Efficient and Distributed Algorithms for Large-Scale Generalized Canonical Correlations Analysis

Xiao Fu[†], Kejun Huang[†], Evangelos E. Papalexakis[*], Hyun-Ah Song[‡],
Partha Pratim Talukdar[⋆], Nicholas D. Sidiropoulos[†], Christos Faloutsos[‡], Tom Mitchell[§]

[†]Department of Electrical and Computer Engineering, University of Minnesota
[*]Computer Science & Engineering Department, University of California Riverside
[⋆]Department of Computational and Data Sciences, Indian Institute of Science
[‡]Computer Science Department, Carnegie Mellon University
[§]Machine Learning Department, Carnegie Mellon University

***Abstract*—Generalized canonical correlation analysis (GCCA) aims at extracting common structure from multiple 'views', i.e., high-dimensional matrices representing the same objects in different feature domains – an extension of classical two-view CCA. Existing (G)CCA algorithms have serious scalability issues, since they involve square root factorization of the correlation matrices of the views. The memory and computational complexity associated with this step grow as a quadratic and cubic function of the problem dimension (the number of samples / features), respectively. To circumvent such difficulties, we propose a GCCA algorithm whose memory and computational costs scale *linearly* in the problem dimension and the number of nonzero data elements, respectively. Consequently, the proposed algorithm can easily handle very large sparse views whose sample and feature dimensions both exceed $100,000$ – while the current approaches can only handle thousands of features / samples. Our second contribution is a distributed algorithm for GCCA, which computes the canonical components of different views in parallel and thus can further reduce the runtime significantly (by $\geq 30\%$ in experiments) if multiple cores are available. Judiciously designed synthetic and real-data experiments using a multilingual dataset are employed to showcase the effectiveness of the proposed algorithms.**

***Keywords*—Lagre-scale generalized canonical correlation analysis, distributed GCCA, multilingual word embeddings**

I. Introduction

Canonical Correlation Analysis (CCA) [1] is used to extract low-dimensional representations from two different views of certain entities (e.g., speech and video features of the same object). CCA has a wide spectrum of applications, including clustering, regression, brain imaging, and natural language processing [2]–[4]. Classical CCA focuses on two views. Generalized CCA (GCCA) [5] aims to handle more than two views, and it has recently drawn renewed attention, motivated by modern applications [6]–[9].

Computationally, GCCA poses very challenging optimization problems. Starting from the 1960s, different GCCA formulations and algorithms have been considered [5], [10], such as the sum-of-correlations (SUMCOR), sum-of-squared-correlations (SSQCOR) and maximal-variance (MAX-VAR) formulations, to name a few. In recent years, along with the increasing ability of collecting more and more multiview data and the demand for integrating and analyzing such data, the study of GCCA algorithms has been gaining momentum [6], [7], [11]. Different approaches including power-method like algorithms [11], block coordinate descent (BCD) combined with local linearization [6], and semidefinite relaxation [7] were proposed to tackle the SUMCOR formulation. Recently, Rastogi *et al* [8] proposed an efficient approximation to the MAX-VAR problem and applied it to multilingual analytics.

Despite the long history of GCCA research, many challenges remain. First, the aforementioned algorithms mostly consider extracting the first canonical component of each view, and then use a deflation method to find the other ones; but deflation suffers from error propagation. Second, many (G)CCA algorithms have serious scalability issues since they employ a whitening process (i.e., multiplying the square roots of the correlation matrices of the views to the data) – which destroys sparsity, can easily exhaust memory resources, and also requires a large number of flops even for moderate-size views. There is a recent push towards designing scalable algorithms for CCA [12], [13], but mostly focusing on the two-view case. Third, to the best of our knowledge, no distributed GCCA algorithm has appeared in the literature. Designing distributed algorithms for GCCA is well-motivated by many reasons, including privacy, confidentiality, and data security/integrity concerns; in addition to better scaling, faster computation, and suitability for cloud computing.

Contributions In this work, our interest lies in the SUMCOR formulation where pairwise highly correlated reduced-dimension views are sought. Our design is under a setting where the views are large sparse matrices. Such a scenario is well-motivated in practice, e.g., in multilingual analytics. Our contributions are as follows:

• **Scalable GCCA**: We propose a highly scalable SUMCOR GCCA algorithm. The proposed algorithm does not need to explicitly instantiate the correlation matrices and whitening matrices, resulting in substantial memory savings. Computationally, the key components of the algorithms are sparse matrix-vector multiplications and singular value decomposition of (typically very) "thin" matrices, which are both

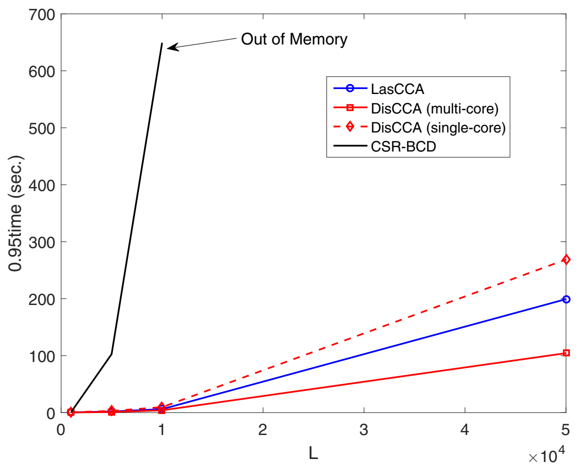

Fig. 1. `0.95_time` (runtime for capturing 95% of the total sum of pairwise correlations among the views) versus size of the views. 5 components are sought; 5 views; each view is a $L \times 0.8L$ matrix; data density $\rho = 5 \times 10^{-3}$. **LasCCA** and **DisCCA** are the proposed algorithms; CSR-BCD is a competitor which uses whitening.

lightweight and suitable for big data analytics. A sneak peek of the experimental results is provided in Fig. 1, where one can verify that our method (**LasCCA**) scales seamlessly to bigger views, while the competitor using whitening runs out of memory when the size of the views gets large. In addition, no deflation process is involved in the proposed algorithm.

• **Distributed GCCA**: We also propose a distributed GCCA algorithm for SUMCOR. To our best knowledge, this is the first distributed algorithm for GCCA. In the distributed algorithm, the views are stored in different nodes of a network, and only a small amount of information needs to be exchanged between the nodes and a coordinator. Most of the computations are carried out locally and in parallel at the nodes. By exploring more nodes/cores, running time can be further reduced by the distributed algorithm by at least 30% compared to that of the proposed scalable algorithm in our experiments (cf. the performance of **DisCCA (multi-core)** in Fig. 1).

Reproducibility: We open-source our code at the first authors' homepage, where a `MATLAB` demo is available.

II. Problem Statement

Let $\boldsymbol{Y}_i \in \mathbb{R}^{L\times M_i}$ denote the ith (centered) view, where $\boldsymbol{Y}_i(\ell,:)$ corresponds to the ℓth data point (or entity) in the ith view, L is the number of entities, and M_i denotes the number of features of the ith view. The SUMCOR GCCA problem can be expressed as [14]:

$$\max_{\{\boldsymbol{Q}_i\in\mathbb{R}^{M_i\times K}\}_{i=1}^{I}} \sum_{i=1}^{I}\sum_{j\neq i}^{I} \mathrm{Tr}(\boldsymbol{Q}_i^T\boldsymbol{X}_i^T\boldsymbol{X}_j\boldsymbol{Q}_j) \tag{1a}$$

$$\text{s.t. } \boldsymbol{Q}_i^T\left(\boldsymbol{X}_i^T\boldsymbol{X}_i\right)\boldsymbol{Q}_i = \boldsymbol{I}_K,\ \forall i; \tag{1b}$$

where $\boldsymbol{X}_i = (1/\sqrt{L})\boldsymbol{Y}_i \in \mathbb{R}^{L\times M_i}$ denotes the ith normalized view, $\boldsymbol{Q}_i \in \mathbb{R}^{M_i\times K}$, and K is the number of canonical components that we seek. By this normalization, $(\boldsymbol{X}_i^T\boldsymbol{X}_i)$ is an estimate of the auto-correlation matrix of the ith view and $(\boldsymbol{X}_i^T\boldsymbol{X}_j)$ an estimate of the cross-correlation matrix between the ith and the jth views; i.e., we seek K-dimensional representations of the views such that they are pair-wise highly correlated, and we aim to extract K canonical components *simultaneously*. Our approach builds upon the block coordinate descent (BCD) framework – i.e., updating each $\boldsymbol{Q}_i$ for $i = 1,\ldots,I$ in turn while fixing the other $\boldsymbol{Q}_j$'s for $j\neq i$. By doing so, the subproblem w.r.t. $\boldsymbol{Q}_i$ can be expressed as

$$\max_{\boldsymbol{Q}_i}\ \mathrm{Tr}\left(\boldsymbol{Q}_i^T\boldsymbol{X}_i^T\sum_{j\neq i}\boldsymbol{X}_j\boldsymbol{Q}_j\right) \tag{2a}$$

$$\text{s.t. } \boldsymbol{Q}_i^T\left(\boldsymbol{X}_i^T\boldsymbol{X}_i\right)\boldsymbol{Q}_i = \boldsymbol{I}_K. \tag{2b}$$

To solve the subproblem (2), it is tempting to do the following: Assume that $\boldsymbol{X}_i^T\boldsymbol{X}_i$ has full rank and thus admits a square root decomposition, i.e., $\boldsymbol{X}_i^T\boldsymbol{X}_i = (\boldsymbol{X}_i^T\boldsymbol{X}_i)^{1/2}(\boldsymbol{X}_i^T\boldsymbol{X}_i)^{1/2}$. Then, let $\boldsymbol{Q}_i = (\boldsymbol{X}_i^T\boldsymbol{X}_i)^{-1/2}\boldsymbol{Z}_i$. We can re-write (2) as

$$\max_{\boldsymbol{Z}_i^T\boldsymbol{Z}_i=\boldsymbol{I}}\ \mathrm{Tr}\left(\boldsymbol{Z}_i^T(\boldsymbol{X}_i^T\boldsymbol{X}_i)^{-1/2}\boldsymbol{X}_i^T\sum_{j\neq i}\boldsymbol{X}_j\boldsymbol{Q}_j\right). \tag{3}$$

In some existing approaches such as those in [6], [7], the key steps are essentially identical to the above change of variables – although they consider the simplest case where $K=1$. The upshot of this reformulation is that Problem (3) is essentially a Procrustes projection problem [15] and can be solved to optimality by applying singular value decomposition (SVD) to $(\boldsymbol{X}_i^T\boldsymbol{X}_i)^{-1/2}\boldsymbol{X}_i^T\sum_{j\neq i}\boldsymbol{X}_j\boldsymbol{Q}_j$. Such a method works well when M_i is small. However, instantiating the whitening matrix $(\boldsymbol{X}_i^T\boldsymbol{X}_i)^{-1/2}$ poses serious complexity issues: First, the matrix $(\boldsymbol{X}_i^T\boldsymbol{X}_i)^{1/2}$ is very likely to be dense even when $\boldsymbol{X}_i$ is sparse. Consequently, computing its inverse requires $\mathcal{O}(M_i^3)$ flops (e.g., when $M_i = 100,000$, this step requires $\mathcal{O}(10^{15})$ flops). More importantly, storing a dense and large $M_i\times M_i$ matrix is almost prohibitive – when $M_i = 100,000$, storing $(\boldsymbol{X}_i^T\boldsymbol{X}_i)^{1/2}$ needs 75 GB memory. In this work, we will propose a new BCD approach that completely avoids instantiating $(\boldsymbol{X}_i^T\boldsymbol{X}_i)^{-1/2}$, and thus is highly scalable in terms of both memory and computational complexity.

III. Large-Scale GCCA (LasCCA)

Instead of applying the change of variables $\boldsymbol{Q}_i = (\boldsymbol{X}_i^T\boldsymbol{X}_i)^{-1/2}\boldsymbol{Z}_i$ and reformulating (2) to (3), we let $\boldsymbol{G}_i = \boldsymbol{X}_i\boldsymbol{Q}_i$ for all i. Note that $\boldsymbol{G}_i \in \mathbb{R}^{L\times K}$ is a very "thin" matrix since in practice $L \gg K$ holds and K, i.e., the number of canonical components sought, is usually small and can be controlled by the designer. Assuming that $\mathrm{rank}(\boldsymbol{X}_i) = M_i$, we have $\boldsymbol{Q}_i = (\boldsymbol{X}_i^T\boldsymbol{X}_i)^{-1}\boldsymbol{X}_i^T\boldsymbol{G}_i$. Substituting the above form of $\boldsymbol{Q}_i$ to Problem (2) leads to the following optimization problem:

$$\max_{\boldsymbol{G}_i}\ \mathrm{Tr}\left(\boldsymbol{G}_i^T\boldsymbol{H}_i\right)$$
$$\text{s.t. } \boldsymbol{G}_i^T\boldsymbol{G}_i = \boldsymbol{I}_K,\quad \boldsymbol{G}_i \in \mathcal{R}(\boldsymbol{X}_i), \tag{4}$$

where $\boldsymbol{H}_i = \boldsymbol{X}_i(\boldsymbol{X}_i^T\boldsymbol{X}_i)^{-1}\boldsymbol{X}_i^T\sum_{j\neq i}\boldsymbol{X}_j(\boldsymbol{X}_j^T\boldsymbol{X}_j)^{-1}\boldsymbol{X}_j^T\boldsymbol{G}_j$, $\mathcal{R}(\boldsymbol{X})$ denotes the range space of $\boldsymbol{X}$, and the constraints in (4) together ensure that Problem (4) and Problem (2) are equivalent. Note that one may also substitute $\boldsymbol{X}_i\boldsymbol{Q}_i = \boldsymbol{G}_i$

Algorithm 1: Large-Scale GCCA (LasCCA)

```
   input : $\{\boldsymbol{X}_i, \boldsymbol{G}_i^{(0)}\}_{i=1}^I$; $K$
1  $r \leftarrow 1$;
2  repeat
3      for $i = 1 : I$ do
4          $\mathcal{G}_i^{(r)} = \{\boldsymbol{G}_1^{(r+1)}, \ldots, \boldsymbol{G}_{i-1}^{(r+1)}, \boldsymbol{G}_{i+1}^{(r)}, \ldots, \boldsymbol{G}_I^{(r)}\}$
5          $\boldsymbol{H}_i^{(r)} \leftarrow$ H_Compute $\left(\{\boldsymbol{X}_i\}_{i=1}^I, \mathcal{G}_i^{(r)}\right)$;
6          $\boldsymbol{U}_i^{(r)}\boldsymbol{\Sigma}_i^{(r)}\boldsymbol{V}_i^{(r)} \leftarrow \mathrm{svd}(\boldsymbol{H}_i^{(r)})$;        $\mathcal{O}(LK^2)$
7          $\boldsymbol{G}_i^{(r+1)} \leftarrow \boldsymbol{U}_i^{(r)}(\boldsymbol{V}_i^{(r)})^T$;
8      end
9      $r \leftarrow r + 1$;
10 until some stopping criterion is reached;
   output: $\{\boldsymbol{G}_i\}$
```

Algorithm 2: H_Compute (Computation of $\boldsymbol{H}_i$)

```
   input : $\{\boldsymbol{X}_i\}$; $\mathcal{G}_i$.
1  for $j = 1, \ldots, I$ do
2      if $j \neq i$ then
3          $\boldsymbol{R}_j \leftarrow \mathrm{CG}(\boldsymbol{X}_j, \boldsymbol{G}_j)$;        $\mathcal{O}(\mathrm{nnz}(\boldsymbol{X}_j)K)$
4          $\boldsymbol{C}_j \leftarrow \boldsymbol{X}_j\boldsymbol{R}_j$;                    $\mathcal{O}(\mathrm{nnz}(\boldsymbol{X}_j)K)$
5      end
6  end
7  $\boldsymbol{P}_i \leftarrow \sum_{j\neq i}\boldsymbol{C}_j$;
8  $\boldsymbol{E}_i \leftarrow \mathrm{CG}(\boldsymbol{X}_i, \boldsymbol{P}_i)$;                $\mathcal{O}(\mathrm{nnz}(\boldsymbol{X}_i)K)$
9  $\boldsymbol{H}_i \leftarrow \boldsymbol{X}_i\boldsymbol{E}_i$;                        $\mathcal{O}(\mathrm{nnz}(\boldsymbol{X}_i)K)$
   output: $\boldsymbol{H}_i$
```

into (2) and write $\boldsymbol{H}_i$ as $\boldsymbol{H}_i = \sum_{j\neq i}\boldsymbol{X}_j(\boldsymbol{X}_j^T\boldsymbol{X}_j)^{-1}\boldsymbol{X}_j^T\boldsymbol{G}_j$; but our reformulation allows us to derive a simple solution as will be seen shortly. The key difference between our reformulation in (4) and that in (3) is that we avoided using the square root decomposition of the correlation matrices, i.e., $(\boldsymbol{X}_i^T\boldsymbol{X}_i)^{-1/2}$. As we will verify later, such a change will significantly reduce memory cost and computational complexity. At first glance, the solution to Problem (4) looks unclear because of the constraint $\boldsymbol{G}_i \in \mathcal{R}(\boldsymbol{X}_i)$. Nevertheless, the solution is in fact simple: Let us first relax the constraint $\boldsymbol{G}_i \in \mathcal{R}(\boldsymbol{X}_i)$ for the moment. Then, an optimal solution to the relaxed problem is the Procrustes projection; i.e.,

$$\boldsymbol{G}_i = \boldsymbol{U}_i\boldsymbol{V}_i^T, \tag{5}$$

where $\boldsymbol{U}_i\boldsymbol{\Sigma}_i\boldsymbol{V}_i = \mathrm{svd}(\boldsymbol{H}_i)$ denotes the economy-size SVD of $\boldsymbol{H}_i$. We note that such $\boldsymbol{G}_i$ satisfies $\boldsymbol{G}_i \in \mathcal{R}(\boldsymbol{H}_i)$ and so $\boldsymbol{G}_i \in \mathcal{R}(\boldsymbol{X}_i)$. Since the solution of the relaxed problem conforms to the constraint of the original problem, (5) is also an optimal solution of (4).

Under this assumption, one can see that the SVD step is not difficult in terms of both computational and memory costs. In fact, $\boldsymbol{H}_i \in \mathbb{R}^{L\times K}$ is a very "thin" matrix (recall that K is usually much smaller than L) – meaning that the memory burden is very light. In addition, computing SVD of $\boldsymbol{H}_i$ costs $\mathcal{O}(LK^2)$ flops, which is merely linear in L. The overall BCD algorithm is presented in Algorithm 1, which we name *large-scale generalized CCA* (LasCCA). At this point, we have not elaborated how to compute $\boldsymbol{H}_i$ but only put an operator called H_Compute$(\cdot,\cdot)$ there (cf. line 2 in Algorithm 1); but computing $\boldsymbol{H}_i$ is the most difficult part as it involves inversions of the correlations. Next, we focus on this key computation.

A. Building Block: Iterative Least Squares

As we have mentioned, computing and storing $(\boldsymbol{X}_j^T\boldsymbol{X}_j)^{-1}$ may not be affordable when constructing $\boldsymbol{H}_i$ at each step. Fortunately, explicit computation of this term is not necessary when calculating $\boldsymbol{H}_i$. In fact, what has been used for constructing $\boldsymbol{H}_i$ is $(\boldsymbol{X}_j^T\boldsymbol{X}_j)^{-1}\boldsymbol{X}_j\boldsymbol{G}_j$ for different j's, which is, again, a very thin matrix that only costs $\mathcal{O}(LK)$ memory. To obtain this thin matrix, consider the following unconstrained linear least squares (LS) problem

$$\min_{\boldsymbol{R}_j} \ \|\boldsymbol{X}_j\boldsymbol{R}_j - \boldsymbol{G}_j\|_F^2 . \tag{6}$$

Assume that $L \geq M_j$ and $\mathrm{rank}(\boldsymbol{X}_j) = M_j$. The optimal solution $\boldsymbol{R}_j^\star$ can be written as $\boldsymbol{R}_j^\star = (\boldsymbol{X}_j^T\boldsymbol{X}_j)^{-1}\boldsymbol{X}_j\boldsymbol{G}_j$. This means that to obtain the key ingredient of building up $\boldsymbol{H}_i$, what one needs to do is to solve a LS problem. The merit of looking at the problem of computing $(\boldsymbol{X}_j^T\boldsymbol{X}_j)^{-1}\boldsymbol{X}_j\boldsymbol{G}_j$ from a LS point of view is that the LS problem can be solved iteratively using very lightweight iterations that can also take advantage of sparsity of $\boldsymbol{X}_j$ – e.g., by stochastic gradient and accelerated gradient. In this work, we propose to employ a powerful tool for unconstrained LS – the *conjugate gradient* (CG) algorithm. There is rich literature on CG since it has been the workhorse for solving large-scale LS problems for decades; see [16] for details. The take-home point is that, to solve Problem (6), CG only requires multiplications of $\boldsymbol{X}_j$ and a matrix of size $M_j \times K$ – if $\boldsymbol{X}_j$ is sparse, such multiplications can be easily carried out with a per-iteration complexity of $\mathcal{O}(\mathrm{nnz}(\boldsymbol{X}_j)K)$ flops, where $\mathrm{nnz}(\cdot)$ counts the non-zero elements. Another advantage of using CG is that even in the worst case, CG is provably convergent within a finite number of iterations [16]; in practice, CG almost always converges to a very good accuracy level within 20 iterations.

Using CG, $\boldsymbol{H}_i$ can be computed efficiently. The corresponding algorithm is presented in Algorithm 2. In short, we first use CG to form $\boldsymbol{R}_j = (\boldsymbol{X}_j^T\boldsymbol{X}_j)^{-1}\boldsymbol{X}_j\boldsymbol{G}_j$ as we explained. Then, $\boldsymbol{C}_j = \boldsymbol{X}_j(\boldsymbol{X}_j^T\boldsymbol{X}_j)^{-1}\boldsymbol{X}_j\boldsymbol{G}_j$ can be formed by multiplying $\boldsymbol{X}_j$ to $\boldsymbol{R}_j$. If $\boldsymbol{X}_j$ is sparse, then this step is easy. Then, the matrix $\boldsymbol{P}_i = \sum_{j\neq i}\boldsymbol{X}_j(\boldsymbol{X}_j^T\boldsymbol{X}_j)^{-1}\boldsymbol{X}_j^T\boldsymbol{G}_j$ can be obtained by summation over the index set $\{1,\ldots,I\}/\{i\}$. By solving $\boldsymbol{E}_i = \arg\min_{\boldsymbol{E}} \|\boldsymbol{P}_i - \boldsymbol{X}_i\boldsymbol{E}\|_F^2$ using CG we obtain $\boldsymbol{E}_i = (\boldsymbol{X}_i^T\boldsymbol{X}_i)^{-1}\boldsymbol{X}_i^T\boldsymbol{P}_i$. Finally, by another multiplication of a sparse matrix $\boldsymbol{X}_i$ and the thin matrix $\boldsymbol{E}_i$, we get $\boldsymbol{H}_i$. We factor out each iteration's processing details in the H_Compute method and Algorithm 1. As one can see, the major computations all have the same complexity order – and it is linear in $\mathrm{nnz}(\boldsymbol{X}_i)$.

IV. Distributed GCCA (DisCCA)

In practice, there are many scenarios in which distributed computing is desirable. For example, sometimes the views may be collected and stored at different nodes within a network, and directly exchanging the views may not be allowed for security and/or legal reasons. Another reason is multi-core parallel computing-based acceleration. Instead of computing $\boldsymbol{G}_i$ sequentially as in Algorithm 1, it may be more appealing

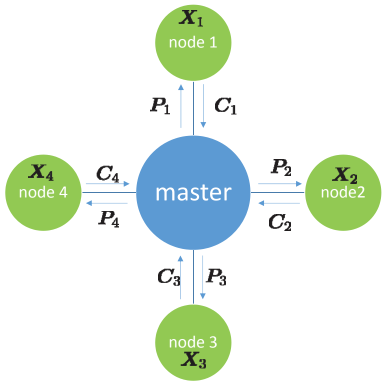

Fig. 2. Architecture of the considered network for DisCCA and information exchange among distributed nodes in DisCCA.

to compute G_i for all i in parallel in a distributed fashion – which may well expedite the whole process when I is large.

Let us consider a network structure as in Fig. 2. There, the nodes can be machines that are located in different places or cores within one machine. The views are stored at the nodes and are not allowed to be exchanged. The master is a coordinator who collects and distributes some derivative information, and we wish to leave the heavy computations to the nodes. To see how we approach this problem, let us consider the following modification to LasCCA. Specifically, instead of updating G_i sequentially and cyclically as in Algorithm 1, the nodes locally update G_i simultaneously, based on the previous iterate, which results in the following H_i-update strategy:

$$H_i^{(r)} \leftarrow \texttt{DH\_Compute}\left(\{X_i\}_{i=1}^I, \{G_j^{(r)}\}_{j\neq i}\right) \tag{7}$$

where `DH_Compute` is a distributed version of Algorithm 2 (cf. Algorithm 3). The update in (7) is different from that in Algorithm 1: In Algorithm 1, the update of $G_i^{(r)}$ uses the information of $G_j^{(r+1)}$ for $j < i$ and $G_j^{(r)}$ for $j > i$, while this update in (7) only uses $G_j^{(r)}$ for all $j \neq i$. By this slight modification, a significant part of (7) can be computed in parallel at different nodes. A detailed implementation of the `DH_Compute` algorithm is presented in Algorithm 3, where one can see that the master only gathers matrices of size $L \times K$, i.e., C_j, and adds them over different sets of j's.

Using `DH_Compute`, the complete distributed implementation of GCCA is easy to derive. After all $G_i^{(r+1)}$'s are computed by different nodes, we update the G_i that brings maximal improvement to the objective function of (1). Using this idea, the whole distributed CCA (DisCCA) algorithm is presented in Algorithm 4. One can see that the lines 6-11 implement the idea of selecting a block to update. In optimization theory, adding these lines makes the algorithm fall into the category of the *maximum block improvement* (MBI) framework [17] which is a greedy variant of the Gauss-Seidel (GS)-type BCD. Compared to GS-type BCD (cf. LasCCA), every update of MBI gives a larger improvement of the objective, which is clearly favorable. On the other hand, MBI "wastes" most of the computations since it computes all the blocks but updates only one. Hence, there is clearly an interesting resource-time trade-off between choosing LasCCA

Algorithm 3: `DH_Compute` (Distributed Computation of H_i)

```
input : $\{X_i\}$; $\{G_i\}$.
1  $R_i \leftarrow \texttt{CG}(X_i, G_i)$;                    (node)
2  $C_i \leftarrow X_i R_i$;                                  (node)
3  node $i$ for all $i$ sends $C_i$ to master;
4  $P_i \leftarrow \sum_{j\neq i} C_j$;                       (master)
5  master sends $P_i$ to node $i$;
6  $E_i \leftarrow \texttt{CG}(X_i, P_i)$;                    (node)
7  $H_i \leftarrow X_i E_i$;                                  (node)
output: $H_i$
```

Algorithm 4: Distributed GCCA (DisCCA)

```
input : $\{X_i, G_i^{(0)}\}_{i=1}^I$; $K$
1  $r \leftarrow 1$;
2  repeat
3      $H_i^{(r)} \leftarrow \texttt{DH\_Compute}\left(\{X_i\}_{i=1}^I, \{G_j^{(r)}\}_{j\neq i}\right)$;
4      $U_i^{(r)}\Sigma_i^{(r)}V_i^{(r)} \leftarrow \text{svd}(H_i^{(r)})$.       (node)
5      $G_i^{(r+1)} \leftarrow U_i^{(r)}(V_i^{(r)})^T$.                         (node)
6      $v_i^{(r)} \leftarrow \text{Tr}((G_i^{(r+1)})^T H_i^{(r)})$;             (node)
7      node $i$ sends $v_i^{(r)}$ to master;
8      $i^\star \leftarrow \arg\max_{i\in\{1,\dots,I\}} \; v_i^{(r)}$;         (master)
9      master sends an update command to node $i^\star$;
10     $G_{i^\star}^{(r+1)} \leftarrow G_{i^\star}^{(r+1)}$;                    (node)
11     $G_i^{(r+1)} \leftarrow G_i^{(r)}$ for $i \neq i^\star$;                 (node)
12     $r \leftarrow r + 1$;
13 until some stopping criterion is reached;
output: $\{G_i\}$
```

and DisCCA in practice. Another remark is that both LasCCA and DisCCA monotonically increase the objective value of (1) following the properties of generic BCD and MBI, which is desired in large-scale GCCA since every iteration makes progress towards the ultimate goal.

V. Experiments

A. Synthetic-Data Experiments

1) Baselines: Our first baseline is an algorithm that employs a change of variables strategy as proposed in [6], [7]. This follows a BCD approach where each subproblem is as in (3). The original algorithms were not designed to handle the $K > 1$ case, but this can be fixed by solving each subproblem in (3) using a Procrustes projection. This baseline will be referred to as *correlation square root-based block coordinate descent* (CSR-BCD). In addition, we also use the *multiview latent semantic analysis* (MLSA) algorithm that was proposed by Rastogi *et al.* [8] to deal with large-scale multiview analysis. MLSA aims at solving the MAX-VAR GCCA problem, which has a different objective function i.e., $\min_{G^TG=I_K,\{Q_i\}} \sum_{i=1}^I \|X_iQ_i - G\|_F^2$. This formulation conceptually shares the same goal with SUMCOR – which is to seek highly correlated X_iQ_i's – and thus using it as a baseline is reasonable.

2) Data Generation: First, we let $Z \in \mathbb{R}^{L\times M}$ be a common latent factor of the different views, where Z is a randomly generated sparse matrix whose non-zero entries follow the i.i.d. zero-mean unit-variance Gaussian distribution. Then, a sparse matrix $A_i \in \mathbb{R}^{M\times M}$ is multiplied to Z, resulting in

TABLE I

AVERAGE CORRELATIONS CAPTURED BY THE ALGORITHMS. $M = 0.8 \times L$; $K = 5$; $I = 5$; $\rho = 5 \times 10^{-3}$; "†" MEANS "OUT-OF-MEMORY"; $v_{\text{opt}} = 100$.

Algorithm	L			
	1,000	5,000	10,000	50,000
LasCCA	99.87	99.30	99.05	99.05
DisCCA	99.60	98.73	98.35	98.24
CSR-BCD	**100.00**	**100.00**	**100.00**	†
MLSA	98.57	63.32	44.60	7.76

$\boldsymbol{X}_i = \boldsymbol{Z}\boldsymbol{A}_i$. The overall sparsity level of $\boldsymbol{X}_i$ is controlled to satisfy experiment specifications. This way, the views have a perfectly correlated common latent factor, i.e., $\boldsymbol{Z}$. We fix $M_1 = \ldots = M_I = M$, and test the algorithms under different sizes and sparsity levels of $\boldsymbol{X}_i$, where the sparsity level ρ_i is defined as $\rho_i = {}^{\text{nnz}(\boldsymbol{X}_i)}/_{LM}$, and we let $\rho_i = \rho$ for all i.

3) Evaluation: We evaluate the synthetic data experiment by observing the captured sum of pairwise correlations ("correlations captured" in short) between the views. Since the data are perfectly correlated in the $\boldsymbol{Z}$-domain, the optimal correlation value is $v_{\text{opt}} = I(I-1)K$. We also define a metric called "`0.95_time`", which records the time that is needed for an algorithm to capture 95% of the optimal value of the total sum of correlations, i.e., $I(I-1)K$.

4) Results: Table I shows the performance of the algorithms under various L's (and M's). Here, we fix $\rho = 5 \times 10^{-3}$ and change L and $M = 0.8 \times L$. We seek $K = 5$ canonical components of each of the $I = 5$ views. MLSA requires first truncating the rank of views to a certain number, and thus we use the first 100 singular values and vectors to approximate each view. We let all the iterative algorithms run for 20 iterations in this experiment, and the results are averaged over 10 random trials. We initialize CSR-BCD, LasCCA and DisCCA using the same random initialization; As one can see from the table, in terms of captured correlations, the CSR-BCD algorithm, gives the best performance when $L \leq 10,000$ – it always reaches the optimal objective value, although the problem is non-convex and NP-hard. However, when $L = 50,000$, the algorithm exhausts the memory quota (32 GB RAM). The proposed algorithms, i.e., LasCCA and DisCCA, give slightly lower objective values compared to that of CSR-BCD, which are already very good – they both capture more than 98% of the total correlations, even under the case CSR-BCD cannot run, i.e., when $L = 50,000$. Another observation is that when $L = 1,000$, MLSA works very well, i.e., 98% of the correlations is captured. However, when L increases, the performance degrades rapidly since the approximation of the views becomes much coarser.

The corresponding `0.95_time` performance of the algorithms in Table I can be found in Fig. 1, which we showed in the beginning of the paper. As one can see, the runtime of CSR-BCD grows rapidly with L increasing. On the other hand, the proposed algorithms scale well. In particular, DisCCA gives the best `0.95_time` performance using a multi-core implementation, which is 104 seconds when $L = 50,000$. LasCCA is slower compared to DisCCA since it only uses one core, taking 199 seconds to capture 95% of the total sum of correlations. Obviously, if we implement DisCCA using a single core, it is slower than LasCCA since DisCCA needs more resources and computations before deciding which block to update. This result suggests that if only a single core is used, LasCCA may be more preferable.

TABLE II

THE AVERAGE CORRELATIONS CAPTURED AFTER 20 ITERATIONS UNDER DIFFERENT ρ'S. $L = 120,000$; $M = 100,000$; $K = 5$; $I = 5$; "†" MEANS "OUT-OF-MEMORY" AND "–" MEANS "NOT APPLICABLE"; $v_{\text{opt}} = 100$.

Algorithm	measure	ρ $(\times 10^{-3})$					
		0.05	0.08	0.10	0.25	0.50	0.75
LasCCA	corr. captured	**99.40**	**99.24**	**99.12**	**98.81**	**98.74**	**98.75**
	`0.95_time`	31.18	36.49	41.88	81.39	136.44	189.40
DisCCA (multi-core)	corr. captured	98.51	98.04	97.93	97.07	96.67	96.85
	`0.95_time`	**13.20**	**16.15**	**20.05**	**43.87**	**85.81**	**128.91**
DisCCA (single-core)	corr. captured	98.51	98.04	97.93	97.07	96.67	96.85
	`0.95_time`	46.06	54.99	65.53	135.33	241.70	337.00
CSR-BCD	corr. captured	†	†	†	†	†	†
	`0.95_time`	–	–	–	–	–	–
MLSA	corr. captured	63.28	55.08	51.44	36.14	22.52	14.87
	`0.95_time`	–	–	–	–	–	–

Table II shows the captured correlations of a larger problem (i.e., $L = 120,000$ and $M_i = 100,000$) under various sparsity levels of $\boldsymbol{X}_i$. In this simulation, CSR-BCD cannot start to run; it leads to out-of-memory under all the ρ's since it destroys sparsity at the very first step. LasCCA and DisCCA, on the other hand, work well under different ρ's: Both algorithms give objective values that are larger than 96 (recall that $v_{\text{opt}} = 100$) under all ρ's. When ρ increases, the captured correlations of both algorithms decrease very gracefully. The performance of MLSA also decreases along with ρ increasing, but more rapidly. In terms of `0.95_time`, one can see that DisCCA (using multiple cores) still outperforms LasCCA as in the previous simulation. The runtimes of both DisCCA and LasCCA grows (roughly) linearly with ρ, which is consistent with our analysis – the major computations in DisCCA and LasCCA have complexity order of $\mathcal{O}(\text{nnz}(\boldsymbol{X}_i)K)$ flops.

B. Real-Data Experiments

1) Multilingual Data: We also test our algorithms on a large-scale parallel corpus of multilingual data. The data were originally introduced in [18] and can be downloaded from http://wordvectors.org/eacl14-data.tar. We have $I = 4$ views, which are constructed from English, Spanish, German and French, respectively. The raw data contain a matrix per language that records the Point-wise Mutual Information (PMI) of the word co-occurrence for that language. We view the rows of the matrix as our data points and the columns as features. We use English words to form our first view, $\boldsymbol{X}_1$, which contains $L = 180,834$ words defined by $M_1 = 130,000$ features. We set $M_1 = \ldots = M_4 = 130,000$, and the features correspond to the columns in each view that have the M_i highest energies. There are 1.21% non-zero entries out of the 2.3508×10^{10} entries of $\boldsymbol{X}_1$. Using dictionaries, we pick out the corresponding words in French, German and Spanish to form $\boldsymbol{X}_2$, $\boldsymbol{X}_3$ and $\boldsymbol{X}_4$, respectively. For those English terms which do not have a translation in $\boldsymbol{X}_i$ for $i > 1$, we simply let the corresponding row of $\boldsymbol{X}_i$ be zero. Consequently, the X_i's are all $180,834 \times 130,000$ sparse matrices.

TABLE III
ENGLISH WORD EMBEDDING EVALUATION. $K = 100$.

TASK	DisCCA	LasCCA	MLSA	SVD
EN-WS-353-REL	**0.5915**	0.5762	0.5462	0.5672
EN-WS-353-SIM	**0.7275**	0.7026	0.6936	0.6777
EN-WS-353-ALL	**0.6465**	0.6353	0.6051	0.6188
EN-MTurk-287	0.6311	0.6365	**0.6763**	0.6068
EN-YP-130	**0.5201**	0.4726	0.448	0.4363
EN-RW-STANFORD	0.434	**0.4464**	0.4361	0.4408
EN-MEN-TR-3k	0.7183	0.7126	0.678	**0.7252**
EN-SIMLEX-999	0.3822	0.4057	**0.419**	0.3438
EN-MTurk-771	**0.6245**	0.6187	0.5919	0.5893
EN-MC-30	0.6901	0.697	0.6198	**0.7435**
EN-RG-65	0.6244	0.6505	0.592	**0.6872**
Average	**0.5991**	0.5958	0.5733	0.5851

TABLE IV
ENGLISH WORD EMBEDDING EVALUATION. $K = 300$.

TASK	DisCCA	LasCCA	MLSVA	SVD
EN-WS-353-REL	**0.6548**	0.5887	0.5374	0.6415
EN-WS-353-SIM	**0.7494**	0.7118	0.6901	0.7417
EN-WS-353-ALL	**0.6888**	0.6509	0.611	0.6804
EN-MTurk-287	**0.6131**	0.568	0.5414	0.548
EN-YP-130	**0.546**	0.5402	0.4781	0.4987
EN-RW-STANFORD	**0.4741**	0.4547	0.4484	0.4606
EN-MEN-TR-3k	0.7615	0.7369	0.7079	**0.7668**
EN-SIMLEX-999	0.4259	**0.456**	0.4354	0.4123
EN-MTurk-771	**0.6796**	0.63	0.6101	0.6481
EN-MC-30	0.8134	0.7791	0.7422	**0.8472**
EN-RG-65	**0.7605**	0.7255	0.7635	0.7214
Average	**0.6516**	0.6220	0.5969	0.6333

2) Evaluation: In the real experiments, we evaluate the quality of the output embeddings of the English words. We employ the evaluation tool provided at wordvectors.org [19], which automatically evaluates the embedded English words (i.e., $\boldsymbol{X}_1\boldsymbol{Q}_1$) on several word embedding tasks by comparing the algorithm-learned embeddings with the judgment of humans. The outputs are scores between zero and one, and a score equal to one means a perfect alignment between the learned result and human judgment. In all the real experiments, we evaluate the results on the first 11 tasks of wordvectors.org.

3) Results: In Tables III and IV, we show the scores given by DisCCA, LasCCA and MLSA when $K = 100$ and $K = 300$, respectively. We run the proposed algorithms for 10 iterations and observe the results. For MLSA, we follow the procedure in [8] to truncate the rank of the views to 640 as pre-processing. We use the results given by MLSA to initialize our algorithms. The first observation is that all the algorithms work better using a larger K, which is reasonable since more dimensions are used. One can see that DisCCA gives the best evaluation scores on 6 and 8 tasks under the two cases, respectively. SVD also gives good scores and performs the best on 3 and 2 tasks, respectively. LasCCA works reasonably well but slightly worse relative to DisCCA. This might be because on such a large-scale problem, DisCCA's greedy update strategy helps get to a good solution quicker, and LasCCA may need more iterations to obtain an equally good solution. MLSA is not very promising in such case. The results in this subsection are encouraging, since they clearly show that using the considered GCCA formulation and the proposed algorithms, the performance of large-scale multiview data analytics has been improved.

VI. CONCLUSION

In this paper, we investigated the problem of computing canonical components of large-scale sparse multiview data. A BCD-based algorithmic framework was proposed based on a judicious equivalent reformulation of the SUMCOR GCCA problem. Under this framework, two highly scalable algorithms, namely, LasCCA and DisCCA, based on different update strategies were proposed. DisCCA is also the first distributed algorithm for large-scale GCCA. Simulations and real experiments show that both of the proposed algorithms scale well to large-size problems and give promising results when applied to a word embedding problem.

ACKNOWLEDGMENT

This work is supported by the National Scientific Foundation (NSF) under Project NSF IIS-1247632, NSF IIS-1447788, and NSF IIS-1247489.

REFERENCES

[1] H. Hotelling, "Relations between two sets of variates," *Biometrika*, vol. 28, no. 3/4, pp. 321–377, 1936.

[2] Q. Zhang, L. Zhang, B. Du, W. Zheng, W. Bian, and D. Tao, "MMFE: Multitask multiview feature embedding," in *ICDM 2015*. IEEE, 2015, pp. 1105–1110.

[3] S. Bickel and T. Scheffer, "Multi-view clustering." in *Proc. ICDM 2004*, vol. 4, 2004, pp. 19–26.

[4] Y. Cui, X. Z. Fern, and J. G. Dy, "Non-redundant multi-view clustering via orthogonalization," in *Proc. ICDM 2007*, 2007, pp. 133–142.

[5] J. D. Carroll, "Generalization of canonical correlation analysis to three or more sets of variables," in *Proceedings of the 76th annual convention of the American Psychological Association*, vol. 3, 1968, pp. 227–228.

[6] A. Tenenhaus and M. Tenenhaus, "Regularized generalized canonical correlation analysis," *Psychometrika*, vol. 76, no. 2, pp. 257–284, 2011.

[7] J. Rupnik, P. Skraba, J. Shawe-Taylor, and S. Guettes, "A comparison of relaxations of multiset cannonical correlation analysis and applications," *arXiv preprint arXiv:1302.0974*, 2013.

[8] P. Rastogi, B. Van Durme, and R. Arora, "Multiview LSA: Representation learning via generalized cca," in *NAACL*, 2015.

[9] R. Arora and K. Livescu, "Multi-view learning with supervision for transformed bottleneck features," in *Proc. ICASSP 2014*, 2014, pp. 2499–2503.

[10] J. R. Kettenring, "Canonical analysis of several sets of variables," *Biometrika*, vol. 58, no. 3, pp. 433–451, 1971.

[11] L.-H. Zhang, L.-Z. Liao, and L.-M. Sun, "Towards the global solution of the maximal correlation problem," *Journal of Global Optimization*, vol. 49, no. 1, pp. 91–107, 2011.

[12] Y. Lu and D. P. Foster, "Large scale canonical correlation analysis with iterative least squares," in *NIPS*, 2014, pp. 91–99.

[13] Z. Ma, Y. Lu, and D. Foster, "Finding linear structure in large datasets with scalable canonical correlation analysis," in *ICML 2015*, 2015.

[14] D. R. Hardoon, S. Szedmak, and J. Shawe-Taylor, "Canonical correlation analysis: An overview with application to learning methods," *Neural computation*, vol. 16, no. 12, pp. 2639–2664, 2004.

[15] P. Schönemann, "A generalized solution of the orthogonal Procrustes problem," *Psychometrika*, vol. 31, no. 1, pp. 1–10, 1966.

[16] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," 1994.

[17] B. Chen, S. He, Z. Li, and S. Zhang, "Maximum block improvement and polynomial optimization," *SIAM Journal on Optimization*, vol. 22, no. 1, pp. 87–107, 2012.

[18] M. Faruqui and C. Dyer, "Improving vector space word representations using multilingual correlation." Association for Computational Linguistics, 2014.

[19] ——, "Community evaluation and exchange of word vectors at wordvectors.org," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Baltimore, USA: Association for Computational Linguistics, June 2014.