T$. Hence, Line 3 of Algorithm 5 is often significantly faster in practice.

Algorithm 5 Direct$_k$
Input: A, b, X', k, and α
Output: top-$k(X)$

1: $t = AA^T$
2: $t_2 = AX' - b$
3: solve: $t\,\xi = t_2$
4: $\mathcal{I}$ = the indices of top-$\alpha k(X')$
5: $X_k = X'[\mathcal{I}] - (A^T\xi)[\mathcal{I}]$
6: **return** X_k

5.2 Signal Reconstruction in Very Large Settings

Recall that in SRP often n is a low dimensional vector with $n \ll m$. In this subsection we briefly describe how to extend DIRECT to handle cases where even n is very large (and still $n \ll m$). For example, let n be 10^6 and m be 10^{12}. 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. To see why, even if there is only one non-zero value in *every column*, then we $O(m)$ storage to even represent this matrix.

Interestingly, the similarity-joins based techniques proposed in § 4 do not require to completely materialize even sparse representation of A for estimating AA^T. Also recall that as discussed in § 5.1, 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 4 applies bottom-k sketch for the sets that their size is more than $\log m$. Thus, choosing the signature size in the bottom-k sketch to be in $O(\log m)$, Algorithm 4 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. Also, since AA^T is a sparse matrix, we only store the non-zero values of matrix t, rather than the complete n by n matrix. Line 2 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 non-zero 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 non-zero elements of AA^T and t' is a n by 1 vector, and finding the n by 1 vector ξ is doable, using methods like Gauss-Jordan. Finally, similar to Algorithm 5, 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. EXPERIMENTAL EVALUATION

6.1 Experimental Setup

Hardware and Platform. All our experiments were performed on a Macintosh machine with a 2.6 GHz CPU and 8GB memory. The algorithms were implemented using Python2.7 and Matlab.

Datasets. We conducted our experiments using both real-world and synthetic datasets to demonstrate the efficacy of our algorithms over graphs with diverse values for 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. The size of SRP that we tackle are 2-3 orders of magnitude larger than prior work such as [44].

The real datasets were derived from a p2p dataset from SNAP repository of Stanford university[4]. The p2p dataset is a snapshot of the Gnutella network in August 2002 with 10876 nodes and 39994 edges. Nodes represent the hosts and the links represent the connection between the hosts. We generated three different datasets from the p2p datasets with increasing number of edges and source-destination pairs. Each of the derived datasets is a subgraph of the

[4]SNAP Dataset: https://snap.stanford.edu/data/p2p-Gnutella04.html

Table 1: Dataset Characteristics

Network	#Nodes	#Edges	#Source-Destination pairs
N_1	274	281	827
N_2	1123	1278	9330
N_3	1231	1421	21058
p2p-1	369	4549	136K
p2p-2	612	3486	373K
p2p-3	1438	7081	2M

overall p2p graph and was obtained by Forest Fire model [29, 30]. The characteristics of each of these datasets dubbed *p2p-1*, *p2p-2* and *p2p-3* can be found in Table 1. The synthetic datasets were constructed as a random, Erdős-Rényi graph [15] by varying number of nodes with edge formation probability of 0.5. Disconnected components were discarded from further calculations. The characteristics of synthetic networks, N_1, N_2 and N_3, can also be found in Table 1.

Constructing Traffic Matrices. Once we sample the network and obtain a connected graph, we consider all possible source destination pairs, i.e., $\#\text{nodes} \times (\#\text{nodes} - 1)$, to be as individual flows. For each source-destination pair we calculated the shortest path between them (network policies are not considered here as our algorithm is oblivious of the route chosen). Traffic matrix is a collection of all such routes in the following manner, each of the rows corresponds to an edge used in routing and each of the columns corresponds to a source-destination pair. Every cell, $c[i, j]$ is a '1' if edge$[i]$ is involved in routing traffic for source-destination$[j]$ else is assigned a value '0'. A visual glimpse of the routing matrix is given in Figure 2.

We used a *Pareto* traffic generation model, a popular stochastic model of the traffic flows for generating self-similar traffic observed in network communication [8, 20].
The distribution is parametrized by a scale parameter x_m (set to 20) and a shape parameter α (set to 1). x_m is the minimum value of the distribution of traffic represented by the scale parameter while the shape parameter α indicates the 'steepness of the slope' of the distribution curve. The *prior* to the experiments (X') was obtained as a function of gravity model from [43].

6.2 Algorithms Evaluated

We evaluated the two algorithms described in our paper for SRP against two representative baselines. The first is an exact algorithm DIRECT that solves SRP using the Lagrangian dual form described in Section 3. The Matlab implementation of DIRECT is provided in Figure 7. The second is an approximate algorithm that speeds up DIRECT using techniques from similarity joins for computing AA^T that is described in Section 4.

Our first baseline formulates the SRP as a quadratic programming problem and uses existing software packages. We refer to this baseline as QP. The Matlab implementation of QP is provided in Figure 8. As our second baseline, we consider the method proposed in [43]. This is an approximate method that generates weighted least-squares estimate of the tomo-gravity (WLSE). It works in two

```
function [x] = direct(A,b,xp)
    tmp = A*A';
    tmp2 = A*xp - b;
    % compute inv(A*A') * (A*xp - b):
    tmp = tmp \ tmp2;
    x = xp - A'*tmp;
end;
```

Figure 7: Matlab implementation of DIRECT

```
function [t,f] = QP( A,b,xp,maxiter)
    m = size(A,2);
    % Define QP parameters
    H = eye(m);
    options = optimset('Algorithm',
        'interior-point-convex',
        'MaxIter',maxiter);
    % Before R2011a:
    %  options = optimset('Algorithm',
    %     'interior-point','MaxIter',
    %     maxiter);

    % Construct the QP, invoke solver
    [t,f] = quadprog(H,-xp,[],[],A,b,[],[],
        [],options);
end;
```

Figure 8: QP implementation in Matlab

stages by initially computing a fast approximate (but not necessarily feasible) solution. For example, this can be done by setting the prior X' as the initial solution. However, this solution will result in some unresolved residue, $b' = b - AX'$. WLSE then solves the smaller problem $AX'' = b'$ by computing the pseudo-inverse of A, using MPP, which as explained in § 7 uses singular value decomposition. Figure 9 shows the Matlab implementation of WLSE [43]. Note that both these baselines are extensively used in solving SRP. In additions to the baselines, we also evaluated the performance of applying compressive sensing [34] for our smallest network setting, i.e., N_1.

6.3 Experimental Results

We evaluate the performance of our algorithms against two measures - time and accuracy. Our algorithm DIRECT is an exact algorithm and we show that it provides significant speedup over prior methods. In fact, it even allows us to solve problem instances that are too large for the baselines. Next, we compare our approximate method against the solution provided by DIRECT and show that it is able to achieve solutions with small error and at a fraction of time of DIRECT.

6.3.1 *Performance Improvement of* DIRECT *over Baseline Algorithms*

In our first series of algorithms, we compare the exact algorithm DIRECT with the exact baseline QP and the approximate baseline WLSE. The evaluation was conducted over the synthetic networks N_1, N_2, N_3. We chose these smaller datasets as the baseline methods failed for larger problem instances. Note that both DIRECT and QP generate the optimal solution and thereby compare the runtime

```
function [x] = wlse(A,b,xp)
    % equivalently transform b=A*x into
    bw = b - A*xp;
    % solve xw=A*bw by computing the
    % pseudo-inverse of A (through svd)
    xw = pinv(full(A)) * bw;
    % transform tw back to t
    x = xp + xw;
end;
```

Figure 9: WLSE implementation in Matlab [43]

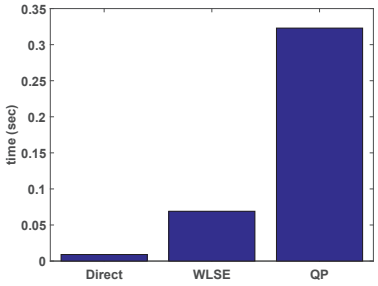


Figure 10: DIRECT **v.s. baselines in** $N_1 : n = 281$ **and** $m = 827$

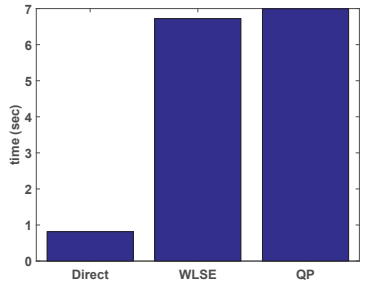


Figure 11: DIRECT **v.s. baselines in** $N_2 : n = 1,278$ **and** $m = 9,330$

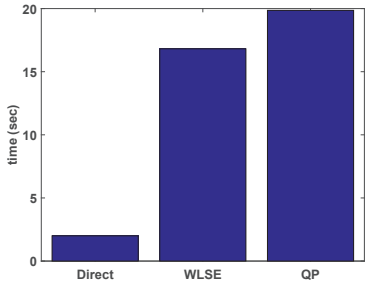


Figure 12: DIRECT **v.s. baselines in** $N_3 : n = 1,421$ **and** $m = 21,058$

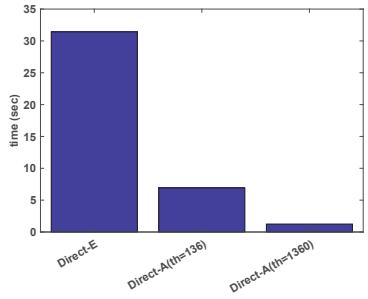


Figure 13: Execution time of DIRECT-E**,** DIRECT-A (τ=136)**, and** DIRECT-A (τ=1360) **in p2p-1**

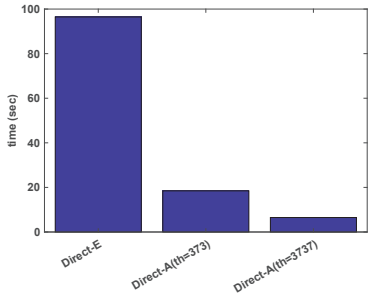


Figure 14: Execution time of DIRECT-E**,** DIRECT-A (τ=373)**, and** DIRECT-A (τ=3737) **in p2p-2**

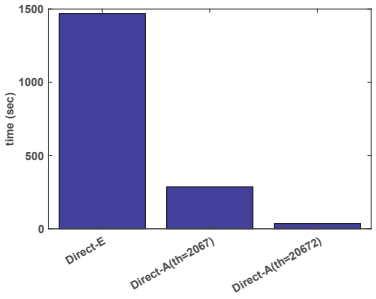


Figure 15: Execution time of DIRECT-E**,** DIRECT-A (τ=2067)**, and** DIRECT-A (τ=20672) **in p2p-3**

performance of the algorithms. Figures 10-12 show the results for networks N_1, N_2 and N_3 respectively.

We can see that as expected `WLSE` outperforms `QP` as the former is an approximate method. However, its efficacy decreases as the network size increases. DIRECT outperforms both the baselines for all the networks and often by a factor of 10-20. Note that while the performance of `WLSE` is adequate for near real-time performance for smaller graphs, the performance becomes unacceptable for larger graphs. On the other hand, DIRECT produces results efficiently for both small and large graphs.

In addition to comparing with the baselines, for the smallest network setting (N_1) we also used compressive sensing [34] for estimating the values of the source-destination pairs. Since the objective in compressive sensing is the expensive ℓ_0-optimization, even for our smallest setting it took 23.414 seconds. Not only significantly slower than the both baselines, compressive sensing also adds a large error in the estimation of flow values, especially for the variable with larger flows. In this experiment while the average flow for top-2% of the variables is 1511.2, its average error for these variables is 494.9. Moreover, since compressive sensing tends to minimize the number of non-zero values, it underestimates a lot of variables with large flows as zero. For example it estimated the variables with the true flows of 1741, 1094, 1001 and 889 in the top-2% as zero. That is, it estimated 25% of the top-2% variables as zero.

6.3.2 *Effectiveness of Similarity Join based approach*

Having shown the superiority of DIRECT over the baselines, we now evaluate the exact version of DIRECT (Algorithm 2) and its approximate counterpart (Algorithm 3) 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. Note that our algorithms take advantage of the sparse representation of matrix A and can perform the linear algebraic operations without materializing the entire matrix. We also evaluate the performance of our algorithms to two different threshold values of $(m/1000)$ and $(m/100)$, where m is 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 two metrics : performance and accuracy. We measure the former through execution time. We measure the accuracy of the signal reconstruction through *bucketized error* where we bucketize the source-destination pairs by the exact value of their flows and compute the error of the approximation algorithm within each bucket. The bucketization is often more illuminating for scenarios such as network traffic engineering where the signal exhibits a heavy tailed distribution and often the practitioner is interested in accurately estimating large flows. After finding the optimal flow assignments using the algorithm DIRECT-E, we sort the source-destination pairs in descending order, based on the amount of flow passing through them. For example, let a flow assignment by DIRECT-E be $\{(SD_1 : 3), (SD_2 : 24), (SD_3 : 7), (SD_4 : 75), (SD_5 : 5), (SD_6 : 12)\}$. The sorted SD pairs are $\{(SD_4 : 75), (SD_2 : 24), (SD_6 : 12), (SD_3 : 7), (SD_5 : 5), (SD_1 : 3)\}$. We then partition the SD pairs into 50 equal size buckets (each bucket contains 2% of SD pairs[5]). In the provided example, assume that we partition them into 3 buckets $B_1 : \{(SD_4 : 75), (SD_2 : 24)\}$, $B_2 : \{(SD_6 : 12), (SD_3 : 7)\}$, and $B_3 : \{(SD_5 : 5), (SD_1 : 3)\}$. For every SD pair, we consider the difference between the values computed by DIRECT-A and the

[5]We have found out the knee point of the cumulative flow is around 2%.

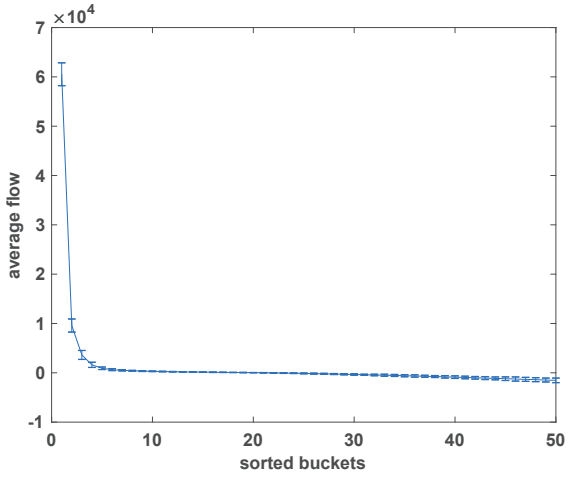


Figure 16: Absolute Error of the DIRECT-A ($\tau = 1360$) in p2p-1

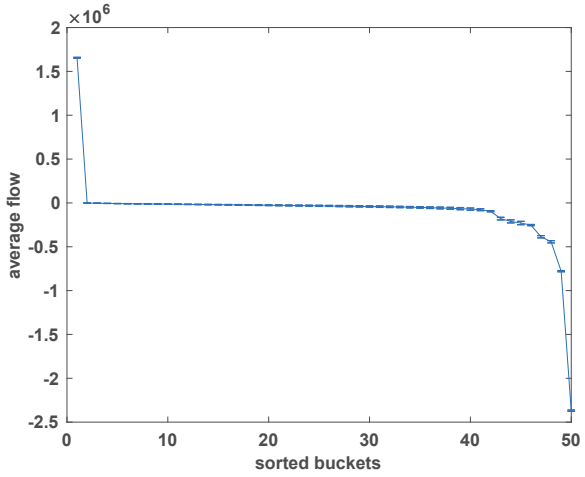


Figure 17: Absolute Error of the DIRECT-A ($\tau = 3737$) in p2p-2

one by DIRECT-E as the error of that SD pair, and compute the average for each bucket. In our example, let $\{(SD_1 : 5), (SD_2 : 24), (SD_3 : 6), (SD_4 : 79), (SD_5 : 5), (SD_6 : 11)\}$ be the assigned values by DIRECT-A. Then the average errors for the buckets B_1, B_2, and B_3 are 2, 1, and 1, respectively. It was observed in [16, 43] that for many tasks in network traffic engineering such as routing optimization, even a relative error of *few* 10s of percent is considered tolerable. As we shall show later, our algorithms often achieve substantially lower errors while providing results in a handful of seconds.

p2p-1 (136K Source-Destination pairs) Figure 13 shows the comparative performance of the exact and approximate version of DIRECT. DIRECT-E takes almost 30 seconds in computing the exact solution while DIRECT-A with threshold = 1360 was able to produce answers in less than 3 seconds. Furthermore, we observed that running DIRECT-A for 20 seconds provides the same solution as DIRECT-E. Figure 16 shows the quality of solution provided by DIRECT-A. We can see that the solution provided by DIRECT-A is very close to that of DIRECT-E even though the former provided a 90% time savings for a mid sized network. As expected, increasing the threshold results in a significant speedup.

p2p-2 (373K Source-Destination pairs) This network has 373K source-destination pairs with 3486 edges sampled form the *SNAP p2p* dataset. This network is twice as big as the previous network. While DIRECT-E takes about 90 seconds for computing the exact solution, our approximate algorithm with threshold=3737 computes the result within 5 seconds. Figure 14 shows the performance gain is much as 90%. Furthermore, the execution time of this algorithm is fast enough to be interactive even for large enough networks. Figure 17 shows that the improved performance did not result in a large error. Instead, the bucketized error is quite small.

p2p-3 (2M Source-Destination pairs) This network has 2M source-destination pairs with 7081 edges sampled form the *SNAP p2p* dataset. Figure 15 we can see that DIRECT-E takes much as 1500 seconds 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 seconds. This is a significant reduction in execution time with a speedup of much as 97% of the running time of DIRECT-E. Figure 18 shows that the results are very close to the exact answer produced by DIRECT-E.

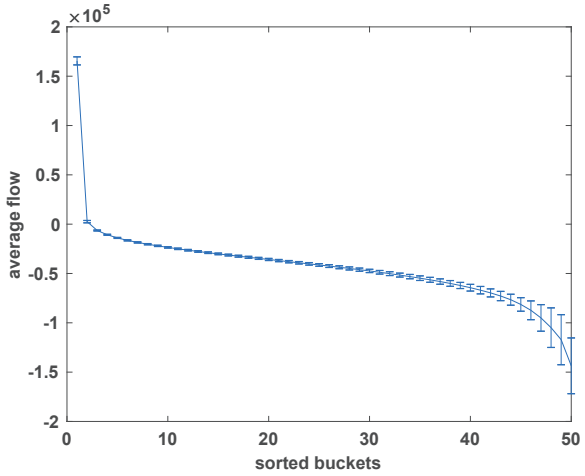


Figure 18: Absolute Error of the DIRECT-A ($\tau = 20672$) in p2p-3

7. RELATED WORK

Traffic Matrix Computation from Link Loads: The problem of inference of traffic matrices in IP networks from link load measurements and routing configuration information has been extensively studied [40, 43]. See [32] for a survey of commonly used techniques. These include formulating SRP as a linear or quadratic programming problem [18], using Bayesian inference techniques [37], statistical likelihood methods such as Expectation-Maximization [7] weighted least squares [43], regularization based entropy penalization [44] using convex optimization theory etc. Note that one cannot use Compressive Sensing (CS) [6] to solve SRP due to two reasons. Compressive Sensing efficiently approximates the ℓ_0 loss function by ℓ_1 loss function when one has the prior knowledge that the signal is sparse. However, in many application scenarios one often has even more additional knowledge such as the prior x'. Incorporating an arbitrary (possibly non sparse) prior into CS is non-trivial. None of these methods can scale for large communication networks and provide results in a near interactive fashion.

Linear Algebraic Techniques for Solving SRP: There has been extensive work on solving the system of linear equations using a wide variety of techniques such as computing the pseudoinverse

of A [41], performing Singular Value Decomposition (SVD) on A [11], and iterative algorithms for solving the linear system [41]. However, none of these methods scale for large-scale IP networks. A key bottleneck in these approaches is often the computation of the pseudo inverse for matrix A. Note that any matrix B such that $ABA = A$ is defined as a pseudo inverse for A. It is possible to identify "the infinitely many possible generalized inverses" [14], each with its own advantages and disadvantages. Moore-Penrose Pseudo inverse (MPP) [2,33,36] is one of the most well-known and widely used pseudo inverse. MPP is the pseudo inverse that has the smallest Frobenius norm, minimizes the least-square fit in over-determined systems, and finds the shortest solution in the under-determined ones. However, none of the pseudo-inverse definitions suits our purpose of finding the solution X that minimizes the ℓ_2 distance from a prior. Furthermore, computing pseudo inverses is often done by SVD [38] that is computationally very expensive.

Applications of SRP: There are very many applications of SRP in diverse domains. In addition to traffic matrix computation, it can also be used to perform traffic analysis attack that can be used to infer revealing information about the users of a P2P network [19]. For example, similar to traffic matrix computation setting, one can identify the amount of traffic flow between any pair of users in a P2P network from the link load information and the routing algorithm of the P2P application. This could also reveal information such as which hosts use P2P applications or even the amount of P2P traffic between users. Another popular application is tomographic reconstruction (TR) which is a multi-dimensional linear inverse problem with wide range of applications in medical imaging [21, 24] such as CT scans (computed tomography). Informally, a CT scan takes multiple 2D projections (b) through X-rays from different angles (A) and the objective is to reconstruct the 3D image from the various 2D projections. Note that many 3D images may produce the same projections necessitating the use of priors to choose an appropriate reconstruction. Other applications include remote sensing in astronomy [11], reconstruction of acoustic sources [26], etc.

8. 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 number of applications in diverse domains including network traffic engineering, astronomy, medical imaging etc. 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.

9. ACKNOWLEDGMENTS

The work of Abolfazl Asudeh, Azade Nazi, Jees Augustine, and Gautam Das was supported in part by AT&T, the National Science Foundation under grant 1343976, and the Army Research Office under grant W911NF-15-1-0020. Nan Zhang was supported in part by the National Science Foundation, including under grants 1343976, 1443858, 1624074, 1760059, and by the Army Research Office under grant W911NF-15-1-0020.

10. REFERENCES

[1] K. Beyer, R. Gemulla, P. J. Haas, B. Reinwald, and Y. Sismanis. Distinct-value synopses for multiset operations. *Communications of the ACM*, 52(10):87–95, 2009.

[2] A. Bjerhammar. *Application of calculus of matrices to method of least squares: with special reference to geodetic calculations*. Elander, 1951.

[3] M. Boehm, M. W. Dusenberry, D. Eriksson, A. V. Evfimievski, F. M. Manshadi, N. Pansare, B. Reinwald, F. R. Reiss, P. Sen, A. C. Surve, et al. Systemml: Declarative machine learning on spark. *PVLDB*, 9(13):1425–1436, 2016.

[4] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[5] A. Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[6] E. J. Candes, J. K. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on pure and applied mathematics*, 59(8):1207–1223, 2006.

[7] J. Cao, D. Davis, S. Vander Wiel, and B. Yu. Time-varying network tomography: router link data. *Journal of the American statistical association*, 95(452):1063–1075, 2000.

[8] B. Chandrasekaran. Survey of network traffic models. *Waschington University in St. Louis CSE*, 567, 2009.

[9] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*. IEEE, 2006.

[10] E. Cohen and H. Kaplan. Tighter estimation using bottom k sketches. *PVLDB*, 1(1):213–224, 2008.

[11] I. J. Craig and J. C. Brown. Inverse problems in astronomy: a guide to inversion strategies for remotely sensed data. *Research supported by SERC. Bristol, England and Boston, MA, Adam Hilger, Ltd., 1986, 159 p.*, 1986.

[12] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 240–251. ACM, 2002.

[13] B. Ding and A. C. König. Fast set intersection in memory. *PVLDB*, 4(4):255–266, 2011.

[14] I. Dokmanić and R. Gribonval. Beyond moore-penrose part ii: The sparse pseudoinverse. *arXiv:1706.08701*, 2017.

[15] P. Erdos and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[16] B. Fortz and M. Thorup. Optimizing ospf/is-is weights in a changing world. *IEEE journal on selected areas in communications*, 20(4):756–767, 2002.

[17] D. Ge, X. Jiang, and Y. Ye. A note on the complexity of l p minimization. *Mathematical programming*, 129(2), 2011.

[18] O. Goldschmidt. Isp backbone traffic inference methods to support traffic engineering. In *Internet Statistics and Metrics Analysis (ISMA) Workshop*, pages 1063–1075, 2000.

[19] Y. Gong. Identifying p2p users using traffic analysis. 2005. `www.symantec.com/connect/articles/identifying-p2p-users-using-traffic-analysis`.

[20] J. Gordon. Pareto process as a model of self-similar packet traffic. In *Global Telecommunications Conference, 1995. GLOBECOM'95., IEEE*, volume 3, pages 2232–2236. IEEE, 1995.

[21] P. Grangeat and J.-L. Amans. *Three-dimensional image reconstruction in radiology and nuclear medicine*, volume 4. Springer Science & Business Media, 2013.

[22] M. Hadjieleftheriou, X. Yu, N. Koudas, and D. Srivastava. Hashed samples: selectivity estimators for set similarity selection queries. *PVLDB*, 1(1):201–212, 2008.

[23] P. C. Hansen. *Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion*. SIAM, 1998.

[24] W. T. Hrinivich, D. A. Hoover, K. Surry, C. Edirisinghe, D. D'Souza, A. Fenster, and E. Wong. Ultrasound guided high-dose-rate prostate brachytherapy: Live needle segmentation and 3d image reconstruction using the sagittal transducer. *Brachytherapy*, 15:S195, 2016.

[25] Z. Kaoudi, J.-A. Quiané-Ruiz, S. Thirumuruganathan, S. Chawla, and D. Agrawal. A cost-based optimizer for gradient descent optimization. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 977–992. ACM, 2017.

[26] Y. Kim and P. Nelson. Optimal regularisation for acoustic source reconstruction by inverse methods. *Journal of sound and vibration*, 275(3):463–487, 2004.

[27] A. Kumar, J. Naughton, J. M. Patel, and X. Zhu. To join or not to join?: Thinking twice about joins before feature selection. In *Proceedings of the 2016 International Conference on Management of Data*, pages 19–34. ACM, 2016.

[28] J. L. Lagrange. *Mécanique analytique*, volume 1. Mallet-Bachelier, 1853.

[29] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636. ACM, 2006.

[30] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.

[31] B. McMahan and D. Ramage. Federated learning: Collaborative machine learning without centralized training data. Technical report, Technical report, Google, 2017.

[32] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: Existing techniques and new directions. *ACM SIGCOMM Computer Communication Review*, 32(4):161–174, 2002.

[33] E. Moors. On the reciprocal of the general algebraic matrix, abstract. *Bull. Amer. Math. Soc.*, 26:394–395, 1920.

[34] D. Needell and J. A. Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3), 2009.

[35] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16(3), 2014.

[36] R. Penrose. A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, volume 51, pages 406–413. Cambridge University Press, 1955.

[37] C. Tebaldi and M. West. Bayesian inference on network traffic using link count data. *Journal of the American Statistical Association*, 93(442):557–573, 1998.

[38] L. N. Trefethen and D. Bau III. Numerical linear algebra. philadelphia: Society for industrial and applied mathematics. Technical report, ISBN 978-0-89871-361-9, 1997.

[39] D. Tsirogiannis, S. Guha, and N. Koudas. Improving the performance of list intersection. *PVLDB*, 2(1):838–849, 2009.

[40] P. Tune and M. Roughan. Maximum entropy traffic matrix synthesis. *ACM SIGMETRICS Performance Evaluation Review*, 42(2):43–45, 2014.

[41] C. R. Vogel. *Computational methods for inverse problems*. SIAM, 2002.

[42] C. Zhang, A. Kumar, and C. Ré. Materialization optimizations for feature selection workloads. *ACM Transactions on Database Systems (TODS)*, 41(1):2, 2016.

[43] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast accurate computation of large-scale ip traffic matrices from link loads. In *ACM SIGMETRICS Performance Evaluation Review*, volume 31, pages 206–217. ACM, 2003.

[44] Y. Zhang, M. Roughan, C. Lund, and D. Donoho. An information-theoretic approach to traffic matrix estimation. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 301–312. ACM, 2003.