Overlapping Clusters for Distributed Computation

Reid Andersen
Microsoft Corporation
Redmond, WA
reidan@microsoft.com

David F. Gleich
Purdue University
West Lafayette, IN
dgleich@purdue.edu

Vahab Mirrokni
Google Research
New York, NY
mirrokni@google.com

ABSTRACT

Most graph decomposition procedures seek to partition a graph into disjoint sets of vertices. Motivated by applications of clustering in distributed computation, we describe a graph decomposition algorithm for the paradigm where the partitions intersect. This algorithm covers the vertex set with a collection of overlapping clusters. Each vertex in the graph is well-contained within some cluster in the collection. We then describe a framework for distributed computation across a collection of overlapping clusters and describe how this framework can be used in various algorithms based on the graph diffusion process. In particular, we focus on two illustrative examples: (i) the simulation of a randomly walking particle and (ii) the solution of a linear system, e.g. PageRank. Our simulation results for these two cases show a significant reduction in swapping between clusters in a random walk, a significant decrease in communication volume during a linear system solve in a geometric mesh, and some ability to reduce the communication volume during a linear system solve in an information network.

1. INTRODUCTION

Graph partitioning is a broad term for processes that take a graph and break it into pieces. These processes are applied throughout scientific computation because they model the conundrum of distributed computations, namely, how to balance local work and communication. In a typical application, a graph serves as a surrogate for the computational domain [14]. Each vertex denotes a piece of information and each edge denotes dependencies between information. Usually, the goal is a partition of the vertices where no piece is too large, and the number of edges crossing between partitions is small. Such a partition implies that most of the work is within a piece and only minimal work takes place between pieces. Thus, a good partitioning minimizes the amount of communication during a distributed computation.

But why partition the *vertices*? Alternatives such as hypergraph partitioning or partitioning the edges often show better results [38, 18, 9]. These models minimize the total number of communication messages and their volume by constructing a hyper-graph to model the actual computation structure more accurately. A concern with such methods is that they make distributed computations more difficult to implement.

We tackle the problem from a different angle. In light of the plethora of local resources available on modern computers and the paucity of communication bandwidth and latency [17], we ask why *partition* at all? Partitioning minimizes the total amount of storage under the constraint of minimizing communication volume. In this paper, we want to design scalable algorithms that address communication problems using a set of *overlapping clusters*, or vertex partitions that intersect. With overlapping clusters, we store more of the graph than required. The framework then affords the ease of implementation of vertex partitioning, and by removing the minimum storage constraint, our hope is that this technique allows us to reduce communication in a distributed computation. As we shall see in the results section, such an idea is a simple means to improve a distributed computation. For a standard test problem – computing the PageRank vector – we observe a drastic reduction in communication volume for geometric graphs and a modest reduction in communication volume for some information networks.

Our main application for this procedure is a distributed diffusive process on a graph. This differs from the goal of much of the overlapping clustering work in the statistical physics communities (see ref. [15] for a survey). There, the goal is to use the clusters to extract information from the network. Many of the methods employed are also rather expensive for a large graph, for example, building overlapping clusters by finding large cliques. As mentioned above, we use a particularly well known diffusion process as our test problem: the PageRank random surfer [32]. Diffuse processes such as PageRank often become linear systems, and our overlapping clustering computation framework also applies to solving a linear system. This lets us compute Katz indices [22], hitting times, commute times, and semi-supervised learning on graphs [40]. We discuss linear systems further in Sections 3 and 6 Our goal is to have a single set of overlapping clusters that will make all of these diffusive graph processes fast, not just PageRank. Thus, much of this paper is spent analyzing random walks on overlapping clusters. This metric is indicative of the performance of any diffusive process. Random walks are, themselves, a key enabling routine for many approximate graph computations (e.g. nearest tree [11] and *MaxCut* [19]) and distributed graph computations (e.g. random spanning tree [34]).

In the remainder of the paper, we explore overlapping clusters. We first review existing work on overlapping clustering in matrix computations and graph computations (Section 3). Next we formally define overlapping covers and specify how we use them in distributed computation of a random walk (Section 4). For a concrete example, we study a random walk process on a cycle graph. For this graph, we prove that overlapping clusters need fewer communication steps than do partitions (Theorem 1). This proof uses an optimal set of overlapping clusters for the graph. In the remainder of the paper, we provide a scheme to find a good set of overlapping clusters for the purpose of distributed computation (Section 5). Our algorithm combines several primitive operations: (i) determining a good set of clusters, (ii) computing a containment score for these clusters to find good vertices, (iii) covering the graph with highly contained vertices, and (iv) combining clusters to optimize performance. These steps involve a few interesting sub-algorithms. For example, we use a personalized PageRank clustering algorithm [3] to determine a good set of clusters in a local graph computation framework. Also, we use an approximate set covering algorithm to cover the vertices with a small set of clusters.

With a computational algorithm in hand, we investigate its performance in the final section. These experiments evaluate the clusters from our algorithm in terms of total storage and communication. We compare against the METIS graph partitioner [21] and the GRACLUS graph partitioner [13]. To evaluate the clusters in a computational setting, we simulate random walks and measure the number of times these walks switch between clusters. We use this as a proxy for the performance of a diffusion process on a graph. Again, we compare against METIS and GRACLUS. These comparisons show that our procedure reduces the probability of swapping between clusters by a factor of between 1.15 to 100.

Let us explicitly note that this paper is intended as a proof of concept demonstration that overlap can help. We do *not* evaluate our procedures in a true distributed environment. Instead, we simulate what communication would have occurred in such an environment. Because of this nature and that many of our results appear to depend strongly on the individual properties of the graph, we make our codes and experiments available for others to use and reproduce: `https://dgleich.com/projects/overlapping-clusters`

2. NOTATION

Throughout the paper, we are concerned with graphs. Thus, we adopt standard graph theoretic notation. A graph $G = (V, E)$ consists of its vertex set V and edge set E. We use $\deg(v)$ to denote the degree of a vertex. Sets are denoted with a capital letter. For a set of vertices S, then the *volume* of the set measures the degree of all vertices in the set:

$$\mathrm{Vol}(S) = \textstyle\sum_{v \in S} \deg(v).$$

Just like a physical volume quantifies space, the volume of a set of vertices is the amount of storage space they require. Another useful measure is the boundary of a set $\partial(S)$:

$$\partial(S) = \{(u, v) \in E : u \in S, v \in \bar{S}\}.$$

In this definition, we denote the complement of a set S by $\bar{S}$. The complement is respect to the set vertices for the entire graph, thus $\bar{S} = \{v \in V : v \notin S\}$. Finally, let $\mathrm{cut}(S, T) = |\{(u, v) \in E : u \in S, v \in T\}|$ be the generalization of the size of a boundary to the size of a particular boundary. Notice that these definitions are related by: $\deg(v) = \mathrm{Vol}(\{v\}) = |\partial(\{v\})| = \mathrm{cut}(\{v\}, V - \{v\})$. Also, $|\partial(S)| = \mathrm{cut}(S, \bar{S})$.

Our final definition is the conductance of a set:

$$\mathrm{Cond}(S) = \frac{|\partial S|}{\min\bigl(\mathrm{Vol}(S), \mathrm{Vol}(\bar{S})\bigr)}.$$

Conductance is a useful metric for graph partitioning because it is small when there are few edges leaving a big group of vertices and large elsewhere.

3. RELATED WORK

Our proposal for overlapping clusters is novel in its implementation, but not so in concept. As always seems to be the case with clustering algorithms, statisticians studied the problem of overlap in the 1970s [10]. These have also been studied for a long time in the field of domain decomposition. There, overlapping domains are used to solve a partial differential equation (PDE) using a Schwarz method. Given a domain that is decomposed into overlapping pieces, Schwarz methods solve the PDE by iterative solving the PDE on each piece and communicating between the domains via the overlapping boundary. Within the numerical linear algebra community, these ideas have been generalized to solve many linear systems $Ax = b$ in either an *additive* or *multiplicative* Schwarz method [37, 6, 28]. The difference between the methods is not important for this paper, but suffice it to say that our distributed PageRank technique discussed in Section 6 is equivalent to an additive method. These techniques have been applied to solving PageRank [7] and recently this community began studying techniques to generate overlap from an existing graph partition [16]. Yet another recent and related idea from numerical linear algebra is the notion of a communication avoiding algorithm, see, for example, ref. [31]. A communication avoiding algorithm substitutes local computation instead of communication. In ref. [31]., they design a matrix-power kernel to reduce the communication required for k consecutive sparse matrix-vector products. The matrix-power kernel will create overlap among the vertices managed by each processor. A key difference between our work and the two previous projects is that they start with a partitioning and add overlap; instead, we build a set of overlapping clusters and then add a mapping from vertices to clusters.

Others recognized the benefits of overlapping clusterings too. These have also been studied in social network analysis [30, 1], and inherent multi-assignment clustering [36]. In some settings, like discovering communities in social networks, the clusters are naturally overlapping and by restricting our attention to non-overlapping clustering, we may lose valuable information about the structure of communities in a social network. This fact have been formally observed in the context of social networks [30] and information networks [1].

Graph partitioning and finding sets with small conductance are important problems that have also been attacked theoretically. Several approximation algorithms exist for them [35, 4, 25, 2, 3]. One of the authors has recently studied the analogous problems with overlapping clusters and established both complexity results and new polylogarithmic approximation algorithms for these problems [23].

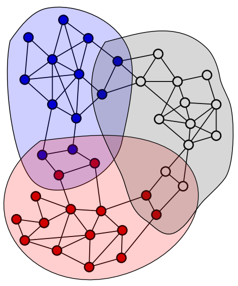
Many decentralized or distributed graph computations also use overlap among a partition. For example, ref. [33] proposes a PageRank algorithm for a peer-to-peer environment, and ref. [20] uses a specific type of overlap to compute minimum spanning trees in a MapReduce environment.

4. DISTRIBUTED COMPUTATION WITH OVERLAPPING CLUSTERS

In this section, we formally define an overlapping cover for a graph and specify how to use it in a distributed random walk. The number of swaps between clusters will be one of our distributed computation metrics. We then show the relationship between our overlapping clusters and partitioning. This will let us formally show an instance where overlapping clusters minimize swaps compared to a partitioning.

4.1 Overlapping covers

An overlapping cover $(\mathcal{C}, \tau)$ consists of a collection $\mathcal{C}$ of *clusters*, which are subsets of V, and a *mapping* $\tau : V \to \mathcal{C}$ that associates each vertex $v \in V$ with a single cluster $\tau(v) \in \mathcal{C}$. We require that the collection $\mathcal{C}$ cover the graph, and that each vertex v be contained in its associated cluster $\tau(v)$. For this reason, τ will define a partition of the vertices. As further explained below, the idea is that τ will give the *best* cluster for a given overlapped vertex.



At left, we illustrate an overlapping cover with three clusters denoted by the red, blue, and gray regions. The function τ is given by the color of the vertices themselves, not the regions. For instance, the gray cluster contains two vertices from the whose associated cluster is the blue cluster; but the blue cluster contains no vertices whose associated cluster is the gray one. These two "blue" vertices are stored in both blue and gray clusters, providing an instance of overlap. In our model, the adjacency information for the graph G is distributed among the clusters in $\mathcal{C}$. Each cluster stores the adjacency list of its member vertices, but not the adjacency lists of other vertices in the graph. Thus, $\operatorname{Vol}(C)$ is a proxy for the space required to store the cluster C, since $\operatorname{Vol}(C)$ is the total size of the adjacency lists of vertices in C. We define $\operatorname{MaxVol}(\mathcal{C})$ to be the maximum volume of any cluster in $\mathcal{C}$, and define $\operatorname{TotalVol}(\mathcal{C})$ to be the sum of the volumes of all clusters in the collection. We will also commonly refer to the volume ratio of a collection: $\operatorname{TotalVol}(\mathcal{C})/\operatorname{Vol}(V)$.

4.2 A random walk in overlapping clusters

Given a maximum volume MaxVol as an upper bound for the volume of a graph on each machine, we need to divide a large graph into several clusters and store each cluster on one machine. The main drawback of this process is that any type of distributed computation or random walk incurs some communication cost or swapping probability among clusters. To model that cost, the random walk process is defined in the usual way. Given a starting vertex X_0, a sequence of vertices $X_0 \dots X_T$ is obtained by choosing X_{t+1} uniformly at random from the neighbors of X_t.

We can simulate the random walk process in G using a collection $(\mathcal{C}, \tau)$ of overlapping clusters. For each time step t let Y_t be the current vertex and C_t be the current active cluster. The current vertex Y_t is some vertex in V, and the current active cluster C_t is some cluster in $\mathcal{C}$ that contains Y_t. Initially, Y_0 is a specified starting vertex, and $C_0 = \tau(Y_0)$. To advance to the next time step, the active cluster C_t chooses the next vertex Y_{t+1} uniformly at random from the neighbors of Y_t. If the new vertex Y_{t+1} is contained in C_t, then the current cluster remains the active cluster. If the new vertex Y_{t+1} is not contained in C_t, then the cluster $\tau(Y_{t+1})$ becomes the active cluster. The sequence of active clusters $\{C_i\}$ is completely determined by the sequence of vertices $\{Y_i\}$,

$$C_{t+1} = \begin{cases} \tau(Y_{t+1}) & \text{if } Y_{t+1} \notin C_t, \\ C_t & \text{otherwise.} \end{cases}$$

Also, the state of the walk simulation at time t is completely described by the pair (Y_t, C_t).

4.3 The frequency of cluster swapping

Our goal is to minimize the number of times the active cluster C_t must be changed during the simulation of the walk. A good collection $\mathcal{C}$ will allow us to simulate a random walk without constantly switching between clusters.

We define $\operatorname{Swaps}(X_0, \dots, X_T)$ to be the number of times the active cluster C_t changes during the walk $X_0, \dots, X_T$. We also define $\rho(X_0, \dots, X_T) = \operatorname{Swaps}(X_0, \dots, X_T)/T$ to be the fraction of steps where the active cluster changes during the walk. For a given vertex v, we define

$$\rho_T(v) = E\left[(1/T) \operatorname{Swaps}(X_0, \dots, X_T) \mid X_0 = v\right]$$

to be the expected number of swaps in a walk starting from v. We also define

$$\rho_T = \frac{1}{n} \sum_{v \in V} \rho_T(v)$$

to be the average number swaps in a walk from a starting vertex chosen uniformly at random. Finally, we define ρ_∞ to be the limit of $\rho_T(G)$ as $T \to \infty$.

We can easily estimate these quantities by repeatedly simulating a random walk. However, we can also compute ρ_∞ exactly. To do so, define a directed graph $G(\mathcal{C}, \tau)$ that describes the transition between the pairs (Y_t, C_t). Presuming that a random walk on $G(\mathcal{C}, \tau)$ has a single ergodic class – which it will if the initial graph is connected and undirected – then let $\hat{\pi}$ be the stationary distribution. The swapping probability ρ_∞ is then

$$\rho_\infty = \sum_{C} \sum_{u \in C} \hat{\pi}(u, C) \frac{\operatorname{cut}(u, \bar{C})}{\deg(u)}.$$

4.4 Relationship with partitioning

Recall, the traditional way to find such clusters is to partition the graph into several non-overlapping parts. Based on a partitioning $\mathcal{P} = (P_1, \dots, P_m)$, we can define an overlapping cover $(\mathcal{P}, \tau)$ where $\tau(v) = P_i$ if $v \in P_i$. In this case, the stationary distribution of $(\mathcal{P}, \tau)$ is the standard stationary distribution of a random walk on an undirected graph: $\hat{\pi}(u) = \deg(u)/\operatorname{Vol}(G)$. The swapping probability of $(\mathcal{P}, \tau)$ is then

$$\rho_\infty = \sum_{C \in \mathcal{P}} \sum_{u \in C} \hat{\pi}(u) \frac{\operatorname{cut}(u, \bar{C})}{\deg(u)} = \frac{1}{\operatorname{Vol}(G)} \sum_{C \in \mathcal{P}} |\partial(C)|.$$

Therefore, the goal for partitioning is to minimize the the number of edges among clusters: $\sum_{C \in \mathcal{P}} |\partial(C)|$. The advantages of partitioning the graph are that it is simple and uses the minimal storage to store the whole graph, since each node in the original graph corresponds to only one

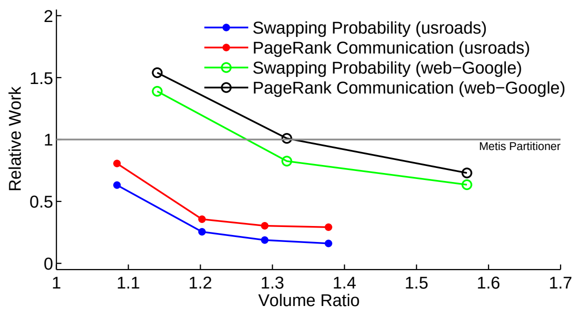


Figure 1: Overlap improves performance for the US road network and a WWW network; see §7.1.

node on exactly one machine. On the other hand, this advantage may result in large communication or swapping probability among clusters. Using overlapping covers may improve the communication cost and the switching cost of the clustering by a large factor. See Figure 1 for an example of this phenomenon. For the graph of the US road network and a web-graph, adding overlap improves the performance of a distributed PageRank computation and the swapping probability of a random walk considerably compared to a partition from the METIS partitioner. The vertical axis is the relative work compared to METIS, and the horizontal axis is the overlap expressed as a volume ratio. Of course, storing overlapping clusters take more space. It is not hard to see that there is a trade-off between the total volume of the clusters TotalVol, and the swapping probability ρ_∞. Next, we show that for cycle graphs, storing overlapping clusters with TotalVol $= 2\,\mathrm{Vol}(G)$ improves the cost function ρ_∞ significantly.

4.5 A clear win for overlap

THEOREM 1. *Consider a large cycle C_n of $n = M\ell$ nodes for a large number $M > 0$, and let the maximum volume of a cluster* MaxVol *be ℓ. Let P be the optimal partitioning of G to non-overlapping clusters of size at most* MaxVol *and ρ^*_∞ be the swapping probability of P. There exists an overlapping cover with* TotalVol *of $2\mathrm{Vol}(G)$ whose swapping probability ρ'_∞ is less than $\frac{\rho^*_\infty}{\Omega(\mathrm{MaxVol})}$.*

PROOF. In a cycle of size n, any cluster of size at most ℓ nodes has a cut size of at least 2. We know that partition $\mathcal{P}$ is at least M clusters. Thus, the size of the cut $\partial(\mathcal{P})$ is at least $2M$. Therefore, the optimal partition $\mathcal{P}$ is to divide the cycle into M paths of consecutive nodes each of which have ℓ nodes, Thus, the swapping probability $\rho^*_\infty = \frac{\partial(\mathcal{P})}{\mathrm{Vol}(C_n)} = \frac{2M}{n} = \frac{2}{\ell}$.

We now illustrate an overlapping cover $(\mathcal{C}, \tau)$ of TotalVol $= 2\,\mathrm{Vol}(G)$ with swapping probability $\rho_\infty \leq \frac{4}{\ell^2}$: $\mathcal{C}$ has $2M$ clusters $(C_1, \ldots, C_M, C'_1, \ldots, C'_M)$, where $C_i = \{v | i\ell - \ell + 1 \leq v \leq i\ell\}$ for $1 \leq i \leq M$, and $C'_i = \{v | i\ell - \frac{\ell}{2} + 1 \leq v \leq i\ell + \frac{\ell}{2}\}$ for $1 \leq i \leq M - 1$, and $C'_M = \{v | 1 \leq v \leq \frac{\ell}{2}\} \cup \{v | M\ell - \frac{\ell}{2} \leq v \leq M\ell\}$. Let *centers* of clusters C_i and C'_i be the two middle nodes of each cluster, e.g., centers of C_i are $i\ell - \frac{\ell}{2}$ and $i\ell - \frac{\ell}{2} + 1$. center of a cluster C is denoted by center(C). For any node $v \in V(G)$, let $\tau(v)$ the cluster whose center is the closest center to v. In particular, $\tau(\mathrm{center}(C)) = C$.

Consider a long enough random walk W. Note that centers of C_i are borders of C'_{i-1} and C'_{i+1}. In particular, on walk W whenever we swap to a new cluster, we go to the center of another cluster. As a result, the expected number of steps between two swaps on walk W is the expected time that a random walk starting at the center of a cluster leaving that cluster. Consider cluster C_1, and let a_i be the expected number of steps before a uniform random walk leaves cluster C_1 starting from node i. Also let $a_0 = a_{\ell+1} = 0$. Since the random walk is uniform, we know that for $1 \leq i \leq \ell$, $a_i = 1 + \frac{1}{2}(a_{i+1} + a_{i-1})$. Solving this recurrence relation, we get $a_i = i(\ell - i)$. Thus, $a_{\mathrm{center}(C_1)} = \frac{\ell(\ell+2)}{4}$. As a result, starting from any cluster in $\mathcal{C}$, the expected time to leave the cluster and swap to another cluster is $\frac{\ell(\ell+2)}{4}$. As discussed before, after swapping to a new cluster, we start from the center of another cluster. Therefore, in average, after $\frac{\ell(\ell+2)}{4}$ steps of the random walk, a swapping occur. Thus, the swapping probability $\rho'_\infty = \frac{4}{\Omega(\ell^2)}$. $\square$

4.6 Hardness of optimality

Before stating the heuristic algorithms to identify the clusters, let us remark that the corresponding optimization problems are NP-hard *and*, most likely, hard to approximate as well. To be formal, consider the following optimization problem: *Given a graph $G(V, E)$, an upper bound for the volume each cluster ℓ, and an upper bound on total volume of T. Find an overlapping cover of G with clusters of volume at most ℓ and total volume of at most T with the minimum swapping probability ρ_∞.* This problem is NP-hard and we sketch a proof of this hardness result.

To do so, we use a hardness result for the minimum bisection problem. *Given a graph, the goal of the minimize bisection problem is to find a minimize size edge set whose deletion splits the graph into two parts with an (almost)-equal number of nodes.* Consider our problem with $\ell = \mathrm{Vol}(G)/2$ and $T = \mathrm{Vol}(G)$, i.e., the total volume of clusters in the overlapping cluster is the same as the volume of G. In this special case, the goal is really to partition the graph into two equal volume clusters and our solution will minimize the edge set between these clusters – thus solving the minimize bisection problem. Hence, the hardness of the minimum bisection problem translates to our problem as well [8]. Other than the NP-hardness, we think that combining inapproximability results for minimum bisection [5, 24] shows the hardness of approximation of our problem. We leave a formal proof of this to the longer version of this manuscript, and proceed to develop a multi-stage scalable heuristic clustering algorithm for the purpose of efficient distributed computation

5. FINDING OVERLAPPING CLUSTERS

In this section, we give an overview of the algorithm for computing the overlapping cover. Given a graph and an upper bound of MaxVol for the volume of the graph stored on each machine, we need find such an overlapping cover that results in a small ρ_∞. The algorithm has 4 main parts:

Identify candidate clusters In the first stage, we want to find clusters with maximum volume MaxVol and small conductance. The idea is to produce a bag of *good* clusters that we'll wheedle down and combine.

Compute well-contained sets For each cluster, we compute a containment score for each vertex. The vertices with highest containment are assigned as *cores* of the cluster. A random walk from a core vertex should take a while to leave the cluster.

Cover with cluster cores We now find a subset of the candidates based on three measures: (i) the total volume should not be too large; (ii) the sum of the cut sizes should be small; and (iii) for each node v, we should pick a cluster C such that v is in the core of C. This is a set-cover problem.

Combine clusters Finally, we combine any small clusters until the final size of each is about MaxVol.

5.1 Candidate clusters

As previously mentioned, we desire a large set of clusters with "high" containment. This implies that we want a small cutsize. We use two procedures to generate candidate clusters. The first is a local clustering algorithm based on personalized PageRank [3]. The second is repeatedly running the METIS algorithm with randomization enabled.

In the first, we compute personalized PageRank vectors with a local algorithm (one that does not access the entire graph) and then generate clusters by sweeping through the ordering induced by the personalized PageRank vector rescaled by inverse degree. In the sweep, we pick the cluster with minimum conductance less than MaxVol. A parameter ε controls the accuracy of the PageRank vector and affects the final size of the cluster. We proceed through all vertices of the graph and compute a PageRank vector from each vertex with a random value of ε chosen to give a cluster with volume between 10 and MaxVol. We will skip looking for a cluster at vertex if that vertex is already in k clusters. This helps keep the work down. The value of α controls the size of the clusters, and $\alpha = 0.99$ produces clusters of a reasonable size. (In this case, α is the link-following probability. This is different from the convention in [3].)

In the second, we repeatedly run METIS with different random seeds and different partition sizes to generate a range of partitions. Each partition is considered a cluster, and thus running this procedure at least twice yields a set of overlapping clusters.

5.2 Core vertices

Given a cluster C, we want to find the vertices that are the most central in C in the sense that a random walk takes a long time to exit the cluster when started from those vertices. We quantify this with the expected leave time for a random walk in a cluster. This derivation is just an application of first-transition analysis in the random walk to the non-cluster vertices. Let $X_0, \ldots$ be the random walk. Also let $u(v) = E[\min(T) \mid X_0 = v, \ldots, X_T \notin C]$ be the expected exit time from the cluster. Trivially, $u(v) = 0$ if $v \notin C$. Applying the memory-less hypothesis and note about $v \notin C$, we have

$$u(v) = 1 + \sum_{(v,r)\in E} \frac{u(r)}{\deg(v)} = 1 + \sum_{\substack{(v,r)\in E \\ r\in C}} \frac{u(r)}{\deg(v)}.$$

Now let $\mathbf{u}$ be a vector of these leavetimes for each vertex in C and let B be the random walk transition matrix restricted to vertices in C. This second equation states $\mathbf{u} = \mathbf{e} + B\mathbf{u}$ where $\mathbf{e}$ is a vector of all ones. Assuming that there is at least one vertex not in C, then this equation is a non-singular linear system. We can quickly approximate its solution using the Neumann series:

$$\mathbf{u} \approx \mathbf{u}_k = B\mathbf{u}_k + \mathbf{e} = \textstyle\sum_{j=0}^{k} B^j \mathbf{e} \qquad \mathbf{u}_0 = \mathbf{e}.$$

This formulation has the attractive property that $\mathbf{u}_k$ is related to the expected leavetime in walks of length at most k. Thus, we can quickly compute expected leavetimes in time $k\,\mathrm{Vol}(C)$, where k is a large but finite random walk time. We use $k = 150$ and additionally modify B to be a PageRank Markov chain with $\alpha = 0.99$. This latter modification changes the answer only slightly, but avoids a few convergence problems when the random walks are slow to exit the cluster. Let $\mathsf{leavetime}(C, v) = u(v)$ which we treat as our containment score. Vertices in a cluster with high leavetime are considered core vertices, which we describe further next.

5.3 Covering the graph

Consider a set of clusters $\mathcal{B} = (C_1, C_2, \ldots, C_m)$. For any cluster C, let $\mathsf{Core}(C)$ be the well-contained set of vertices in C. (These nodes should have long leavetimes.) From $\mathcal{B}$, our goal is to find a minimum cost overlapping cover $(\mathcal{C}, \tau)$ such that for each node $v \in V(G)$, there exists a cluster $C \in \mathcal{C}$ whose core include v, i.e., $v \in V(C)$. Also, it is required that for each node v, $v \in \mathsf{Core}(\tau(v))$. Two parameters are important in determining the cost of the overlapping cover. One parameter is $\mathrm{TotalVol}(\mathcal{C})$, and the other parameter is the total number of edges outgoing from clusters in $\mathcal{C}$. As a heuristic solution, we can formalize this problem as a set cover problem as follows. Each cluster C has some cost $\mathsf{Cost}(C) = \mathrm{Vol}(C) + \gamma|\partial(C)|$, where γ is an appropriate scaling factor. We find it isn't particularly sensitive to this choice and use γ of 0.2.

To find a minimum-cost overlapping cover, we need to find a set of clusters that cover all the nodes of the network. This can be formalized as a set cover problem. In an instance of the set cover problem, we are given a family of sets $S_1, \ldots, S_m$ of a ground set V, each with a cost $c(S_i)$, we need to pick a minimum-cost subfamily of S_i's that cover all elements of V. Given a set of clusters $\mathcal{B}$, we construct a set cover instance as follows: we set S_i in the set cover problem to be $\mathsf{Core}(C_i)$, and the cost $c(S_i)$ to be $\mathsf{Cost}(C)$ as defined above.

The set cover problem is NP-hard, and not approximable better than a factor of $\Omega(\log n)$. Nevertheless, the following simple greedy algorithm gives an $O(\log n)$-approximation algorithm for this problem. It also assigns core vertices in the final step.

```
1. Let $\mathcal{C} = \emptyset$, and $T = V(G)$.
2. While some nodes of $V(G)$ are uncovered,
   (a) Find a cluster $C \notin \mathcal{B}\backslash\mathcal{C}$ that maximizes $\frac{|\mathsf{Core}(C)\cap T|}{\mathsf{Cost}(C)}$.
   (b) Add $C$ to $\mathcal{C}$, and let $T = T\backslash\mathsf{Core}(C)$.
3. For all nodes $v$,
   (a) Find a cluster $C' \in \mathcal{C}$ such that $v \in C'$, and $\mathsf{leavetime}(v, C')$
       is maximized,
   (b) Set $\tau(v) = C'$.
```

5.4 Combining the Clusters

We take into account the maximum allowed volume of the cluster MaxVol while computing the candidate clusters, but we did not use the parameter MaxVol while selecting the final set of clusters C using the set cover greedy algorithm. In this section, we show that we can combine the clusters resulting from the set cover algorithm, and only improve the the total volume, the total cost, and the parameter ρ_∞. Then, we give a heuristic to combine a set of clusters given a maximum volume constraint on each combined cluster.

In order to prove that various cost functions only improve by combining the clusters, we need to prove that these cost functions are subadditive. A function f is subadditive if for any two cluster $A, B \subset V(G)$, $f(A \cup B) \leq f(A) + f(B)$. We first observe this for Vol and ∂ cost functions.

REMARK 1. *For any two clusters C_1 and C_2:*

1. $\mathrm{Vol}(C_1 \cup C_2) \le \mathrm{Vol}(C_1) + \mathrm{Vol}(C_2)$,
2. $\partial(C_1 \cup C_2) \le \partial(C_1) + \partial(C_2)$,

The above remark implies that after combining any set of clusters, the cost function Vol, function ∂, and as a result function Cost will not increase. Now, we show that combining clusters only improve the cost function ρ_∞. Consider an overlapping cover $(\mathcal{C}, \tau)$, and two cluster $C_1, C_2 \in \mathcal{C}$. Let $\mathcal{C}' = \mathcal{C} \backslash \{C_1, C_2\} \cup \{C_1 \cup C_2\}$, i.e, $\mathcal{C}'$ is $\mathcal{C}$ after removing clusters C_1 and C_2 and adding the cluster $C_1 \cup C_2$. Also, construct the mapping τ' as follows: if $\tau(v)$ is C_1 or C_2, then $\tau(v) = C_1 \cup C_2$; otherwise, $\tau'(v) = \tau(v)$. While performing a (random) walk W on the overlapping cover $(\mathcal{C}', \tau')$, we know that whenever we switch from $C_1 \cup C_2$, the same change has to happen in the overlapping cover $(\mathcal{C}, \tau)$. This fact proves that cost function ρ_∞ is not larger for $(\mathcal{C}', \tau')$ compared to $(\mathcal{C}, \tau)$.

The above discussion shows that combining the cluster can only decrease all cost functions that we have considered so far. Here, we give a heuristic for combining the clusters that attempts to maximize such decrease in the cost. The algorithm reads an overlapping cluster $(\mathcal{C}, \tau)$, and a parameter MaxVol and runs the following algorithm:

```
1. Let $\mathcal{C}' = \emptyset$, $\mathcal{B} = \mathcal{C}$, and $T = \emptyset$.
2. While $\mathcal{B}$ is nonempty
   (a) Find $C \in \mathcal{B}$ that minimizes $\mathrm{Vol}(C \cap T)$.
   (b) Let $C' = C$, $\mathcal{B} = \mathcal{B} - \{C\}$, $T = T \cup C$.
   (c) While $\mathcal{B}$ is nonempty
       i. Find $C \in \mathcal{B}$ with $\mathrm{Vol}(C \cup C') \le$ MaxVol and
          maximum $\mathrm{Vol}(C \cap C')/\mathrm{Vol}(C)$.
       ii. Break if no such $C$ exists, otherwise
           let $C' = C' \cup C$, $\mathcal{B} = \mathcal{B} - \{C\}$, $T = T \cup C$.
   (d) Add cluster $C'$ to $\mathcal{C}'$ and update $\tau$ to $C'$ for all ag-
       glomerated clusters.
```

6. SOLVING LINEAR SYSTEMS WITH OVERLAPPING CLUSTERS

We now describe how to solve a certain class of linear systems using overlapping clusters. The convergence analysis of this algorithm places some restriction on where it may be applied, but these are not too restrictive. Many linear systems, including the PageRank linear system [32] and the linear system for Katz scores [22], satisfy these restrictions. Note that this algorithm for PageRank is not new. McSherry discussed its essence in ref. [29].

We first describe the algorithm in general, and then discuss how it applies to the PageRank linear system. Consider a linear system $A\mathbf{x} = \mathbf{b}$. Each row and column of A corresponds to a vertex in our graph. For each cluster, assign that cluster to a processor. The processor holds the columns of the linear system associated with the vertices in its cluster. For a cluster/processor, we allocate a solution vector $\mathbf{x}_C$ and a residual vector $\mathbf{r}_C$. These vectors store information about the solution and residual for all vertices in the cluster. If it helps, think about them as sparse vectors defined only on the vertices of the cluster. Our goal is to compute a sequence of $\mathbf{x}_C^{(k)}$ and $\mathbf{r}_C^{(k)}$ on each processor such that

$$\mathbf{x} \approx \sum_C \mathbf{x}_C^{(k)} \qquad \mathbf{r} \approx \sum_C \mathbf{r}_C^{(k)}$$

where $A\mathbf{x} \approx \mathbf{b}$ and $\mathbf{b} - A\mathbf{x} = \mathbf{r}$ is small. On each processor, we only store a non-zero value in the vector $\mathbf{x}_C$ for the vertices in the cluster, however, we store a non-zero value the vector $\mathbf{r}_C$ for the vertices in the cluster *and* the vertices on the boundary of the cluster. We call the latter $\mathbf{f}_C$ when we refer only to these vertices. Let A_C be the local matrix among vertices in C and let B_C be the matrix from vertices in C to the boundary. In each iteration, we compute

$$\mathbf{x}_C^{(k+1)} = \mathbf{x}_C^{(k)} + A_C^{-1}\mathbf{r}_C(k).$$

This is, we locally solve the linear system. This has the effect of making $\mathbf{r}_C^{(k+1)} = 0$ for all vertices in the cluster. However, we then must update the residual for all boundary vertices. Recall that we use $\mathbf{f}_C$ to denote the "foreign" residual: $\mathbf{f}_C = B_C A_C^{-1} \mathbf{r}_C^{(k)}$. After performing this update, we communicate $\mathbf{f}_C(v)$ to the processor with $\tau(v)$. The remote processor, say C', adds this element into $\mathbf{r}_{C'}^{(k+1)}$. The local processor subsequently sets $\mathbf{f}_C(v) = 0$. Thus, at the end of an iteration $\mathbf{f}_C = 0$ (all the foreign residual was communicated elsewhere), and $\mathbf{r}_C^{(k+1)}$ is only non-zero where other processors communicated their values to C. Take note that *all communication occurs via residuals.* This property is a hallmark of additive Schwarz methods, which locally solve for the residual. Consequently, this algorithm *is* an additive Schwarz method. Also note that $\mathbf{b}$ does not appear in the above iteration. Each solution vector $\mathbf{x}_C$ is set to 0 initially, and thus $\mathbf{r}_C(v)$ is initially set to $\mathbf{b}(v)$ when $\tau(v) = C$. Put another way, the residual is assigned to the "core" vertices. The relationship with the additive Schwarz method lets us conclude that this approach will converge for linear systems with M-matrices and symmetric positive definite systems [37].

When we apply this algorithm, we do not always communicate the foreign residual as described above. Instead, we only communicate elements of the foreign residual if they exceed a solution dependent tolerance. This further reduces communication. *The final communication volume metric is the number of foreign residual elements communicated during the course of the linear system solve.*

PageRank as a linear system is $(I - \alpha P^T)\mathbf{x} = \mathbf{e}$, where P is the random walk normalization of an adjacency matrix and α is the link-following probability in PageRank. This "A" is an M-matrix and thus the above procedure will converge – although PageRank admits a much simpler convergence analysis, which we omit due to space. For PageRank, we solve each local system using the PageRank push algorithm proposed by McSherry [29], using a queue as described by Andersen et al. [3]. This algorithm updates the foreign residuals along with the local residuals. At each iteration, it "pushes" the residual rank to the boundary of the cluster, updating the solution $\mathbf{x}_C$ within the current cluster. After completing all of these push operations, it communicates the residuals to the core vertices as above.

There is no closed form solution for the PageRank vector of an undirected graph, although it is usually nearby the standard normalized degree stationary distribution. Nonetheless, using this approach, it is actually possible to solve a PageRank system with *zero* communication. This occurs when the core vertices are sufficiently far from the boundary, so that the residual rank arriving at the boundary is negligible. In a real implementation, there would be a few small communication steps in aggregating the solution vector $\mathbf{x}$ and checking the residual $\mathbf{r}$; however, we do not model those communication steps in our PageRank communication metric. This result cannot occur in a partitioning because

Table 1: All graphs are undirected and connected. Edges are counted twice and some graphs have self-loops. The first group are geometric networks and the second are information networks.

Graph	$\lvert V\rvert$	$\lvert E\rvert$	max deg	$\lvert E\rvert/\lvert V\rvert$
`onera`	85567	419201	5	4.9
`usroads`	126146	323900	7	2.6
`annulus`	500000	2999258	19	6.0
`email-Enron`	33696	361622	1383	10.7
`soc-Slashdot`	77360	1015667	2540	13.1
`dico`	111982	2750576	68191	24.6
`lcsh`	144791	394186	1025	2.7
`web-Google`	855802	8582704	6332	10.0
`as-skitter`	1694616	22188418	35455	13.1
`cit-Patents`	3764117	33023481	793	8.8

there no boundary around the cluster.

To summarize, when solving a problem with overlapping clusters, solve locally within each cluster and communicate the residual on the boundary to the core vertices identified by the map τ.

7. EXPERIMENTAL RESULTS

At this point, we provide experimental evidence for the ability of our heuristic technique to (i) reduce the swapping probability (Table 2) and (ii) reduce the communication in a distributed PageRank solve (Table 3). Before getting to that demonstration, we first discuss the datasets we utilize for our experiments.

7.1 Data Sets

The data we use to evaluate our techniques comes in two classes: geometric networks and information networks. (We include social networks within information networks.) Our technique is extremely effective at the geometric networks, whereas it is less effective at information networks. Thus, we focus more on the latter. All of our experimental data comes from the following public sources: the University of Florida Sparse Matrix collection [12], the Stanford SNAP collection [26], the Library of Congress subject headings (`lcsh` graph) [39], and the National Highway Planning Network (`usroads` graph – `http://www.fhwa.dot.gov/planning/nhpn/`). The `annulus` graph is a random triangulation of points in an large annulus. Please see Table 1 for information about the size of the networks. We removed the direction of any edges and only consider the largest connected component of the network.

7.2 Cluster performance

The first stage of our algorithm is to generate a set of clusters with small conductance using a local personalized PageRank algorithm [3]. This algorithm has well known properties and has been used in other experimental probes of graph clusters [27]. This algorithm has a few parameters we can control. Although our final goal may be to have clusters of the network with a MaxVol of 10% of the total graph volume, we often found it effective to find smaller clusters and combine these together. We investigate three regimes. The first, called *small*, seeks clusters up to MaxVol of 10000 with a small average volume. The second, called *med*, seeks clusters up to MaxVol of 10000 with a large average volume.

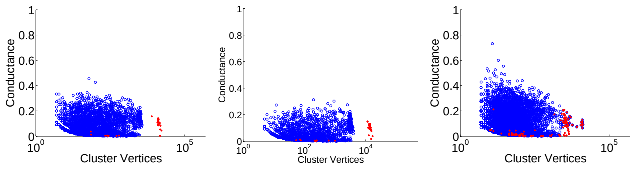


Figure 2: Cluster conductance for three different regimes on the graph lcsh. From left: small, medium, and big. The red dots are the results with the final set of combined clusters.

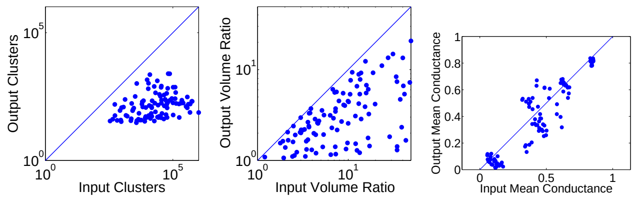


Figure 3: Cluster combinations. See §7.3 for details.

The third, called *big*, seeks clusters up to MaxVol of 10% of the total graph volume with a average volume comparable to the medium set. The conductance of the sets generated by these algorithms is plotted in Figure 2 for the graph `lcsh`.

7.3 Combine performance

We now show that we can effectively combine clusters. See Figure 3 for relationships between the input to the cluster combination process and the output in three measures. This experiment summarizes the result of all the cluster combination experiments we performed on the information networks during the sweep experiment discussed in the next section. The first measure is the number of clusters. In this measure, the algorithm effectively combines up to 1,000,000 initial clusters down to a few hundred or thousand. The second measure is the volume ratio of the input clusters to the volume ratio of the output clusters. This ratio always decreases, though not always by a large margin. The final measure is the average conductance of the input clusters to the average conductance of the output clusters. Here, we find the average conductance *can* increase, although it most often decreases.

7.4 Distributed computation measures

At last we come to our experiment demonstrating the benefit of overlapping clusters. In this experiment, MaxVol is 10% of the edges in the graph. For each graph in Table 1, we then compute two partitions: one with METIS and one with GRACLUS. We asked for more than 10 partitions when the algorithms returned partitions exceeding MaxVol; and then combined small partitions using a heuristic based on the intersection in boundary vertices until all partitions were about size MaxVol.

Tables For the overlapping experiments, we ran our cluster identification algorithm in the *small*, *med*, and *big* regimes described in Section 7.2. We examined these cases with a target overlap T of 1, 2, 5, 10, and 30. Recall that the PageRank clustering algorithm looks for a cluster around a starting vertex. We sequentially iterate through all the vertices of the graph and start a cluster from that vertex if it is not already in T clusters. This setup has 15 different experiments for each graph. In the tables, we label each method by the

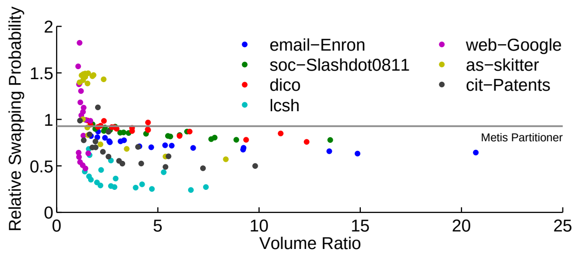


Figure 4: Swapping probability relative to METIS for all our overlapping experiments.

combination of regime and overlap, i.e. "med.5" is the med regime with an overlap of 5. We also use METIS to generate overlapping clusters. We do so by running it 2, 4, or 10 times to generate the same number of partitions. Each run uses a different seed and a different number of partitions designed to produce clusters with volume of 1000 and 10000. This gives us the methods "metis.2", "metis.4", and "metis.10".

After generating the clusters, we combine them into a set of clusters with MaxVol up to 10% of the edges in the graph. Given the resulting partitions and overlapping clusterings, we estimate the swapping probability via sampling and measure the communication volume of solving PageRank on the overlapping clusters using the procedure from Section 6. The results for swapping probability are shown in Table 2. Each row of the table shows the swapping probability for the best partition result, along with the average conductance of the clusters in that partition. When the partitioning results from GRACLUS are superior to those of METIS, we show them in the table and denote them with a $*$. The next set of columns shows the best swapping probability from any of the overlapping clustering experiments. We also list the performance ratio (overlap / partition), volume ratio, average conductance, and method for the best overlap result as well. In Table 3 we see the same data, but for PageRank communication volume instead. The PageRank problem used $\alpha = 0.85$ and a relative tolerance of $1/n$ where n is the number of vertices in the graph.

The results for swapping probability show that our overlapping cluster method reduces it by a factor of 10 to 100 on geometric networks and 1-2 for information networks. The results for PageRank communication are perhaps more compelling for geometric networks. In all cases, we only needed a trivial amount of communication compared to the partitioning approach. (The zero result is not an error, see §6.) For information networks, they are much less compelling. We reduce the communication for three graphs: `lcsh`, `web-Google`, and `cit-Patents`. These graphs have the lowest average conductance scores among the information networks, suggesting a possible a priori performance estimate.

Finally, Figure 4 shows the reduction in swapping probability occurred in almost every experiment we performed with the overlapping clusters. The figure also shows that slightly smaller swapping probabilities may be obtained for significantly less total volume.

8. CONCLUSION AND FUTURE WORK

Minimizing communication in a distributed computation with an information network is a challenging problem. Here, we have tried to address the issue of whether careful use of *overlap* or redundancy of the distributed data can help reduce the communication. Towards that end, we proposed a computational framework with overlapping clusters. These overlapping groups of vertices are simple to use, and we propose a procedure to find a good set of them. The results on our large information networks are encouraging as preliminary support for the use of overlapping clusters. We are consistently able to reduce the swapping probability of a random walk in a network with only a moderate amount of overlap. Typically, this metric decreases almost monotoni-

Table 2: Swapping probability results. See §7.4, *Tables* paragraph for more information.

Graph	Swap. Prob. of Partition	Avg. Cond.	Swap. Prob. of Overlap	Perf. Ratio	Vol. Ratio	Avg. Cond.	Method
`onera`	7.6×10^{-4}	0.02	1×10^{-4}	0.129	2.82	0.03	Med.30
`usroads`	1.3×10^{-4}*	<0.01	1×10^{-6}	0.008	1.49	0.01	Med.30
`annulus`	1×10^{-4}	<0.01	5×10^{-6}	0.049	1.17	<0.01	Med.10
`email-Enron`	0.02	0.39	0.013	0.650	14.86	0.47	Big.30
`soc-Slashdot`	0.03	0.66	0.026	0.867	13.52	0.65	Med.30
`dico`	0.04	0.82	0.03	0.750	12.35	0.82	Big.30
`lcsh`	0.003*	0.06	0.0007	0.233	6.63	0.12	Med.30
`web-Google`	7.8×10^{-4}*	0.02	4.6×10^{-4}	0.592	1.43	0.02	Big.30
`as-skitter`	0.005	0.1	0.004	0.549	8.36	0.2	Big.30
`cit-Patents`	0.0064	0.13	0.0034	0.524	3.25	0.42	Small.10

Table 3: PageRank computation volume. See §7.4, *Tables* paragraph for more information.

Graph	Comm. of Partition	Avg. Cond.	Comm. of Overlap	Perf. Ratio	Vol. Ratio	Avg. Cond.	Method
`onera`	18654	0.02	48	0.003	2.82	0.03	Med.30
`usroads`	3256*	<0.01	0	0.000	1.49	0.01	Med.30
`annulus`	12074	<0.01	2	0.000	0.01	<0.01	Med.10
`email-Enron`	194536*	0.4	235316	1.210	1.7	0.46	Metis.2
`soc-Slashdot`	875435*	0.68	1.3×10^{6}	1.480	1.78	0.74	Metis.2
`dico`	1.5×10^{6}*	0.79	2.0×10^{6}	1.320	1.53	0.84	Metis.2
`lcsh`	73000*	0.06	48777	0.668	2.17	0.08	Small.5
`web-Google`	201159*	0.02	167609	0.833	1.57	0.04	Metis.10
`as-skitter`	2.4×10^{6}	0.1	3.9×10^{6}	1.645	1.93	0.24	Metis.10
`cit-Patents`	8.7×10^{6}	0.13	7.3×10^{6}	0.845	1.34	0.16	Metis.4

cally with increasing overlap. For such applications: *overlap*, do not *divide*. In the future, we plan to investigate how caching small blocks of vertex information would compare against our centralized heuristic approach.

The results in our second metric – the communication volume of a distributed linear system solve – are more murky. Our heuristic algorithm to identify good overlapping clusters is able to find a good set in geometric meshes. In large information networks, using a set of overlapping partitions by repeatedly running METIS and using our cluster combination algorithm seems to perform better. In a few cases, these clusters actually reduce the communication. For the communication problem on an information network, there seems to be a tension between large overlap and total communication volume. We are most successful when we find a set of clusters that does not increase the total volume stored too much. Currently, our bag of clusters from the local PageRank algorithm does not appear to identify a good set for this task. We plan to study variations on this heuristic to identify better sets of clusters from the algorithm. Also, we plan to investigate some of the overlapping community detection algorithms from the statistical physics community.

Our goal is to highlight possible advantages of overlapping clustering over partitioning in the context of distributed computing. We believe that our evaluation of overlapping clustering for two related objective functions along with our preliminary theoretical analysis have shown the effectiveness of this idea. Evaluating these ideas in a true distributed setting is an interesting subject of future research.

9. REFERENCES

[1] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466:761–764, 2009.

[2] R. Andersen. A local algorithm for finding dense subgraphs. In *SODA*, pages 1003–1009, 2008.

[3] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using PageRank vectors. In *FOCS*, 2006.

[4] S. Arora, S. Rao, and U. V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2), 2009.

[5] P. Berman and M. Karpinski. Approximation hardness of bounded degree min-csp and min-bisection. In *Automata, Languages and Programming*, volume 2380 of *LNCS*, pages 782–782. Springer Berlin / Heidelberg, 2002.

[6] R. Bru, F. Pedroche, and D. B. Szyld. Additive Schwarz iterations for Markov chains. *SIAM J. Matrix Anal. Appl.*, 27(2):445–458, 2005.

[7] R. Bru, F. Pedroche, and D. B. Szyld. Cálculo del vector PageRank de Google mediante el método iterativo de Schwarz. In J. P. A. et al., editor, *Congreso de Métodos Numéricos en Ingeniería*, Granada, Spain, 2005. In Spanish.

[8] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Inf. Process. Lett.*, 42(3):153–159, 1992.

[9] U. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst.*, 10(7):673–693, 1999.

[10] A. J. Cole and D. Wishart. An improved algorithm for the Jardine-Sibson method of generating overlapping clusters. *The Computer Journal*, 13(2):156–163, 1970.

[11] A. Czumaj, O. Goldreich, D. Ron, C. Seshadhri, A. Shapira, and C. Sohler. Finding cycles and trees in sublinear time. *arXiv*, cs.DM:1007.4230, 2010.

[12] T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. Submitted., 2010.

[13] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(11):1944–1957, November 2007.

[14] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White. *The Sourcebook of Parallel Computing.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[15] S. Fortunato. *arXiv*, physics.soc-ph:0906.0612v2, 2010.

[16] D. Fritzsche. Fast graph partitioning and applications to preconditioning. Presentation at SIAM Combinatorial Scientific Computing 2009, October 2009. Available online `http://www.math.temple.edu/~daffi/pablo/csc09.pdf`.

[17] S. L. Graham, M. Snir, and C. A. Patterson, editors. *Getting up to Speed: The Future of Supercomputing.* National Academies Press, 2005.

[18] B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel Comput.*, 26(12):1519–1534, November 2000.

[19] S. Kale and C. Seshadhri. Combinatorial approximation algorithms for maxcut using random walks. In *Innovations in Computer Science*, pages 367–388, 2011.

[20] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 938–948, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[21] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[22] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, March 1953.

[23] R. Khandekar, G. Kortsarz, and V. Mirrokni. Overlapping vs. non-overlapping clustering: Conductance and density, 2010.

[24] S. Khot. Ruling out ptas for graph min-bisection, densest subgraph and bipartite clique. In *FOCS*, pages 136–145, Washington, DC, USA, 2004. IEEE Computer Society.

[25] F. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

[26] J. Leskovec. The Stanford large network dataset collection. `http://snap.stanford.edu/data/index.html`, 2010.

[27] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *WWW*, pages 695–704, New York, NY, USA, 2008. ACM.

[28] A. A. Maleev. On the SOR method with overlapping subsystems. *Comp. Math. and Math. Phys.*, 46(6):919–929, June 2006.

[29] F. McSherry. A uniform approach to accelerated PageRank computation. In *WWW*, pages 575–582, 2005.

[30] N. Mishra, R. Schreiber, I. Stanton, and R. E. Tarjan. Clustering social networks. In *WAW2007*, pages 56–67, 2007.

[31] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick. Minimizing communication in sparse matrix solvers. In *SC09*, pages 1–12, New York, NY, USA, November 2009. ACM.

[32] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford University, November 1999.

[33] J. X. Parreira, C. Castillo, D. Donato, S. Michel, and G. Weikum. The juxtaposed approximate PageRank method for robust PageRank approximation in a peer-to-peer web search network. *The VLDB Jour.*, 17(2):291–313, March 2008.

[34] A. D. Sarma, D. Nanongkai, G. Pandurangan, and P. Tetali. Near-optimal sublinear time bounds for distributed random walks. *arXiv*, cs.DS:0911.3195, 2009.

[35] D. A. Spielman and S.-H. Teng. Spectral partitioning works: Planar graphs and finite element meshes. In *FOCS*, pages 96–105, 1996.

[36] A. P. Streich, M. Frank, D. Basin, and J. M. Buhmann. Multi-assignment clustering for boolean data. In *ICML*, 2009.

[37] D. B. Szyld and A. Frommer. Weighted max norms, splittings, and overlapping additive schwarz iterations. *Numerische Mathematik*, 83:259–278, 1999.

[38] Ümit V. Çatalyürek and C. Aykanat. A fine-grain hypergraph model for 2d decomposition of sparse matrices. In *PDPS*, pages 1199–1204, April 2001.

[39] Various. The Library of Congress subject headings. `http://id.loc.gov`, 2009. Accessed in September 2009.

[40] D. Zhou, J. Huang, and B. Schölkopf. Learning from labeled and unlabeled data on a directed graph. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, pages 1036–1043, New York, NY, USA, 2005. ACM Press.