Fast Multilevel Co-Clustering

Haifeng Xu*
Centre for Computational Mathematics
University of Waterloo
Canada
haifeng.ustc@gmail.com

Hans De Sterck
Department of Applied Mathematics
University of Waterloo
Canada
hdesterck@uwaterloo.ca

Geoffrey Sanders
Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, USA
sanders29@llnl.gov

ABSTRACT

There are clear indications that many online social networks contain multilevel overlapping cluster structure, but it is difficult to unravel this structure using existing methods. We propose new fast algorithms for finding the multilevel overlapping co-cluster structure of feature matrices that encode social network relations. Starting from the weighted bipartite graph structure of the feature matrix, the algorithms use new graph agglomeration procedures to recursively coarsen the bipartite graphs that represent the relations between the co-clusters on increasingly coarser levels. New fast heuristic coarsening routines are described that circumvent the bottleneck of all-to-all similarity computations by exploiting measures of direct connection strength between row and column variables in the feature matrix. For traditional (single-level) co-clustering problems of gene expression data, our algorithms compare favourably to current methods in terms of speed, scalability and accuracy. Our approach uncovers multilevel overlapping cluster structure in a proof-of-concept application to a data set that relates LinkedIn users to their skills and expertise. The scalability of our methods is illustrated on problems with input size up to tens of millions of data elements, where our fastest method needs about a second and produces more accurate results than a leading existing method that requires more than 1,000 seconds.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Clustering

Keywords

Co-Clustering, Multilevel Methods, Online Social Networks

1. INTRODUCTION

Hierarchical organization is omnipresent in complex networks in nature and society [12]. Often this hierarchical organization is 'hard-coded' into societal structures. For example, universities (where professors do research and teach students a variety of skills) consist of faculties, departments, research groups, etc. These hierarchical structures are overlapping; for example, some professors may be active in multiple departments or faculties, and many skills (often even the more specialized ones) are taught in multiple degree programs. In a similar way, it is to be expected that many of the currently emerging online social networks also contain inherent overlapping hierarchical organization, in particular when they focus on a specific dimension of the human condition, like, e.g., the professional dimension. Consider for example the LinkedIn social network, where users connect to their business relations and acquaintances, and list user-defined 'skills and expertise' on their user profiles that can be endorsed by their connections. Similar to the case of universities, it is clear that in a social network like LinkedIn there must be hierarchical overlapping groups of users with similar skills and professions, and hierarchical overlapping groups of skill keywords that characterize professional groups.

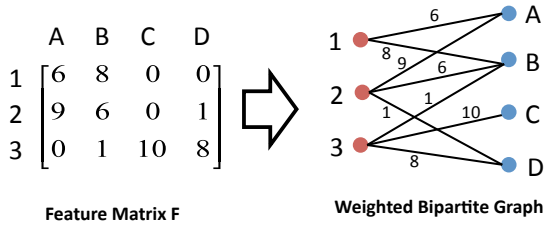

Figure 1: Feature matrix F and induced weighted bipartite graph. Red dots correspond to row variables of F, and blue dots to column variables. The rows and columns may, e.g., represent LinkedIn users and skills, with the weights indicating how often a user's skill was endorsed by the user's connections.

However, in contrast to the example of universities, in emerging social networks this hierarchy is not 'hard-coded' into the structure of the network; if it were, it would seriously impede the growth and dynamical evolution of these networks. Since the hierarchy is not explicitly hard-coded into the structure of the network but is nevertheless present, it is at once a very interesting and a challenging problem to try to automatically generate a representation of this hierarchy from the social network data. This paper addresses this problem by proposing new fast algorithms for finding the multilevel overlapping co-cluster structure of nonnegative feature matrices that encode social network relations.

Our new fast multilevel co-clustering (FMCC) algorithms

*Currently at Department of Computer Science, University of Southern California.

KDD'14, August 24–27, 2014, New York, USA.

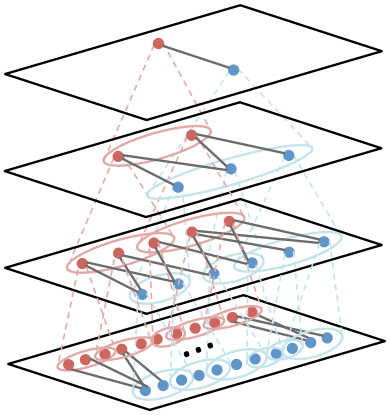

Figure 2: Bipartite graph hierarchy obtained by the FMCC algorithms. The input feature matrix is located at the bottom of the diagram.

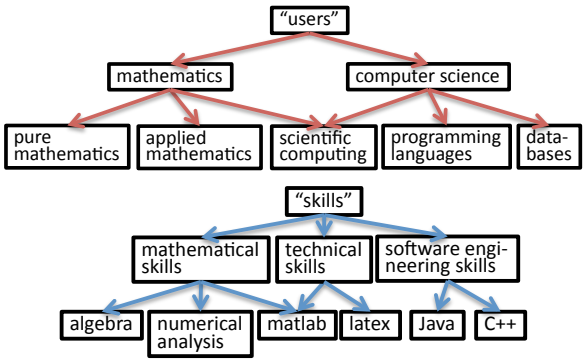

Figure 3: Joint taxonomy of users and skills, corresponding to the top three levels of Fig. 2.

are inspired by algebraic multigrid (AMG) solvers for linear systems [2, 3]. Starting from the weighted bipartite graph structure that is induced by the feature matrix F (Fig. 1), our algorithms proceed by recursively coarsening the bipartite graphs that represent the relations between the co-clusters on increasingly coarser levels (Fig. 2). The graph nodes on increasingly coarser levels represent overlapping groups (co-clusters) of nodes on the next finer level (Fig. 2), and the hierarchical relations between the variables at increasingly coarser levels can be represented in tree-like structures, both for the row variables (red dots in Figs. 1 and 2) and for the column variables (blue dots). For instance, Fig. 3 gives a hypothetical example of what the top levels of these tree-like structures could look like in the case of a subnetwork of LinkedIn users that would contain mathematicians and computer scientists. At a high (coarse) level the users are clustered into two groups corresponding to mathematicians and computer scientists, and each of these has subgroups. Note that the 'mathematics' and 'computer science' co-clusters may overlap, since users in the 'scientific computing' subgroup (for example) may be considered partially mathematicians and partially computer scientists (Fig. 3 top part). Similarly, the skills are clustered in high-level groups and subgroups with possible overlap (Fig. 3 bottom part), with the relations between the groups of users and groups of skills at any level encoded in the weighted bipartite graph at that level (Fig. 2). Note also the connection to ontology generation. At any level the number of row clusters does not necessarily equal the number of column clusters.

Our goal in this paper is to develop efficient multilevel co-clustering algorithms that automatically produce hierarchical co-clusterings as in the tree-like taxonomies of Fig. 3, with coarse bipartite graphs that describe the connections between the co-clusters at all levels of the hierarchical clustering as in Fig. 2. Our work is situated in the general area of co-clustering (or biclustering), which has important applications, for example in analysis of microarray data [11] and in text mining [6]. The goal is to simultaneously find clusters in the row and column variables, and to obtain information on how these clusters relate. While many algorithms have been developed for this problem, most algorithms are expensive and do not scale well to large problem sizes; most also are unable to uncover hierarchical levels of co-clusters.

Main contributions of this paper:

- We propose new fast agglomerative algorithms for finding the multilevel overlapping co-cluster structure of nonnegative feature matrices F by recursively coarsening the bipartite graph induced by F.
- Two new fast heuristic coarsening routines are described that circumvent the bottleneck of all-to-all similarity computations by exploiting measures of direct connection strength between row and column variables in F.
- We show that our FMCC algorithms compare favourably in terms of accuracy to current methods for traditional (single-level) co-clustering of microarray data.
- Scalability tests for problems with tens of millions of data elements show that FMCC is more accurate and up to 1,000 times faster than leading competitors.
- We demonstrate how our approach can uncover multilevel overlapping cluster structure in a proof-of-concept application to a data set that relates LinkedIn users to their skills and expertise, presenting, to our knowledge, the first analysis of this kind for LinkedIn data.

2. RELATED WORK

Our work is most directly related to AMG methods for linear systems that describe discretizations of Laplacian partial differential equations (PDEs) on unstructured computational meshes [3, 2]. AMG methods coarsen these discretized Laplacian operator matrices to increasingly coarser meshes, and since these operator matrices can be interpreted as graph Laplacian matrices, the AMG process is really equivalent to a graph coarsening process, which can be extended to more general graphs than the graphs arising from discretizing Laplacian PDEs. This was first realized by Brandt and co-workers, who applied AMG graph coarsening approaches to graph-based image segmentation and to clustering and manifold detection [13, 9]. Only recently AMG has been applied to rectangular (instead of square) matrices, in the context of low-rank matrix decompositions [5]. This work used the classical approach of coarsening row and column variables by computing similarities between all rows and between all columns, which is expensive. Compared to existing clustering and co-clustering methods, our algorithms are hierarchical and agglomerative, they produce overlapping ('fuzzy') clusters, and they employ heuristics that aim to provide relevant agglomerates with modest computational effort and good scalability properties. The graph coarsening approach is also closely related to clustering by normalized cut minimization [13, 9]. Our algorithms are more efficient and more scalable than standard hierarchical

approaches [18] because we agglomerate many nodes in each recursive step using fast and accurate heuristics. AMG approaches have been shown to scale in parallel to very large problem sizes [1], which now reach more than 1 trillion unknowns on more than 1 million compute cores. Since our co-clustering algorithms (only described in serial in this paper) use similar building blocks, they also have tremendous potential for scalability in parallel. In the numerical evaluation section of this paper, we compare our new methods with existing methods. We compare extensively with nonnegative matrix factorization (NMF), which has been used as a clustering or co-clustering method in a variety of fields [16, 4, 7]. NMF often gives accurate results, but it is expensive and does not scale well to large problem sizes. In [8] hierarchical document trees are constructed using an efficient partitional rank-2 NMF approach, but clusters are not overlapping and no coarse bipartite graphs are constructed relating row to column clusters. We also compare with standard hierarchical clustering algorithms [18] and with specialized co-clustering algorithms for microarray data [11].

3. FAST MULTI-LEVEL CO-CLUSTERING

3.1 Problem Statement and Basic Ideas

PROBLEM 1. *Let $X = \{x_1, ..., x_m\}$ and $Y = \{y_1, ..., y_n\}$ be two variable sets and $F \in \mathbb{R}^{m \times n}$ the nonnegative feature matrix relating them. We are interested in simultaneously finding the hierarchical clustering of X and Y based on F.*

Starting from feature matrix $F \in \mathbb{R}^{m \times n}$, the goal of the FMCC algorithms is to generate a hierarchy of increasingly coarser bipartite graphs as in Fig. 2, with the co-cluster nodes on all levels forming overlapping clusters of nodes on the next finer level, resulting in tree-like structures as in Fig. 3. Consider the resulting hierarchy with n_{lev} levels, $l = 1, 2, \ldots, n_{lev}$, where $l = 1$ refers to the finest level. FMCC determines coarse feature matrices $F^{[l]}$ of size $m^{[l]} \times n^{[l]}$ on all levels ($l = 1, 2, \ldots, n_{lev}$), where $m^{[l]}$ denotes the number of x-points on level l, and $n^{[l]}$ denotes the number of y-points. Throughout this paper, we use the terms node, point and variable interchangeably. Note that $F^{[1]} = F$, $m^{[1]} = m$ and $n^{[1]} = n$. The $F^{[l]}$ matrices encode the strength relations (weighted bipartite graphs) between the nodes on each level l as illustrated in Fig. 2. In the recursive coarsening process, new clusters are initiated on the current fine level by judiciously selecting *seed points* on the current fine level (each seed point generates a new cluster). These seed points (on the current fine level) are called C-points (for 'coarse points'), and the points that are not chosen as seed points are called F-points (for 'fine points'). We define coarse sets $C_x^{[l]}$ and $C_y^{[l]}$ on all levels l that list the seed points that generate new x-clusters and y-clusters on level l, which will be represented by the nodes on levels $l+1$ ($|C_x^{[l]}| = m^{[l+1]}$ and $C_y^{[l]} = n^{[l+1]}$). The sets $F_x^{[l]}$ and $F_y^{[l]}$ contain lists of points that are not chosen as seed points for clusters. FMCC generates membership matrices $M_x^{[l]}$ of size $m^{[l]} \times m^{[l+1]}$ ($l = 1, 2, \ldots, n_{lev} - 1$), with elements between 0 and 1 and row sums 1, where row i of $M_x^{[l]}$ encodes how much fine-level variable $x_i^{[l]}$ on level l belongs to each of the $m^{[l+1]}$ x-clusters that are represented by the nodes on level $l+1$ (i.e., we use a concept of overlapping, 'fuzzy' membership, encoding the relations between coarse and fine nodes in the tree-like structures of Figs. 2 and 3)). Similarly, the $M_y^{[l]}$ are $n^{[l]} \times n^{[l+1]}$ matrices with row sums 1 that encode the memberships of the y-clusters on levels l.

In what follows, we describe how our algorithms coarsen the bipartite graph induced by $F = F^{[1]}$ to produce a coarsened bipartite graph on the first coarse level (level 2). This is done recursively in the multilevel co-clustering algorithm (see Fig. 2). For simplicity we drop the superscript 1 for the fine-level quantities, i.e., we use F, C_x, C_y, F_x, F_y, M_x and M_y. The bipartite graph induced by matrix F is called G_F, and $A \in \mathbb{R}^{(m+n)\times(m+n)}$ is the adjacency matrix of G_F:

$$A = \begin{bmatrix} 0_{m\times m} & F_{m\times n} \\ (F^T)_{n\times m} & 0_{n\times n} \end{bmatrix}. \quad (1)$$

For any matrix Q, q_{ij}, Q_i, and Q^j denote the (i,j)th entry, the ith row and the jth column of Q. We use a subscript c to refer to the feature matrix on the first coarse level (level 2), $F_c = F^{[2]}$, to the coarse adjacency matrix on that level, A_c, and to the numbers of clusters on that level, m_c and n_c.

Our basic approach, inspired by AMG, is to form the coarse bipartite graph by computing its adjacency matrix, A_c, from A, using a triple matrix-matrix product of the form

$$A_c = P^T A P, \quad (2)$$

where the matrix $P \in \mathbb{R}^{(m+n)\times(m_c+n_c)}$, which is called *interpolation matrix* in AMG, encodes how F-points are related to C-points and is formed based on the weights between them as given by F. AMG-style algorithms consist of two main parts: a procedure to choose choose C-points (seed points for new clusters), and a procedure to determine P (once the C-points are known). The procedure to choose C-points is called C/F-splitting, since it splits the points on the finest level into the sets of C-points and F-points.

Interpolation matrix P needs to be chosen judiciously; in particular, we choose forms of P that maintain the bipartite graph structure on the coarse level, i.e., we choose P such that A_c obtains the form

$$A_c = \begin{bmatrix} 0_{m_c\times m_c} & (F_c)_{m_c\times n_c} \\ (F_c^T)_{n_c\times m_c} & 0_{n_c\times n_c} \end{bmatrix}. \quad (3)$$

Equation (2) describes a graph agglomeration process: coarse x-node i is related to coarse y-node j in F_c by summing the fine-level connection weights (from F) between the fine nodes that are connected to cluster i (as encoded in P) and the fine nodes that are connected to cluster j; this process takes into account the overlaps between the clusters. In what follows we will propose two fundamentally different ways to specify P, with different properties in terms of efficiency and accuracy. Before discussing how to form P, we will first describe new efficient strategies for selecting seed points for new clusters in the co-clustering framework (C/F-splitting).

3.2 Cluster Seed Point Selection

Algorithms 1, 2 and 3 give three procedures for C/F-splitting that are considered in this paper. Algorithm 1 (Classical C/F-splitting) employs the standard approach of computing all-to-all similarities between all x-points and between all y-points, and for this reason is often prohibitively expensive. We propose the new Algorithms 2 (Separate C/F-splitting) and 3 (Alternating C/F-splitting) that avoid the all-to-all similarity bottleneck by exploiting measures of direct connection strength between row and column variables in the feature matrix. All three C/F-splitting algo-

Algorithm 1 Classical C/F-splitting

Input: feature matrix F, position parameter $\rho \in [0,1]$
Output: C_x, F_x, C_y, F_y

1: Create strength matrix S_x for x-points based on $D_x = FF^T$, and S_y for y-points based on $D_y = F^TF$.
2: Sort x-points and y-points in increasing order according to their number of strong connections in S_x and S_y.
3: Initialize all x-points to be unassigned.
4: Choose x_c at position ρ in the unassigned ordered list, and assign x_c to C_x; for any unassigned x_i that is strongly connected to x_c in S_x, assign x_i to F_x.
5: Repeat step 4 until all x-points are assigned.
6: Repeat steps 3 to 5 for y-points.

rithms employ the following AMG-inspired strength of connection heuristic:

DEFINITION 1. *Given a (square or rectangular) matrix Q and a strength threshold $\theta \in (0,1]$, we say that row variable x_i is strongly connected to column variable y_j if $q_{ij} \geq \theta\,(\max_k q_{kj})$ or $q_{ij} \geq \theta\,(\max_k q_{ik})$. We denote by S the binary strength matrix of the same size as Q that has 1s in locations of strong connections, and 0s otherwise.*

Classical C/F-splitting (Algorithm 1), as in [5], forms similarity matrices $D_x = FF^T$ and $D_y = F^TF$ between row and column variables, respectively, and then applies classical C/F-splitting approaches from AMG to the row and column variables separately, using the concept of strong connections in the D_x and D_y matrices. Position parameter $\rho \in [0,1]$ determines whether points with few (small ρ) or many (large ρ) strong connections are preferentially chosen as seed points for new clusters. It is often beneficial to choose ρ close to 1 because points with many strong connections are similar to many other points and may thus be considered good representatives for a cluster that groups points with that similarity. However, in other cases it may be better to make sure that small clusters generate seed points first (ρ small), so the optimal value depends on the application. Points strongly connected to a new seed point become F-points since they can now be represented on the coarse level by that C-point (and potentially also other C-points that may be selected later), and thus do not need to start their own cluster. These F-points will be made part of the clusters they are similar to in the second, bipartite graph coarsening part of the algorithm. Algorithm 1 may lead to desirable C/F-splittings, but it is expensive. In our presentation of the algorithm components here in Section 3 we will for simplicity comment on computational cost for the specific case of a dense feature matrix F of size $m \times m$; Section 4 contains a more comprehensive discussion of algorithm complexity that also discusses sparse and rectangular feature matrices. For a dense feature matrix F of size $m \times m$, the computational cost of classical C/F-splitting algorithm1 is $O(m^3)$, which is often prohibitive when m is large.

Separate C/F-splitting (Algorithm 2) avoids the $O(m^3)$ all-to-all similarity computation by computing a strength matrix S directly for the feature matrix F according to Definition 1. The x-points and y-points are then coarsened separately. Seed points for the row and column clusters are chosen according to the number of strong connections in S, and F-points are assigned based on the fraction of common strong connections they have with the seed points, according to user-specified overlap ratios α_x and $\alpha_y \in (0,1]$ (which we normally choose around 0.5). This process is cheaper than

Algorithm 2 Separate C/F-splitting

Input: feature matrix F and its strength matrix S, position parameter $\rho \in [0,1]$, overlap ratio $\alpha_x, \alpha_y \in (0,1]$
Output: C_x, F_x, C_y, F_y

1: Sort x-points and y-points in increasing order according to their number of strong connections in S.
2: Initialize all x-points to be unassigned.
3: Choose an x-point at position ρ in the unassigned and ordered list, say x_c, and assign x_c to C_x; for any unassigned point x_i, if $\frac{(S_i, S_{x_c})}{\sum_j S_{ij}} \geq \alpha_x$, assign x_i to F_x.
4: Repeat step 4 until all the x-points are assigned.
5: Repeat steps 2 to 4 for y-points.

classical C/F-splitting because it effectively computes similarities in the binary S matrix, and, most importantly, similarities need only be computed to the seed points, resulting in a complexity of $O(m^2 m_c)$ for the case of the square and dense feature matrix, where m_c is the number of clusters on the first coarse level. Note that typically $m_c \ll m$ (using *aggressive* coarsening), but m_c may depend on m. Nevertheless, separate C/F-splitting is often significantly faster and more scalable than classical C/F-splitting, and the numerical results presented below show it gives accurate results.

Algorithm 3 Alternating C/F-splitting

Input: feature matrix F and its strength matrix S, position parameter $\rho \in [0,1]$
Output: C_x, F_x, C_y, F_y

1: Sort the x-points and y-points in increasing order according to their number of strong connections in S. Initialize all x, y-points to be unassigned.
2: If all the x-points are assigned, go to step 4;
Otherwise: choose the x-point at position ρ in the unassigned and ordered list, say x_c, and assign it to the set C_x; Assign all the unassigned y-points that are strongly connected to x_c to the set F_y.
3: Pick the y-point at position ρ in the ordered list of $\{y_j | S_{x_c j} = 1\}$, say y_c. Then, re-assign y_c (which is in F_y) to C_y; Assign all the unassigned x-points that are strongly connected to y_c to the set F_x.
4: If all the y-points are assigned, go to step 5; Otherwise, execute step 2 and 3, with the role of x and y exchanged.
5: Repeat steps 2 to 4 until all the x, y-points are assigned.

Alternating C/F-splitting (Algorithm 3) also computes a strength matrix S directly for the feature matrix F, and it further reduces the computational cost by avoiding the computation of any similarities between row variables or between column variables. Instead, our idea is to directly use the strength matrix S, not only to select seed points, like in the Separate C/F-Splitting method, but also to infer F-points. This proceeds in an alternating fashion: we first select a seed point for a new cluster among the x-points, and then assign all unassigned y-points that are strongly connected to this seed point to F_y. We then select a seed point for a new cluster among the y-points, and assign all unassigned x-points that are strongly connected to this seed point to F_x. This alternating procedure is executed repeatedly until all row and column variables are assigned as C-points or F-points. In this way, we indirectly use the information encoded in S to determine which row and column variables should become seed points of new clusters rather than becoming F-points.

The computational cost of this procedure is only $O(m^2)$ for the case of a square dense matrix, since it is dominated by the $O(m^2)$ cost of building S. Numerical results show that this C/F-splitting leads to accurate results, despite being much faster than Algorithms 1 and 2. One additional step is required in Algorithm 3 to assure that every point in C_x is strongly connected to a point in C_y, and vice versa, which is required for building the coarse bipartite graph feature matrix (to avoid zero rows or columns which lead to problems in normalization). We achieve this by making sure that any newly assigned point in C_x is strongly connected to a point in C_y in step 4 of the algorithm, and similarly for newly assigned points in C_y. As a consequence, the number of clusters in the x-variables and in the y-variables on any level is the same for Algorithm 3. This makes Algorithm 3 less general than Algorithms 1 and 2, but this limitation is not unreasonable (it also exists, for example, for NMF), and the significantly lower cost of Algorithm 3 makes it an attractive option especially for large data sizes.

3.3 Choosing Interpolation

Once the C/F-splittings are known, P can be determined. **Membership matrices:** First define the matrix $P_{xy} \in \mathbb{R}^{m\times n_c}$ as the submatrix of F that consists of all columns F^j with $j \in C_y$, and $P_{yx} \in \mathbb{R}^{n\times m_c}$ as the *transpose* of the submatrix of F that consists of all rows F_i with $i \in C_x$. The membership matrices M_x and M_y are then formed as follows: first compute $M_x = FP_{yx}$, and then linearly normalize each row of $M_x$to sum up to 1; and compute $M_y = F^T P_{xy}$, and then linearly normalize each row of M_y to sum up to 1. The following property can be verified easily (see [15]):

THEOREM 1. *If each row and column of F has at least one non-zero, then the same holds for the rows and columns of M_x and M_y under Separate C/F-Splitting (Algorithm 2) and under Alternating C/F-Splitting (Algorithm 3).*

Diagonal Interpolation: Our first choice for P is

$$P = \begin{bmatrix} M_x & 0 \\ 0 & M_y \end{bmatrix} \in \mathbb{R}^{(m+n)\times(m_c+n_c)}. \quad (4)$$

This choice guarantees that A_c has the form (3), with

$$F_c = (M_x)^T F M_y = ((P_{yx})^T F^T) F (F^T P_{xy}). \quad (5)$$

Anti-Diagonal Interpolation: Our second, novel, approach is to choose P as

$$P = \begin{bmatrix} 0 & P_{xy} \\ P_{yx} & 0 \end{bmatrix} \in \mathbb{R}^{(m+n)\times(m_c+n_c)}. \quad (6)$$

In this case

$$F_c = (P_{yx})^T F^T P_{xy}. \quad (7)$$

Note that the cost of (7) is much smaller than (5), which contains the triple matrix product $F^T F F^T$ (instead of a single factor F in (7)). This cost saving results from our new idea to use connections between fine x-points and coarse y-points (and vice versa), as encoded directly in F, to specify the weights of F that should be agglomerated in determining the coarse bipartite graph weights using $A_c = P^T A P$. This is in contrast to using connections between fine and coarse x-points (M_x), and between fine and coarse y-points (M_y), which are not directly encoded in F, but require computing similarities (parts of FF^T and $F^T F$), which is significantly more expensive. When using anti-Diagonal interpolation, M_x and M_y can be computed as above, but to save work, one can only compute them on the coarsest levels if they are not be needed on finer levels. This can save a significant amount of work. Also, the following slightly modified approach is a compelling possibility for problems where $m \gg n$: compute M_y as above (which is much cheaper than computing M_x), and compute a different approximation for the x-membership matrix as follows: $\widehat{M}_x = P_{xy} F_c^T$. This can save large amounts of work, as discussed below.

The following theorem can be verified easily ([15]):

THEOREM 2. *If each row and column of F has at least one non-zero, then the same holds for the rows and columns of $F_c = (P_{yx})^T F^T P_{xy}$ (anti-diagonal interpolation) and $F_c = (M_x)^T F M_y$ (diagonal interpolation), both in the case of Separate C/F-Splitting and Alternating C/F-Splitting.*

We consider the following FMCC variants in this paper, using the building blocks that were presented in this section:

method	C/F-splitting	interpolation
FMCC-C-D	Classical	Diagonal
FMCC-C-aD	Classical	anti-Diagonal
FMCC-S-D	Separate	Diagonal
FMCC-S-aD	Separate	anti-Diagonal
FMCC-A-D	Alternating	Diagonal
FMCC-A-aD	Alternating	anti-Diagonal

We conclude this section by discussing some further details that are required to arrive at a practical algorithm.

Noise Filtering and Rescaling of Membership Matrices: Since noise in F may have a large influence on the values computed in products like FF^T and $F^T F$, it is important for most applications to reduce the effect of noise (as illustrated by some prototypical examples in [15]). In particular, when computing the membership matrices we first filter F as follows: we set all weak connections in F to zero according to the notion of strength given in Definition 1, with a noise threshold $\lambda \in (0, 1]$ (which we normally choose around 0.3). The filtered feature matrix, $\widehat{F}$, is then used in computing M_x and M_y: $M_x = \widehat{F} P_{yx}$ and $M_y = \widehat{F}^T P_{xy}$. Further, we rescale M_x and M_y as follows (before row-normalizing to 1), in order to combat spurious element growth due to noise amplification that is known to occur in AMG coarse matrices due to the repeated triple products (2). We rescale row i of M_x by finding the maximum and minimum of row i of M_x, and rescaling $(M_x)_{ij}$ to $(M_x)_{ij} \exp(\gamma \frac{(M_x)_{ij} - min}{max})$, where $\gamma > 0$ is a parameter (which we normally choose between 1.5 and 3). The rescaling maintains the minimum value, and scales the maximum value up by $\exp(\gamma)$.

Normalization of F_c: The coarse graph weights computed directly by the agglomeration $A_c = P^T A P$ depend on the size of the clusters. In most applications it is desirable to determine the graph weights as the average weights of the connections between the points that make up the clusters, and to remove the influence of the size of the clusters. In the case of Diagonal P, we first form $F_c = (M_x)^T F M_y$ and then divide $(F_c)_{ij}$ by $(\sum_{k=1}^{m} (M_x)_{ki})$ $(\sum_{k=1}^{n} (M_y)_{kj})$ (the sizes of the i and j clusters). For anti-Diagonal P, we first compute $F_c = (P_{yx})^T M_y$, and since M_y is already row-normalized, we only need to divide the jth column of F_c by $\sum_{k=1}^{n} (M_y)_{kj}$.

4. COMPUTATIONAL COST

Table 1 shows detailed expressions for the approximate total computational cost (additions, multiplications and com-

Classical C/F-Splitting	$2N(c+r)+5(m^2+n^2)$
Seperate C/F-Splitting	$6N+2N(\min(m_c,c)+$ $\min(n_c,r))+2mm_c+2nn_c$
Alternating C/F-Splitting	$6N+m_cr+n_cc$
F_c using Diagonal P	$2N(\min(m_c,c)+\min(n_c,r))$ $+2Nn_c+2mm_cn_c$
F_c using anti-Diagonal P	$2N\min(n_c,r)+$ $2nn_c\min(m_c,c)+$
(standard: M_x)	$2N\min(m_c,c)$
(option +: $\widehat{M_x}$)	$2cm_cn_c$

Table 1: Approximate total operation count.

parisons) of each of the FMCC algorithm components. Note that we only count the cost on the finest level, which is normally dominant. The expressions in Table 1 apply to the general case, including matrices that are dense or sparse. More specific cases for which it is easier to compare the different algorithm variants are discussed in Table 2. In the tables, N is the total number of nonzeros of F, c is the maximum number of nonzeros in any column of F, and r is the maximum number of nonzeros in any row. Theorem 3 is a useful tool in deriving the cost estimates of Table 1:

THEOREM 3. *([17]) Let $A \in \mathbb{R}^{m\times l}$ and $B \in \mathbb{R}^{l\times n}$, let c_i be the number of nonzeros in the ith column of A and let r_i be the number of nonzeros in the i'th row of B. Then computation of AB can be implemented with number of operations (multiplications and additions) bounded by $2\sum_{i=1}^{l} c_i r_i$.*

Classical C/F-Splitting: Note first that $N = \sum_j c_j = \sum_i r_i$. Then it follows easily from Theorem 3 that the approximate operation counts for computing FF^T and F^TF are bounded by $2Nc$ and $2Nr$, respectively. Assuming that FF^T is dense, the operation count to build S_x is approximately $5m^2$: determining row maximums and column maximums requires m^2 operations each, determining strong connections requires 2 comparisons per matrix element (for a total of $2m^2$ operations), and computing column sums of S_x requires an additional m^2 operations, approximately. Similarly, computing S_y approximately requires $5n^2$ operations.
Separate and Alternating C/F-Splitting: Building S requires $6N$ operations ($2N$ to find row and column maximums of F, $2N$ to find strong connections, and $2N$ to compute row and column sums of S). Separate C/F-Splitting requires $2N\min(m_c,c)$ to compute similarities between the rows of S and the coarse rows of S; $2N\min(n_c,r)$ to compute similarities for the columns of S; and $2mm_c+2nn_c$ to assign C-points and F-points for the x-variables and y-variables. Alternating C/F-Splitting is much less expensive: no similarities are computed, and assigning C-points and F-points happens directly in S with cost bounded by m_cr+n_cc.
Computing F_c Using Diagonal P: The cost for computing M_x is bounded by $2N\min(m_c,c)$, and similar for M_y. (For simplicity, we consider the cost without noise filtering, rescaling and row sum normalization, because they do not essentially change the order of the cost estimates.) Matrix F_c can be computed in two ways: $F_c = M_x^T(FM_y)$ or $F_c = (M_x^TF)M_y$. The first option costs $2Nn_c+2mm_cn_c$ and is the best choice when $m_c > n_c$; we assume this and retain this option in Table 1 and the rest of this paper.
Computing F_c Using anti-Diagonal P: Here we also have two options: $F_c = P_{yx}^TM_y$ or $F_c = M_x^TP_{xy}$. The first option is preferred when $m_c > n_c$, so we choose this option. Computing M_y requires $2N\min(n_c,r)$ operations, and $P_{yx}^TM_y$ requires $2nn_c\min(m_c,c)$ operations. Computing M_x adds $2N\min(m_c,c)$ operations, but this additional cost (which can be significant, especially if $m \gg n$), can be avoided by computing the x-membership matrix in the alternative way $\widehat{M_x} = P_{xy}F_c^T$, which only requires $2cm_cn_c$ work, and which we indicate with 'option +' in Table 1.

	dense, $m=n$	sparse, $m \gg n$
Classical C/F	$4m^3$	$2N(c+r)+5m^2$
Seperate C/F	$4m^2m_c$	$2N(c+r)+2mm_c$
Alternating C/F	$6m^2$	$6N$
Diagonal P	$6m^2m_c$	$2N(c+r+n_c)+$ $2mm_cn_c$
anti-Diagonal P	$2m^2m_c+$	$2Nr+$
(standard: M_x)	$2m^2m_c$	$2Nc$
(option +: $\widehat{M_x}$)	$2mm_c^2$	$2cm_cn_c$
FMCC-S-D	$10m^2m_c$	$2N(2c+2r+n_c)+$ $2mm_cn_c$
FMCC-S-aD	$8m^2m_c$	$2N(2c+2r)$
FMCC-S-aD+	$6m^2m_c$	$2N(c+2r)$
FMCC-A-D	$6m^2m_c$	$2N(c+r+n_c)+$ $2mm_cn_c$
FMCC-A-aD	$4m^2m_c$	$2N(c+r)$
FMCC-A-aD+	$2m^2m_c$	$2Nr$

Table 2: Examples of total operation count estimates for the specific cases of a dense, square matrix and a sparse, 'tall-and-skinny' matrix.

Special Cases: Table 2 shows specializations of the general estimates in Table 1 for the special cases of a dense, square matrix and a sparse, 'tall-and-skinny' ($m \gg n$) matrix. For the dense, square matrix $m = n$, and we further assume $m_c = n_c \ll m = n$. We also assume in Table 2, for simplicity, that $m_c \ll m$ and $n_c \ll n$ (i.e., we consider large coarsening ratios of, e.g., 10 to 100, which depend on application and parameter settings but can be achieved naturally in many cases). For the dense, square matrix, Table 2 shows the advantage of our new separate and alternating coarsenings over the classical coarsening, which computes all-to-all similarities for row and column variables ($O(m^3)$). Our new methods were designed to avoid this costly step. They are both proportional to m^2 in cost, but note that m_c can be large, and it may depend on m, so alternating splitting can be significantly less expensive than separate splitting. This effect can also be seen when comparing the FMCC-S variants with the FMCC-A variants. The cost difference between the Diagonal, anti-Diagonal and anti-Diagonal+ variants does not involve a difference in order, but savings can still be significant, up to a factor 5. The third column of Table 2 shows the case of a sparse, 'tall-and-skinny' matrix ($m \gg n$). We assume the matrix is sparse without dense rows or columns ($c \ll m$ and $r \ll n$), and we further assume $c \le m_c$ and $r \le n_c$. An example in the term-document context may be a large collection of m emails (with m growing daily) that use a (nearly) fixed vocabulary of n words, with $m \gg n$. The cost of classical C/F-splitting is prohibitive since it is proportional to m^2. It can be expected here that c and m_c grow with m, whereas r and n_c remain approximately constant. Thus, the cost of separate splitting would be superlinear in m, while the cost of alternating splitting is linear (N can be expected to be linear in m). The cost for the Diagonal variants contains terms proportional to Nc and mm_c, which can

be expected to grow superlinear in m. The FMCC-A-aD+ variant completely eliminates these terms, and shows linear behaviour in terms of m (for constant n).

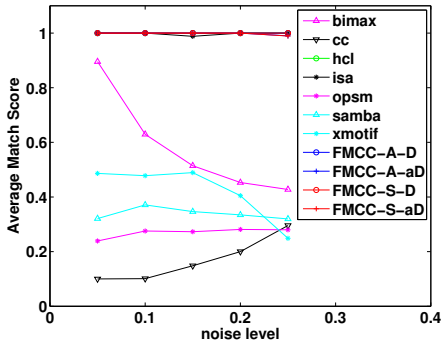

Figure 4: (Experiment 1, Scenario 1) Average match scores at different noise levels without overlap. $(\theta, \rho, \lambda, \gamma, \alpha_x, \alpha_y) = (0.7, 0.7, 0.3, 2.7, 0.4, 0.4)$**.**

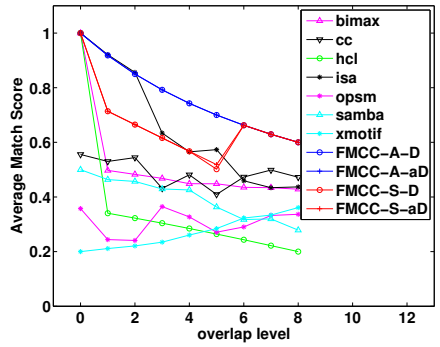

Figure 5: (Experiment 1, Scenario 2) Average match scores at different overlap levels without noise. $(\theta, \rho, \lambda, \gamma, \alpha_x, \alpha_y) = (0.5, 0.1, 0.3, 2, 0.6, 0.4)$**.**

5. EXPERIMENTS

5.1 Non-hierarchical Setting: One-Level Co-Clustering of Gene Expression Data

EXPERIMENT 1. *Standard artificial test problem from [11]. The rows and columns of data matrix F correspond to genes and experimental samples. Scenario 1 (varying noise) ($m = 100$, $n = 50$) has 10 non-overlapping transcription modules with 10 genes and 5 samples. Noise is added as in [11]. Scenario 2 (varying overlap) introduces overlap between the modules without noise (as in [11]). (Data matrices and test results with specialized methods are available online [11].)*

Figures 4 and 5 show experimental results for Scenarios 1 and 2 (we use Gene Match Score as defined in [11] to evaluate the results). For Scenario 1 (varying noise), each test is repeated 10 times per noise level. The figures compare the four main variants of FMCC (Alternating versus Separate splitting, and Diagonal versus anti-Diagonal P) with specialized gene clustering methods that were evaluated in [11]. All four FMCC variants produce accurate co-clusterings. Note that we have chosen slightly different algorithm parameters for Scenarios 1 and 2, but the results are not very sensitive to these differences. Also, remarkably, FMCC is as accurate or more accurate than the specialized methods from [11]. (The red curves occlude the blue curves in Figure 4.)

EXPERIMENT 2. *Real Gene Expression Data from [4].* **Leukemia Data Set** *(5000 × 38, denoted as ALL-AML) Consists of 27 acute lymphoblastic leukemia (ALL) and 11 acute myelogenous leukemia (AML) samples. This data set has become a benchmark in the cancer classification community [4].* **Medulloblastoma Data Set** *(5893×34) Captures childhood brain tumors known as medulloblastomas. There are two known histological subclasses: 25 classic samples and 9 desmoplastic samples. Similar to [4], we also compare the clustering methods using part of the data (subsets of rows).*

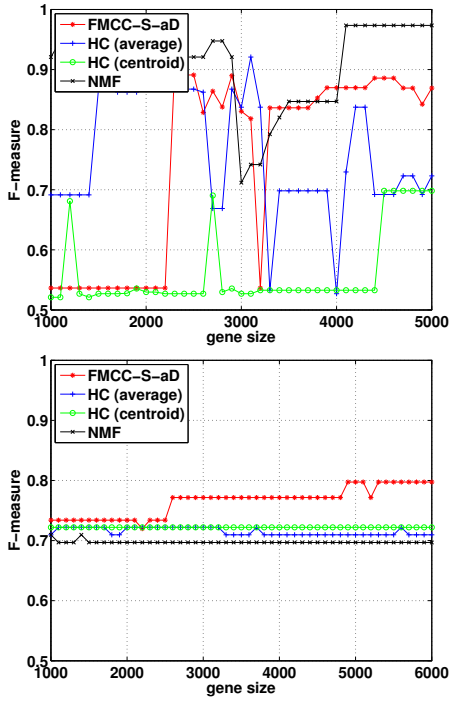

Figure 6: (Experiment 2) F-measure Comparison with NMF and Standard Hierarchical Clustering (HC) on Real Gene Data. Top: ALL-AML; Bottom: Medulloblastoma. $(\theta, \rho, \lambda, \gamma, \alpha_x, \alpha_y) = (0.5, 0.9, 0.4, 1.7, 0.55, 0.55)$ **for ALL-AML and** $(0.4, 0.5, 0.3, 1.7, 0.3, 0.3)$ **for Medulloblastoma.**

We use the standard F-measure to evaluate the real gene expression data [14]. Figure 6 (ALL-AML, top panel) shows that FMCC-S-aD is more accurate than standard hierarchical clustering methods (in particular, average and centroid linkage as implemented in Matlab). (FMCC-S-aD is considered here as a representative for the FMCC methods.) NMF is known to give highly accurate results, but it is expensive and does not scale well to large problem sizes. It is remarkable that FMCC-S-aD gives results that are close in accuracy to NMF. The Medulloblastoma results (bottom panel) confirm that FMCC-S-aD gives good results compared to hierarchical clustering and NMF in terms of accuracy.

5.2 Hierarchical Setting: Unraveling Hierarchical Co-Clusters in Complex Data Sets

EXPERIMENT 3. *Artificial Multilevel Data. (See Figure 7.) Similar to test problems in [10, 12], our artificial data has two cluster levels: there are 16 small clusters (8 points each) embedded in 4 big clusters (with 4 small clusters per big cluster) for both the rows and columns. $F \in \mathbb{R}^{128\times128}$ has*

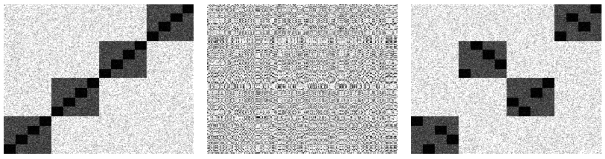

Figure 7: (Experiment 3) Color Maps of F with σ=1. Left: original; Middle: randomly permuted; Right: clustering found by FMCC-S-aD.

integer values between 0 and 10: For each of the dark, grey and light regions, there is a "basic value" b (b =10 for dark, 5 for grey and 0 for light). The values in each region are randomly chosen as $i \in \{0, 1, ..., 10\}$ with a probability proportional to the Gaussian distribution $\exp(-(i-b)^2/2\sigma^2)$*, where σ is a noise parameter.*

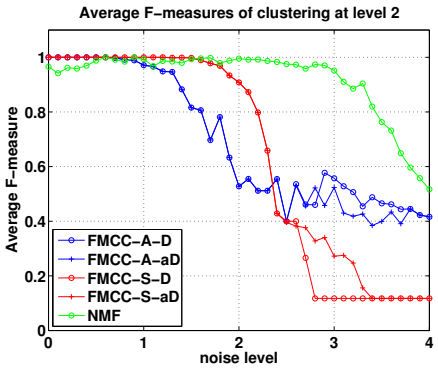

Figure 8: (Experiment 3) F-measure Comparison with NMF. Level 2 (16 clusters). $(\theta, \rho, \lambda, \gamma, \alpha_x, \alpha_y) = (0.7, 0.7, 0.3, 1.7, 0.5, 0.5)$**.**

Fig. 7 (right panel) shows how FMCC-S-aD recovers the nested co-clusters at all levels with remarkable accuracy. Note that the orders of the clusters are arbitrary, which accounts for the difference in ordering of the blocks and sub-blocks between the left and right panels. The accuracy of FMCC up to moderate noise levels (with a comparison of the 4 main FMCC variants) is confirmed in Fig. 8. Each test is repeated 10 times per noise level. NMF also gives good accuracy and maintains it up to higher noise levels (results for level 3 are similar). It is important to point out, though, that NMF needs to be executed two separate times with pre-specified k-values 4 and 16 in order to find the 16 clusters at level 2 and 4 clusters at level 3, and NMF does not directly reveal the nesting of the clusters. In contrast, FMCC does not require to specify the desired number and finds clusterings with the correct numbers of clusters automatically on both levels, discovering the nested relation of the clusters at the same time. Figure 9 compares the accuracy and execution time performance of FMCC and NMF for increasing problem size in Experiment 3, for F ranging in size from 256×256 to 8192×8192 and for $\sigma = 1$. We execute NMF in two different ways: with a fixed number of 100 iterations, or up to a fixed tolerance. In the top panel we make two interesting observations. First, the aD+ variants of FMCC (which compute the less expensive approximations $\widetilde{M_x}$ for the x membership matrices) are less accurate than the aD variants, as expected, but they become almost as accurate for large problem sizes, where they are most useful in terms of reducing the cost. This can be interpreted as a 'big data' effect: simple algorithms may provide accurate results as

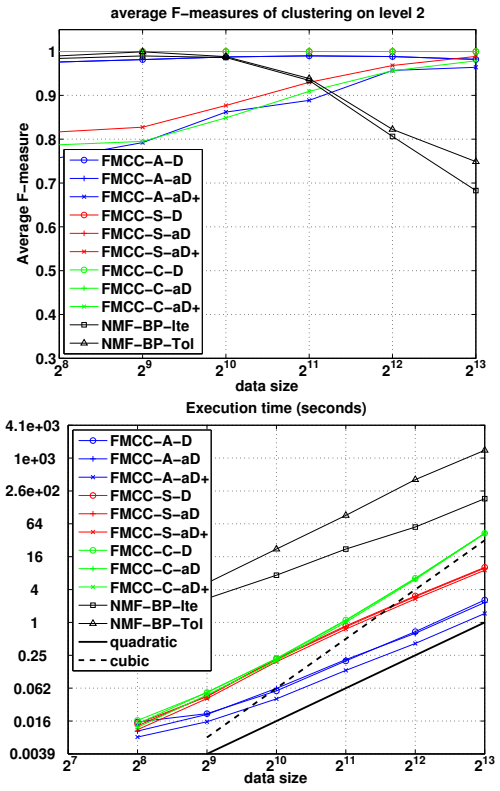

Figure 9: (Experiment 3) Accuracy and time performance for different data size m (F is of size $m \times m$). Top: average F-measure of small group clustering (level 2, 16 clusters); Bottom: runtime performance. FMCC parameters: $(\theta, \rho, \lambda, \gamma, \alpha_x, \alpha_y) = (0.7, 0.7, 0.3, 1.7, 0.5, 0.5)$**.**

long as they can take advantage of large amounts of data. On the contrary, and remarkably, the accuracy of the NMF results degrades seriously when problem sizes increase. The bottom panel of Fig. 9 compares execution times. We use the NMF implementation of [7] that is available online and is considered an efficient implementation of NMF. For fairness of runtime comparison, all algorithms we compare are implemented in Matlab. Experiment 3 involves square and dense feature matrices (with density around 30%), and as such we can compare the performance results with the estimates provided in the middle column of Table 2. Figure 9 confirms that the 'Separate' and 'Alternating' FMCC-S and FMCC-A variants have runtime complexity quadratic in m, while the 'Classical' FMCC-C variants are cubic in m. For the 'Alternating' variants, it is confirmed that FMCC-A-aD+ is indeed faster than FMCC-aD and FMCC-D (by a factor of about 2). (The advantage of the aD+ variant does not show up in the numerical tests for the FMCC-S variants because their cost is dominated by the separate clustering algorithm, which contains steps that cannot take advantage of Matlab's fast matrix-matrix multiplication routines.) Both NMF variants are much slower than the FMCC algorithms, with the tolerance-based NMF (which is more accurate) being about 1,000 times slower than the time of the fastest FMCC variant for the largest problem size tested. (Note that NMF is also much less accurate at this problem size.)

Experiment 4. *Real LinkedIn Data on Users and 'Skills & Expertise'. The values in F (176 users $\times$ 47 skills) are*

integers denoting how many times a skill for a user was endorsed by his or her connections. (Most values are zero.) The data set was collected in a manual process in January 2013 by starting from 3 randomly picked seed users (in the fields of Numerical Analysis, Physics and Computer Science) and then doing a breadth first search of all their endorsers until an endorser's profile cannot be visited (note that a standard LinkedIn user can only visit other users who are within a distance of 3 connections). In total, 1247 users were collected with 7074 listed skills. Many of these skills show up very rarely or overlap with each other, and some users have very few skills. To get a more compact data set, we first filtered skills out if less than 70 of the 1247 users listed them; then, users were filtered out if they had less than 7 remaining skills. Note that this dataset is relatively small, but we are not aware of a practical way to obtain larger data sets (note that automated collection is prohibited). Nevertheless, this data set is of sufficient size to illustrate the potential of our new methods for unraveling multilevel co-cluster structure of social network data.

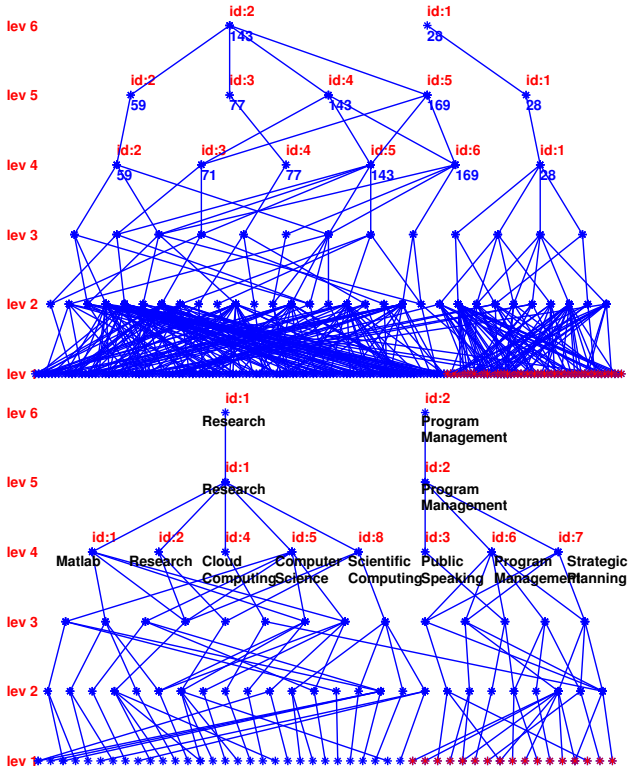

Figure 10: (Experiment 4) Hierarchical trees found by FMCC-S-aD. Top: tree of users; Bottom: tree of skills. $(\theta, \rho, \lambda, \gamma, \alpha_x, \alpha_y) = (0.7, 0.7, 0.3, 2.7, 0.4, 0.4)$.

Figure 10 shows the user and skill co-clusters and their strong connections obtained by FMCC-S-aD. It turns out that the LinkedIn dataset contains two well-defined groups of users. The first group is related to computer scientists and mathematicians doing university research. The second group is related to business people who are mainly involved with start-ups in the cloud computing area. These two groups are clearly identified by running NMF on F with $k = 2$. The first confirmation that FMCC-S-aD obtains accurate results is by comparing the multilevel co-clustering with the NMF result. In Fig. 10 we have plotted the FMCC user and skill tree-like structures, but we have ordered the users and skills in a particular way: the red dots in the top panel correspond to the users that were grouped in the 'research' group by NMF, and the red dots in the bottom panel correspond to skills determined to be related to the 'research' group by NMF. Remarkably, FMCC-S-aD finds largely the same grouping at level 6, with very few strong connections (blue lines) anywhere in the tree between the subtree corresponding to the red dots (which we will call the 'business' side of the tree) and its complement (which we will call the 'research' side). The F-measure w.r.t. to the NMF clustering is 0.96 for the user clusters, and 1 for the skill clusters. At level 6 in the skills tree, the seed skill keyword of the first cluster is indeed 'research', the seed of the second cluster is 'program management', and their subclusters have seed keywords in more specific parts of research and business, respectively. Note that the blue lines in the trees of Fig. 10 indicate strong connections in the following sense: For the x-tree, for example, blue lines are drawn between points on consecutive levels if the corresponding membership value in M_x is strong according to Definition 1, with strength value $\theta_M = 0.8$.

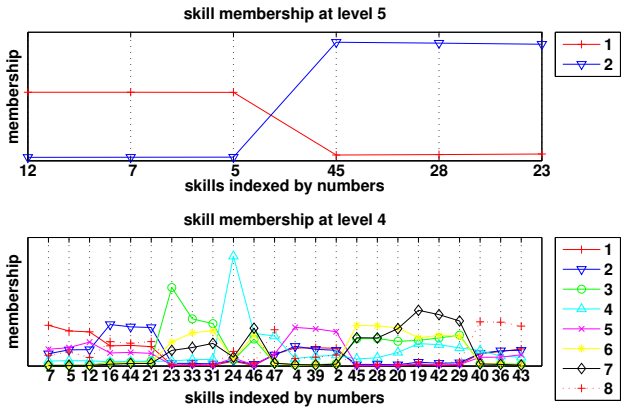

Figure 11: Skill Memberships at different levels.

2	C++	29	Entrepreneurship
4	Java	31	Business Analysis
5	LaTeX	33	Project Management
7	Matlab	36	Numerical Analysis
12	Databases	39	Software Engineering
16	Data Mining	40	Scientific Computing
19	Start-ups	42	Business Development
20	Team Building	43	Mathematical Modeling
21	Data Analysis	44	Business Intelligence
23	Public Speaking	45	Management Consulting
24	Cloud Computing	46	New Business Devpmt.
28	Change Mgmt.	47	HPC

Table 3: Skills names (with some abbreviations) associated with the indices in Fig. 11.

More detailed investigation of the results confirms that FMCC indeed uncovers relevant multilevel structure in the LinkedIn data set. Table 3 lists a subset of the 47 skills that have large memberships in the skill clusters obtained by MFCC at high levels of the hierarchy. Figure 11 shows a plot of part of the skill membership vectors (columns of M_y) for the skills clusters at levels 5 and 4 in the skill tree

of Fig. 10. For each cluster, the membership value of the three most dominant skill keywords is plotted. For example, at level 5 in Fig. 11, one can see that the two clusters are well-separated (no overlapping skill keywords), and the dominant skills in 'research' cluster 1 are the technical skills 'Databases', 'Matlab' and 'Latex' (with labels 12, 7 and 5, identified in Table 3). Cluster 2 at level 5 is dominated by business-related skills 'Management Consulting', 'Change Management' and 'Public Speaking'. The clusters at level 6 in Fig. 11 are the same as at level 5, but the clusters at level 4 are overlapping subclusters of the clusters at the higher level: clusters 3, 6 and 7 on level 4 are subclusters of the 'Program Management' cluster on level 5, and the other clusters on level 4 are subclusters of the 'Research' cluster on level 5 (see Fig. 10). Clusters 3, 6 and 7 on level 4 correspond to social media skills, management skills and entrepreneurship skills, respectively, and their keywords overlap in the bottom panel of Fig. 11. Similarly, skill clusters 1, 2, 4, 5 and 8 on level 4 correspond to skills in research tools, data mining, high performance computing, software engineering, and scientific computing, respectively. Note also that there is some 'crosstalk' between skills in the 'research' side and the 'business' side of the tree, which results mainly from some professors in database research being active in start-up companies in the area of cloud computing in our data set. A similar analysis confirms for the groups and subgroups in the user tree of Fig. 10 that, for example on level 5, group 1 corresponds to business users, and groups 2, 3, 4 and 5 correspond to researchers active in Numerical Analysis, HPC, Data Science, and Software Engineering, respectively (details not shown due to space constraints). The coarse bipartite graph on level 5 connects the user groups to the skill groups they are related to.

6. CONCLUSIONS AND FUTURE WORK

We have developed new fast algorithms for multilevel co-clustering. They incorporate novel graph agglomeration formulations and fast ways for coarsening rectangular matrices that employ measures of direct connection strength between row and column variables in the feature matrix. Our methods can be scaled up efficiently while maintaining adequate accuracy, and are therefore attractive options for traditional (single-level) co-clustering problems with large data sizes, taking advantage of speed gains offered by the multilevel framework that uses fast, scalable and accurate heuristics. More importantly, our methods offer new tools for unraveling the multilevel overlapping cluster structure of social networks, which we have demonstrated for a proof-of-concept LinkedIn data set relating users to their skills and expertise. It would be interesting to test this on larger LinkedIn and other available social network data.

Further technical developments that will be useful include user-friendly interfaces for interactive visual exploration of the large co-cluster trees and their bipartite graph connections that may be generated by FMCC, reliable methods to automatically identify 'prominent' clusters (as opposed to intermediate clusters that are similar to neighbour clusters with which they will be merged at the next coarser level), methods to compute distances between multilevel cluster trees and their distances to the original data (in order to facilitate comparison of multilevel clustering results and evaluation of the quality of the cluster trees), and parallel implementations of the FMCC algorithms for big data problems. This requires significant further work both at the theoretical and practical levels. All code, data and numerical tests for this paper will be made available online at publication time.

7. REFERENCES

[1] A. Baker, R. Falgout, T. Kolev, and U. M. Yang. Scaling Hypre's multigrid solvers to 100,000 cores. In *High Performance Scientific Computing: Algorithms and Applications*, pages 261–279, 2012.

[2] A. Brandt, S. McCormick, and J. Ruge. Sparsity and its Applications; Algebraic multigrid (AMG) for sparse matrix equations. Cambridge University Press, 1984.

[3] W. Briggs, V. Henson, and S. McCormick. *A multigrid tutorial*. SIAM, 2000.

[4] J. Brunet, P. Tamayo, T. Golub, and J. Mesirov. Metagenes and molecular pattern discovery using matrix factorization. *PNAS*, 101:4164–4169, 2004.

[5] H. De Sterck. A self-learning algebraic multigrid method for extremal singular triplets and eigenpairs. *SIAM Journal on Scientific Computing*, 34:A2092–A2117, 2012.

[6] I. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proc. KDD*, pages 269–274, 2001.

[7] J. Kim and H. Park. Toward faster nonnegative matrix factorization: A new algorithm and comparisons. In *Proc. ICDM*, pages 353–362, 2008.

[8] D. Kuang and H. Park. Fast rank-2 nonnegative matrix factorization for hierarchical document clustering. In *Proc. KDD*, pages 739–747, 2013.

[9] D. Kushnir, M. Galun, and A. Brandt. Fast multiscale clustering and manifold identification. *Pattern Recognition*, 39:1876–1891, 2006.

[10] A. Lancichinetti, S. Fortunato, and J. Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11:033015, 2009.

[11] A. Prelić, S. Bleuler, P. Zimmermann, A. Wille, P. Bühlmann, W. Gruissem, L. Hennig, L. Thiele, and E. Zitzler. A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics*, 22(9):1122–1129, 2006.

[12] M. Sales-Pardo, R. Guimera, A. Moreira, and L. Amaral. Extracting the hierarchical organization of complex systems. *PNAS*, 104:15224–15229, 2007.

[13] E. Sharon, M. Galun, D. Sharon, R. Basri, and A. Brandt. Hierarchy and adaptivity in segmenting visual scenes. *Nature*, 442:810–813, 2006.

[14] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, 2000.

[15] H. Xu. Fast multilevel co-clustering. Master's research report, University of Waterloo, 2013.

[16] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *Proc. SIGIR*, pages 267–273, 2003.

[17] R. Yuster and U. Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1:2–13, 2005.

[18] Y. Zhao and G. Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *Proc. ICKM*, pages 515–524, 2002.