# RankMap: A Framework for Distributed Learning From Dense Data Sets

Azalia Mirhoseini, Eva L. Dyer, Ebrahim M. Songhori, Richard Baraniuk, *Fellow, IEEE*, and Farinaz Koushanfar

*Abstract*—This paper introduces RankMap, a platform-aware end-to-end framework for efficient execution of a broad class of iterative learning algorithms for massive and dense data sets. Our framework exploits data structure to scalably factorize it into an ensemble of lower rank subspaces. The factorization creates sparse low-dimensional representations of the data, a property which is leveraged to devise effective mapping and scheduling of iterative learning algorithms on the distributed computing machines. We provide two APIs, one matrix-based and one graph-based, which facilitate automated adoption of the framework for performing several contemporary learning applications. To demonstrate the utility of RankMap, we solve sparse recovery and power iteration problems on various real-world data sets with up to 1.8 billion nonzeros. Our evaluations are performed on Amazon EC2 and IBM iDataPlex servers using up to 244 cores. The results demonstrate up to two orders of magnitude improvements in memory usage, execution speed, and bandwidth compared with the best reported prior work, while achieving the same level of learning accuracy.

*Index Terms*—Dense and big data, iterative machine learning (ML), large-scale distributed computing, sparse matrix factorization, low rank approximation, union of subspaces.

## I. INTRODUCTION

**M**ANY modern learning algorithms are based on exploring the underlying patterns, correlations, and dependencies present across the signals in the data set. Some prominent examples of such algorithms and their applications are linear or penalized regression [29], power iterations [30], belief propagation [48], and expectation maximization [38], [39]. In all of these settings, solving the underlying objective function requires iterative updates of parameters of interest until convergence is achieved. Such iterative updates often require matrix multiplications that involve the data dependence or Gram matrix. In scenarios where data are too large to fit on a single computing node and must be distributed, iterative dependence-based updates become challenging as they incur large computation and communication costs.

A. Mirhoseini, E. M. Songhori, and R. Baraniuk are with the Department of Electrical and Computer Engineering, Rice University, Houston, TX 77005 USA (e-mail: azalia@google.com; ebrahim@rice.edu; richb@rice.edu).

E. L. Dyer is with the Department of Physical Medicine and Rehabilitation, Northwestern University, Chicago, IL 60611 USA (e-mail: evaLdyer@gmail.com).

F. Koushanfar is with the Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093 USA (e-mail: farinaz@ucsd.edu).

To facilitate parallel computing, a number of distributed abstractions that target iterative learning algorithms have been developed, e.g., Pregel [35], Spark [50], and GraphLab [34]. These abstractions adopt a graph-parallel model, which consists of a *sparse graph* and a kernel function that runs in parallel on each vertex [25]. Performance gains are achieved due to the communication-minimizing partitioning of the graph and effective control of data movement.

While graph-parallelism has been shown to accelerate machine learning (ML) and signal processing tasks for sparse graphs, this approach cannot be readily applied when the data exhibit a *nonsparse dependence matrix*. The storage of such data in a graph format becomes very inefficient as it requires storing a large number of edges (pairwise nonzero correlation values) for each vertex (data sample). In addition, finding efficient graph cuts and partitions is infeasible when dense dependencies exist. Data with dense dependencies appear in a wide range of fields, such as computer vision, medical image processing, boundary element methods and their applications, and N-body problems [8], [26]. Thus, finding efficient solutions for running iterative learning algorithms on densely dependent data is of paramount importance.

In this paper, we introduce RankMap, a novel distributed framework for efficient execution of a broad class of iterative learning algorithms on data sets with dense but structured dependencies. Our key observation is that, despite the apparent high dimensionality of data, in many settings, dense data sets are low rank or lie on a union of much lower dimensional subspaces. We exploit this property to reduce the overhead associated with processing dense data dependencies—a factor which has rendered the currently available graph-parallel abstractions impractical for processing dense data sets. RankMap provides a set of interfaces and transformations that enable efficient data-aware content analysis, as well as coordinated mapping and optimization to the specifics of the underlying hardware components. RankMap significantly improves the runtime and energy consumption of the learning algorithms by reducing the amount of required computation, distributed system communication, and storage.

To accelerate large matrix multiplications required to compute an iterative update, we decompose dense but structured data and rewrite it as a product of two matrices with far fewer nonzeros than the original data. The decomposed data are then used in subsequent iterative learning algorithms in lieu of the original dense data. We introduce a host of automated methods for partitioning the decomposed factors and ordering the computation flow in a distributed setting. The partitioning algorithm is efficient (within a bound from the optimum) and has a constant runtime. We introduce

two different representations and accompanying computational models (a matrix-based and a vertex-centric model) to compute an update. Depending on the data domain and the sparsity of the decomposed components, there are different regimes where each of these two models delivers highest efficiency.

We provide APIs for both matrix-based and vertex-centric iterative update models on the transformed data. Our APIs are open-source and available at [3]. Our matrix-based API uses the general message passing interface (MPI). Our vertex-centric API is based upon the GraphLab programming model. We develop an efficient mapping of the iterative computations on the sparsified decomposed data within the constraints of the GraphLab distributed framework. Both APIs are written in C++. We evaluate RankMap on the Amazon Elastic Cloud (EC2) computing service and IBM iDataPlex computer cluster. Our experiments utilize up to 244 cores on 12 large computing nodes.

Our explicit contributions are as follows.
1) We propose RankMap, a large-scale learning framework that proposes sparse transformations for accelerating iterative learning algorithms on dense but structured data.
2) We introduce a scalable transformation, which maps structured (low rank) data onto two matrices, which contain far fewer number of nonzeros. A systematic way to tune the transformation error to achieve a desired level of accuracy in the learning applications is provided.
3) We develop efficient distributed computational models to conduct iterative updates on the decomposed data. Highly effective partitioning methods for the decomposed data along with data-aware performance bounds are provided.
4) We perform proof-of-concept evaluation on applications, including eigenvalue decomposition, denoising, and classification that demonstrate up to two orders of magnitude improvement in runtime and memory footprint.

This paper is organized as follows. In Section II, we provide a global overview of RankMap. In Section III, we review related work. In Section IV, we introduce our novel data transformation algorithm. In Section V, we study the impact of the decomposition error on learning and provide a method for automatically tuning RankMap to produce a user-specified learning error. In Section VI, we introduce schemes for cost-efficient distributed partitioning, along with the details of our graph and matrix-based computational models. In Section VII, we provide evaluation results on multiple synthetic and real-world data sets. Finally, in Section VIII, we discuss the practicality of our framework, describe domain-specific use-cases of each of the proposed computational models, and conclude.

## II. RankMap Framework

### A. Overview and Approach

In this paper, we introduce RankMap, a distributed data-aware framework that efficiently executes learning algorithms applied to the data. The main idea underlying our
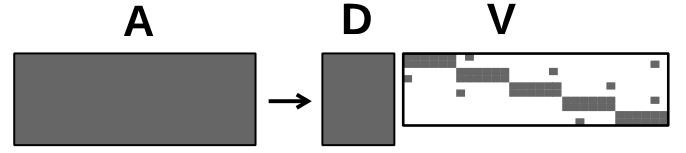


Fig. 1. *Schematic of decomposing a dense data matrix into the product of a small dense matrix and a large sparse matrix.*

approach is to leverage structure in large collections of data to decompose the correlation (Gram) matrix of the data, such that the system costs (e.g., runtime, memory, and energy) associated with iterative learning algorithms are significantly reduced.

Let the data matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ denote a collection of $n$ signals of $m$-dimensions, and $\mathbf{G} = \mathbf{A}^T \mathbf{A}$ denote the Gram matrix. Many learning algorithms iteratively update a solution vector, denoted by $\mathbf{x}$, according to an update function of the following form:

$$\mathbf{x}^{\text{iter}+1} = f(\mathbf{G}\mathbf{x}^{\text{iter}}) \tag{1}$$

where $f(\cdot)$ is a low-complexity function and iter is the current iteration. Examples are included in Section II-B.

When $\mathbf{G}$ is massive and dense, each distributed update in (1) becomes very expensive. To cope with large data sizes, RankMap creates an approximation to $\mathbf{G}$, denoted by $\widehat{\mathbf{G}}$, to reduce the cost of an update. To be more specific, our aim is to decompose the data matrix $\mathbf{A}$ into two components, i.e., $\widehat{\mathbf{A}} = \mathbf{DV}$, where $\mathbf{D} \in \mathbb{R}^{m \times l}$ contains a subset of columns from $\mathbf{A}$, $\mathbf{V} \in \mathbb{R}^{l \times n}$ is a sparse matrix, and rank$(\mathbf{A}) \leq l \ll n$ (see Fig. 1). After decomposing $\mathbf{A}$, we then efficiently partition the decomposed data and perform distributed updates using $\widehat{\mathbf{G}} = (\mathbf{DV})^T(\mathbf{DV})$. Mapping the original dense data to a decomposed model directly reduces the memory usage, and the costly computational operations and communication incurred by the iterative updates.

When $\mathbf{A}$ is low rank, it is possible to construct a reduced decomposition that is exact ($\mathbf{A} = \mathbf{DV}$). However, we demonstrate that for many real-world data sets, we can achieve significant performance improvements in exchange for a small decomposition error ($\mathbf{A} \approx \mathbf{DV}$). We discuss the connection between the decomposition error and the accuracy of a target learning method as well as strategies for tuning the decomposition to achieve a desired level of accuracy in the iterative learning algorithms.

RankMap consists of three main components (see Fig. 2): 1) a scalable data decomposition that shrinks the size of the data set by leveraging the data's structure; 2) a data partitioning scheme along with an execution flow for performing iterative updates on the decomposed data that significantly reduces the distributed computing costs; and 3) a systematic method for tuning the decomposition error (denoted by $\delta_D$) to achieve the desired level of approximation error in the learning algorithms (denoted by $\delta_L$).

### B. Target Applications

Our framework can be used for a broad class of optimization problems that are solved via iterative updates based upon the
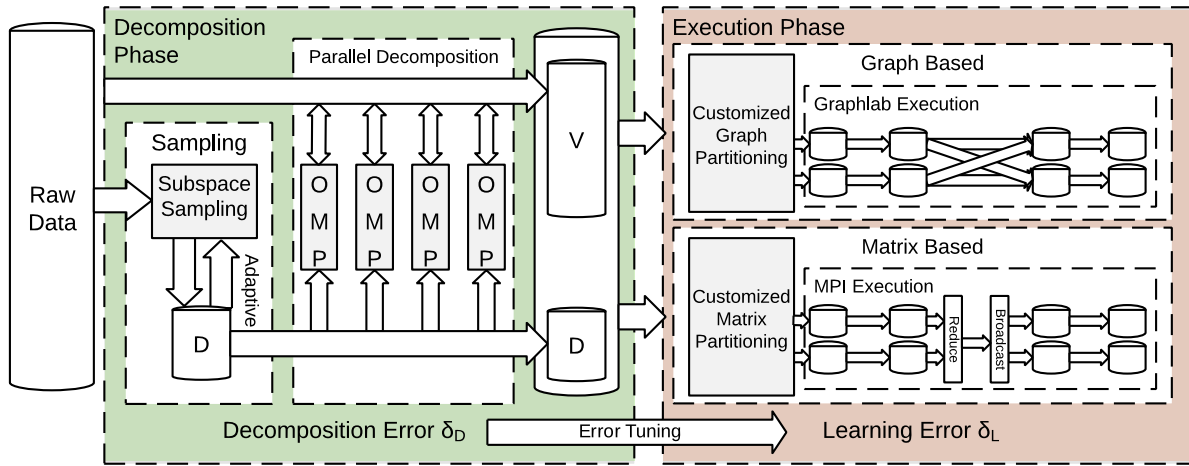
Fig. 2. *Overview of RankMap framework.* The method is divided into two main phases, the decomposition phase (left) and the execution phase (right). To execute iterative updates, we provide two computational models: a matrix-based model (implemented in MPI) and a graph-based model (implemented in GraphLab).

Gram matrix. A large number of objective functions used in ML, e.g., penalized regression methods, such as the LASSO or BPDN [9] and ridge regression [29], are typically solved using iterative updates. In all these settings, the complexity of executing these methods is dominated by costly iterative computation on the Gram matrix of the data set.

To ground RankMap in real-world problems, we now discuss two particular learning algorithms that are evaluated in this paper: 1) sparse approximation and 2) the power method for eigenvalue decomposition.

*1) Sparse Approximation for Image Denoising and Classification:* Sparse representation is used in a wide range of signal processing and ML applications, including denoising [9], classification [47], clustering [18], and outlier detection [17]. The sparse approximation objective function can be written in terms of the $\ell_1$-norm as follows:

$$\arg \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2 + \lambda \|\mathbf{x}\|_1 \qquad (2)$$

where $\mathbf{x}$ is a sparse representation of $\mathbf{y}$ with respect to $\mathbf{A}$ and $\lambda$ is a regularization coefficient (increasing this parameter promotes sparser solutions).

In an image denoising application, $\mathbf{y}$ is a noisy image, $\mathbf{x}$ is a sparse coefficient vector, and $\mathbf{A}\mathbf{x}$ is a denoised approximation of $\mathbf{y}$. In a classification application, the sparse coefficient vector $\mathbf{x}$ is used to determine which class a test signal $\mathbf{y}$ belongs to. This can be done by first measuring the sum of the coefficients in each class and then finding the class that has largest number of nonzero coefficient.

In the sequel, we evaluate the performance of RankMap for accelerating the first-order methods for sparse approximation via $\ell_1$-minimization. This sparse approximation problem can be solved using the following iterative soft-thresholding algorithm [11]:

$$\mathbf{x}^{\text{iter}+1} = f(\mathbf{x}^{\text{iter}} - \gamma(\mathbf{G}\mathbf{x}^{\text{iter}} - \mathbf{A}^T \mathbf{y})) \qquad (3)$$

where $f(\cdot)$ is a low-complexity thresholding operation (e.g., a soft-thresholding operator [11]) to account for the

term $\lambda \|\mathbf{x}\|_1$ at each iteration, and $\gamma$ is the step size. In our evaluations, we employ a variant of this algorithm called fast iterative shrinkage-thresholding algorithm (FISTA) [6]. FISTA is an example of projected gradient descent (PGD) methods [32], which provide a generalization of standard gradient descent methods for certain classes of nonsmooth objective functions. RankMap can be readily applied to cost functions that can be solved using PGD.

*2) Power Method for Eigenvalue Decomposition:* The power method is a simple and iterative algorithm that can be used to sequentially find the eigenvectors and eigenvalues of a matrix in descending order. Recall that an eigenvector $\mathbf{x}$ of a matrix $\mathbf{A}$ satisfies the following relationship $\mathbf{A}\mathbf{x} = \sigma \mathbf{x}$, where $\sigma$ is the eigenvalue associated with the eigenvector $\mathbf{x}$. To find an eigenvector of the symmetric matrix $\mathbf{G} = \mathbf{A}^T \mathbf{A}$, the power method utilizes the following iterative update:

$$\mathbf{x}^{\text{iter}+1} = \frac{\mathbf{G}\mathbf{x}^{\text{iter}}}{\|\mathbf{G}\mathbf{x}^{\text{iter}}\|_2}. \qquad (4)$$

Once the power method converges to an estimate of an eigenvector $\mathbf{x}$, the contribution of this eigenvector is removed from $\mathbf{A}$, and the power method is applied again to the residual to find the next eigenvector.

In both applications described earlier, the main cost of each iteration is due to the computation of $\mathbf{G}\mathbf{x}$, especially when $\mathbf{G}$ is large, dense, and distributed onto multiple computing nodes. For example, as a case-study in our evaluations, we perform image reconstruction on a data set where $\mathbf{A}$ is a collection of light field image patches of size $18{,}496 \times 100{,}000$. In this case, to reconstruct a single noisy image patch $\mathbf{y}$, more than 3.6 billion floating point multiplications are required to perform $\mathbf{G}\mathbf{x}^{\text{iter}} = \mathbf{A}^T \mathbf{A}\mathbf{x}^{\text{iter}}$ per iteration.

## III. BACKGROUND AND RELATED WORK

In this section, we provide background on methods for matrix factorization and describe related work.

### A. Methods for Matrix Factorization

High-dimensional data can be modeled by the low-rank structures that are present in the data. Extracting low-dimensional structures not only reduces dimensionality, but also mitigates the effect of noise and improves the performance of learning and inference tasks [17], [19].

*1) Singular Value Decomposition:* In settings where the column span of $\mathbf{A}$ admits a low-rank model, the singular value decomposition (SVD) provides a powerful tool for forming low-rank approximations. Let $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ be the SVD. The best rank-$k$ approximation of $\mathbf{A}$ is given by $\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$, where $\mathbf{U}_k \in \mathbb{R}^{m \times k}$ and $\mathbf{V}_k \in \mathbb{R}^{n \times k}$ are the truncated left and right singular vectors (the first $k$ columns of $\mathbf{U}$ and $\mathbf{V}$) and $\mathbf{\Sigma}_k \in \mathbb{R}^{k \times k}$ contains the first $k$ singular values of $\mathbf{A}$ along its diagonal. The rank of $\mathbf{A}_k$ equals the number of nonzero singular values. The truncated SVD also provides the solution to principal components analysis (PCA), which seeks to find a $k$-dimensional subspace that best approximates $\mathbf{A}$ in the least-squares sense [46].

The complexity of computing the SVD directly is $m^2 n$. Thus, for large data sets, the power method is used to find the eigenvectors of $\mathbf{A}^T \mathbf{A}$ and $\mathbf{A}\mathbf{A}^T$, which correspond to the right and the left singular vectors, respectively.

*2) Sparse Factorization:* The SVD provides a closed-form solution for finding the best rank-$k$ approximation to a matrix. However, in many settings, enforcing sparse structure, either in the left or right singular vectors can provide a more faithful and compact decomposition of the data. Two widely used sparse factorization methods include sparse PCA (SPCA) [51] and dictionary learning [4]. However, these approaches are often not applied to large data sets, since computing an update of both the left and the right factor matrices, at each iteration, is costly. To solve SPCA on big data sets, a generalized power method can be employed [30]. The basic idea behind using the power method to find sparse principal components is to simply threshold the output of each power iteration to ensure the resulting eigenvectors are sparse. Unfortunately, the convergence of this method is much slower than standard power iterations.

*3) Column Subset Selection-Based Matrix Factorization:* An alternative strategy for low-rank matrix factorization is to form a decomposition based upon columns (or rows) from the data. Column subset selection (CSS)-based solutions form an approximate matrix decomposition in which one factorized component is a subset of the columns of the data itself [16]. CSS-based approaches have been used to provide a scalable and efficient strategy for finding approximate solutions to least-squares regression [15], Gramian matrix decomposition [16], image denoising and clustering [17], and also in spectral clustering [21]. After selecting columns from $\mathbf{A}$, the remaining unsampled columns are completed by finding the least-squares projection onto the subspace spanned by the sampled columns.

### B. Generic Distributed Abstractions

A number of successful distributed abstractions for processing large data sets on clusters have been proposed.

Examples include MapReduce [13], Apache Spark [49], and SystemML [23]. However, these models become less efficient for applications when direct data-parallelism does not exist. Several new distributed abstractions have been proposed that model data dependence in a graph format, most notably Pregel [35] and GraphLab [34]. They use a vertex-centric computation model, in which the user-defined programs are executed on each vertex in parallel. As graph-based abstractions are suited for sparse data sets, efficient data partitioning is not possible when the graph representation of the data is densely connected. Furthermore, such tools mostly rely on the communication between the vertices for computation. When the data are densely connected, the resulting communication congestion makes the computation dramatically slow. Because of this, most of these tools are designed based on the assumption that the input data are sparse [25], [35], [49].

By design, MapReduce-based solutions are not guaranteed to be fast, instead they provide easy and reusable programming frameworks that operate on very large data sets on a distributed computing platform. Users only have to deal with writing the functions of the algorithm in the given MapReduce-based programming model. MapReduce, on the other hand, controls the distributed cluster, manages data partitioning and data transfers between the various parts of the system, and provides fault tolerance.

## IV. COLUMN SELECTION-BASED SPARSE DECOMPOSITION

In this section, we present a scalable method for matrix decomposition (the decomposition phase in Fig. 2), which we call column selection-based sparse decomposition (CSSD).

### A. Overview of CSSD Method

The main idea behind CSSD is to first select a subset of columns from $\mathbf{A}$, and then use this subset of columns as a basis from which we form sparse representations of the remaining columns. We thus factorize the data as $\mathbf{A} = \mathbf{D}\mathbf{V}$, where $\mathbf{D}$ is formed by subsampling and normalizing the selected columns of $\mathbf{A}$. Each column of $\mathbf{V}$ is then computed by finding the sparse approximation of the corresponding column of $\mathbf{A}$ with respect to $\mathbf{D}$. This sparse approximation problem can be solved by an efficient greedy routine called *orthogonal matching pursuit* (OMP) [12]. We provide pseudocode for CSSD in Algorithm 1.

*1) Step 1 (Sequential Column Selection):* In order to ensure that the total approximation error in our factorization is sufficiently small, we must ensure that the columns selected from $\mathbf{A}$ to form $\mathbf{D}$ well approximate the range of the original matrix. Thus, we employ a sequential method to adaptively select columns that are not well approximated by the current set of columns [14].

Adaptive column selection methods select a new batch of columns according to the following probability distribution:

$$p(i) \propto \frac{\left\| \mathbf{A}_S \mathbf{A}_S^+ \mathbf{a}_i - \mathbf{a}_i \right\|_2}{\left\| \mathbf{a}_i \right\|_2} \tag{5}$$

where $p(i)$ equals the probability of selecting the $i$th column from $\mathbf{A}$ (denoted by $\mathbf{a}_i$), $S$ contains the indices of columns

---

**Algorithm 1** Column Selection Sparse Matrix Decomposition

---

**Input:** Matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, error tolerance $\delta_D$, number of columns to select at each iteration $l_s$, and the maximum number of columns to select $l$.

**Output:** A sparse matrix $\mathbf{V} \in \mathbb{R}^{l \times n}$ and a dense matrix $\mathbf{D} \in \mathbb{R}^{m \times l}$ such that for each column of $\mathbf{a}$ of $\mathbf{A}$, $\|\mathbf{a} - \mathbf{D}\mathbf{v}\|_2 \leq \delta_D$.

**Initialize:** Initialize $\mathbf{D}$ by adding and normalizing $l_s$ columns from $\mathbf{A}$ with uniform random sampling.

**Step 1: Sequential column selection**

**while** $ncols(\mathbf{D}) < l$ **do**
  I. Update $\mathbf{D}$ by selecting and normalizing $l_s$ columns from $\mathbf{A}$ according to the distribution in (5).
  II. If the $\ell_2$-norm of each column of $\mathbf{E} = \mathbf{A} - \mathbf{D}\mathbf{D}^+\mathbf{A}$ is less than $\delta_D$, return $\mathbf{D}$ and proceed to Step 2 to compute $\mathbf{V}$.
**end while**

**Step 2. Sparse approximation**

I. Compute $\mathbf{V}$ by applying Batch OMP to solve (6) with error tolerance $\delta_D$.

---

already selected, and $\mathbf{A}_S$ denotes the submatrix of sampled columns. We can flexibly execute this subsampling approach by either specifying the maximum number of columns to select and/or specifying the maximum amount of error in each unsampled column of $\mathbf{A}$.

*2) Step 2 (Sparse Approximation):* After selecting a subset of $l$ columns $\mathbf{A}_S \in \mathbb{R}^{m \times l}$, we normalize each column, such that all the columns in matrix $\mathbf{D}$ have unit norm. Now, to form the sparse matrix $\mathbf{V}$, we find a sparse representation of the remaining columns in $\mathbf{A}$ (i.e., $\mathbf{A}_{-S}$) in terms of the normalized dictionary $\mathbf{D}$. The problem is formally written as follows:

$$\min_{\mathbf{v}} \ \|\mathbf{v}\|_0 \quad \text{s.t.} \quad \frac{\|\mathbf{a}_i - \mathbf{D}\mathbf{v}\|_2}{\|\mathbf{a}_i\|_2} \leq \delta_D, \quad \forall i \notin S \qquad (6)$$

where $\|\mathbf{v}\|_0$ counts the number of nonzero coefficients in $\mathbf{v}$ and $\delta_D$ is a user-specified parameter, which controls the decomposition error.

We employ a matching pursuit-based solver called Batch OMP [43] to solve (6). We can enforce sparsity either by the number of nonzeros in each column of $\mathbf{V}$ (i.e., $\|\mathbf{v}\|_0$) or by the total amount of approximation error for each column.

*B. Complexity Analysis*

The complexity of sequential column selection (Step 1) is $\mathcal{O}(l^2 m + lmn)$. The complexity terms correspond to computing $\mathbf{D}^+$ and $\mathbf{D}\mathbf{D}^+\mathbf{A}$, respectively. The projection $\mathbf{D}\mathbf{D}^+\mathbf{a}$ can be computed for each column of $\mathbf{A}$ independently. The complexity of sparse approximation (Step 2), using the Batch OMP method [43], is $\mathcal{O}(lmn + k^2 ln)$, where $k < l$ is the average number of nonzeros per column of $\mathbf{V}$. Similarly, for each column of $\mathbf{A}$, Batch OMP is applied independently. Let $n_c$ be the number of parallel processing nodes. By storing $\mathbf{D}$ (which is a small $m \times l$ matrix) and a uniform fraction of columns of $\mathbf{A}$ in each node [i.e., $(n/n_c)$ columns], the overall complexity of Algorithm 1 in a distributed setting can be written as $\mathcal{O}((n/n_c)(lm + k^2 l) + l^2 m)$.

Note that CSSD is linear in terms of both the number of data samples $n$, and the number of processors $n_c$. This is a key feature of our approach that makes our framework applicable to very large data sets in distributed settings.

*C. Computational Benefits of CSSD*

CSSD provides computational benefits when the size of the decomposition is small (i.e., $l$ is small relative to $m$) and/or when matrix $\mathbf{V}$ is sparse. In general, predicting the amount of savings in computation is a function of: 1) the structure of the data and 2) the amount of accuracy required from the learning algorithm. We now discuss some key factors that impact the decomposition results.

*1) Impact of Data Structure:* Predicting the size and sparsity of the decomposition provided by CSSD for an arbitrary data set is challenging; however, when the data lie on a single subspace (i.e., exhibit low-rank structure) or lie on multiple low-dimensional subspaces, CSSD provides a more compact representation of the data. For example, when data are exactly low rank and its rank is $r < m$, we must select $r$ linearly independent columns from $\mathbf{A}$ to form an exact decomposition (zero error), i.e., $l = r$. When the data are approximately low rank, there exists a large body of work that characterizes the performance of the sequential column selection method (Step 1) used to form $\mathbf{D}$ [14], [24]. In particular, the selection strategy in Step 2 of Algorithm 1 provides exponential decrease in the factorization error with each batch of columns that we select from $\mathbf{A}$ [14]. More specifically, assume that at each iteration, we select $l_s > (r/\epsilon)$ samples from the columns of $\mathbf{A}$ and let $l = t l_s$ denote the set of columns selected after $t$ iterations. Let $\mathbf{A}_r$ denote the best rank $r$ approximation to $\mathbf{A}$ and let $\widetilde{\mathbf{A}} = \mathbf{A}_S \mathbf{A}_S^+ \mathbf{A}$ denote the approximation of $\mathbf{A}$ based upon the $l$ selected columns $\mathbf{A}_s$. Then, according to [14], the difference between the expected value of the approximation error, i.e., $\|\mathbf{A} - \widetilde{\mathbf{A}}\|_F^2$ and that of the best rank $r$ approximation $\|\mathbf{A} - \mathbf{A}_r\|_F^2$ decreases exponentially with rate $\epsilon^t$.

Another low-dimensional signal model that has recently gained traction models data with multiple low-dimensional subspaces (union of subspaces). For example, images of objects under different illumination conditions [41], motion trajectories of point correspondences [31], neural data [17], to structured sparse, and block-sparse signals [5] are all well approximated by a union of low-dimensional subspaces. When $\mathbf{A}$ lies on a union of subspaces, this effectively bounds the sparsity level of each column of $\mathbf{V}$ [18]. This insight is based upon the fact that when we form a representation of a column of $\mathbf{A}$ with respect to other columns in the same data set (as in CSSD), the sparsity level of each column is bounded by the dimension of the subspace the signal lies on. For instance, if $\mathbf{A}$ lives on a union of multiple $r$-dimensional subspaces of $\mathbb{R}^n$ and we select at least $r$ linearly independent columns from each subspace, then no more than $r$ nonzeros are required to represent a signal. In other words, the number of nonzeros per column of $\mathbf{V}$ is no more the dimension of the subspace the signal lives in.

*2) Impact of Increasing the Decomposition Error:* The decomposition error of CSSD is controlled by the parameter $\delta_D$ in Algorithm 1. In the case where we set $\delta_D = 0$, then we are guaranteed an exact decomposition of the data. Exact decomposition occurs when $r$ linearly independent columns are selected from $\mathbf{A}$, where $r = \text{rank}(\mathbf{A})$. In this case, the selected columns in $\mathbf{A}_S$ will fully represent the data and thus $\|\mathbf{A} - \mathbf{A}_S \mathbf{A}_S^+ \mathbf{A}\| = 0$, i.e., exact decomposition occurs.

While CSSD can produce an exact decomposition when the data are exactly low rank (or lies on a union of subspaces), in practice, data sets are approximately low rank. In this case, we can introduce a small amount of error into the decomposition by setting $\delta_D > 0$. By introducing some error into the decomposition, we observe that both the number of selected columns in Step 1 of Algorithm 1 and the sparsity level of $\mathbf{V}$ can be reduced further. In Fig. 8, we show how increasing the decomposition error produces a more compact decomposition. In Section V, we study and discuss the impact of the decomposition error $\delta_D$ on the accuracy of a learning algorithms that we apply RankMap to.

## V. TUNING DECOMPOSITION ERROR FOR A TARGET LEARNING ACCURACY

In Section IV C.2 we discussed the computational benefits associated with introducing some approximation error into a CSSD decomposition. Naturally, as we increase the decomposition error (controlled by $\delta_D$), the accuracy of our learning algorithm will be affected. Thus, the key question is how much decomposition error we can afford to achieve a certain degree of accuracy in learning. The answer to this question heavily depends on the specific learning algorithm and the application of interest.

Previous theoretical studies have established a connection between the total error in a factorization of a kernel (or Gram) matrix and the accuracy of certain popular learning algorithms, including kernel ridge regression and kernel SVM [10]. While for some learning algorithms, our framework can exploit previous work to relate $\delta_D$ and the learning error, which we denote by $\delta_L$; the aim of this section is to propose a generic approach for tuning the factorization error to achieve a specified learning accuracy. We do this by iteratively remapping of the data to find a compact decomposition that satisfies a learning error (specified by the user).

### A. Error Tuning

Given an already established relationship between the decomposition error and a specific algorithm, a practitioner who uses our framework can easily specify the error parameter $\delta_D$ for CSSD to achieve a particular learning accuracy. Alternatively, if a practitioner specifies a target accuracy for a learning algorithm, the decomposition error $\delta_D$ can be tuned in order to achieve a particular learning error $\delta_L$.

Our strategy for guaranteeing that we have small learning error is to solve CSSD for a particular $\delta_D$, map the resulting decomposition via the methods described in Section VI, and then compute the accuracy of the learning algorithm $\delta_L$.

We then iteratively add columns to $\mathbf{D}$, such that the decomposition error $\delta_D$ is small enough to ensure that $\delta_L$ is within the error-specified tolerance. Depending on the underlying computing resources available, RankMap can be applied for multiple values of $\delta_D$ in parallel and the largest value of $\delta_D$ (most compact representation) that achieves a particular value of $\delta_L$ is selected.

When computing resources are constrained and thus running the algorithm for multiple values of $\delta_D$ in parallel is not possible, we use a bisection method. In essence, the idea is to: 1) set the factorization error to predefined maximum value $\delta_D^{max}$ (0.4 in our experiments) and evaluate $\delta_L$ and 2) if $\delta_L$ is below a target value then we stop, otherwise we decrease $\delta_D$ by half. By exponentially decreasing $\delta_D$, we are also guaranteed to decrease $\delta_L$ exponentially, provided that there is a polynomial relationship between the two quantities. We observe a polynomial relationship holds both in theory [10] and in practice. In Section VII, we provide empirical results, which demonstrate the connection between the decomposition error and learning accuracy for numerous data sets and algorithms of interest [see Figs. 7(a) and 8(b)].

## VI. DISTRIBUTED EXECUTION AND DATA PARTITIONING

In this section, we introduce our approach for applying iterative updates on the decomposed data (the execution phase in Fig. 2). We describe an execution flow for dependence matrix-based updates [i.e., $\mathbf{Gx} = \mathbf{V}^T (\mathbf{D}^T \mathbf{D}) \mathbf{Vx}$] and introduce an efficient method for partitioning the decomposed data in a distributed setting. We also provide performance bounds on memory usage, the number of flop operations, and the number of communicated bytes across the computing nodes.

### A. Computation Flow

We propose two computational models for the distributed implementation of an update in (1). Recall that at each iteration, we must compute $\mathbf{z} = \mathbf{Gx} = \mathbf{V}^T (\mathbf{D}^T \mathbf{D}) \mathbf{Vx}$. We break this computation into four steps: 1) $\mathbf{p} = \mathbf{Vx}$; 2) $\mathbf{r} = \mathbf{Dp}$; 3) $\mathbf{p} = \mathbf{D}^T \mathbf{r}$; and 4) $\mathbf{z} = \mathbf{V}^T \mathbf{p}$. The output vector $\mathbf{z}$ is used to produce an update of $\mathbf{x}^{\text{iter}+1} = f(\mathbf{z} + \mathbf{b})$, where $\mathbf{b}$ is an offset vector and $f(\cdot)$ is a low-complexity function, such as a soft-thresholding operator (sparse approximation) or normalization (power method). To carry out the computation described earlier, we propose and implement a matrix-based and vertex-based model to apply the iterative updates on the decomposed factors. We now describe our implementation of both models.

### B. Matrix-Based Model

Fig. 3 shows the schematic of our proposed matrix-based model. In this model, the data are stored in arrays. The sparse matrix $\mathbf{V}$ is stored and operated upon using the compressed sparse column (CSC) format. The matrix $\mathbf{D}$ and vector $\mathbf{x}$ are stored using regular dense arrays. By doing so, we exploit sparsity in $\mathbf{V}$. We use C++ Eigen Library for array manipulation and MPI for distributed computing.
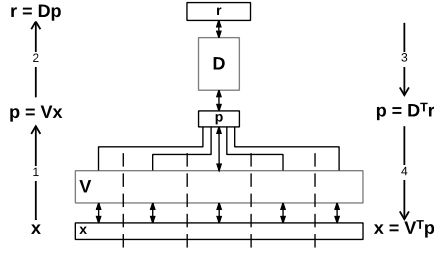
Fig. 3.   *Distributed design of matrix-based model.*



Fig. 4.   *Distributed design of graph-based model.*

*1) Distributed Partitioning:* We partition columns of $\mathbf{V}$ uniformly across the computing nodes to achieve a balanced partitioning. Let us assume that there are $n_c$ computing nodes. Thus, $(n/n_c)$ number of columns are assigned to each node. The vector $\mathbf{x}$ is also divided into chunks of size $(n/n_c) \times 1$. Each chunk is then allocated to the node that hosts the corresponding columns of $\mathbf{V}$.

Matrix-vector multiplications $\mathbf{Vx}$ are performed locally on the columns of $\mathbf{V}$ and the portion of $\mathbf{x}$ that reside on the same computing node. The resulting $l \times 1$ vectors are then sent to a central node to create $\mathbf{p} = \mathbf{Vx}$. Next, $\mathbf{D}^T(\mathbf{Dp})$ is computed locally in the central node. The resulting $l \times 1$ vector is then *broadcasted* back to all the computing nodes where it is multiplied by the local $\mathbf{V}^T$ to update the vector $\mathbf{x}$.

*2) Performance Bounds:* We now provide bounds on the memory usage, computation, and communication required by our proposed matrix-based model. We also provide the performance bounds for baseline, in which we perform iterative analysis on $\mathbf{A}^T\mathbf{A}$ in matrix format. Let *nnz(.)* denote the number of nonzeros of its input and $n_c$ denote the number of computing nodes. Recall that $\mathbf{D}$ is an $m \times l$ matrix and $\mathbf{V}$ is an $l \times n$ matrix. Note that in our target data scenarios, $m \ll n$ and $l \ll n$. In many cases, the rank of decomposition $l$ is often much smaller than the dimensions of data $m$. When data exhibit union of subspace property, $\mathbf{V}$ will be sparse, i.e., $nnz(\mathbf{V}) < ln < mn$.

| Matrix-based | Baseline | RankMap |
|---|---|---|
| Memory usage $\propto$: | | |
| # non-zeros | $mn$ | $(nnz(\mathbf{V}) + lm) + n + m$ |
| Computation $\propto$ : | | |
| # additions | $2mn + mn_c$ | $2(nnz(\mathbf{V}) + lm + ln_c)$ |
| # multiplications | $2mn$ | $2(nnz(\mathbf{V}) + lm)$ |
| Communication $\propto$: | | |
| #edges | $2lm$ | $2ln_c$ |

Since $\mathbf{V}$ is stored in a CSC format, only the nonzero values are stored and operated on. The matrix $\mathbf{D}$ is stored in a regular dense matrix format. The communication corresponds to sending and receiving the $l \times 1$ vectors from each computing node to the central node. Clearly, for smaller $l$ and sparser $\mathbf{V}$, both memory footprint and the number of arithmetic operations are reduced. The number of edges, which correspond to the number of broadcasted and reduced values, directly corresponds to $l$ and the number of computing nodes $n_c$.
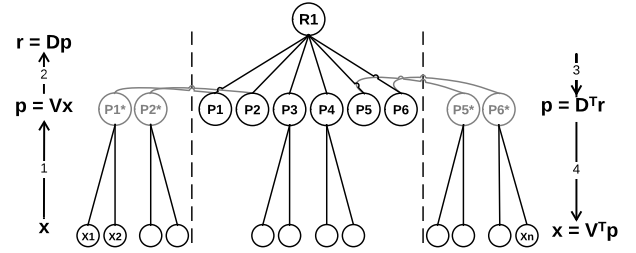
## C. Graph-Based Model

Fig. 4 shows an schematic of our proposed graph model. The decomposed data are three-layer graph denoted by $\mathcal{G}_A(S_X, S_P, S_R)$ with vertex sets $S_X = \{X_i\}_{i=1}^n$ in the bottom layer, $S_P = \{P_i\}_{i=1}^l$ in the middle layer, and $S_R = \{R_1\}$ in the top layer. Each nonzero element in $\mathbf{V}$, e.g., $\mathbf{V}_{ij}$, is represented by an edge, which connects $X_i$ to $P_j$. Each column of $\mathbf{D}$, e.g., $\mathbf{D}_i$, is represented by an edge, which connects $P_i$ to $R_1$. Value of vertices in $S_X$ corresponds to the elements of $\mathbf{x}$.

We use GraphLab Distributed API [34] to implement this model. While GraphLab is a highly optimized distributed engine for graph-based computation on iterative data, we perform extensive customizations in order to adapt GraphLab to our factorized setting. We also force GraphLab to use our developed graph partitioning method as opposed to its automated partitioning schemes. Our proposed partitioning is customized to the factorized data and significantly improves the performance.

*1) Distributed Partitioning:* In the graph-based model, we partition $\mathcal{G}_A(S_X, S_P, S_R)$ with the aim of balancing the number of components assigned to each node and also minimizing the internode communications characterized by the edges. Since the edge distribution of $\mathcal{G}_A$ is highly nonuniform ($l \ll n$), a vertex partitioning inevitably results in many undesirable edge cuts across the computing nodes. Instead, we apply a vertex-cut method in which the goal is to partition graph edges evenly, such that the number of vertices that are spanned across multiple partitions is minimized. As a result of edge partitioning, vertices may reside on two or more computing nodes. In this case, we assign one of the copies to be the *master* vertex and the others to be the *replica* vertices (these definitions are borrowed from GraphLab [25]). The replicas directly cause (expensive) internode communication costs.

Fig. 4 shows the graph-based distributed design. Our detailed edge partitioning method is as follows: 1) distribute master of vertices $X_i \in S_X$ uniformly onto the available computing nodes, such that vertex chunks of size $(n/n_c)$ are assigned to each node; 2) add the edges between vertices $X_i \in S_X$ and $P_j \in S_P$ to the node in which the corresponding master of $X_i$ resides; 3) add master of vertices $P_i \in S_P$ and $R_1 \in S_R$ to a *central node*; and 4) add the edges between the vertices $P_i \in S_P$ and $R_1 \in S_R$ to the central node.

The proposed edge partitioning algorithm is highly efficient in that it does not induce any replicas for vertices in $S_X$ and $S_R$. However, from Step 2), replicas of vertices in $S_P$

may exist in computing nodes other than the central node. At the beginning of an iteration, master vertices in $S_P$ and their replicas perform vertex updates with respect to $S_X$. The replicas send the updated values to their own master vertices in the central node. The master vertices in $S_P$ *reduce* the received values ($\mathbf{p} = \mathbf{Vx}$). Then, master vertex $R_1$ performs a vertex update ($\mathbf{r} = \mathbf{Dp} - \mathbf{y}$). Next, master vertices in $S_P$ complete vertex update with respect to $S_R$ and *broadcast* the results to their own replicas ($\mathbf{p} = \mathbf{D}^T\mathbf{r}$). Finally, master vertices in $S_X$ update themselves ($\mathbf{x} = \mathbf{V}^T\mathbf{p}$). We integrate and implement the proposed customized partitioning and distributed computation flow with the distributed GraphLab API [25].

*2) Performance Bounds:* We now provide bounds on the memory usage, computation, and communication required by our proposed graph-based model.
1) *Memory Usage:*
   # edges $\propto nnz(\mathbf{V}) + l$.
   # vertices $\propto n + \sum_{1 \le i \le l} \text{rep}(P_i)$.
2) *Computation (Per Iteration):*
   # additions $\propto 2(nnz(\mathbf{V}) + ml) + \sum_{1 \le i \le l} \text{rep}(P_i)$.
   # multiplications $\propto 2(nnz(\mathbf{V}) + ml)$.
3) *Communication:*
   # edge-cuts $\propto 2 \sum_{1 \le i \le l} \text{rep}(P_i)$.

| Graph-based | Baseline | RankMap |
|---|---|---|
| Memory usage $\propto$: | | |
| # edges | $mn$ | $nnz(\mathbf{V}) + l$ |
| # vertices | $n + mn_c$ | $n + \sum_{1 \le i \le l} rep(P_i)$ |
| Computation $\propto$ : | | |
| # additions | $2mn + mn_c$ | $2(nnz(\mathbf{V}) + ml) + \sum_{1 \le i \le l} rep(P_i)$ |
| # multiplications | $2mn$ | $2(nnz(\mathbf{V}) + lm)$ |
| Communication $\propto$: | | |
| #cuts | $2lm$ | $2 \sum_{1 \le i \le l} rep(P_i)$ |

Each of the computing nodes receives approximately $(1/n_c)(n + \sum_{1 \le i \le l} \text{rep}(P_i))$ vertices and $(1/n_c)nnz(\mathbf{V})$ edges. The central node has $l$ additional edges between the master vertices in $S_P$ and $R_1$. The computation cost is induced by vertex update operations. The communication overhead is incurred by the message passing across master and replica vertices in $S_P$.

*Bound on total replicas:* From the discussions described earlier, it is clear that reducing number of replicas of $S_P$ reduces the communication overhead. The following are the bounds on the total number of replicas:

$$l \le \sum_{1 \le i \le l} \text{rep}(P_i) \le ln_c.$$

The inequalities hold, since each $P_i$ is replicated at least once and at most $n_c$ times (one replica per computing node). Both $l$ and $n_c$ are much smaller than the size of the graph. Thus, RankMap's graph-based model readily provides efficient/balanced computation and reduced communication without using complicated and costly graph partitioning algorithms. The minimum communication is achieved when $\mathbf{V}$ is block diagonal.

## VII. EVALUATIONS

In this section, we evaluate the performance of RankMap on a variety of data sets. Our evaluations explore: 1) the scalabil-

ity of CSSD and its ability to produce sparse representations; 2) the connection between decomposition error and learning accuracy for multiple learning applications, including face recognition, image denoising, and PCA; 3) RankMap's performance improvement in terms of runtime, and memory over prior work; and 4) the performance of our distributed matrix- and graph-based models for different structured data sets.

### A. Evaluation Setup

*1) Data Sets:* We evaluate RankMap on both real and structured synthetic data sets. Our real data sets include Light Field data [2], hyper spectral images [1], a dictionary of video frames [28], and a collection of images of different faces under varying illumination conditions [22].

We apply RankMap to two different Light Field data sets. The first data set, which we refer to as Light Field (i), consists of 10 k randomly selected atoms from a $5 \times 5$ Light Field array (collected from Chess Images). Each Light Field patch consists of 25 $8 \times 8$ patches, which produces a data set of size 1.6 k $\times$ 10 k (128 MB). The second data set, which we refer to as Light Field (ii), consists of 100 k randomly selected atoms from a $17 \times 17$ Light Field array (collected from all available Light Fields in the archive). Each Light Field patch consists of 289 $8 \times 8$ patches, which produces a data set of size $18\,496 \times 100$ k (14.7 GB). The hyper spectral data set (Salinas) is taken from a region of a remote sensing scene in Salinas, CA. Each pixel in the scene contains information from 203 spectral bands and produces a data set of size $203 \times 54\,129$ (87.9 MB). The video dictionary data set (VideoDict) contains patches of an image over multiple frames and produces a data set of size $1764 \times 100\,000$ (1.41 GB). The face image data set (Faces) consists of 631 images of ten different people faces under varying illumination conditions. Each image is $48 \times 84$ pixels, which produces a data set of size $4032 \times 631$. In addition to real-world data sets, we generate synthetic data for $n = 10$ M, $m = 1$ k with varying $l$, and sparsity levels in $\mathbf{V}$.

*2) Computing Platform:* To evaluate the decomposition methods on Light Field (i), an eight-core CPU (Intel Core i7 processor) with 12 GB of RAM is used. For computations on Light Field data set (ii), we instancied a cluster of 16 m3.large nodes (machines) on Amazon EC2. Each node has 16 cores (two Intel Xeon processors) at 7.5 GB of RAM per node. The synthetic data sets are evaluated on IBM iDataPlex computing cluster, which has 2304 cores in 192 Westmere nodes (12 processor cores per node) at 48 GB of RAM per node.

*3) Distributed Tools:* All RankMap's APIs are available to the public [3]. The RankMap framework's sparse matrix-based model is implemented using Eigen Library to represent data in a compressed column storage (CCS) format [27]. It uses MPI's standard system to distribute the data and computation and is written in C++. We have also implemented the distributed update on the factorized data on Apache Spark [49].

The RankMap framework's sparse graph-based design is implemented using *GraphLab*, a high-level graph-parallel abstraction [25]. GraphLab enables vertex-update-based computations. We implemented RankMap's customized
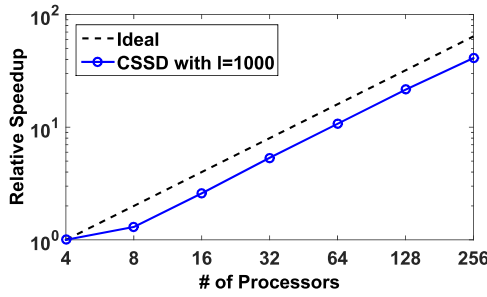
Fig. 5. *CSSD's runtime scaling behavior as the number of processors increases.*

partitioning using GraphLab's *ingress* class. The proposed architectures are mapped efficiently into GraphLab API (Section VI-C). Note that the GraphLab framework is designed to accelerate distributed learning for sparse graphs and thus it is not suited to process dense data until we sparsify the data using CSSD.

### B. Scaling of CSSD

Fig. 5 shows how the runtime of CSSD scales as the number of processors increases for the VideoDict data set. We increase the number of cores from 4 to 256 (on IBM iDataPlex cluster). The dotted line shows the ideal scale-out behavior. As can be seen, CSSD is highly parallel as it almost linearly scales with the number of processors. Thus, it can be applied to very large data sets.

### C. Sparse Approximation

To evaluate the performance of RankMap for sparse approximation, we use the FISTA [6] to solve the $\ell_1$-minimization problem in (2). We study the utility of RankMap for two applications: sparse representation-based classification for face recognition and image denoising (see Section II-B for more details on these applications).

*1) Sparse Representation-Based Classification for Face Recognition:* To employ sparse approximation for classification, our aim is to use a collection of labeled images (training set) as our dictionary **A** and then form a sparse representation of a test image **y** in terms of **A**. After finding a sparse coefficient vector **x**, we can then determine, which signals in the testing set (columns of **A**) are selected to represent the test signal **y**. Based upon the class of the selected columns, we then make a decision about which class the test signal lies in. One easy way to do this is to simply sum the absolute value of the coefficients in **x** in a certain class and then find the class with maximum sum.

In Fig. 6, we provide a demonstration of sparse representation-based classification for face recognition. We show the test image of interest on top and the corresponding sparse coefficient vector obtained by solving (2) with FISTA, where $\lambda = 1$. We solve FISTA with the full Gram matrix $\mathbf{A}^T\mathbf{A}$ and approximate Gram provided by CSSD, where the decomposition error $\delta_D = 0.05$ ($l = 62$).

To understand the connection between the decomposition error and learning accuracy for face recognition,
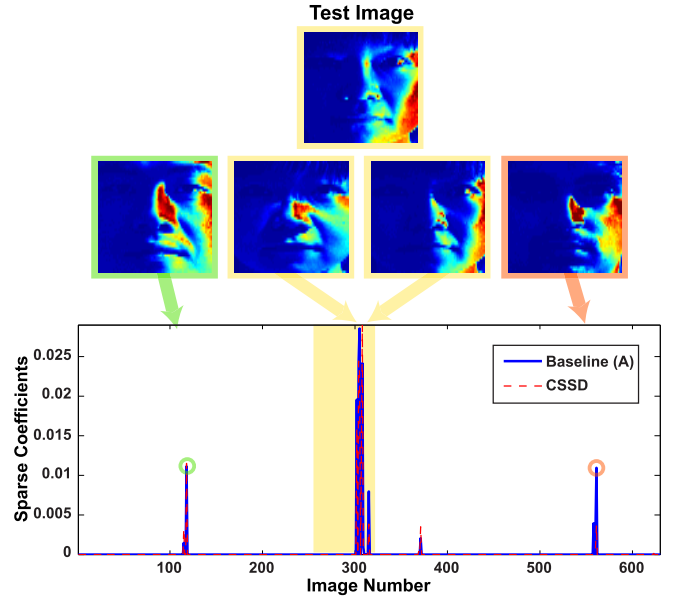


Fig. 6. *Sparse representation-based face recognition.* We show the sparse coefficients obtained with the original Gram matrix (blue solid lines) and approximate Gram matrix with CSSD for $\delta_D = 0.05$ (red dashed lines). On top, we show the test image, and four training images that produce significant nonzero coefficients. Both the baseline and our approach result in correct classification, as their largest coefficient is associated with a training example from the same class as the test image. The block of coefficients corresponding to images in the correct class is highlighted.

we solve (2) using FISTA for two different regularization parameters $\lambda = \{0, 1\}$, where $\lambda = 0$ corresponds to the least-squares solution and $\lambda = 1$ produces sparse solutions. We vary the decomposition error $\delta_D = \{0.4, 0.2, 0.1, 0.05\}$ and solve FISTA for 30 different test images (after removing them from training) for each of these decompositions. We calculate the learning accuracy by measuring the $\ell_2$-norm between the solution obtained with the full and approximate Gram [Fig. 7(a)], the sum of coefficients in the correct class [Fig. 7(b)], and the relative density of **V** (the number of nonzeros in **V** versus the number of nonzeros in **A**) [Fig. 7(c)]. In Fig. 7(b), we also display the minimum sum of coefficients required to correctly classify the test image. In this case, we obtain the correct class with this approach when $\delta_D < 0.2$. These results suggest that even when we allow a significant amount of decomposition error, correct classification is still possible.

*2) Light Field Image Denoising:* We evaluate RankMap's performance in denoising Light Field data. A Light Field is a multidimensional array of images where each image is captured from a slightly different viewpoint. To reconstruct and denoise light fields, $\ell_1$-minimization (2) is employed to find a sparse representation of a Light Field image with respect to an overcomplete Light Field dictionary consisting of a large number of Light Field image patches collected from many scenes [36]. This dictionary can be used to reconstruct light field patches for the purpose of superresolution and denoising.

We study the performance of RankMap for reconstructing light field patches from noisy observations (image denoising).
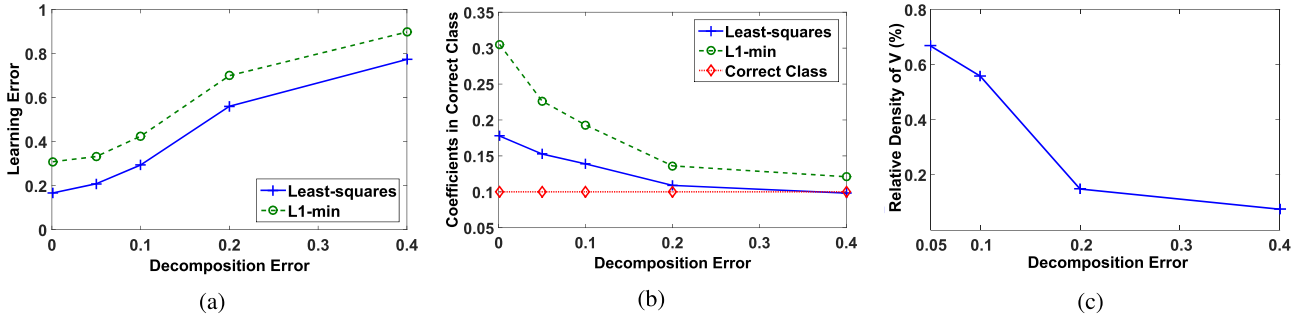
Fig. 7. *Studying the impact of the decomposition error on learning accuracy.* For a range of decompositions (varying $\delta_D$), we show (a) learning accuracy, which measures the two norm between the solution obtained with the full and approximate Gram, (b) sum of coefficients in the correct class, the minimum sum of coefficients required to correctly classify the test image is also shown, and (c) relative number of nonzeros in **V** versus the number of nonzeros in **A**.

TABLE I

*Runtime to Reach a Specific Output PSNR.* WE APPLY FISTA TO SOLVE THE DENOISING PROBLEM ON A AS WELL AS TWO DECOMPOSITIONS OF A OBTAINED BY SETTING $l = 240$ AND 1000 (IN ALGORITHM 1). THE LOWER-DIMENSIONAL DECOMPOSITION ($l = 240$) PROVIDES AN OUTPUT WITH 30-DB PSNR MUCH FASTER, DUE TO ITS LOWER MEMORY FOOTPRINT AND COMPUTING REQUIREMENTS. SIMILARLY, THE DECOMPOSITION OBTAINED FOR $l = 1000$ PRODUCES AN OUTPUT WITH 35-DB PSNR MUCH FASTER THAN A.

| PSNR (dB) | $l = 240$ | $l = 1000$ | baseline (**A**) |
|---|---|---|---|
| 25 | 4.2 | 72 | 487 |
| 30 | 13.9 | 162 | 1050 |
| 35 | - | 356 | 2051 |
| 40 | - | - | 3171 |

In all of our denoising experiments, Light Field (ii) is used. We first apply CSSD for decomposing the dictionary corresponding to the Light Field (ii) data set, for two different values of $l = 240$ and $l = 1000$. The decomposition error $\delta_D$ is set to 0.1. After decomposing the data, we then evaluate FISTA on the decomposed data with the matrix-based model. We compare RankMap's performance with that of a tailored distributed MPI-based model to evaluate FISTA on the original data set (**A**) using regular dense matrix representations. This implementation is denoted as the *baseline* in our evaluations.

Table I shows the total runtime of FISTA to achieve different peak signal-to-noise ratios (PSNRs). The PSNR is the ratio between the maximum possible power of a signal and the power of the corrupting noise is used to measure the performance of denoising algorithms. The PSNR is defined as $10 \log_{10}((\text{MAX}/\sqrt{\text{MSE}}))$ (dB), where $MAX$ is the maximum pixel value of the original image patch and MSE is the mean square reconstruction error defined as $((\|\mathbf{y} - \widehat{\mathbf{y}}\|^2)/m)$. Typically, in image noise reduction applications, PSNR values of 30 dB and higher are desired [4], [7], [45].

In all the experiments, a batch of ten noisy patches are used as the input and the norm of the noise is set to 0.3 times the norm of the input vector (PSNR = 21.14). We observe that RankMap can achieve the same PSNR orders of magnitude faster than the baseline implementation. For instance, if our desired PSNR is 30.0 dB, running FISTA on the decomposed data takes 13.9 s ($l = 240$) and $162s$ s ($l = 1000$), while it takes 1050 s for the baseline. However, as expected,

the baseline (**A**) reaches higher PSNRs in comparison with those achieved from running FISTA on the decomposed data. Thus, RankMap can be used to tradeoff learning accuracy for speed.

### D. Power Method

We also evaluate our framework on power method for three data sets: Salinas, VideoDict, and Light Field (i) (see Section II-B for more discussion of the power method). The matrix-based model is used and the experiments are run on 64 cores on an IBMiDataplex cluster. We run CSSD with various decomposition errors ($\delta_D$) that belong to the following set: $\{0.4, 0.2, 0.1, 0.05, 0.001\}$ and run the power method on each of the decomposition results. Fig. 8(a) shows the sparsity of **V** as we vary the error. As expected, for larger error tolerances, a sparser decomposition is achieved. Fig. 8(b) shows the impact of the decomposition error on the accuracy of the results of the power method. Here, the learning error ($\delta_L$) is defined to be the normalized accumulated error of the first 100 eigenvalues. By lowering the decomposition error, we can observe significant improvements in the accuracy of the power method. Finally, Fig. 8(c) shows the corresponding normalized runtimes to find the first 100 eigenvalues. Our results suggest that significant speedups are achieved in comparison with the baseline.

### E. Graph- Versus Matrix-Based Models

We compare the performance of RankMap's vertex and matrix-based models for various synthetic decomposed data. The purpose of these evaluations is to determine the advantages of each of the model, with respect to the structure of the data. In all the experiments, the iterative update in (3) is applied on a random input vector **y**. The experiments are done on an IBM iDataPlex computing cluster. In Fig. 8(a)–(c), the runtime results for the dense matrix-based implementation (i.e., regular deployment of the decomposed matrices without using CCS format) are provided to demonstrate the efficiency achieved by exploiting sparsity in **V** through sparse matrix-based and graph-based models. For the former model, we report analysis based on our C++ MPI implementation, and for the latter model, we report analysis based on our modification of GraphLab engine to implement RankMap.
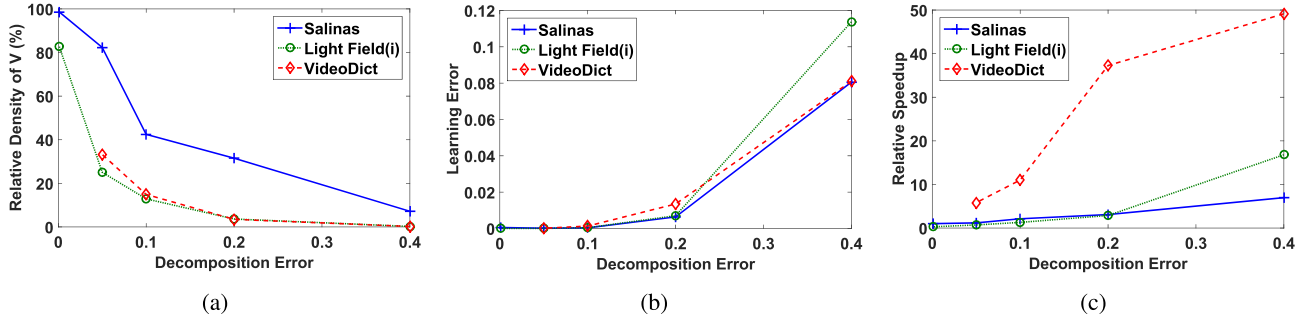
Fig. 8. *Power method results for three data sets: Salinas, VideoDict, and Light Field (i).* In (a), we show the effect of varying the decomposition error ($\delta_D$) on the sparsity of the factor **V**. Reported values are the number of nonzeros in **V** normalized to the number of nonzeros in **A**. In (b), we show the learning error ($\delta_L$) and in (c) relative speedup of power method to find the first 100 eigenvalues. Relative values are runtime of power method using the decomposed factors derived from CSSD normalized to the corresponding runtime using **A**.
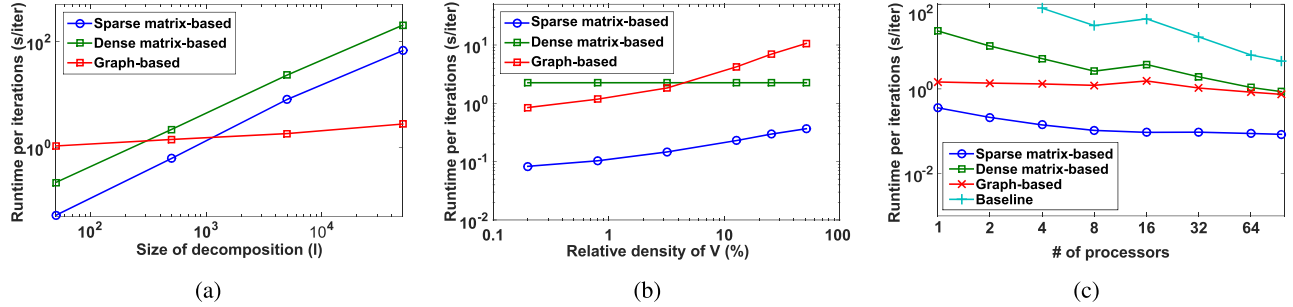


Fig. 9. *Comparison of matrix- and graph-based computational models on synthetic block-diagonal data.* We compare runtime versus the (a) size of the factorization, (b) relative density of **V**, and (c) number of processors.

The performance of RankMap for different (block diagonal) **V** values, with fixed number of nonzeros (set to $100M$), is shown in Fig. 9(a). As $l$ increases, the density-level of **V** decreases. The graph-based model's performance is more consistent as $l$ increases. However, the matrix-based model's performance degrades for larger $l$ values. This observation can also be explained due to the fact that the communication overhead of the matrix-based model is more affected by larger $l$ values.

Fig. 9(b) shows the performance for a fixed $l = 500$ on block-diagonal matrices **V**, for varying densities of **V**. As density increases, the performance decreases in both models. However, the performance degradation in graph-based model is worse due to the overhead of representing a large number of edges. Finally, Fig. 9(c) shows the scaling performance of the models for various number of processors. When the number of processors is less than 12, the computations are done on a single node. Thus, the reverse scaling behavior while increasing the number of processors from 8 to 16 is due to the high overhead of the internode communication cost. For comparison purposes, we report the scaling of the baseline approach that operates on the original dense $m = 1$ k by $n = 10$ M data set, instead of its decomposed form. It can be seen that as the number of processors increases, the performance gap between different methods shrink. However, even with a large number of processors ($\geq 100$), the decomposed models perform up to two orders of magnitude better than the baseline.

These experiments provide insight into the use case of each model. Depending on the structure of the decomposed data and

TABLE II

MEMORY ANALYSIS. RankMap ACHIEVES SIGNIFICANT IMPROVEMENT VIA ITS ADAPTIVE AND SPARSITY-INDUCING APPROACH

| Memory size (MB) | Original data | Least-squares | RankMap |
|---|---|---|---|
| Salina | 87.9 | 86.9 | 38.2 |
| VideoDict | 1411.2 | 835.0 | 279.2 |
| Light Field (ii) | 14796.8 | 1634.8 | 190.1 |

the specifications of the platform, an appropriate model should be selected. A more systematic domain-specific approach for model selection is the subject of future work.

*F. Memory Analysis*

Table II compares the required memory for storing matrices **V** and **D**. The memory usage of the original matrix **A** is also provided. We also provided the memory savings for the case in which **D** is formed in the same fashion as CSSD, but **V** is computed via least squares, as opposed to OMP. RankMap results in up to $77.8\times$ (memory usage) improvement over $\mathbf{A}^T\mathbf{A}$ and $8.6\times$ improvement over the adaptive norm-2 projection-based decomposition. The approximation error for both decomposition methods is set to $\delta_D = 0.1$.

*G. Comparison With Spark*

We have already shown the results based on our implementation of RankMap on GraphLab in Fig. 9(a)–(c). Now, we provide runtime comparisons between RankMap and an
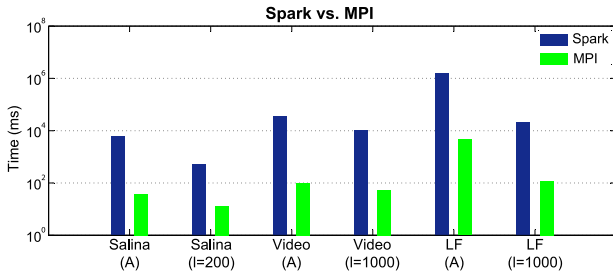
Fig. 10.    *Average runtime per power method iteration for RankMap and a Spark implementation of the baseline method.*

Apache Spark-based implementation of the power method on the baseline **A** versus on the decomposed data **DV**. Recall that the core iterative update function used in power method is provided in (4). We report the average runtime per iteration with Spark and our implementations on the same hardware.

Fig. 10 shows the average runtime per iteration on a cluster of eight nodes, eight core per processor for Salinas, VideoDict, and Light Field (ii) data sets. As expected, our carefully tailored implementation of RankMap based on C++ MPI performed significantly better than Spark by more than two orders of magnitude in some cases. This gap in performance is in part due to our particular implementation of RankMap, which carefully partitions the data, such that the computation per core is balanced and the communication is reduced (see Section VI-C2 for performance bounds). In addition, Spark provides fault tolerance, which causes extra overhead due to data replications.

## VIII. DISCUSSION

This paper introduced RankMap, a novel distributed framework for applying a host of iterative learning algorithms on large-scale dense and structured data sets. Our framework leverages *low-dimensional structure* in data sets in order to quickly map a large data set with dense dependencies into lower dimensional components with sparse representations. We introduce two computational models, a matrix- and graph-based model that can be used to execute distributed learning algorithms. Our framework provides an efficient partitioning of the computational flow that guarantees load balancing and significantly lowers communication overhead. We apply our matrix- and graph-based models to numerous real-world and synthetic data sets and demonstrate significant improvements in the runtime and memory footprint.

There is an unavoidable cost associated with factorizing the data. For extremely large data sets, however, this initial cost can pay off a lot. The performance gain achieved on the subsequent computations justifies the one-time decomposition overhead. For example, we decompose Light Field (ii) data set (Section VII) on a cluster of four nodes (each with 12 cores) on IBM iDataPlex. For $l = 240$, the decomposition is completed in less than 15 min. For ten sample patches (each of length 18 k), the overall reconstruction time is reduced from more than 1000 to below 20 s. Thus, the off-line decomposition overhead can be justified once considering that there are thousands of patches in a single light field. Moreover,

the same dictionary can be used to reconstruct other light field data sets.

There are a number of existing column sampling-based methods that aim to improve the performance of specific learning objectives, such as least squares [37], $\ell_2$-minimization with square root $\ell_1$ penalty [40], and SVM [20]. RankMap is unique in that uses column sampling to improve the performance of a broad class of ML algorithms that operate on the Gram matrix. Moreover, RankMap relies of the sparsity of the decomposition for further improvement in runtime, energy, and memory usage. Finally, to the best of our knowledge, RankMap is the first end-to-end framework that is equipped with open-source supported APIs [3].

Sparse matrix factorization approaches, such as SPCA and KSVD, have objectives similar to CSSD; however, their complexity makes them difficult to apply to massive data sets. As we sample the data set instead of learning a factorization of the data, our proposed decomposition is faster and scalable. While our sampling-based approach is effective, the decomposition phase in RankMap (see Fig. 2) can be readily replaced by other sparse decomposition approaches. Tradeoffs between the time to compute a factorization (via learning or sampling) and how sparse we can make the decomposition are likely to exist. Although outside of the scope of this paper, it would be interesting to study the utility of learning in terms of its later computational benefit.

Our graph-based and matrix-based computational models provide advantages in different data regimes. Thus, it is natural to ask which model to select for data processing. Both models follow the same computational flow and operate only on the nonzeros. In practice, we observe that the matrix-based approach is faster: this is especially true when we exploit sparsity in the decomposition with a sparse matrix-based approach. This is likely due to the fact that the graph-based model requires extra overhead to store and operate on the vertices and edges. The main advantage of the graph-based model is in its reduced communication cost (Section VI-C2). When **V** is completely block diagonal, the communication becomes almost independent of the number of computing nodes $n_c$ and is only proportional to $2l$. In contrast, the communication cost in the matrix-based model is always proportional to $2ln_c$. This difference may result in a better overall performance of the graph-based approach, especially for larger values of $l$ and $n_c$. In general, the performance of each model is highly dependent on the specifications of the available computing nodes, including the communication bandwidth and computation power (e.g., maximum floating point operations per second). Our evaluations in Section VII-E provide further insight into the differences of the two models.

Throughout our experiments, we used FISTA, an accelerated gradient descent method, as an optimizer. Our computation/communication and memory minimizing framework can also be applied to other optimization methods, such as stochastic gradient descent (SGD) [42] and stochastic coordinate descent (SCD) [33]. Both SGD and SCD operate on the entire $m \times n$ data set; however, each iterative update is performed on a subset of rows (as in SGD) or along the columns (in SCD). For this reason, the convergence of

stochastic method is slower. SGD can be integrated within RankMap by breaking the $m$ rows of matrix $\mathbf{A}$ into batches, and performing RankMap's decomposition on each batch. SCD can also be applied to the factorized data set $\mathbf{DV}$ instead of $\mathbf{A}$. In general, RankMap is not limited to a particular optimizer; it is beneficial whenever there is a need to store and/or iteratively perform matrix multiplication on large data sets.

In this paper, we showed how sparse matrix factorization and adaptive sampling can be used to speed up iterative optimization algorithms on large data sets. We have mainly explored RankMap's use for computational gains; however, recent theoretical results have shown that subsampling data can also be beneficial for reducing the generalization error in learning problems [44]. RankMap provides a new computational framework for exploring the gains and tradeoffs that exist when learning from adaptively selected examples from large datasets.

## REFERENCES

[1] *Hyperspectral Remote Sensing Scenes*, accessed on 2014. [Online]. Available: http://www.ehu.eus/ccwintco/index.php?title=Hyperspectral_Remote_Sensing_Scenes

[2] *The Light Field Archive*. [Online]. Available: http://lightfield.stanford.edu/

[3] *Rankmap APIs*. [Online]. Available: https://github.com/azalia/RankMap

[4] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006.

[5] R. G. Baraniuk, V. Cevher, M. F. Duarte, and C. Hegde, "Model-based compressive sensing," *IEEE Trans. Inf. Theory*, vol. 56, no. 4, pp. 1982–2001, Apr. 2010.

[6] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imag. Sci.*, vol. 2, no. 1, pp. 183–202, 2009.

[7] E. J. Candès, M. B. Wakin, and S. P. Boyd, "Enhancing sparsity by reweighted $\ell_1$ minimization," *J. Fourier Anal. Appl.*, vol. 14, nos. 5–6, pp. 877–905, 2008.

[8] K. Chen, "On a class of preconditioning methods for dense linear systems from boundary elements," *SIAM J. Sci. Comput.*, vol. 20, no. 2, pp. 684–698, 1998.

[9] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 33–61, 1999.

[10] C. Cortes, M. Mohri, and A. Talwalkar, "On the impact of kernel approximation on learning accuracy," in *Proc. TAISTATS*, 2010, pp. 113–120.

[11] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Commun. Pure Appl. Math.*, vol. 57, no. 11, pp. 1413–1457, 2004.

[12] G. M. Davis, S. G. Mallat, and Z. Zhang, "Adaptive time-frequency decompositions," *Opt. Eng.*, vol. 33, no. 7, pp. 2183–2191, 1994.

[13] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[14] A. Deshpande, L. Rademacher, S. Vempala, and G. Wang, "Matrix approximation and projective clustering via volume sampling " in *Proc. SODA*, 2006, pp. 1117–1126.

[15] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay, "Clustering large graphs via the singular value decomposition," *Mach. Learn.*, vol. 56, nos. 1–3, pp. 9–33, 2004.

[16] P. Drineas and M. W. Mahoney, "On the Nyström method for approximating a Gram matrix for improved kernel-based learning," *J. Mach. Learn. Res.*, vol. 6, pp. 2153–2175, Dec. 2005.

[17] E. L. Dyer, T. A. Goldstein, R. Patel, K. P. Kording, and R. G. Baraniuk. (2015). "Self-expressive decompositions for matrix approximation and clustering." [Online]. Available: https://arxiv.org/abs/1505.00824

[18] E. L. Dyer, A. C. Sankaranarayanan, and R. G. Baraniuk, "Greedy feature selection for subspace clustering," *J. Mach. Learn. Res.*, vol. 14, no. 1, pp. 2487–2517, 2013.

[19] E. Elhamifar and R. Vidal, "Sparse subspace clustering: Algorithm, theory, and applications," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 11, pp. 2765–2781, Nov. 2013.

[20] S. Fine and K. Scheinberg, "Efficient SVM training using low-rank kernel representations," *J. Mach. Learn. Res.*, vol. 2, pp. 243–264, Mar. 2002.

[21] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, "Spectral grouping using the Nyström method," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 2, pp. 214–225, Feb. 2004.

[22] A. S. Georghiades, P. N. Belhumeur, and D. Kriegman, "From few to many: Illumination cone models for face recognition under variable lighting and pose," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 6, pp. 643–660, Jun. 2001.

[23] A. Ghoting *et al.*, "SystemML: Declarative machine learning on MapReduce," in *Proc. ICDE*, 2011, pp. 231–242.

[24] A. Gittens and M. W. Mahoney, "Revisiting the Nyström method for improved large-scale machine learning," in *Proc. ICML*, 2013, pp. 567–575.

[25] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed graph-parallel computation on natural graphs," in *Proc. OSDI*, 2012, pp. 17–30.

[26] A. G. Gray and A. W. Moore, "'N-body' problems in statistical learning," in *Proc. NIPS*, 2001, pp. 521–527.

[27] G. Guennebaud and B. Jacob. *Eigen, a C++ Template Library for Linear Algebra*, accessed on Oct. 25, 2015. [Online]. Available: http://eigen.tuxfamily.org/

[28] Y. Hitomi, J. Gu, M. Gupta, T. Mitsunaga, and S. Nayar, "Video from a single coded exposure photograph using a learned over-complete dictionary," in *Proc. ICCV*, 2011, pp. 287–294.

[29] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[30] M. Journée, Y. Nesterov, P. Richtárik, and R. Sepulchre, "Generalized power method for sparse principal component analysis," *J. Mach. Learn. Res.*, vol. 11, pp. 517–553, Mar. 2010.

[31] K. Kanatani, "Motion segmentation by subspace separation and model selection," in *Proc. 8th IEEE ICCV*, vol. 2. Jul. 2001, pp. 586–591.

[32] C.-J. Lin, "Projected gradient methods for nonnegative matrix factorization," *Neural Comput.*, vol. 19, no. 10, pp. 2756–2779, 2007.

[33] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar, "An asynchronous parallel stochastic coordinate descent algorithm," *J. Mach. Learn. Res.*, vol. 16, pp. 285–322, Feb. 2015.

[34] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein, "GraphLab: A new parallel framework for machine learning," in *Proc. UAI*, 2010, pp. 340–349.

[35] G. Malewicz *et al.*, "Pregel: A system for large-scale graph processing," in *Proc. SIGMOD*, 2010, pp. 135–146.

[36] K. Marwah, G. Wetzstein, Y. Bando, and R. Raskar, "Compressive light field photography using overcomplete dictionaries and optimized projections," *ACM Trans. Graph.*, vol. 32, no. 4, p. 46, 2013.

[37] X. Meng, M. A. Saunders, and M. W. Mahoney, "LSRN: A parallel iterative solver for strongly over- or underdetermined systems," *SIAM J. Sci. Comput.*, vol. 36, no. 2, pp. 95–118, 2014.

[38] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis*. Hoboken, NJ, USA: Wiley, 2015.

[39] U. Nodelman, C. R. Shelton, and D. Koller. (2012). "Expectation maximization and complex duration distributions for continuous time Bayesian networks." [Online]. Available: https://arxiv.org/abs/1207.1402

[40] V. Pham and L. El Ghaoui, "Robust sketching for multiple square-root LASSO problems," in *Proc. AISTATS*, 2015, pp. 753–761.

[41] R. Ramamoorthi, "Analytic PCA construction for theoretical analysis of lighting variability in images of a Lambertian object," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 10, pp. 1322–1333, Oct. 2002.

[42] B. Recht, C. Re, S. Wright, and F. Niu, "HOGWILD: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. NIPS*, 2011, pp. 693–701.

[43] R. Rubinstein, M. Zibulevsky, and M. Elad, "Efficient implementation of the K-SVD algorithm using batch orthogonal matching pursuit," *CS Technion*, vol. 40, no. 8, pp. 1–15, 2008.

[44] A. Rudi, R. Camoriano, and L. Rosasco, "Less is more: Nyström computational regularization," in *Proc. NIPS*, 2015, pp. 1657–1665.

[45] J.-L. Starck, E. J. Candès, and D. L. Donoho, "The curvelet transform for image denoising," *IEEE Trans. Electron. Packag. Manuf.*, vol. 11, no. 6, pp. 670–684, Jun. 2002.

[46] R. C. Thompson, "Principal submatrices IX: Interlacing inequalities for singular values of submatrices," *Linear Algebra Appl.*, vol. 5, no. 1, pp. 1–12, 1972.

[47] J. Wright, Y. Ma, J. Mairal, G. Sapiro, T. S. Huang, and S. Yan, "Sparse representation for computer vision and pattern recognition," *Proc. IEEE*, vol. 98, no. 6, pp. 1031–1044, Jun. 2010.

[48] J. S. Yedidia *et al.*, "Generalized belief propagation," in *Proc. NIPS*, 2000, pp. 689–695.

[49] M. Zaharia *et al.*, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. NSDI*, 2012, p. 2.

[50] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. HotCloud*, 2010, p. 10.

[51] H. Zou, T. Hastie, and R. Tibshirani, "Sparse principal component analysis," *J. Comput. Graph. Statist.*, vol. 15, no. 2, pp. 265–286, Jun. 2006.

**Azalia Mirhoseini** received the Ph.D. degree from Rice University, Houston, TX, USA, where she was involved in algorithms and architectures for performance efficient data analytics.

She is currently with Google Brain, Mountain View, CA, USA, where she is involved in research on large scale deep learning.

Dr. Mirhoseini is a recipient of the National Gold Medal in Iran Mathematics Olympiad in 2004, the Microsoft Women Graduate Student Scholarship in 2010, the IBM Ph.D. Student Scholarship in 2012, and the Schlumberger Ph.D. Student Fellowship in 2013. Her dissertation received the 2015 Best Ph.D. thesis award at ECE department, Rice University.

**Eva L. Dyer** is currently a Research Scientist with the Department of Physical Medicine and Rehabilitation, Northwestern University, Chicago, IL, USA. Her current research interests include the intersection of machine learning, signal processing, and neuroscience.

Dr. Dyer was a recipient of a NSF Graduate Research Fellowship, a NLM Fellowship in computational biology and medicine, a Presidential Fellowship from the College of Engineering, Rice University, Houston, TX, USA, and a Distinguished Texas Instruments Fellowship. She has also received numerous awards including a Best Ph.D. Presenter Award from the Department of Electrical and Computer Engineering, Rice University, and an Eliahu Jury Award for Undergraduate Scholarship from the University of Miami, Coral Gables, FL, USA.

**Ebrahim M. Songhori** received the B.Sc. degree in computer engineering from the University of Tehran, Tehran, Iran, in 2011, and the M.Sc. degree from Rice University, Houston, TX, USA, in 2014. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Rice University, under the supervision of Prof. F. Koushanfar (University of California at San Diego, La Jolla, CA, USA).

His current research interests include computer architecture, secure multi-party computation, embedded system design, FPGA and ASIC design, and high-performance computing.

**Richard Baraniuk** (S'85–M'93–SM'98–F'01) is currently the Victor E. Cameron Professor of Electrical and Computer Engineering, Rice University, Houston, TX, USA, and the Founder and the Director of OpenStax, Rice University. His current research interests include new theory, algorithms, and hardware for sensing, signal processing, and machine learning.

Dr. Baraniuk has received National Young Investigator Awards from the National Science Foundation and the Office of Naval Research, the Rosenbaum Fellowship from the Isaac Newton Institute of Cambridge University, the ECE Young Alumni Achievement Award from the University of Illinois, the IEEE Signal Processing Society Best Paper, Best Column, Education, and Technical Achievement Awards, and the IEEE James H. Mulligan, Jr. Medal. He is a Fellow of the AAAS.

**Farinaz Koushanfar** received the M.A. degree in statistics and the Ph.D. degree in electrical engineering and computer science from University of California at Berkeley, Berkeley, CA, USA, in 2005.

She was a Professor of Electrical and Computer Engineering, Rice University, Houston, TX, USA. She is currently a Professor and the Henry Booker Faculty Scholar of Electrical and Computer Engineering, University of California at San Diego, La Jolla, where she is directing the Adaptive Computing and Embedded Systems Laboratory. Her M.S. degree is from UCLA and her B.S. from Sharif University of Technology, both in Electrical Engineering. Her current research interests include embedded and cyber-physical systems design, embedded systems security, design automation (DA), and in particular DA of domainspecific/mobile computing, and machine learning applications. She currently serves as an Associate Partner of the Intel Collaborative Research Institute for Secure Computing to aid developing solutions for the next generation of embedded secure devices. She is the Founder of Women ExCEl and a Co-Founder of the Trust-Hub.

Dr. Koushanfar received a number of awards and honors for her research, mentorship and teaching, including the Presidential Early Career Award for Scientists and Engineers from President Obama, the ACM SIGDA Outstanding New Faculty Award, MIT TR-35, and the Young Faculty/CAREER Awards from NSF, DARPA, ONR, and ARO.