# Finding Dense Subgraphs via Low-Rank Bilinear Optimization

Dimitris S. Papailiopoulos Ioannis Mitliagkas Alexandros G. Dimakis Constantine Caramanis

The University of Texas at Austin

### Abstract

Given a graph, the Densest k-Subgraph (DkS) problem asks for the subgraph on k vertices that contains the largest number of edges. In this work, we develop a new algorithm for DkS that searches a low-dimensional space for provably dense subgraphs. Our algorithm comes with novel performance bounds that depend on the graph spectrum. Our graph-dependent bounds are surprisingly tight for real-world graphs where we find subgraphs with density provably within 70% of the optimum. These guarantees are significantly tighter than the best available worst case a priori bounds.

Our algorithm runs in nearly linear time, under spectral assumptions satisfied by most graphs found in applications. Moreover, it is highly scalable and parallelizable. We demonstrate this by implementing it in MapReduce and executing numerous experiments on massive real-world graphs that have up to billions of edges. We empirically show that our algorithm can find subgraphs of significantly higher density compared to the previous state of the art.

## 1. Introduction

Given a graph  $\mathcal{G}$  on n vertices with m edges and a parameter k, we are interested in finding an induced subgraph on k vertices with the largest average degree, also known as the *maximum density*. This is the *Densest k-Subgraph* (DkS) – a fundamental problem in combinatorial optimization with applications in numerous fields including social sciences, communication networks, and biology (see *e.g.* (Hu et al., 2005; Gibson et al., 2005; Dourisboure et al., 2007; Saha et al., 2010; Miller et al., 2010; Bahmani et al., 2012)).

DkS is a notoriously hard problem. It is NP-hard by reduc-

DIMITRIS @ UTEXAS.EDU IOANNIS @ UTEXAS.EDU DIMAKIS @ AUSTIN.UTEXAS.EDU CONSTANTINE @ UTEXAS.EDU

tion to MAXCLIQUE. Moreover, Khot showed in (Khot, 2004) that, under widely believed complexity-theoretic assumptions, DkS cannot be approximated within an arbitrary constant factor.<sup>1</sup> The best known approximation ratio was  $n^{1/3+\epsilon}$  (for some small  $\epsilon$ ) due to (Feige et al., 2001). Recently, (Bhaskara et al., 2010) introduced an algorithm with approximation ratio  $n^{1/4+\epsilon}$ , that runs in time  $n^{O(1/\epsilon)}$ . Such results, where the approximation factor scales as a polynomial in the number of vertices, are too pessimistic for real-world applications. This resistance to better approximations, despite the long history of the problem, suggests that DkS is probably very hard in the worst case.

**Our Contributions.** In this work we move beyond the worst case framework. We present a novel DkS algorithm that has two key features: i) it comes with approximation guarantees that are surprisingly tight on real-world graphs and ii) it is fully parallelizable and can scale up to graphs with billions of edges.

Our algorithm combines spectral and combinatorial techniques; it relies on examining candidate subgraphs obtained from vectors lying in a low-dimensional subspace of the adjacency matrix of the graph. This is accomplished through a framework called the *Spannogram*, which we define below.

Our approximation guarantees are graph-dependent: they are related to the spectrum of the adjacency matrix of the graph. Let opt denote the average degree (i.e., the density) of the densest k-subgraph, where  $0 \le \text{opt} \le k - 1$ . Our algorithm takes as input the graph, the subgraph size k, and an accuracy parameter  $d \in \{1, \ldots, n\}$ . The output is a subgraph on k vertices with density  $\text{opt}_d$ , for which we obtain the following approximation result:

**Theorem 1.** For any unweighted graph, our algorithm outputs in time  $O\left(\frac{n^{d+2} \cdot \log n}{\delta}\right)$  a k-subgraph that has density

 $\mathsf{opt}_d \geq 0.5 \cdot (1-\delta) \cdot \mathsf{opt} - 2 \cdot |\lambda_{d+1}|,$ 

with probability  $1 - \frac{1}{n}$ , where  $\lambda_i$  is the *i*th largest, in mag-

Proceedings of the 31<sup>st</sup> International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32. Copyright 2014 by the author(s).

<sup>&</sup>lt;sup>1</sup>approximation ratio  $\rho$  means that there exists an algorithm that produces in polynomial time a number A, such that  $1 \leq \frac{\text{opt}}{A} \leq \rho$ , where opt is the optimal density.

nitude, eigenvalue of the adjacency matrix of the graph. If the graph is bipartite, or if the largest d eigenvalues of the graph are positive, then our algorithm runs in time  $O(n^{d+1} + T_d)$ , and outputs a k-subgraph with density

$$\operatorname{opt}_d \geq \operatorname{opt} - 2 \cdot |\lambda_{d+1}|,$$

where  $T_d$  is the time to compute the *d* leading eigenvectors of the adjacency matrix of the graph.

Our bounds come close to  $2 + \epsilon$  and  $1 + \epsilon$  factor approximations, when  $\lambda_{d+1}$  is significantly smaller than the density of the densest k-subgraph. In the following theorem, we give such an example. However, we would like to note that in the worst case our bounds might not yield something meaningful.

**Theorem 2.** If the densest-k-subgraph contains a constant fraction of all the edges, and  $k = \Theta(\sqrt{E})$ , then we can approximate DkS within a factor of  $2 + \epsilon$ , in time  $n^{O(1/\epsilon^2)}$ . If additionally the graph is bipartite, we can approximate DkS within a factor of  $1 + \epsilon$ .

The above result is similar to the  $1 + \epsilon$  approximation ratio of (Arora et al., 1995) for dense graphs, where the densest*k*-subgraph contains a constant fraction of the  $\Omega(n^2)$  edges, where  $k = \Omega(n)$ . The innovation here is that our ratio also applies to *sparse graphs* with sublinear number of edges.

**Computable upper bounds.** In addition to these theoretical guarantees, our analysis allows us to obtain a graphdependent upper bound for the optimal subgraph density. This is shown in Fig. 3 in our experimental section, where for many graphs our algorithm is provably within 70% from the upper bound of opt. These are far stronger guarantees than the best available *a priori* bounds. This illustrates the potential power of graph-dependent guarantees that, however, require the execution of an algorithm.

**Nearly-linear time approximation.** Our algorithm has a worst-case running time of  $O\left(\frac{n^{d+2} \cdot \log n}{\delta}\right)$ . Under some mild spectral assumptions, a randomized version of our algorithm runs in nearly-linear time.

**Theorem 3.** Let the d largest eigenvalues of the graph be positive, and let the d-th, (d + 1)-st largest have constant ratio:  $\left|\frac{\lambda_d}{\lambda_{d+1}}\right| \geq C$ . Then, we can modify our algorithm to output, with probability  $1 - \delta$ , a k-subgraph with density  $(1 - \epsilon)^2 \cdot \operatorname{opt}_d$ , in time  $O\left(m \cdot \log n + \frac{n}{\epsilon^d} \cdot \log\left(\frac{1}{\epsilon \cdot \delta}\right)\right)$ , where m is the number of edges.

We found that the above spectral condition holds for all  $d \le 5$ , in many real-world graphs that we tested.

**Scalability.** We develop two key scalability features that allow us to scale up efficiently on massive graphs.

*Vertex sparsification:* We introduce a pre-processing step that eliminates vertices that are unlikely to be part of

the densest k-subgraph. The elimination is based on the vertices' weighted leverage scores (Mahoney & Drineas, 2009; Boutsidis et al., 2009) and admits a provable bound on the introduced error. We empirically found that even with a negligible additional error, the elimination dramatically reduced problem sizes in all tested datasets.

*MapReduce implementation:* We show that our algorithm is *fully-parallelizable* and tailor it for the MapReduce framework. We use our MapReduce implementation to run experiments on Elastic MapReduce (EMR) on Amazon. In our large-scale experiments, we were able to scale out to thousands of mappers and reducers in parallel over 800 cores, and find large dense subgraphs in graphs with billions of edges.

#### 1.1. Related work

*DkS algorithms*: One of the few positive results for DkS is a  $1 + \epsilon$  approximation for dense graphs where  $m = \Omega(n^2)$ , and in the linear subgraph setting  $k = \Omega(n)$  (Arora et al., 1995). For some values of  $m = o(n^2)$  a  $2 + \epsilon$  approximation was established by (Suzuki & Tokuyama, 2005). Moreover, for any  $k = \Omega(n)$  a constant factor approximation is possible via a greedy approach by (Asahiro et al., 2000), or via semidefinite relaxations by (Srivastav & Wolf, 1998) and (Feige & Langberg, 2001). Recently, (Alon et al., 2013) established new approximate solver for graphs with small " $\epsilon$ -rank," using an approximate solver for low-rank perturbed versions of the adjacency matrix.

There is a vast literature on algorithms for detecting communities and well-connected subgraphs: greedy schemes (Ravi et al., 1994), optimization approaches (Jethava et al., 2012; d'Aspremont et al., 2010; Ames, 2011), and the truncated power method (Yuan & Zhang, 2011). We compare with various of these algorithms in our evaluation section.

The Spannogram framework: We present an exact solver for *bilinear* optimization problems on matrices of constant rank, under  $\{0, 1\}$  and sparsity constraints on the variables. Our theory is a generalization of the Spannogram framework, originally introduced in the foundational work of (Karystinos & Liavas, 2010) and further developed in (Asteris et al., 2014; Papailiopoulos et al., 2013), that obtains exact solvers for low-rank *quadratic optimization* problems with combinatorial constraints, such as sparse PCA.

*MapReduce algorithms for graphs*: The design of MapReduce algorithms for massive graphs is an active research area as Hadoop becomes one of the standards for storing large data sets. The related work by Bahmani *et al.* (Bahmani et al., 2012) designs a novel MapReduce algorithm for the *densest subgraph* problem. This densest subgraph problem requires finding a subgraph of highest *normalized density* without enforcing a specific subgraph size k. Surprisingly, without a subgraph size restriction, the densest subgraph becomes polynomially solvable and therefore fundamentally different from what we consider in this paper.

### 2. Proposed Algorithm

The *density* of a subgraph indexed by a vertex set  $S \subseteq \{1, \ldots, n\}$  is equal to the average degree of the vertices within S:

$$\mathsf{den}(\mathcal{S}) = \frac{\mathbf{1}_{\mathcal{S}}^T \mathbf{A} \mathbf{1}_{\mathcal{S}}}{|\mathcal{S}|}$$

where **A** is the adjacency matrix  $(A_{i,j} = 1 \text{ if } (i, j) \text{ is an} edge, else <math>A_{i,j} = 0$ ) and the indicator vector  $\mathbf{1}_{S}$  has 1s in the entries indexed by S and 0 otherwise. Observe that  $\mathbf{1}_{S}^{T}\mathbf{A}\mathbf{1}_{S} = \sum_{i,j\in S} A_{i,j}$  is twice the number of edges in the subgraph with vertices in S.

For a fixed subgraph size |S| = k, we can express DkS as a quadratic optimization:

DkS: opt = 
$$(1/k) \cdot \max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A} \mathbf{1}_{\mathcal{S}}$$

where |S| = k denotes that the optimization variable is a *k*-vertex subset of  $\{1, \ldots, n\}$ .

The bilinear relaxation of DkS. We approximate DkS via approximating its bipartite version. This problem can be expressed as a bilinear maximization:

$$\mathsf{DBkS}: \quad \mathsf{opt}^{\mathsf{B}} = (1/k) \cdot \max_{|\mathcal{X}| = k} \max_{|\mathcal{Y}| = k} \mathbf{1}_{\mathcal{X}}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}}.$$

As we see in the following lemma, the two problems are fundamentally related: a good solution for the bipartite version of the problem maps to a "half as good" solution for DkS. The proof is given in the Supplemental Material.

**Lemma 1.** A  $\rho$ -approximation algorithm for DBkS implies a  $2\rho$ -approximation algorithm for DkS.

#### 2.1. DkS through low rank approximations

At the core of our approximation lies a constant rank solver: we show that DBkS can be solved in polynomial time on constant rank matrices. We solve constant rank instances of DBkS instead of DkS due to an important implication: DkS is NP-hard even for rank-1 matrices with 1 negative eigenvalue, as we show in the Supplemental Material.

The exact steps of our algorithm are given in the pseudocode tables referred to as Algorithms 1-3.<sup>2</sup> The output of our algorithm is a k-subgraph  $Z_d$  that has density  $opt_d$  that comes with provable guarantees. We present our theoretical guarantees in the next subsection.

Our main algorithmic innovation, the constant rank solver for DBkS (Algorithms 2-3), is called many times: in lines 5, 8, and 15 of our general DkS approximation, shown as Algorithm 1. We describe its steps subsequently.

Algorithm	1	low-rank	an	nrovima	tions	for	DkS
Algorithm	L	10w-rank	ap	рголина	uons	101	

1:  $[\mathbf{V}_d, \mathbf{\Lambda}_d] = \text{EVD}(\mathbf{A}, d)$ 2: if  $\mathcal{G}$  is bipartite then 3:  $\mathbf{B} =$ bi-adjacency of  $\mathcal{G}$  $[\mathbf{V}_d, \mathbf{\Sigma}_d, \mathbf{U}_d] = \text{SVD}(\mathbf{B}, d)$ 4:  $\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg \max_{|\mathcal{X}| + |\mathcal{Y}| = k} \mathbf{1}_{\mathcal{X}}^T \mathbf{V}_d \mathbf{\Sigma}_d \mathbf{U}_d^T \mathbf{1}_{\mathcal{Y}}.$ 5: 6:  $\mathcal{Z}_d = \mathcal{X}_d \cup \mathcal{Y}_d$ 7: else if The first d eigenvalues of A are positive then 8:  $\{\mathcal{X}_d, \mathcal{X}_d\} = \arg \max_{|\mathcal{X}| = |\mathcal{Y}| = k} \mathbf{1}_{\mathcal{X}}^T \mathbf{V}_d \mathbf{\Lambda}_d \mathbf{V}_d^T \mathbf{1}_{\mathcal{Y}}.$ 9:  $\mathcal{Z}_d = \mathcal{X}_d$ 10: else for  $i = 1 : \frac{\log n}{\delta}$  do 11: draw n fair coins and assign them to vertices 12:  $\mathcal{L}$  = vertices with heads;  $\mathcal{R} = \{1, \dots, n\} - \mathcal{L}$ 13: 14:  $\mathbf{B}_{d}^{i} = [\mathbf{V}_{d} \mathbf{\Lambda}_{d} \mathbf{V}_{d}^{T}]_{\mathcal{L},\mathcal{R}}$  $\{\mathcal{X}^i, \mathcal{Y}^i\} = \arg \max_{|\mathcal{X}|+|\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{B}_d^i \mathbf{1}_{\mathcal{Y}}.$ 15: 16: end for  $\{\mathcal{X}^i, \mathcal{Y}^i\} = \arg \max_{1 \le i \le n} \mathbf{1}_{\mathcal{X}^i}^T \mathbf{B}_d^i \mathbf{1}_{\mathcal{Y}^i}$ 17: 18:  $\mathcal{Z}_d = \mathcal{X}_d \cup \mathcal{Y}_d$ 19: end if 20: Output:  $Z_d$ 

**Constant rank solver for DBkS.** In the following we present an exact solver for DBkS on constant rank approximations of **A**. Our DkS algorithm makes a number of calls to the DBkS low-rank solver on slightly different (some times rectangular) matrices. The details of the general low-rank solver are in the Supplemental Material.

**Step 1:** Obtain  $\mathbf{A}_d = \sum_{i=1}^d \lambda_i \mathbf{v}_i \mathbf{v}_i^T$ , a rank-d approximation of  $\mathbf{A}$ . Here,  $\lambda_i$  is the *i*-th largest in magnitude eigenvalue and  $\mathbf{v}_i$  the corresponding eigenvector.

**Step 2:** Use  $\mathbf{A}_d$  to obtain  $O(n^d)$  candidate subgraphs. For any matrix  $\mathbf{A}$  we can solve DBkS by exhaustively checking all  $\binom{n}{k}^2$  pairs  $(\mathcal{X}, \mathcal{Y})$  of k-subsets of vertices. Surprisingly, if we want to find the  $\mathcal{X}, \mathcal{Y}$  pairs that maximize  $\mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}}$ , i.e., the bilinear problem on the rank-d matrix  $\mathbf{A}_d$ , then we show that only  $O(n^d)$  candidate pairs need to be examined.

**Step 3:** Check all *k*-set pairs  $\{\mathcal{X}, \mathcal{Y}\}$  obtained by Step 2, and output the one with the largest density on the low-rank weighted adjacency  $\mathbf{A}_d$ .

In the next section, we derive the constant rank-solver using two key facts. First, for each fixed vertex set  $\mathcal{Y}$ , we show that it is easy to find the optimal set  $\mathcal{X}$  that maximizes  $\mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}}$  for that  $\mathcal{Y}$ . Since this turns out to be easy, then the challenge is to find the number of different vertex sets  $\mathcal{Y}$  that we need to check. Do we need to exhaustively check all  $\binom{n}{k}$  k-sets  $\mathcal{Y}$ ? We show that this question is equivalent

<sup>&</sup>lt;sup>2</sup>In the pseudocode of Algorithm 2, top<sub>k</sub>( $\mathbf{v}$ ), denotes the indices of the k largest signed elements of  $\mathbf{v}$ .

to searching the span of the first d eigenvectors of  $\mathbf{A}$ , and collecting in a set  $S_d$  the top-k coordinates of all vectors in that d-dimensional space. By modifying the Spannogram theory of (Karystinos & Liavas, 2010; Asteris et al., 2014), we show how this set has size  $O(n^d)$  and can be constructed in time  $O(n^{d+1})$ . This will imply that DBkS can be solved in time  $O(n^{d+1})$  on  $\mathbf{A}_d$ .

**Computational Complexity.** The worst-case time complexity of the constant-rank DBkS solver on  $\mathbf{A}_d$  is  $O(\mathsf{T}_d + n^{d+1})$ , where  $\mathsf{T}_d$  is the time to compute the first *d* eigenvectors of  $\mathbf{A}$ . Under conditions satisfied by many real world graphs, we show that we can modify our algorithm and obtain a randomized one that succeeds with probability  $\delta$  and is  $\epsilon$  far from the optimal rank-*d* solver, while its complexity reduces to *nearly linear* in the number of edges *m* of the graph  $\mathcal{G}: O(m \cdot \log n + \frac{n}{\epsilon^d} \cdot \log(\frac{1}{\epsilon \cdot \delta}))$ .

Algorithm 2 lowrankDBkS  $(k, d, \mathbf{A})$ 1:  $[\mathbf{V}_d, \mathbf{\Lambda}_d] = \text{EVD}(\mathbf{A}, d)$ 2:  $S_d = \text{Spannogram}(k, \mathbf{V}_d)$ 3:  $\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg \max_{|\mathcal{X}|=k} \max_{\mathcal{Y} \in S_d} \mathbf{1}_{\mathcal{X}}^T \mathbf{V}_d \mathbf{\Lambda}_d \mathbf{V}_d^T \mathbf{1}_{\mathcal{Y}}$ 4: **Output:**  $\{\mathcal{X}_d, \mathcal{Y}_d\}$ 1: Spannogram  $(k, \mathbf{V}_d)$ 2:  $S_d = \{\text{top}_k(\mathbf{v}) : \mathbf{v} \in \text{span}(\mathbf{v}_1, \dots, \mathbf{v}_d)\}$ 3: **Output:**  $S_d$ .

#### 2.2. Approximation Guarantees

We approximate DBkS by finding a solution to the constant rank problem

$$\max_{|\mathcal{X}|=k} \max_{|\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}}.$$

We output a pair of vertex sets,  $\mathcal{X}_d$ ,  $\mathcal{Y}_d$ , which we refer to as the *rank-d optimal solution*, that has density

$$\mathsf{opt}_d^\mathsf{B} = (1/k) \cdot \mathbf{1}_{\mathcal{X}_d}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}_d}$$

Our approximation guarantees measure how far  $opt_d^B$  is from  $opt^B$ , the optimal density for DBkS. Our bounds capture a simple core idea: the loss in our approximation comes due to solving the problem on  $A_d$  instead of solving it on the full rank matrix A. This loss is quantified in the next lemma. The detailed proofs of the following results are in the supplemental material.

**Lemma 2.** For any matrix  $\mathbf{A}$ :  $\operatorname{opt}_{d}^{\mathsf{B}} \geq \operatorname{opt}^{\mathsf{B}} - 2 \cdot |\lambda_{d+1}|$ , where  $\lambda_{i}$  is the *i*th largest eigenvalue of  $\mathbf{A}$ .

Using an appropriate pre-processing step and then running Algorithm 2 as a subroutine on a sub-sampled and low-rank version of **A**, we output a *k*-subgraph  $Z_d$  that has density opt<sub>d</sub>. By essentially combining Lemmata 1 and 2 we obtain the following bounds.

**Theorem 1.** Algorithm 1 outputs in time  $O\left(\frac{n^{d+2} \cdot \log n}{\delta}\right) a$ k-subgraph that has density

$$\mathsf{opt}_d = \mathsf{den}(\mathcal{Z}_d) \ge 0.5 \cdot (1 - \delta) \cdot \mathsf{opt} - 2 \cdot |\lambda_{d+1}|,$$

with probability  $1 - \frac{1}{n}$ , where  $\lambda_i$  is the *i*th largest, in magnitude, eigenvalue of the adjacency matrix of the graph. If the graph is bipartite, or if the largest d eigenvalues of the graph are positive, then our algorithm runs in time  $O(n^{d+1} + T_d)$ , and outputs a k-subgraph with density  $opt_d \ge opt - 2 \cdot |\lambda_{d+1}|$ , where  $T_d$  is the time to compute the d leading eigenvectors of the adjacency matrix of the graph.

Using bounds on eigenvalues of graphs, Theorem 1 translates to the following approximation guarantees.

**Theorem 2.** If the densest-k-subgraph contains a constant fraction of all the edges, and  $k = \Theta(\sqrt{E})$ , then we can approximate DkS within a factor of  $2 + \epsilon$ , in time  $n^{O(1/\epsilon^2)}$ . If additionally the graph is bipartite, then we can approximate DkS within a factor of  $1 + \epsilon$ .

**Remark 1.** The above results are similar to the  $1 + \epsilon$  ratio of (Arora et al., 1995), which holds for graphs where the densest-k-subgraph contains  $\Omega(n^2)$  edges.

**Graph dependent bounds.** For any given graph, after running our constant rank solver on  $\mathbf{A}_d$ , we can compute an upper bound to the optimal density opt via bounds on opt<sup>B</sup>, since it is easy to see that opt<sup>B</sup>  $\geq$  opt. Our graph-dependent bound is the minimum of three upper bounds on the unknown optimal density:

Lemma 3. The optimal density of DkS can be bounded as

opt 
$$\leq \min\left\{ (1/k) \cdot \mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d} + |\lambda_{d+1}|, \ k-1, \ \lambda_1 \right\}.$$

In our experimental section, we plot the above upper bounds, and show that for most tested graphs our algorithm performs *provably* within 70% from the upper bound on the optimal density. These are far stronger guarantees than the best available *a priori* bounds.

### 3. The Spannogram Framework

In this section, we describe how our constant rank solver operates by examining candidate vectors in a low-dimensional span of A.

Here, we work on a rank-*d* matrix  $\mathbf{A}_d = \mathbf{v}_1 \mathbf{u}_1^T + \ldots + \mathbf{v}_d \mathbf{u}_d^T$  where  $\mathbf{u}_i = \lambda_i \mathbf{v}_i$ , and we wish to solve:

$$\max_{|\mathcal{X}|=|\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^{T} \left( \mathbf{v}_{1} \mathbf{u}_{1}^{T} + \ldots + \mathbf{v}_{d} \mathbf{u}_{d}^{T} \right) \mathbf{1}_{\mathcal{Y}}.$$
 (1)

Observe that we can rewrite (1) in the following way

$$\max_{\substack{|\mathcal{X}|=|\mathcal{Y}|=k}} \mathbf{1}_{\mathcal{X}}^{T} \left[ \mathbf{v}_{1} \cdot \underbrace{(\mathbf{u}_{1}^{T} \mathbf{1}_{\mathcal{Y}})}_{c_{1}} + \ldots + \mathbf{v}_{d} \cdot \underbrace{(\mathbf{u}_{d}^{T} \mathbf{1}_{\mathcal{Y}})}_{c_{d}} \right]$$
$$= \max_{|\mathcal{Y}|=k} \left( \max_{|\mathcal{X}|=k} \mathbf{1}_{\mathcal{X}}^{T} \mathbf{v}_{\mathcal{Y}} \right), \qquad (2)$$

where  $\mathbf{v}_{\mathcal{Y}} = \mathbf{v}_1 \cdot c_1 + \ldots + \mathbf{v}_d \cdot c_d$  is an *n*-dimensional vector generated by the *d*-dimensional subspace spanned by  $\mathbf{v}_1, \ldots, \mathbf{v}_d$ .

We will now make a key observation: for every fixed vector  $\mathbf{v}_{\mathcal{Y}}$  in (2), the index set  $\mathcal{X}$  that maximizes  $\mathbf{1}_{\mathcal{X}}^T \mathbf{v}_{\mathcal{Y}}$  can be easily computed. It is not hard to see that for any fixed vector  $\mathbf{v}_{\mathcal{Y}}$ , the k-subset  $\mathcal{X}$  that maximizes  $\mathbf{1}_{\mathcal{X}}^T \mathbf{v}_{\mathcal{Y}} = \sum_{i \in \mathcal{X}} [\mathbf{v}_{\mathcal{Y}}]_i$  corresponds the set of k largest signed coordinates of  $\mathbf{v}_{\mathcal{Y}}$ . That is, the locally optimal k-set is  $top_k(\mathbf{v}_{\mathcal{Y}})$ .

We now wish to find all possible locally optimal sets  $\mathcal{X}$ . If we could possibly check all vectors  $\mathbf{v}_{\mathcal{Y}}$ , then we could find all locally optimal index sets top<sub>k</sub>( $\mathbf{v}_{\mathcal{Y}}$ ).

Let us denote as  $S_d$  the set of all k-subsets  $\mathcal{X}$  that are the optimal solutions of the inner maximization of (2) for any vector  $\mathbf{v}$  in the span of  $\mathbf{v}_1, \ldots, \mathbf{v}_d$ 

$$\mathcal{S}_d = \{ \operatorname{top}_k([\mathbf{v}_1 \cdot c_1 + \ldots + \mathbf{v}_d \cdot c_d]) : c_1, \ldots, c_d \in \mathbb{R} \}.$$

Clearly, this set contains all possible locally optimal  $\mathcal{X}$  sets of the form top<sub>k</sub>( $\mathbf{v}_{\mathcal{Y}}$ ). Therefore, we can rewrite DBkS on  $\mathbf{A}_d$  as

$$\max_{|\mathcal{Y}|=k} \max_{\mathcal{X}\in\mathcal{S}_d} \mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}}.$$
 (3)

The above problem can now be solved in the following way: for every set  $\mathcal{X} \in S_d$  find the locally optimal set  $\mathcal{Y}$  that maximizes  $\mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}}$ , that is, this will be  $\operatorname{top}_k(\mathbf{A}_d \mathbf{1}_{\mathcal{X}})$ . Then, we simply need to test all such  $\mathcal{X}, \mathcal{Y}$  pairs on  $\mathbf{A}_d$  and keep the optimizer.

Due to the above, the problem of solving DBkS on  $A_d$  is equivalent to constructing the set of k-supports  $S_d$ , and then finding the optimal solution in that set. How large can  $S_d$  be and can we construct it in polynomial time? Initially one could expect that the set  $S_d$  could have size as big as  $\binom{n}{k}$ . Instead, we show that the set  $S_d$  will be tremendously smaller, as in (Karystinos & Liavas, 2010) and (Asteris et al., 2014).

**Lemma 4.** The set  $S_d$  has size at most  $O(n^d)$  and can be built in time  $O(n^{d+1})$  using Algorithm 2.

#### **3.1.** Constructing the set $S_d$

We build up to the general rank-*d* algorithm by explaining special cases that are easier to understand.

**Rank-1 case**. We start with the d = 1 case, where we have  $S_1 = \{ top_k(c_1 \cdot v_1) : c_1 \in \mathbb{R} \}$ . It is not hard to see that there are only two supports to include in  $S_1$ :  $top_k(v_1)$  and  $top_k(-v_1)$ . These two sets can be constructed in time in time O(n), via a partial sorting and selection algorithm (Cormen et al., 2001). Hence,  $S_1$  has size 2 and can be constructed in time O(n).

**Rank-**2 **case.** This is the first non-trivial *d* which exhibits the details of the Spannogram algorithm.

Let an auxiliary angle  $\phi \in \Phi = [0, \pi)$  and let

$$\mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \sin \phi \\ \cos \phi \end{bmatrix}.^3$$

Then, we re-express  $c_1 \cdot \mathbf{v}_1 + c_2 \cdot \mathbf{v}_2$  in terms of  $\phi$  as

$$\mathbf{v}(\phi) = \sin \phi \cdot \mathbf{v}_1 + \cos \phi \cdot \mathbf{v}_2. \tag{4}$$

This means that we can rewrite the set  $S_2$  as:

$$\mathcal{S}_2 = \{ \operatorname{top}_k(\pm(\mathbf{v}(\phi)), \phi \in [0, \pi) \}$$

Observe that each element of  $\mathbf{v}(\phi)$  is a continuous *spectral curve* in  $\phi$ :  $[\mathbf{v}(\phi)]_i = [\mathbf{v}_1]_i \sin(\phi) + [\mathbf{v}_2]_i \cos(\phi)$ . Consequently, the top/bottom-k supports of  $\mathbf{v}(\phi)$  (*i.e.*,  $top_k(\pm \mathbf{v}(\phi))$ ) are themselves a function of  $\phi$ . How can we find all possible supports?



Figure 1. A rank d = 2 spannogram for n = 5 and two random vectors  $\mathbf{v}_1, \mathbf{v}_2$ . Observe that every two curves intersect in exactly one point. These intersection points define intervals in which a top-k set is invariant.

**The Spannogram.** In Fig. 1, we draw an example plot of five curves  $[\mathbf{v}(\phi)]_i$ ,  $i = 1, \ldots, 5$ , which we call a spannogram. From the spannogram in Fig. 1, we can see that the continuity of these sinusoidal curves implies a "local invariance" property of the top/bottom k supports  $top_k(\pm \mathbf{v}(\phi))$ , in a small neighborhood around a fixed  $\phi$ . So, when does a top/bottom-k support change? The index sets  $top_k(\pm \mathbf{v}(\phi))$  change *if and only if* two curves cross, *i.e.*, when the ordering of two elements  $[\mathbf{v}(\phi)]_i, [\mathbf{v}(\phi)]_j$ changes.

Finding all supports: There are *n* curves and each pair intersects at exactly one point in the  $\Phi$  domain<sup>4</sup>. Therefore, there are exactly  $\binom{n}{2}$  intersection points. These  $\binom{n}{2}$  intersection points define  $\binom{n}{2} + 1$  intervals. Within an interval the top/bottom *k* supports top<sub>k</sub>( $\pm \mathbf{v}(\phi)$ ) remain the same. Hence, it is now clear that  $|S_2| \leq 2\binom{n}{2} = O(n^2)$ .

A way to find all supports in  $S_2$  is to compute the  $\mathbf{v}(\phi_{i,j})$  vectors on the intersection points of two curves i, j, and

<sup>&</sup>lt;sup>3</sup>Observe that when we scan  $\phi$ , the vectors **c**,  $-\mathbf{c}$  express all possible unit norm vectors on the circle.

<sup>&</sup>lt;sup>4</sup>Here we assume that the curves are in *general position*. This can be always accomplished by infinitesimally perturbing the curves as in (Papailiopoulos et al., 2013).

then the supports in the two adjacent intervals of such intersection point. The  $\mathbf{v}(\phi_{i,j})$  vector on an intersection point of two curves *i* and *j* can be easily computed by first solving a set of linear equations  $[\mathbf{v}(\phi_{i,j})]_i = [\mathbf{v}(\phi_{i,j})]_j \Rightarrow (\mathbf{e}_i - \mathbf{e}_j)^T [\mathbf{v}_1 \quad \mathbf{v}_2] \mathbf{c}_{i,j} = \mathbf{0}_{2\times 1}$  for the unknown vector  $\mathbf{c}_{i,j}$ , where  $\mathbf{e}_i$  is the *i*-th column of the  $n \times n$  identity matrix, i.e.,  $\mathbf{c}_{i,j} = \text{nullspace}((\mathbf{e}_i - \mathbf{e}_j)^T [\mathbf{v}_1 \quad \mathbf{v}_2])$ . Then, we compute  $\mathbf{v}(\phi_{i,j}) = [\mathbf{v}_1 \quad \mathbf{v}_2] \mathbf{c}_{i,j}$ . Further details on breaking ties in  $\operatorname{top}_k(\mathbf{v}(\phi_{i,j}))$  can be found in the supplemental material.

*Computational cost:* We have  $\binom{n}{2}$  intersection points, where we calculate the top/bottom k supports for each  $\mathbf{v}(\phi_{i,j})$ . The top/bottom k elements of every  $\mathbf{v}(\phi_{i,j})$  can be computed in time O(n) using a partial sorting and selection algorithm (Cormen et al., 2001). Since we perform this routine a total of  $O(\binom{n}{2})$  times, the total complexity of our rank-2 algorithm is  $O(n^3)$ .

**General Rank-**d **case.** The algorithm generalizes to arbitrary dimension d, as we show in the supplemental material; its pseudo-code is given as Algorithm 3.

**Remark 2.** Observe that the computation of each loop under line 2 of Algorithm 3 can be computed in parallel. This will allow us to parallelize the Spannogram.

```
Algorithm 3 Spannogram(k, \mathbf{V}_d)1: S_d = \emptyset2: for all (i_1, \ldots, i_d) \in \{1, \ldots, n\}^d and s \in \{-1, 1\} do3: \mathbf{c} = s \cdot \text{nullspace} \left( \begin{bmatrix} [(\mathbf{V}_d]_{i_1,:} - [\mathbf{V}_d]_{i_2,:} \\ \vdots \\ [\mathbf{V}_d]_{i_1,:} - [\mathbf{V}_d]_{i_d,:} \end{bmatrix} \right)4: \mathbf{v} = \mathbf{V}_d^T \mathbf{c}5: S = \text{top}_k(\mathbf{v})6: \mathcal{T} = S - \{i_1, \ldots, i_d\}7: for all \binom{d}{k-|\mathcal{T}|} subsets \mathcal{J} of (i_1, \ldots, i_d) do8: S_d = S_d \bigcup (\mathcal{T} \cup \mathcal{J})9: end for10: end for11: Output: S_d.
```

#### **3.2.** An approximate $S_d$ in nearly-linear time

In our exact solver, we solve DBkS on  $A_d$  in time  $O(n^{d+1})$ . Surprisingly, when  $A_d$  has only positive eigenvalues, then we can tightly approximate DBkS on  $A_d$  in nearly linear time.

**Theorem 3.** Let the *d* largest eigenvalues of the graph be positive, and let the *d*-th, (d + 1)-st largest have constant ratio:  $\left|\frac{\lambda_d}{\lambda_{d+1}}\right| \geq C$ . Then, we can output, with probability  $1 - \delta$ , a k-subgraph with density  $(1 - \epsilon)^2 \cdot \operatorname{opt}_d$ , in time  $O\left(m \cdot \log n + \frac{n}{\epsilon^d} \cdot \log\left(\frac{1}{\epsilon \cdot \delta}\right)\right)$ .

The main idea is that instead of checking all  $O(n^d)$  possible k sets in  $S_d$ , we can approximately solve the problem by randomly sampling  $M = O\left(\epsilon^{-d} \cdot \log\left(\frac{1}{\epsilon \cdot \delta}\right)\right)$  vectors in the span of  $\mathbf{v}_1, \ldots, \mathbf{v}_d$ . Our proof is based on the fact that we can "approximate" the surface of the d-dimensional

sphere with M randomly sampled vectors from the span of  $\mathbf{v}_1, \ldots, \mathbf{v}_d$ . This allows us to identify, probability  $1 - \delta$ , near-optimal candidates in  $S_d$ . The modified algorithm is very simple and is given below; its analysis can be found in the supplemental material.

${f Algorithm} \ {f 4}$ Spannogram_approx ( $k, \ {f V}_d, {f \Lambda}_d$ )	
1: for $i = 1 : O\left(\epsilon^{-d} \cdot \log\left(\frac{1}{\epsilon \cdot \delta}\right)\right)$ do	
2: $\mathbf{v} = (\mathbf{\Lambda}_d^{1/2} \cdot \mathbf{V}_d)^T \cdot \operatorname{randn}(d, 1)$	
3: $S_d = S_d \cup \operatorname{top}_k(\mathbf{v}) \cup \operatorname{top}_k(-\mathbf{v})$	
4: end for	
5: <b>Output:</b> $S_d$ .	
	_

# 4. Scaling up

In this section, we present the two key scalability features that allow us to scale up to graphs with billions of edges.

#### 4.1. Vertex Sparsification

We introduce a very simple and efficient pre-processing step for discarding vertices that are unlikely to appear in a top k set in  $S_d$ . This step runs after we compute  $\mathbf{A}_d$  and uses the leverage score,  $\ell_i = \sum_{j=1}^d [\mathbf{V}_d]_{i,j}^2 |\lambda_j|$ , of the *i*-th vertex to decide whether we will discard it or not. We show in the supplemental material, that by appropriately setting a threshold, we can guarantee a provable bound on the error introduced. In our experimental results, the above elimination is able to reduce *n* to approximately  $\hat{n} \approx 10 \cdot k$  for a provably small additive error, even for data sets where  $n = 10^8$ .

### 4.2. MapReduce Implementation

A MapReduce implementation allows scaling out to a large number of compute nodes that can work in parallel. The reader can refer to (Meng & Mahoney, 2013; Bahmani et al., 2012)) for a comprehensive treatment of the MapReduce paradigm. In short, the Hadoop/MapReduce infrastructure stores the input graph as a distributed file spread across multiple machines; it provides a tuple streaming abstraction, where each map and reduce function receives and emits tuples as (*key*, *value*) pairs. The role of the keys is to ensure information aggregation: all the tuples with the same key are processed by the same reducer.

For the spectral decomposition step of our scheme we design a simple implementation of the power method in MapReduce. The details are beyond the scope of this work; high-performance implementations are already available in the literature, e.g. (Lin & Schatz, 2010). We instead focus on the novel implementation of the Spannogram.

Our MapReduce implementation of the rank-2 Spannogram is outlined in Algorithm 4. The Mapper is responsible for the duplication and dissemination of the eigenvectors,  $V_2$ ,  $U_2 = V_2 \Lambda_2$ , to all reducers. Line 3 emits the *j*-th row of  $V_2$  and  $U_2$  once for every node *i*. Since *i* is used as the key, this ensures that every reducer receives  $V_2$ ,  $U_2$  in their entirety.

From the breakdown of the Spannogram in Section 3, it is understood that, for the rank-2 case, it suffices to solve a simple system of equations for every pair of nodes. The Reducer for node *i* receives the full eigenvectors  $V_2$ ,  $U_2$ and is responsible for solving the problem for every pair (i, j), where j > i. Then, Line 6 emits the best candidate computed at Reducer *i*. A trivial final step, not outlined here, collects all  $n^2$  candidate sets and keeps the best one as the final solution.

The basic outline in Algorithm 4 comes with heavy communication needs and was chosen here for ease of exposition. The more efficient version that we implement, does not replicate  $V_2, U_2$  *n* times. Instead, the number of reducers – say  $R = n^{\alpha}$  – is fine-tuned to the capabilities of the cluster. The mappers emit  $V_2, U_2 R$  times, once for every reducer. Then, reducer r is responsible for solving for node pairs (i, j), where  $i \equiv r \pmod{R}$  and j > i. Depending on the performance bottleneck, different choices for  $\alpha$  are more appropriate. We divide the construction of the  $O(n^2)$  candidate sets in  $S_2$  to  $O(n^{\alpha})$  reducers and each of them computes  $O(n^{2-\alpha})$  candidate subgraphs. The total communication cost for this parallelization scheme is  $O(n^{1+\alpha})$ :  $n^{\alpha}$  reducers need to have access to the entire  $\mathbf{V}_2, \mathbf{U}_2$  that has  $2 \cdot 2 \cdot n$  entries. Moreover, the total computation cost for each reducer is  $O(n^{3-\alpha})$ .

Algorithm 5 SpannogramMR( $\mathbf{V}_2, \mathbf{U}_2$ ) 1: Map({[ $\mathbf{V}_2$ ]<sub>j:</sub>, [ $\mathbf{U}_2$ ]<sub>j:</sub>, j}): 2: for i = 1 : n do 3: emit:  $\langle i, \{[\mathbf{V}_2]_{j,1}, [\mathbf{V}_2]_{j,2}, [\mathbf{U}_2]_{j,1}, [\mathbf{U}_2]_{j,2}, j\}\rangle$ 4: end for 1: Reduce<sub>i</sub>( $\langle i, \{[\mathbf{V}_2]_{j,1}, [\mathbf{V}_2]_{j,2}, [\mathbf{U}_2]_{j,1}, [\mathbf{U}_2]_{j,2}, j\}\rangle, \forall j$ ): 2: for each  $j \ge i + 1$  do 3:  $\mathbf{c} = \text{nullspace}([\mathbf{V}]_{i,:} - [\mathbf{V}]_{j,:})$ 4:  $[den_j, \{\mathcal{X}_j, \mathcal{Y}_j\}] = \max_{|\mathcal{Y}|=k, \mathcal{X} \in top_k(\pm \mathbf{V}_2 \mathbf{c})} \mathbf{1}_{\mathcal{X}} \mathbf{V}_2 \mathbf{U}_2^T \mathbf{1}_{\mathcal{Y}}$ 5: end for 6: emit:  $\langle i, \{\mathcal{X}_i, \mathcal{Y}_i\} = \max_j \mathbf{1}_{\mathcal{X}_j} \mathbf{V}_2 \mathbf{U}_2^T \mathbf{1}_{\mathcal{Y}_j} \rangle$ 

### 5. Experimental Evaluation

We experimentally evaluate the performance of our algorithm and compare it to the truncated power method (TPower) of (Yuan & Zhang, 2011), a greedy algorithm by (Feige et al., 2001) (GFeige) and another greedy algorithm by (Ravi et al., 1994) (GRavi). We performed experiments on synthetic dense subgraphs and also massive real graphs from multiple sources. In all experiments we compare the density of the subgraph obtained by the Spannogram to the density of the output subgraphs given by the other algorithms.

Our experiments illustrate three key points: (1) for all tested graphs, our method outperforms – some times significantly – all other algorithms compared; (2) our method is



subgraph v.s. the expected number of edges.

(b) Running times of the Spannogram and power iteration for two top eigenvectors.

Figure 2. Planted clique experiments for random graphs.

highly scalable, allowing us to solve far larger problem instances; (3) our data-dependent upper bound in many cases provide a certificate of near-optimality, far more accurate and useful, than what *a priori* bounds are able to do.

**Planted clique**. We first consider the so-called (and now much studied) Planted Clique problem: we seek to find a clique of size k that has been planted in a graph where all other edges are drawn independently with probability 1/2. We scale our randomized experiments from n = 100 up to 10<sup>5</sup>. In all cases we set the size of the clique to  $k = 3 \cdot \sqrt{n}$ - close to what is believed to be the critical computability threshold. In all our experiments, GRavi, TPower, and the Spannogram successfully recovered the hidden clique. However, as can be seen in Fig. 2, the Spannogram algorithm is the only one able to scale up to  $n = 10^5$  – a massive dense graph with about 2.5 billion edges. The reason is that this graph does not fit in the main memory of one machine and caused all centralized algorithms to crash after several hours. Our MapReduce implementation scales out smoothly, since it splits the problem over multiple smaller problems solved in parallel.

Specifically, we used Amazon Wireless Services' Elastic MapReduce framework (aws). We implemented our map and reduce functions in Python and used the MRJob class (mrj). For our biggest experiments we used a 100-machine strong cluster, consisting of m1.xlarge AWS instances (a total of 800 cores).

The running times of our experiments over MapReduce are shown in Fig. 2(b). The main bottleneck is the computation of the first two eigenvectors which is performed by repeating the power iteration for few (typically 4) iterations. This step is not the emphasis of this work and has not been optimized. The Spannogram algorithm is significantly faster and the benefits of parallelization are clear since it is CPU intensive.

In principle, the other algorithms could be also implemented over MapReduce, but that requires non-trivial dis-



Figure 3. Subgraph density vs. subgraph size (k). We compare our DkS Spannogram algorithm with the algorithms from (Feige et al., 2001) (GFeige), (Ravi et al., 1994) (GRavi), and (Yuan & Zhang, 2011) (tPM). Across all subgraph sizes k, we obtain higher subgraph densities using Spannograms of rank d = 2 or 5. We also obtain a *provable data-dependent upper bound* (solid black line) on the objective. This proves that for these data sets, our algorithm is typically within 80% from optimality, for all sizes up to k = 250, and indeed for small subgraph sizes we find a clique which is clearly optimal. Further experiments on multiple other data sets are shown in the supplemental material.

tributed algorithm design. As is well-known, *e.g.*, (Meng & Mahoney, 2013), implementing iterative machine learning algorithms over MapReduce can be a significant task and schemes which perform worse in standard metrics can be highly preferable for this parallel framework. Careful MapReduce algorithmic design is needed especially for dense graphs like the one in the hidden clique problem.

**Real Datasets.** Next, we demonstrate our method's performance in real datasets and also illustrate the power of our data-dependent bounds. We run experiments on large graphs from different applications and our findings are presented in Fig. 3. The figure compares the density achieved by the Spannogram algorithm for rank 1, 2 and 5 to the performance of GFeige, GRavi and TPower. The figure shows that the rank-2 and rank-5 versions of our algorithm, improve – sometimes significantly – over the other techniques. Our novel data-dependent upper-bound shows that our results on these data sets are provably near-optimal.

The experiments are performed for two community graphs (com-LiveJournal and com-DBLP), a web graph (web-NotreDame), and a subset of the Facebook graph. A larger set of experiments is included in the supplemental material. Note that the largest graph in Figure 3 contains no more than 35 million edges; these cases fit in the main memory of a single machine and the running times are presented in the supplemental material, all performed on a standard Macbook Pro laptop using Matlab. In summary, rank-2 took less than one second for all these graphs while prior work methods took approximately the same time, up to a few seconds. Rank-1 was significantly faster than all other methods in all tested graphs and took fractions of a second. Rank-5 took up to 1000 seconds for the largest graph (LiveJournal).

We conclude that our algorithm is an efficient option for finding dense subgraphs. Different rank choices give a tradeoff between accuracy and performance while the parallel nature allows scalability when needed. Further, our theoretical upper-bound can be useful for practitioners investigating dense structures in large graphs.

### 6. Acknowledgments

The authors would like to acknowledge support from NSF grants CCF 1344364, CCF 1344179, DARPA XDATA, and research gifts by Google, Docomo and Microsoft.

### References

Amazon Web Services, Elastic Map Reduce. URL http:// aws.amazon.com/elasticmapreduce/.

MRJob. URL http://pythonhosted.org/mrjob/.

- Alon, Noga, Lee, Troy, Shraibman, Adi, and Vempala, Santosh. The approximate rank of a matrix and its algorithmic applications: approximate rank. In *Proceedings of the 45th annual* ACM symposium on Symposium on theory of computing, pp. 675–684. ACM, 2013.
- Ames, Brendan PW. Convex relaxation for the planted clique, biclique, and clustering problems. PhD thesis, University of Waterloo, 2011.
- Arora, Sanjeev, Karger, David, and Karpinski, Marek. Polynomial time approximation schemes for dense instances of nphard problems. In STOC, 1995.
- Asahiro, Yuichi, Iwama, Kazuo, Tamaki, Hisao, and Tokuyama, Takeshi. Greedily finding a dense subgraph. *Journal of Algorithms*, 34(2):203–221, 2000.
- Asteris, Megasthenis, Papailiopoulos, Dimitris S, and Karystinos, George N. The sparse principal component of a constant-rank matrix. *IEEE Trans. IT*, 60(4):228–2290, 2014.
- Bahmani, Bahman, Kumar, Ravi, and Vassilvitskii, Sergei. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5):454–465, 2012.
- Bhaskara, Aditya, Charikar, Moses, Chlamtac, Eden, Feige, Uriel, and Vijayaraghavan, Aravindan. Detecting high log-densities: an  $O(n^{1/4})$  approximation for densest k-subgraph. In *STOC*, 2010.
- Boutsidis, Christos, Mahoney, Michael W, and Drineas, Petros. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the twentieth Annual* ACM-SIAM Symposium on Discrete Algorithms, pp. 968–977. Society for Industrial and Applied Mathematics, 2009.
- Cormen, Thomas H, Leiserson, Charles E, Rivest, Ronald L, and Stein, Clifford. *Introduction to algorithms*. MIT press, 2001.
- d'Aspremont, Alexandre et al. Weak recovery conditions using graph partitioning bounds. 2010.
- Dourisboure, Yon, Geraci, Filippo, and Pellegrini, Marco. Extraction and classification of dense communities in the web. In *WWW*, 2007.
- Feige, Uriel and Langberg, Michael. Approximation algorithms for maximization problems arising in graph partitioning. *Journal of Algorithms*, 41(2):174–211, 2001.
- Feige, Uriel, Peleg, David, and Kortsarz, Guy. The dense ksubgraph problem. *Algorithmica*, 29(3):410–421, 2001.

- Gibson, David, Kumar, Ravi, and Tomkins, Andrew. Discovering large dense subgraphs in massive graphs. In *PVLDB*, 2005.
- Hu, Haiyan, Yan, Xifeng, Huang, Yu, Han, Jiawei, and Zhou, Xianghong Jasmine. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 21(suppl 1):i213–i221, 2005.
- Jethava, Vinay, Martinsson, Anders, Bhattacharyya, Chiranjib, and Dubhashi, Devdatt. The lovasz theta function, svms and finding large dense subgraphs. In *NIPS*, 2012.
- Karystinos, George N and Liavas, Athanasios P. Efficient computation of the binary vector that maximizes a rank-deficient quadratic form. *IEEE Trans. IT*, 56(7):3581–3593, 2010.
- Khot, Subhash. Ruling out ptas for graph min-bisection, densest subgraph and bipartite clique. In *FOCS*, 2004.
- Lin, Jimmy and Schatz, Michael. Design patterns for efficient graph algorithms in mapreduce. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, pp. 78–85. ACM, 2010.
- Mahoney, Michael W and Drineas, Petros. Cur matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009.
- Meng, Xiangrui and Mahoney, Michael. Robust regression on mapreduce. In Proceedings of The 30th International Conference on Machine Learning, pp. 888–896, 2013.
- Miller, B, Bliss, N, and Wolfe, P. Subgraph detection using eigenvector 11 norms. In *NIPS*, 2010.
- Papailiopoulos, Dimitris, Dimakis, Alexandros, and Korokythakis, Stavros. Sparse pca through low-rank approximations. In Proceedings of the 30th International Conference on Machine Learning (ICML-13), pp. 747–755, 2013.
- Ravi, Sekharipuram S, Rosenkrantz, Daniel J, and Tayi, Giri K. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.
- Saha, Barna, Hoch, Allison, Khuller, Samir, Raschid, Louiqa, and Zhang, Xiao-Ning. Dense subgraphs with restrictions and applications to gene annotation graphs. In *Research in Computational Molecular Biology*, pp. 456–472. Springer, 2010.
- Srivastav, Anand and Wolf, Katja. *Finding dense subgraphs with semidefinite programming*. Springer, 1998.
- Suzuki, Akiko and Tokuyama, Takeshi. Dense subgraph problems with output-density conditions. In *Algorithms and Computation*, pp. 266–276. Springer, 2005.
- Yuan, Xiao-Tong and Zhang, Tong. Truncated power method for sparse eigenvalue problems. arXiv preprint arXiv:1112.2679, 2011.