Finding Dense Subgraphs via Low-Rank Bilinear Optimization

Dimitris S. Papailiopoulos DIMITRIS@UTEXAS.EDU
Ioannis Mitliagkas IOANNIS@UTEXAS.EDU
Alexandros G. Dimakis DIMAKIS@AUSTIN.UTEXAS.EDU
Constantine Caramanis CONSTANTINE@UTEXAS.EDU
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 reduction to MAXCLIQUE. Moreover, Khot showed in (Khot, 2004) that, under widely believed complexity-theoretic assumptions, DkS cannot be approximated within an arbitrary constant factor.[1] The best known approximation ratio was $n^{1/3+\epsilon}$ (for some small ϵ) 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 $0 \leq \mathsf{opt} \leq k-1$. 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 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* λ_i *is the* i*th largest, in mag-*

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

[1] approximation ratio ρ means that there exists an algorithm that produces in polynomial time a number A, such that $1 \leq \frac{\mathsf{opt}}{\mathsf{A}} \leq \rho$, 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 $O\left(n^{d+1} + \mathsf{T}_d\right)$, and outputs a k-subgraph with density

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

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 λ_{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 graph-dependent 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 $(1-\epsilon)^2 \cdot \mathsf{opt}_d$, 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 $d \leq 5$, 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 approximation results for graphs with small "ϵ-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 $\mathcal{S} \subseteq \{1, \ldots, n\}$ is equal to the average degree of the vertices within $\mathcal{S}$:

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

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

For a fixed subgraph size $|\mathcal{S}| = k$, we can express DkS as a quadratic optimization:

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

where $|\mathcal{S}| = k$ 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 ρ-approximation algorithm for DBkS implies a 2ρ-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.[2] The output of our algorithm is a k-subgraph $\mathcal{Z}_d$ 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 approximations for DkS

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

Constant rank solver for DBkS. In the following we present an exact solver for DBkS on constant rank approximations of $\mathbf{A}$. 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, λ_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

[2] In the pseudocode of Algorithm 2, $\mathrm{top}_k(\mathbf{v})$, 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 $\mathcal{S}_d$ 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 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 δ and is ϵ far from the optimal rank-d solver, while its complexity reduces to *nearly linear* in the number of edges m of the graph $\mathcal{G}$: $O\left(m \cdot \log n + \frac{n}{\epsilon^d} \cdot \log\left(\frac{1}{\epsilon \cdot \delta}\right)\right)$.

Algorithm 2 `lowrankDBkS(k, d, A)`

1: $[\mathbf{V}_d, \mathbf{\Lambda}_d] = \texttt{EVD}(\mathbf{A}, d)$
2: $\mathcal{S}_d = \texttt{Spannogram}(k, \mathbf{V}_d)$
3: $\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg\max_{|\mathcal{X}|=k} \max_{\mathcal{Y} \in \mathcal{S}_d} \mathbf{1}_{\mathcal{X}}^T \mathbf{V}_d \mathbf{\Lambda}_d \mathbf{V}_d^T \mathbf{1}_{\mathcal{Y}}$
4: **Output:** $\{\mathcal{X}_d, \mathcal{Y}_d\}$

`Spannogram(k, V_d)`
$\mathcal{S}_d = \{\text{top}_k(\mathbf{v}) : \mathbf{v} \in \text{span}(\mathbf{v}_1, \ldots, \mathbf{v}_d)\}$
Output: $\mathcal{S}_d$.

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 $\mathsf{opt}_d^{\mathsf{B}}$ is from $\mathsf{opt}^{\mathsf{B}}$, the optimal density for DBkS. Our bounds capture a simple core idea: the loss in our approximation comes due to solving the problem on $\mathbf{A}_d$ instead of solving it on the full rank matrix $\mathbf{A}$. 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 $\mathbf{A}$: $\mathsf{opt}_d^{\mathsf{B}} \geq \mathsf{opt}^{\mathsf{B}} - 2 \cdot |\lambda_{d+1}|$, where λ_i is the ith 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 $\mathbf{A}$, we output a k-subgraph $\mathcal{Z}_d$ that has density opt_d. 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*

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

with probability $1 - \frac{1}{n}$, where λ_i is the ith 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 $O\left(n^{d+1} + \mathsf{T}_d\right)$, and outputs a k-subgraph with density $\mathsf{opt}_d \geq \mathsf{opt} - 2 \cdot |\lambda_{d+1}|$, 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 $\mathsf{opt}^{\mathsf{B}}$, since it is easy to see that $\mathsf{opt}^{\mathsf{B}} \geq \mathsf{opt}$. 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*

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

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 $\mathbf{A}$.

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:

$$\max_{|\mathcal{X}|=|\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \left(\mathbf{v}_1 \mathbf{u}_1^T + \ldots + \mathbf{v}_d \mathbf{u}_d^T\right) \mathbf{1}_{\mathcal{Y}}. \tag{1}$$

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

$$\max_{|\mathcal{X}|=|\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \left[\mathbf{v}_1 \cdot \underbrace{(\mathbf{u}_1^T \mathbf{1}_{\mathcal{Y}})}_{c_1} + \ldots + \mathbf{v}_d \cdot \underbrace{(\mathbf{u}_d^T \mathbf{1}_{\mathcal{Y}})}_{c_d}\right]$$

$$= \max_{|\mathcal{Y}|=k} \left(\max_{|\mathcal{X}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{v}_{\mathcal{Y}}\right), \tag{2}$$

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 $\text{top}_k(\mathbf{v}_\mathcal{Y})$.

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 $\text{top}_k(\mathbf{v}_\mathcal{Y})$.

Let us denote as $\mathcal{S}_d$ 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 $\mathbf{v}_1, \ldots, \mathbf{v}_d$

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

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

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

The above problem can now be solved in the following way: for every set $\mathcal{X} \in \mathcal{S}_d$ 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 $\text{top}_k(\mathbf{A}_d \mathbf{1}_\mathcal{X})$. 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 $\mathbf{A}_d$ is equivalent to constructing the set of k-supports $\mathcal{S}_d$, and then finding the optimal solution in that set. How large can $\mathcal{S}_d$ be and can we construct it in polynomial time? Initially one could expect that the set $\mathcal{S}_d$ could have size as big as $\binom{n}{k}$. Instead, we show that the set $\mathcal{S}_d$ will be tremendously smaller, as in (Karystinos & Liavas, 2010) and (Asteris et al., 2014).

Lemma 4. *The set $\mathcal{S}_d$ 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 $\mathcal{S}_d$

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 $\mathcal{S}_1 = \{\text{top}_k(c_1 \cdot \mathbf{v}_1) : c_1 \in \mathbb{R}\}$. It is not hard to see that there are only two supports to include in $\mathcal{S}_1$: $\text{top}_k(\mathbf{v}_1)$ and $\text{top}_k(-\mathbf{v}_1)$. These two sets can be constructed in time in time $O(n)$, via a partial sorting and selection algorithm (Cormen et al., 2001). Hence, $\mathcal{S}_1$ 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 ϕ 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 $\mathcal{S}_2$ as:

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

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

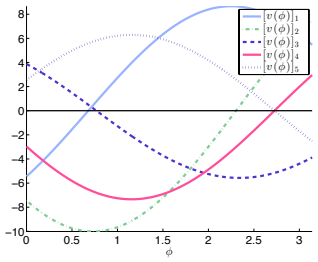


Figure 1. A rank $d = 2$ spannogram for $n = 5$ and two random vectors $\mathbf{v}_1, \mathbf{v}_2$. Observe that every two curves intersect in exactly one point. These intersection points define intervals in which a top-k set is invariant.

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 $\text{top}_k(\pm\mathbf{v}(\phi))$, in a small neighborhood around a fixed ϕ. So, when does a top/bottom-k support change? The index sets $\text{top}_k(\pm\mathbf{v}(\phi))$ 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 Φ domain[4]. 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 $\text{top}_k(\pm\mathbf{v}(\phi))$ *remain the same*. Hence, it is now clear that $|\mathcal{S}_2| \leq 2\binom{n}{2} = O(n^2)$.

A way to find all supports in $\mathcal{S}_2$ is to compute the $\mathbf{v}(\phi_{i,j})$ vectors on the intersection points of two curves i, j, and

[3] Observe that when we scan ϕ, the vectors $\mathbf{c}, -\mathbf{c}$ express all possible unit norm vectors on the circle.

[4] 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 $\mathbf{v}(\phi_{i,j}) = [\mathbf{v}_1 \ \mathbf{v}_2]\mathbf{c}_{i,j}$. Further details on breaking ties in $\text{top}_k(\mathbf{v}(\phi_{i,j}))$ 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: $\mathcal{S}_d = \emptyset$
2: **for** all $(i_1, \ldots, i_d) \in \{1, \ldots, n\}^d$ and $s \in \{-1, 1\}$ **do**
3: $\quad \mathbf{c} = s \cdot \text{nullspace}\left(\begin{bmatrix} [\mathbf{V}_d]_{i_1,:} - [\mathbf{V}_d]_{i_2,:} \\ \vdots \\ [\mathbf{V}_d]_{i_1,:} - [\mathbf{V}_d]_{i_d,:} \end{bmatrix}\right)$
4: $\quad \mathbf{v} = \mathbf{V}_d^T\mathbf{c}$
5: $\quad \mathcal{S} = \text{top}_k(\mathbf{v})$
6: $\quad \mathcal{T} = \mathcal{S} - \{i_1, \ldots, i_d\}$
7: $\quad$ **for** all $\binom{d}{k-|\mathcal{T}|}$ subsets $\mathcal{J}$ of $(i_1, \ldots, i_d)$ **do**
8: $\quad\quad \mathcal{S}_d = \mathcal{S}_d \bigcup (\mathcal{T} \cup \mathcal{J})$
9: $\quad$ **end for**
10: **end for**
11: **Output:** $\mathcal{S}_d$.

3.2. An approximate $\mathcal{S}_d$ in nearly-linear time

In our exact solver, we solve DBkS on $\mathbf{A}_d$ in time $O(n^{d+1})$. Surprisingly, when $\mathbf{A}_d$ has only positive eigenvalues, then we can tightly approximate DBkS on $\mathbf{A}_d$ 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 $(1-\epsilon)^2 \cdot \mathsf{opt}_d$, 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 $\mathcal{S}_d$, 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 $\mathcal{S}_d$. The modified algorithm is very simple and is given below; its analysis can be found in the supplemental material.

Algorithm 4 `Spannogram_approx`$(k,\ \mathbf{V}_d, \mathbf{\Lambda}_d)$

1: **for** $i = 1 : O\left(\epsilon^{-d} \cdot \log\left(\frac{1}{\epsilon\cdot\delta}\right)\right)$ **do**
2: $\quad \mathbf{v} = \left(\mathbf{\Lambda}_d^{1/2} \cdot \mathbf{V}_d\right)^T \cdot \texttt{randn}(d, 1)$
3: $\quad \mathcal{S}_d = \mathcal{S}_d \cup \text{top}_k(\mathbf{v}) \cup \text{top}_k(-\mathbf{v})$
4: **end for**
5: **Output:** $\mathcal{S}_d$.

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 $\mathcal{S}_d$. 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, $\mathbf{V}_2, \mathbf{U}_2 = \mathbf{V}_2\mathbf{\Lambda}_2$, to all reducers. Line 3 emits the j-th row of $\mathbf{V}_2$ and $\mathbf{U}_2$ once for every node i. Since i is used as the key, this ensures that every reducer receives $\mathbf{V}_2, \mathbf{U}_2$ 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 $\mathbf{V}_2, \mathbf{U}_2$ 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 $\mathbf{V}_2, \mathbf{U}_2$ n times. Instead, the number of reducers – say $R = n^\alpha$ – is fine-tuned to the capabilities of the cluster. The mappers emit $\mathbf{V}_2, \mathbf{U}_2$ 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 α are more appropriate. We divide the construction of the $O(n^2)$ candidate sets in $\mathcal{S}_2$ 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^α 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`$(\mathbf{V}_2, \mathbf{U}_2)$

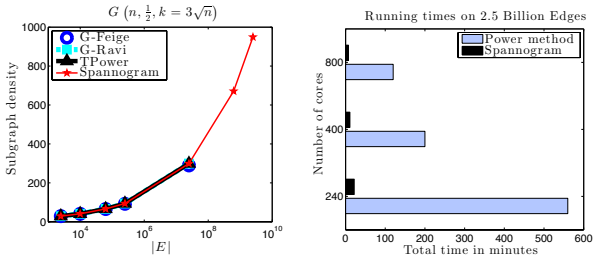
1: **Map**$(\{[\mathbf{V}_2]_{j,:}, [\mathbf{U}_2]_{j,:}, j\})$:
2: **for** $i = 1 : n$ **do**
3: **emit**: $\langle i, \{[\mathbf{V}_2]_{j,1}, [\mathbf{V}_2]_{j,2}, [\mathbf{U}_2]_{j,1}, [\mathbf{U}_2]_{j,2}, j\}\rangle$
4: **end for**

1: **Reduce**$_i(\langle i, \{[\mathbf{V}_2]_{j,1}, [\mathbf{V}_2]_{j,2}, [\mathbf{U}_2]_{j,1}, [\mathbf{U}_2]_{j,2}, j\}\rangle, \forall j)$:
2: **for** each $j \geq i + 1$ **do**
3: $\mathbf{c} = \text{nullspace}([\mathbf{V}]_{i,:} - [\mathbf{V}]_{j,:})$
4: $[\text{den}_j,\ \{\mathcal{X}_j, \mathcal{Y}_j\}] = \max_{|\mathcal{Y}|=k, \mathcal{X} \in \text{top}_k(\pm \mathbf{V}_2 \mathbf{c})} \mathbf{1}_\mathcal{X} \mathbf{V}_2 \mathbf{U}_2^T \mathbf{1}_\mathcal{Y}$
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 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.



(a) Densities of the recovered 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.

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 10^5. 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-

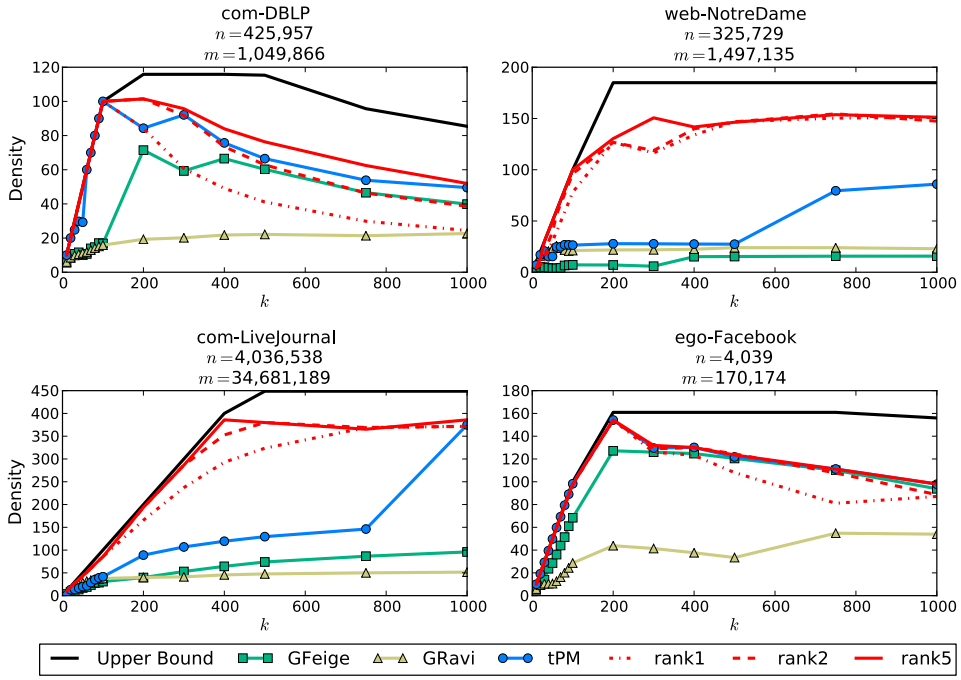


Figure 3. Subgraph density vs. subgraph size (k). We compare our DkS Spannogram algorithm with the algorithms from (Feige et al., 2001) (GFeige), (Ravi et al., 1994) (GRavi), and (Yuan & Zhang, 2011) (tPM). Across all subgraph sizes k, we obtain higher subgraph densities using Spannograms of rank $d = 2$ 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 np-hard 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 $\mathsf{O}(n^{1/4})$ 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 k-subgraph 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 l1 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.

Supplemental Material for: Finding Dense Subgraphs via Low-Rank Bilinear Optimization

1. Proof of Lemma 4: Building the set $\mathcal{S}_d$ for arbitrary d-dimensional subspaces

In our general case, we solve DBkS on

$$\mathbf{A}_d = \mathbf{V}_d \mathbf{U}_d^T = \sum_{i=1}^{d} \mathbf{v}_i \mathbf{u}_i^T$$

where

$$\mathbf{V}_d = [v_1 \ \ \ldots \ \ v_d] \text{ and } \mathbf{U}_d = [\lambda_1 \cdot v_1 \ \ \ldots \ \ \lambda_d \cdot v_d] .$$

Solving the problem on $\mathbf{A}_d$ is equivalent to answering the following combinatorial question:

"how many different top-k supports are there in a d-dimensional subspace: top$_k(c_1 \cdot \mathbf{v}_1 + \ldots + c_d \cdot \mathbf{v}_d)$?"

Here we define $d-1$ auxiliary angles $\phi_1, \ldots, \phi_{d-1} \in \Phi = [0,\ \pi)$ and we rewrite the coefficients $c_1, \ldots, c_d$ as

$$\mathbf{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_d \end{bmatrix} = \begin{bmatrix} \sin\phi_1 \\ \cos\phi_1 \sin\phi_2 \\ \vdots \\ \cos\phi_1 \cos\phi_2 \ldots \sin\phi_{d-1} \\ \cos\phi_1 \cos\phi_2 \ldots \cos\phi_{d-1} \end{bmatrix}.$$

Clearly we can express every vector in the span of $\mathbf{V}_d$ as a linear combination $c_1 \cdot \mathbf{v}_1 + \ldots + c_d \cdot \mathbf{v}_d$ in terms of ϕ:

$$\mathbf{v}(\phi_1, \ldots, \phi_{d-1}) = (\sin\phi_1) \cdot \mathbf{v}_1 + (\cos\phi_1 \sin\phi_2) \cdot \mathbf{v}_2 + \ldots + (\cos\phi_1 \cos\phi_2 \ldots \cos\phi_{d-1}) \cdot \mathbf{v}_d. \tag{1}$$

For notation simplicity let us define a vector that contains all $d-1$ auxiliary phase variables

$$\varphi = [\phi_1, \ldots, \phi_{d-1}].$$

We can use the above derivations to rewrite the set $\mathcal{S}_d$ that contains all top k coordinates in the span of $\mathbf{V}_d$ as:

$$\begin{aligned}
\mathcal{S}_d &= \{\text{top}_k(c_1 \cdot \mathbf{v}_1 + \ldots + c_d \cdot \mathbf{v}_d) : c_1, \ldots, c_d \in \mathbb{R}\} \\
&= \{\text{top}_k \pm (\mathbf{v}(\varphi)) : \varphi \in \Phi^{d-1}\} \\
&= \{\text{top}_k \pm ((\sin\phi_1) \cdot \mathbf{v}_1 + (\cos\phi_1 \sin\phi_2) \cdot \mathbf{v}_2 + \ldots + (\cos\phi_1 \cos\phi_2 \ldots \cos\phi_{d-1}) \cdot \mathbf{v}_d), \varphi \in \Phi^{d-1}\}
\end{aligned}$$

Observe again that each element of $\mathbf{v}(\varphi)$ is a continuous *spectral curve* in the $d-1$ auxiliary variables:

$$[\mathbf{v}(\varphi)]_i = (\sin\phi_1) \cdot [\mathbf{v}_1]_i + (\cos\phi_1 \sin\phi_2) \cdot [\mathbf{v}_2]_i + \ldots + (\cos\phi_1 \cos\phi_2 \ldots \cos\phi_{d-1}) \cdot [\mathbf{v}_d]_i.$$

Consequently, the top/bottom-k supports of $\mathbf{v}(\varphi)$ (*i.e.*, top$_k(\pm\mathbf{v}(\varphi))$) are themselves a function of the $d-1$ variables in φ. How can we find all possible supports?

Remark 1. *In our general problem we wish to find all top and bottom k coordinates that appear in a d-dimensional subspace. In the following discussion, for simplicity we handle the top k coordinates problem. Finding the bottom k trivially follows, by just checking the smallest k coordinates of each vector $c_1 \cdot \mathbf{v}_1 + \ldots + c_d \cdot \mathbf{v}_d$ that we construct using our algorithm.*

1.1. Ranking regions for a single coordinate $[\mathbf{v}(\varphi)]_i$

We now show that for each single coordinate $[\mathbf{v}(\varphi)]_i$, we can partition Φ^{d-1} in regions, wherein the ith coordinate $[\mathbf{v}(\varphi)]_i$ retains the same ranking relative to the other $n-1$ coordinates in the vector $\mathbf{v}(\varphi)$.

Let us first consider for simplicity $[\mathbf{v}(\varphi)]_1$. We aim to find all values of φ where $[\mathbf{v}(\varphi)]_1$ is in one of the the k largest coordinates of $\mathbf{v}(\varphi)$. We observe that this region can be characterized by using n boundary tests:

$$\begin{gathered}[\mathbf{v}(\varphi)]_1 \gtrless [\mathbf{v}(\varphi)]_2 \\ [\mathbf{v}(\varphi)]_1 \gtrless [\mathbf{v}(\varphi)]_3 \\ \vdots \\ [\mathbf{v}(\varphi)]_1 \gtrless [\mathbf{v}(\varphi)]_n\end{gathered}$$

Each of the above boundary tests defines a bounding curve that partitions the Φ^{d-1} domain. We refer to this bounding curve as $\mathcal{B}_{1,j}(\varphi) : \Phi^{d-1} \mapsto \Phi^{d-2}$. A $\mathcal{B}_{1,j}(\varphi)$ curve partitions Φ and defines two regions of φ angles:

$$\mathcal{R}_{1>j} = \{\varphi \in \Phi^{d-1} : [\mathbf{v}(\varphi)]_1 > [\mathbf{v}(\varphi)]_j\} \text{ and } \mathcal{R}_{1\leq j} = \{\varphi \in \Phi^{d-1} : [\mathbf{v}(\varphi)]_1 \leq [\mathbf{v}(\varphi)]_j\} \tag{2}$$

such that $\mathcal{R}_{1>j} \cup \mathcal{R}_{1\leq j} = \Phi^{d-1}$.

Observe that these $n-1$ curves $\mathcal{B}_{1,1}(\varphi), \ldots, \mathcal{B}_{1,n}(\varphi)$ partition Φ in disjoint *cells*, $\mathcal{C}_1^1, \ldots, \mathcal{C}_T^1$, such that

$$\bigcup_{i=1}^{T} \mathcal{C}_i^1 = \Phi^{d-1}.$$

Within each cell $\mathcal{C}_i^1$, the first coordinate $[\mathbf{v}(\varphi)]_1$ retains a fixed ranking relative to the rest of the elements in $\mathbf{v}(\varphi)$, e.g., for a specific cell it might be the largest element, and in another cell it might be the 10th smallest, etc. This happens because for all values of φ in a single cell, the respective ordering $[\mathbf{v}(\varphi)]_1 \gtrless [\mathbf{v}(\varphi)]_2,\ \ldots,\ [\mathbf{v}(\varphi)]_1 \gtrless [\mathbf{v}(\varphi)]_n$ remains the same.

If we have access to a single point, say φ_0, that belongs to a specific cell, say $\mathcal{C}_j^1$, then we can calculate $[\mathbf{v}(\varphi_0)]$ and find the ranking of the first coordinate $[\mathbf{v}(\varphi)]_1$, that remains invariant for all $\varphi \in \mathcal{C}_j^1$. Hence, if we visit all these cells, then we can find all possible rankings that the first coordinate $[\mathbf{v}(\varphi)]_1$ takes in the d-dimensional span of $\mathbf{v}_1, \ldots, \mathbf{v}_d$. In the following subsections, we show that the number of these cells is bounded by $T \leq 2^d \binom{n-1}{d-1}$.

Observe that each bounding curve $\mathcal{B}_{1,i}(\varphi)$ has a one-to-one correspondence to an equation $[\mathbf{v}(\varphi)]_1 = [\mathbf{v}(\varphi)]_j$, which is linear in $\mathbf{c}$:

$$[\mathbf{v}(\varphi)]_1 = [\mathbf{v}(\varphi)]_j \Rightarrow \mathbf{e}_1^T \mathbf{V}_d \mathbf{c} - \mathbf{e}_i^T \mathbf{V}_d \mathbf{c} = 0 \Rightarrow (\mathbf{e}_1 - \mathbf{e}_j)^T \mathbf{V}_d \mathbf{c} = 0. \tag{3}$$

Due to their linear characterization with respect to $\mathbf{c}$, it is easy to see that each $(d-1)$-tuple of bounding curves intersects on a single point in Φ^{d-1}:[1]

$$\begin{matrix}[\mathbf{v}(\varphi)]_1 = [\mathbf{v}(\varphi)]_{i_1} \\ [\mathbf{v}(\varphi)]_1 = [\mathbf{v}(\varphi)]_{i_2} \\ \vdots \\ [\mathbf{v}(\varphi)]_1 = [\mathbf{v}(\varphi)]_{i_{d-1}}\end{matrix} \quad \begin{matrix}(\mathbf{e}_1 - \mathbf{e}_{i_1})^T \mathbf{V}_d \mathbf{c} = 0 \\ (\mathbf{e}_1 - \mathbf{e}_{i_2})^T \mathbf{V}_d \mathbf{c} = 0 \\ \vdots \\ (\mathbf{e}_1 - \mathbf{e}_{i_{d-1}})^T \mathbf{V}_d \mathbf{c} = 0\end{matrix} \quad \Rightarrow \quad \begin{bmatrix}(\mathbf{e}_1 - \mathbf{e}_{i_1})^T \\ (\mathbf{e}_1 - \mathbf{e}_{i_2})^T \\ \vdots \\ (\mathbf{e}_1 - \mathbf{e}_{i_{d-1}})^T\end{bmatrix} \mathbf{V}_d \mathbf{c} = \mathbf{0}_{(d-1)\times 1}.$$

Let us denote the solution of the above linear inverse problem as $\mathbf{c}_{1,i_1,\ldots,i_{d-1}}$. We refer to $\mathbf{c}_{1,i_1,\ldots,i_{d-1}}$ as an intersection vector. For each intersection vector $\mathbf{c}_{1,i_1,\ldots,i_{d-1}}$, we can compute its polar expression and solve for the angles φ that generate it. These $d-1$ input angles correspond exactly to the intersection point of $d-1$ curves specified by the above $d-1$ equations. We denote these $d-1$ angles that generate $\mathbf{c}_{1,i_1,\ldots,i_{d-1}}$, as $\varphi_{1,i_1,\ldots,i_{d-1}}$ which we refer to as the intersection point of the $d-1$ curves $\mathcal{B}_{1,i_1}(\varphi), \ldots, \mathcal{B}_{1,i_{d-1}}(\varphi)$.

Since, the $\varphi_{1,i_1,\ldots,i_{d-1}}$ intersection points are defined for every $d-1$ curves, the total number of intersection points is $\binom{n-1}{d-1}$. In the following subsections, we show how we can visit all cells by just examining these intersection points.

We proceed to show that if we visit the adjacent cells of the intersection points defined for all coordinates, then we can find all top-k supports in the span of $\mathbf{V}_d$.

[1] as a matter of fact, due to the sign ambiguity of the solution, this corresponds to two intersection points. However, the following discussion omits this technical detail for simplicity.

1.2. Visiting all cells = finding all top k supports

Our goal is to find all top-k supports that can appear in the span of $\mathbf{V}_d$. To do so, it is sufficient to visit the cells where $[\mathbf{v}(\varphi)]_1$ is the k-th largest coordinate, then the cells where $[\mathbf{v}(\varphi)]_2$ is the k-largest, and so on. Within such cells, one coordinate (say $[\mathbf{v}(\varphi)]_i$) remains always the k-th largest, while the identities of the bottom $n-k$ coordinates remain the same. This means that in such a cell, we have that

$$[\mathbf{v}(\varphi)]_i \geq [\mathbf{v}(\varphi)]_{j_1}, \ldots, [\mathbf{v}(\varphi)]_i \geq [\mathbf{v}(\varphi)]_{j_{n-k}}$$

for all φ in that cell and some scecific $n-k$ other coordinates indexed by $j_1, \ldots, j_{n-k}$. Hence, although the sorting of the top $k-1$ elements might change in that cell (i.e., the first might become the second largest, and vice versa), the coordinates that participate in the top $k-1$ support will be the same, while at the same time the k-th largest will be $[\mathbf{v}(\varphi)]_i$.

Hence, for each coordinate $[\mathbf{v}(\varphi)]_i$, we need to visit the cells wherein it is the k-th largest. We do this by examining *all* cells wherein $[\mathbf{v}(\varphi)]_i$ retains a fixed ranking. Visiting all these cells (T for each coordinate), is possible by visiting all $n \cdot \binom{n-1}{d-1}$ intersection points of $\mathcal{B}_{i,j}(\varphi)$ curves as defined earlier. Since we know that each cell is adjacent to at least 1 intersection point, then at each of these points we visit all adjacent cells. For each cell that we visit, we compute the support of the largest k coordinates of a vector $\mathbf{v}(\varphi_0)$ with a φ_0 that lies in that cell. We include this top k index set in $\mathcal{S}_d$ and carry the same procedure for all cells. Since we visit all coordinates and all their adjacent cells, this means that we visit all cells $\mathcal{C}_j^i$. This means that this procedure will construct all possible supports in

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

1.3. Constructing the set $\mathcal{S}_d$

To visit all possible cells $\mathcal{C}_j^i$, we now have to check the intersection points, which are obtained by solving the system of $d-1$ equations

$$\begin{aligned} &[\mathbf{v}(\varphi)]_{i_1} = [\mathbf{v}(\varphi)]_{i_2} = \ldots = [\mathbf{v}(\varphi)]_{i_d} \\ \Leftrightarrow &[\mathbf{v}(\varphi)]_{i_1} = [\mathbf{v}(\varphi)]_{i_2}, \ldots, [\mathbf{v}(\varphi)]_{i_1} = [\mathbf{v}(\varphi)]_{i_d}. \end{aligned} \tag{4}$$

We can rewrite the above as

$$\begin{bmatrix} \mathbf{e}_{i_1}^T - \mathbf{e}_{i_2}^T \\ \vdots \\ \mathbf{e}_{i_1}^T - \mathbf{e}_{i_d}^T \end{bmatrix} \mathbf{V}_d \mathbf{c} = \mathbf{0}_{(d-1)\times 1} \tag{5}$$

where the solution is the nullspace of the matrix, which has dimension 1.

To explore all possible candidate vectors, we need to visit all cells. To do so, we compute all possible $\binom{n}{d}$ solution intersection vectors $\mathbf{c}_{i_1,\ldots,i_d}$. On each intersection vector we need to compute the locally optimal support set

$$\text{top}_k\left(\mathbf{V}_d \mathbf{c}_{i_1,\ldots,i_d}\right).$$

Then observe that the coordinates $i_1, \ldots, i_d$ of $\mathbf{V}_d \mathbf{c}_{i_1,\ldots,i_d}$ have the same value, since they all satisfy equation (5). Let us assume that t of them appear in the set $\text{top}_k\left(\mathbf{V}_d \mathbf{c}_{i_1,\ldots,i_d}\right)$. The, finding the top k supports of all neighboring cell is equivalent to checking all different supports that can be generated by taking all $\binom{d}{t}$ possible t-subsets of the $i_1, \ldots, i_d$ coordinates with respect to $\mathbf{V}_d \mathbf{c}_{i_1,\ldots,i_d}$, while keeping the rest of the elements in $\mathbf{V}_d \mathbf{c}_{i_1,\ldots,i_d}$ in their original ranking, as computed in $\text{top}_k\left(\mathbf{V}_d \mathbf{c}_{i_1,\ldots,i_d}\right)$. This, induces at most $O(\binom{d}{d/2})$ local sortings, i.e., top k supports. All these sortings will eventually be the elements of the $\mathcal{S}_d$ set. The number of all candidate support sets will now be $O(\binom{d}{d/2}\binom{n-1}{d}) = O(n^d)$ and the total computation complexity is $O\left(n^{d+1}\right)$, since for each point we compute the top-k support in linear time $O(n)$.

For completeness the algorithm of the spannogram framework that generates $\mathcal{S}_d$ is given below.

1.4. Resolution of singularities

In our proofs, we assumed that the curves in $\mathbf{v}(\phi)$ are in general position. This is needed so that no more than $d-1$ curves intersect at a single point. This assumption is equivalent to requiring that every $d \times d$ submatrix of $\mathbf{V}_d$ is full rank. This "general position" requirement can be handled by introducing infinitesimal perturbations in V_d. The details of the analysis of this method can be found in (Papailiopoulos et al., 2013).

Algorithm 1 Spannogram Algorithm for $\mathcal{S}_d$.

$\mathcal{S}_d = \emptyset$
for all $(i_1, \ldots, i_d) \in \{1, \ldots, n\}^d$ and $s \in \{-1, 1\}$ **do**
  $\mathbf{c} = s \cdot \text{nullspace}\left(\begin{bmatrix} [(\mathbf{V}_d]_{i_1,:} - [\mathbf{V}_d]_{i_2,:} \\ \vdots \\ [\mathbf{V}_d]_{i_1,:} - [\mathbf{V}_d]_{i_d,:} \end{bmatrix}\right)$
  $\mathbf{v} = \mathbf{V}_d^T \mathbf{c}$
  $\mathcal{S} = \text{top}_k(\mathbf{v})$
  $\mathcal{T} = \mathcal{S} - \{i_1, \ldots, i_d\}$
  for all $\binom{d}{k-|\mathcal{T}|}$ subsets $\mathcal{J}$ of $(i_1, \ldots, i_d)$ **do**
    $\mathcal{S}_d = \mathcal{S}_d \bigcup (\mathcal{T} \cup \mathcal{J})$
  end for
end for
Output: $\mathcal{S}_d$.

2. Proof of Lemma 1: Going from DkS to DBkS and back

In this subsection we show how a ρ-approximation algorithm for DBkS for arbitrary matrices, implies a 2ρ-approximation for DkS. Our proof goes through a randomized sampling argument.

Algorithm 2 `randombipartite`($\mathcal{G}$)

$\mathcal{L} = \emptyset, \mathcal{R} = \emptyset$
draw n fair coins, and assign each of them to the n vertice of the graph.
$\mathcal{L} =$ the set of vertices that corresponds to heads
$\mathcal{R} = \{1, \ldots, n\} \backslash \mathcal{L}$
$\mathcal{G}_{\mathsf{B}} = \mathcal{G}$
delete all edges in $\mathcal{G}_{\mathsf{B}}(\mathcal{L})$ and $\mathcal{G}_{\mathsf{B}}(\mathcal{R})$
Output: $\mathcal{G}_{\mathsf{B}}$

2.1. Proof of Lemma 1: Randomized Reduction

Let us denote by $\mathcal{G}(\mathcal{S})$ the subgraph in $\mathcal{G}$ induced by a vertex set $\mathcal{S}$. Let the adjacency matrix of the bipartite graph created by `randombipartite`($\mathcal{G}$) be $\mathcal{G}_{\mathsf{B}}$

$$\mathbf{A}_{\mathsf{B}} = \begin{bmatrix} \mathbf{0}_{n_1 \times n_2} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{0}_{n_2 \times n_1} \end{bmatrix},$$

where $n_1 + n_2 = n$. In the following, we refer to $\mathbf{B}$ as the bi-adjacency matrix of the bipartite graph $\mathcal{G}_{\mathsf{B}}$. Moreover, we denote as $\mathcal{L}$ and $\mathcal{R}$ the two disjoint vertex sets of a bipartite graph.

Before we proceed let us state a simple property on the quadratic form of bipartite graphs.

Proposition 1. *Let* $\mathbf{A}_{\mathsf{B}} = \begin{bmatrix} \mathbf{0}_{n_1 \times n_2} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{0}_{n_2 \times n_1} \end{bmatrix}$ *be the adjacency matrix of a bipartite graph. Then, for any subset of vertices* $\mathcal{S}$*, we have that* $\mathcal{S} = \mathcal{S}_l \cup \mathcal{S}_r$*, with* $\mathcal{S}_l = \mathcal{S} \cap \mathcal{L}$ *and* $\mathcal{S}_r = \mathcal{S} \cap \mathcal{R}$*. Moreover,*

$$\mathbf{1}_{\mathcal{S}}^T \mathbf{A}_{\mathsf{B}} \mathbf{1}_{\mathcal{S}} = 2 \cdot \mathbf{1}_{\mathcal{S}_l}^T \mathbf{B} \mathbf{1}_{\mathcal{S}_r}.$$

Proof. It is easy to see that $\mathcal{S}_l$ and $\mathcal{S}_r$ are the vertex subsets of $\mathcal{S}$ that correspond to either the left or right nodes of the bipartite graph. Since the two sets are disjoint, we have

$$\mathbf{1}_{\mathcal{S}} = \mathbf{1}_{\mathcal{S}_l} + \mathbf{1}_{\mathcal{S}_r}.$$

Then, the quadratic forms on $\mathbf{A}_{\mathsf{B}}$ can be equivalently rewritten as bilinear forms on $\mathbf{B}$:

$$\mathbf{1}_{\mathcal{S}}^T \mathbf{A}_{\mathsf{B}} \mathbf{1}_{\mathcal{S}} = (\mathbf{1}_{\mathcal{S}_l} + \mathbf{1}_{\mathcal{S}_r})^T \begin{bmatrix} \mathbf{0}_{n_1 \times n_2} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{0}_{n_2 \times n_1} \end{bmatrix} (\mathbf{1}_{\mathcal{S}_l} + \mathbf{1}_{\mathcal{S}_r}) = \mathbf{1}_{\mathcal{S}_r} \mathbf{B}^T \mathbf{1}_{\mathcal{S}_l} + \mathbf{1}_{\mathcal{S}_l} \mathbf{B} \mathbf{1}_{\mathcal{S}_r} = 2 \cdot \mathbf{1}_{\mathcal{S}_l}^T \mathbf{B} \mathbf{1}_{\mathcal{S}_r}.$$

□

Due to the above, we consider the following bilinear optimization problem

$$\{\mathcal{X}_{\mathsf{B}}, \mathcal{Y}_{\mathsf{B}}\} = \arg \max_{\substack{|\mathcal{X}|=k_1 \\ |\mathcal{Y}|=k_2 \\ k_1+k_2=k}} \mathbf{1}_{\mathcal{X}}^T \mathbf{B} \mathbf{1}_{\mathcal{Y}}, \tag{6}$$

where the constraint $k_1 + k_2 = k$ forces the left and right vertices to induce a k-subgraph. Due to Proposition 1, the two vertex sets are disjoint, i.e., $\mathcal{X}_{\mathsf{B}} \cap \mathcal{Y}_{\mathsf{B}} = \emptyset$, since the columns and rows of $\mathbf{B}$ index two disjoint vertex sets $\mathcal{L}$ and $\mathcal{R}$, respectively.

Let $\mathcal{S}_{\mathsf{B}} = \mathcal{X}_{\mathsf{B}} \cup \mathcal{Y}_{\mathsf{B}}$ be the k vertices in the union of $\mathcal{X}_{\mathsf{B}}$ and $\mathcal{Y}_{\mathsf{B}}$. Then, we will relate the density of $\mathcal{S}_{\mathsf{B}}$ on the original graph $\mathcal{G}$ to the bipartite we obtain from `randombipartite`($\mathcal{G}$).

Proposition 2. *The density of $\mathcal{S}_{\mathsf{B}}$, the densest k-subgraph of $\mathcal{G}_{\mathsf{B}}$, on the original graph $\mathcal{G}$, is at least*

$$\mathsf{den}(\mathcal{S}_{\mathsf{B}}) = \frac{\mathbf{1}_{\mathcal{S}_{\mathsf{B}}}^T \mathbf{A} \mathbf{1}_{\mathcal{S}_{\mathsf{B}}}}{k} \geq \frac{\mathbf{1}_{\mathcal{S}_{\mathsf{B}}}^T \mathbf{A}_{\mathsf{B}} \mathbf{1}_{\mathcal{S}_{\mathsf{B}}}}{k} = 2 \cdot \frac{\mathbf{1}_{\mathcal{X}_{\mathsf{B}}}^T \mathbf{B} \mathbf{1}_{\mathcal{Y}_{\mathsf{B}}}}{k}.$$

Proof. The result follows immediately by the nonnegativity of the entries in $\mathbf{A}$, and the fact that $\mathbf{A}_{\mathsf{B}}$ contains a subset of the entries of $\mathbf{A}$. The last equality follows from Proposition 1. □

We will now show that $\mathsf{den}(\mathcal{S}_{\mathsf{B}})$ is at least $\mathsf{opt}/2$, in expectation. We will use this fact to show that, if we solve DBkS on $\frac{\log n}{\delta}$ graphs independently created using `randombipartite`($\mathcal{G}$), and by keeping the best solution among them, then the extracted k-subgraph has with high probability density $\mathsf{opt}/(2+\delta)$.

Proposition 3. *Let $\mathcal{G}_{\mathsf{B}}$ be the output of `randombipartite`($\mathcal{G}$). Then, there exists in $\mathcal{G}_{\mathsf{B}}$, a k-subgraph that contains $\frac{k \cdot \mathsf{opt}}{2}$ edges, in expectation.*

Proof. First observe that we can represent the edges of $\mathcal{G}_{\mathsf{B}}$ as random variables $X_{i,j}$. If (i,j) is not an edge in $\mathcal{G}$, then $X_{i,j}$ will be 0 with probability 1. If however (i,j) is an edge in $\mathcal{G}$, then $X_{i,j}$ is 1, i.e., appears in $\mathcal{G}_{\mathsf{B}}$, with the same probability that one of its vertices lands in $\mathcal{L}$, while the second is in $\mathcal{R}$. It is easy to find that this probability is $\Pr\{X_{i,j} = 1\} = 1/2$. Hence,

$$X_{i,j} = \begin{cases} 0, & \text{if } (i,j) \text{ not an edge in } \mathcal{G}, \\ Z, & \text{if } (i,j) \text{is an edge in } \mathcal{G}, \end{cases} \tag{7}$$

where Z is a Bernoulli$(1/2)$ random variable.

Now let $\mathcal{S}_*$ denote the vertex set of the densest k-subgraph on the original graph $\mathcal{G}$, that has density $\mathsf{den}(\mathcal{S}_*) = \mathsf{opt}$. Observe that for that subgraph we have

$$\mathbf{1}_{\mathcal{S}_*}^T \mathbf{A} \mathbf{1}_{\mathcal{S}_*} = \sum_{i,j \in \mathcal{S}_*} A_{i,j} = k \cdot \mathsf{opt}.$$

Let $\mathbf{A}_{\mathsf{B}}$ denote the adjacency matrix of the bipartite graph $\mathcal{G}_{\mathsf{B}}$. Then, we have that the expected quadratic form on the new adjacency $\mathbf{1}_{\mathcal{S}_*}^T \mathbf{A}_{\mathsf{B}} \mathbf{1}_{\mathcal{S}_*}$ is:

$$E\left\{\mathbf{1}_{\mathcal{S}_*}^T \mathbf{A}_{\mathsf{B}} \mathbf{1}_{\mathcal{S}_*}\right\} = E\left\{\sum_{i,j \in \mathcal{S}_*} X_{i,j}\right\} = E\left\{\sum_{\substack{i,j \in \mathcal{S}_* \\ (i,j) \in \mathcal{G}}} Z\right\} = \frac{1}{2} \cdot \sum_{i,j \in \mathcal{S}_*} A_{i,j} = \frac{k \cdot \mathsf{opt}}{2}.$$

□

We will now show that if we run `randombipartite`($\mathcal{G}$) a total number of $3 \log n \cdot \log \log n$ times, then with high probability, at least one $\mathcal{G}_{\mathsf{B}}$ will contain a k-subgraph with density at least $0.5 \cdot \mathsf{opt}$. This will imply that the densest k subgraph of $\mathcal{G}_{\mathsf{B}}$ will have density *at least* $0.5 \cdot \mathsf{opt}$.

Algorithm 3 `DkS_2_approx`($\mathcal{G}$, δ)

for $i = 1 : \frac{\log n}{\delta}$ **do**
  $\mathcal{G}_{\mathsf{B}^i} = \texttt{randombipartite}(\mathcal{G})$
  $\mathbf{B}^i =$ biadjacency of $\mathcal{G}_{\mathsf{B}^i}$
  $\{\mathcal{X}^i, \mathcal{Y}^i\} = \arg\max_{|\mathcal{X}|=k_1, |\mathcal{Y}|=k_2, k_1+k_2=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{B}^i \mathbf{1}_{\mathcal{Y}}$
end for
$\{\mathcal{X}_{\mathsf{B}}, \mathcal{Y}_{\mathsf{B}}\} = \arg\max_i \mathbf{1}_{\mathcal{X}^i \cup \mathcal{Y}^i}^T \mathbf{A} \mathbf{1}_{\mathcal{X}^i \cup \mathcal{Y}^i}$
Output: $\mathcal{S}_{\mathsf{B}} = \mathcal{X}_{\mathsf{B}} \cup \mathcal{Y}_{\mathsf{B}}$

Proposition 4. *Then, with probability at least* $1 - \frac{1}{n}$*, we have*

$$\mathbf{1}_{\mathcal{S}_{\mathsf{B}}}^T \mathbf{A} \mathbf{1}_{\mathcal{S}_{\mathsf{B}}} \geq \frac{1-\delta}{2} \cdot \mathsf{opt}.$$

Proof. In this proof we will use the reverse Markov Inequality which states that for any random variable X, such that $X \leq m$, then, for any $a \leq E\{X\}$, we have

$$\Pr\{X \leq a\} \leq \frac{m - E\{X\}}{m - a}.$$

Let $\mathcal{S}_*$ denote the densest k-subgraph for $\mathcal{G}$. Here, our random variable will be the the quadratic form

$$X = \mathbf{1}_{\mathcal{S}_*}^T \mathbf{A}_{\mathsf{B}}^i \mathbf{1}_{\mathcal{S}_*}.$$

Due to Proposition 3, we have that $E\{X\} = 0.5 \cdot k \cdot \mathsf{opt}$. Hence, set $m = k \cdot \mathsf{opt}$ and $\alpha = \frac{k \cdot \mathsf{opt}}{2} - \delta \cdot \frac{k \cdot \mathsf{opt}}{2}$ to obtain:

$$\Pr\left\{X \leq \frac{k \cdot \mathsf{opt}}{2} - \delta \cdot \frac{k \cdot \mathsf{opt}}{2}\right\} \leq \frac{k \cdot \mathsf{opt} - k \cdot \mathsf{opt}/2}{k \cdot \mathsf{opt} - \frac{k \cdot \mathsf{opt}}{2} + \delta \cdot \frac{k \cdot \mathsf{opt}}{2}} = \frac{1}{1+\delta}$$

Now, observe that if we want to have with probability $1 - \frac{1}{n}$ at least one graph $\mathcal{G}_{B^i}$ where

$$\mathbf{1}_{\mathcal{S}_*}^T \mathbf{A}_{\mathsf{B}}^i \mathbf{1}_{\mathcal{S}_*} > \frac{1-\delta}{2} \cdot k \cdot \mathsf{opt},$$

we need to draw l graphs, such that

$$\left(\frac{1}{1+\delta}\right)^l \leq \frac{1}{n}$$

which yields $l = \frac{\log n}{\log(1+\delta)}$. Since $\delta \in (0, 1)$, we have that a number of

$$\frac{\log n}{\delta}$$

draws suffices (assuming the base-2 logarithm), so that

$$\max_i \mathbf{1}_{\mathcal{X}^i \cup \mathcal{Y}^i}^T \mathbf{A}_{\mathsf{B}}^i \mathbf{1}_{\mathcal{X}^i \cup \mathcal{Y}^i} \geq \max_i \mathbf{1}_{\mathcal{S}_*}^T \mathbf{A}_{\mathsf{B}}^i \mathbf{1}_{\mathcal{S}_*} \geq \frac{1-\delta}{2} \mathbf{1}_{\mathcal{S}_*}^T \mathbf{A} \mathbf{1}_{\mathcal{S}_*}.$$

□

Proposition 4 establishes Lemma 2. What we show in our approximation results is that we can compute a solution with density at least

$$\frac{1-\delta}{2} \cdot \mathsf{opt} - 2|\lambda_{d+1}|,$$

where λ_{d+1} is the $d+1$ absolutely largest eigenvalue of the adjacency matrix $\mathbf{A}$.

3. Proof of Theorem 1

3.1. Low-rank DBkS on bipartite graphs and rectangular matrices

The first important technical proposition that we show, is that we can solve DBkS for any constant rank *rectangular* matrix $\mathbf{B}$ of dimensions $n_1 \times n_2$.

Proposition 5. *Let* $\mathbf{B}$ *be any matrix of size* $n_1 \times n_2$ *and let*

$$\mathbf{B}_d = \sum_{i=1}^{d} \sigma_i \mathbf{v}_i \mathbf{u}_i^T$$

be its singular value decomposition, where $\mathbf{v}_i$ *and* $\mathbf{u}_i$ *is the left and right singular vectors corresponding to the* i*th largest singular value* $\sigma_i(\mathbf{B})$*. Then, we can solve the following problem*

$$\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg \max_{|\mathcal{X}|=k_1, |\mathcal{Y}|=k_2} \mathbf{1}_{\mathcal{X}}^T \mathbf{B}_d \mathbf{1}_{\mathcal{Y}}.$$

in time $O(\min\{n_1, n_2\}^{d+1})$.

Proof. For simplicity we assume that the left singular vectors are scaled by their singular values, hence

$$\mathbf{B}_d = \mathbf{v}_1 \mathbf{u}_1^T + \ldots + \mathbf{v}_d \mathbf{u}_d^T.$$

Let us without loss of generality assume that $n_1 \leq n_2$.

We wish to solve:

$$\max_{|\mathcal{X}|=k_1, |\mathcal{Y}|=k_2} \mathbf{1}_{\mathcal{X}}^T \left(\mathbf{v}_1 \mathbf{u}_1^T + \ldots + \mathbf{v}_d \mathbf{u}_d^T \right) \mathbf{1}_{\mathcal{Y}}. \tag{8}$$

Observe that we can rewrite (8) in the following way

$$\max_{|\mathcal{X}|=k_1, |\mathcal{Y}|=k_2} \mathbf{1}_{\mathcal{X}}^T \left[\mathbf{v}_1 \cdot \underbrace{(\mathbf{u}_1^T \mathbf{1}_{\mathcal{Y}})}_{c_1} + \ldots + \mathbf{v}_d \cdot \underbrace{(\mathbf{u}_d^T \mathbf{1}_{\mathcal{Y}})}_{c_d} \right] = \max_{|\mathcal{Y}|=k_2} \left(\max_{|\mathcal{X}|=k_1} \mathbf{1}_{\mathcal{X}}^T \mathbf{v}_{\mathcal{Y}} \right),$$

where $\mathbf{v}_{\mathcal{Y}} = \mathbf{v}_1 \cdot c_1 + \ldots + \mathbf{v}_d \cdot c_d$ is an n_1-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 $\mathbf{v}_{\mathcal{Y}}$, 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_1-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 to either the set of k_1 largest or k_1 smallest signed coordinates of $\mathbf{v}_{\mathcal{Y}}$. That is, the locally optimal sets are either $\text{top}_{k_1}(\mathbf{v}_{\mathcal{Y}})$ or $\text{top}_{k_1}(-\mathbf{v}_{\mathcal{Y}})$.

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 $\text{top}_{k_1}(\pm \mathbf{v}_{\mathcal{Y}})$.

Let us denote as $\mathcal{S}_d$ the set of all k_1-sized sets $\mathcal{X}$ that are the optimal solutions of the inner maximization of in the above, for *any* vector $\mathbf{v}$ in the span of $\mathbf{v}_1, \ldots, \mathbf{v}_d$

$$\mathcal{S}_d = \{\text{top}_{k_1}(\pm[\mathbf{v}_1 \cdot c_1 + \ldots + \mathbf{v}_d \cdot c_d]) : c_1, \ldots, c_d \in \mathbb{R}\}.$$

Clearly, this set contains all possible locally optimal $\mathcal{X}$ sets of the form $\text{top}_{k_1}(\mathbf{v}_{\mathcal{Y}})$. Therefore, we can rewrite DBkS on $\mathbf{B}_d$ as

$$\max_{|\mathcal{Y}|=k_2} \max_{\mathcal{X} \in \mathcal{S}_d} \mathbf{1}_{\mathcal{X}}^T \mathbf{B}_d \mathbf{1}_{\mathcal{Y}}. \tag{9}$$

The above problem can now be solved in the following way: for every set $\mathcal{X} \in \mathcal{S}_d$ find the locally optimal set $\mathcal{Y}$ that maximizes $\mathbf{1}_{\mathcal{X}}^T \mathbf{B}_d \mathbf{1}_{\mathcal{Y}}$. Again, this will either be $\text{top}_{k_2}(-\mathbf{B}_d \mathbf{1}_{\mathcal{X}})$ or $\text{top}_{k_2}(\mathbf{B}_d \mathbf{1}_{\mathcal{X}})$. Then, we simply need to test all such $\mathcal{X}, \mathcal{Y}$ pairs on $\mathbf{B}_d$ and keep the optimizer.

Due to the above, the problem of solving DBkS on the rectangular matrix $\mathbf{B}_d$ is equivalent to constructing the set of k_1-supports $\mathcal{S}_d$, and then finding the optimal solution in that set. How large can $\mathcal{S}_d$ be and can we construct it in polynomial time? As we showed in the first section of the supplemental material this set has size $O(\binom{n_1}{d})$ and can be constructed in time $O(n_1^{d+1})$.

Observe that in the above we could have equivalently solved the problem by finding all the top k_2 sets in the span of $\mathbf{u}_1, \ldots, \mathbf{u}_d$, say that they belong in set $\mathcal{S}'_d$. Then, we could solve the problem by finding for each k_2 sized set $\mathcal{Y} \in \mathcal{S}'_d$ the optimal k_1 sized set $\mathcal{X}$. Both approaches are the same, and the one with the smallest dimension is selected to reduce the computational complexity.

The algorithm that solves the problem for rectangular matrices is given below.

Algorithm 4 low-rank approximations for DBkS

`lowrankDBkS(`k_1`,` k_2`,` d`,` $\mathbf{B}$`)`

$[\mathbf{V}_d, \mathbf{\Sigma}_d, \mathbf{U}_d] = \texttt{SVD}(\mathbf{B}, d)$

$\mathcal{S}_d = \texttt{Spannogram}(k_1, \mathbf{V}_d)$

$\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg\max_{|\mathcal{Y}|=k_2} \max_{\mathcal{X} \in \mathcal{S}_d} \mathbf{1}_{\mathcal{X}}^T \mathbf{V}_d \mathbf{\Sigma}_d \mathbf{U}_d^T \mathbf{1}_{\mathcal{Y}}$

Output: $\{\mathcal{X}_d, \mathcal{Y}_d\}$

`Spannogram(`k_1`,` $\mathbf{V}_d$`)`

$\mathcal{S}_d = \{\text{top}_k(\mathbf{v}) : \mathbf{v} \in \text{span}(\mathbf{v}_1, \ldots, \mathbf{v}_d)\}$

Output: $\mathcal{S}_d$.

□

3.2. Bipartite graphs part of Theorem 1

In our following derivations, for both cases of a rectangular and square symmetric matrices, we consider the same notation of the output solution and output density for simplicity:

$$\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg \max_{|\mathcal{X}|=k, |\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}} \text{ and } \mathsf{opt}_d^{\mathsf{B}} = \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}_d}}{k},$$

$$\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg \max_{k_1, k_2 : k_1 + k_2 = k} \max_{|\mathcal{X}|=k_1, |\mathcal{Y}|=k_2} \mathbf{1}_{\mathcal{X}}^T \mathbf{B}_d \mathbf{1}_{\mathcal{Y}} \text{ and } \mathsf{opt}_d^{\mathsf{B}} = 2\frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{B} \mathbf{1}_{\mathcal{Y}_d}}{k}.$$

Moreover, as a reminder the optimal solutions and densities for the problems of interest (DkS, DBkS on $\mathbf{A}$, and DBkS on $\mathbf{B}$) are

$$\mathcal{S}* = \arg \max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A} \mathbf{1}_{\mathcal{S}} \text{ and } \mathsf{opt} = \frac{\mathbf{1}_{\mathcal{S}_*}^T \mathbf{A} \mathbf{1}_{\mathcal{S}_*}}{k},$$

$$\{\mathcal{X}_*, \mathcal{Y}_*\} = \arg \max_{|\mathcal{X}|=k, |\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}} \text{ and } \mathsf{opt}^{\mathsf{B}} = \frac{\mathbf{1}_{\mathcal{X}_*}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}_*}}{k},$$

$$\{\mathcal{X}_*, \mathcal{Y}_*\} = \arg \max_{k_1, k_2 : k_1 + k_2 = k} \max_{|\mathcal{X}|=k_1, |\mathcal{Y}|=k_2} \mathbf{1}_{\mathcal{X}}^T \mathbf{B} \mathbf{1}_{\mathcal{Y}} \text{ and } \mathsf{opt}^{\mathsf{B}} = 2\frac{\mathbf{1}_{\mathcal{X}_*}^T \mathbf{B} \mathbf{1}_{\mathcal{Y}_*}}{k}.$$

We continue with bounding the distance between the optimal solution for DBkS and rank-d optimal solution pair $\{\mathcal{X}_d, \mathcal{Y}_d\}$. We have the following result, which is essentially Lemma 2 of our main paper.

Proposition 6. *For any matrix* $\mathbf{A}$*, we have*

$$\mathsf{opt}_d^{\mathsf{B}} \geq \mathsf{opt}^{\mathsf{B}} - 2 \cdot |\lambda_{d+1}|. \tag{10}$$

Moreover, for any rectangular matrix $\mathbf{B}$*, we have*

$$\mathsf{opt}_d^{\mathsf{B}} \geq \mathsf{opt}^{\mathsf{B}} - 2 \cdot \sigma_{d+1}. \tag{11}$$

Proof. Let $\mathcal{X}_*, \mathcal{Y}_*$ be the optimal solution of DBkS on $\mathbf{A}$ and let $\mathcal{X}_d, \mathcal{Y}_d$ be the optimal solution of DBkS on the rank-d matrix $\mathbf{A}_d$. Then, we have

$$\begin{aligned}\mathsf{opt}_d^{\mathsf{B}} &= \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}_d}}{k} = \frac{\mathbf{1}_{\mathcal{X}_d}^T (\mathbf{A}_d + \mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_d}}{k} = \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} + \frac{\mathbf{1}_{\mathcal{X}_d}^T (\mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_d}}{k} \\ &\geq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} - \frac{\|\mathbf{1}_{\mathcal{X}_d}\|_2 \cdot \|(\mathbf{A} - \mathbf{A}_d)\mathbf{1}_{\mathcal{Y}_d}\|_2}{k} \geq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} - |\lambda_{d+1}|, \end{aligned} \tag{12}$$

where the first inequality comes due to Cauchy-Schwarz and the second due to the fact that the norm of the indicator vector is k and the operator norm of $\mathbf{A} - \mathbf{A}_d$ is equal to the $d+1$ largest eigenvalue of $\mathbf{A}$.

Moreover, we have that

$$\begin{aligned}\mathsf{opt}^{\mathsf{B}} &= \frac{\mathbf{1}_{\mathcal{X}_*}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}_*}}{k} = \frac{\mathbf{1}_{\mathcal{X}_*}^T (\mathbf{A}_d + \mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_*}}{k} = \frac{\mathbf{1}_{\mathcal{X}_*}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_*}}{k} + \frac{\mathbf{1}_{\mathcal{X}_*}^T (\mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_*}}{k} \\ &\leq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} + \frac{\mathbf{1}_{\mathcal{X}_*}^T (\mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_*}}{k} \leq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} + \frac{\|\mathbf{1}_{\mathcal{X}_*}\|_2 \|(\mathbf{A} - \mathbf{A}_d)\mathbf{1}_{\mathcal{Y}_*}\|_2}{k} \leq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} + |\lambda_{d+1}| \end{aligned} \tag{13}$$

where the first inequality comes due to the fact that $\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d} \geq \mathbf{1}_{\mathcal{X}_*}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_*}$ and the second and third are similar to the previous bound. We can now combine the above two bound to obtain:

$$\mathsf{opt}_d^{\mathsf{B}} \geq \mathsf{opt}^{\mathsf{B}} - 2 \cdot |\lambda_{d+1}|. \tag{14}$$

In the exact same way, we can obtain the result for rectangular matrices. □

The above proposition, combined with Proposition 1 give us the bipartite part of Theorem 1, where $\mathsf{opt}^{\mathsf{B}} = \mathsf{opt}$, that is

$$\mathsf{opt}_d^{\mathsf{B}} \geq \mathsf{opt}^{\mathsf{B}} - 2 \cdot |\lambda_{d+1}| = \mathsf{opt} - 2 \cdot |\lambda_{d+1}|.$$

3.3. Graphs with their first d eigenvalues positive part of Theorem 1

To establish the part about graphs with the d largest eigenvalues being positive, we use the following result.

Proposition 7. *If* $\mathbf{A}_d$ *is positive semidefinite, then*

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

Proof. This is easy to see by the fact that for any two sets $\mathcal{X}, \mathcal{Y}$ we have

$$\begin{aligned}\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}} &\leq \max_{|\mathcal{X}|=k, |\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}} = \max_{|\mathcal{X}|=k, |\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{V}_d \mathbf{\Lambda}_d^{1/2} \mathbf{\Lambda}_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{Y}} \\ &\leq \max_{|\mathcal{X}|=k, |\mathcal{Y}|=k} \max \left\{ \|\mathbf{\Lambda}_d^{1/2} \mathbf{V}_d \mathbf{1}_{\mathcal{X}}\|^2, \ \|\mathbf{\Lambda}_d^{1/2} \mathbf{V}_d \mathbf{1}_{\mathcal{Y}}\| \right\} \leq \max_{|\mathcal{X}|=k, |\mathcal{Y}|=k} \max \left\{ \mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{X}}, \ \mathbf{1}_{\mathcal{Y}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}} \right\} \\ &\leq \max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}} \end{aligned}$$

where the second inequality comes due to the Cauchy-Schwarz inequality. □

We can combine the above proposition with the first part of Proposition 6 to obtain that

$$\mathsf{opt}_d = \mathsf{opt}_d^{\mathsf{B}} \geq \mathsf{opt} - 2|\lambda_{d+1}(\mathbf{A})|$$

when $\mathbf{A}_d$ is positive semidefinite.

3.4. Arbitrary graphs part of Theorem 1

In the next proposition, we show how to translate a low-rank approximation of $\mathbf{A}$ after we used the random sampling of $\texttt{randombipartite}(\mathcal{G})$. We need this result to to establish the general result of Theorem 1, by connecting the previous spectral bound, with the 2 loss in approximation between DBkS and DkS.

Proposition 8. *Let $\mathbf{A}$ be the adjacency matrix of a graph. Moreover, let the matrices $\mathbf{P}_1$ and $\mathbf{P}_2$ be such that $\mathbf{B} = \mathbf{P}_1\mathbf{A}\mathbf{P}_2$ is the bi-adjecency created by each loop of $\texttt{randombipartite}(\mathcal{G})$, where $\mathbf{P}_1$ is an $n_1 \times n$ matrix indexing the left vertices of the graph, and $\mathbf{P}_2$ is an $n \times n_2$ sampling matrix that indexes the right vertices of the sub-sampled graph. Then,*

$$\mathsf{opt}_d^{\mathsf{B}} \geq \mathsf{opt}^{\mathsf{B}} - 2|\lambda_{d+1}(\mathbf{A})|,$$

where $\mathsf{opt}^{\mathsf{B}}$ is the maximum density on $\mathbf{B} = \mathbf{P}_1\mathbf{A}\mathbf{P}_2$.

Proof. Let without loss of generality assume that $\mathbf{B}$ will be the bipartite subgraph between the first n_1 and the remaining $n_2 = n - n_1$ vertices, such that

$$\mathbf{A} = \begin{bmatrix} \mathbf{C} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{D} \end{bmatrix} \tag{15}$$

Then there are two sampling matrices that pick the corresponding columns and rows

$$\mathbf{P}_1 = [\mathbf{I}_{n_1 \times n_1} \quad \mathbf{0}_{n_1 \times n_2}] \text{ and } \mathbf{P}_2 = \begin{bmatrix} \mathbf{0}_{n_1 \times n_2} \\ \mathbf{I}_{n_2 \times n_2} \end{bmatrix}$$

Then, instead of working on the matrix that is the rank-d best fit for $\mathbf{B}$, we work on

$$\mathbf{B}_d = \mathbf{P}_1\mathbf{A}_d\mathbf{P}_2.$$

Now, we use the bounding techniques of our previous derivations:

$$\begin{aligned} \mathsf{opt}_d = \frac{\mathbf{1}_{\mathcal{X}_d}^T\mathbf{B}\mathbf{1}_{\mathcal{Y}_d}}{k} &= \frac{\mathbf{1}_{\mathcal{X}_d}^T\mathbf{P}_1(\mathbf{A}_d + \mathbf{A} - \mathbf{A}_d)\mathbf{P}_2\mathbf{1}_{\mathcal{Y}_d}}{k} = \frac{\mathbf{1}_{\mathcal{X}_d}^T\mathbf{P}_1\mathbf{A}_d\mathbf{P}_2\mathbf{1}_{\mathcal{Y}_d}}{k} + \frac{\mathbf{1}_{\mathcal{X}_d}^T\mathbf{P}_1(\mathbf{A} - \mathbf{A}_d)\mathbf{P}_2\mathbf{1}_{\mathcal{Y}_d}}{k} \\ &\geq \frac{\mathbf{1}_{\mathcal{X}_d}^T\mathbf{P}_1\mathbf{A}_d\mathbf{P}_2\mathbf{1}_{\mathcal{Y}_d}}{k} - \frac{\|\mathbf{1}_{\mathcal{X}_d}\|_2 \cdot \|\mathbf{P}_1(\mathbf{A} - \mathbf{A}_d)\mathbf{P}_2\mathbf{1}_{\mathcal{Y}_d}\|_2}{k} \geq \frac{\mathbf{1}_{\mathcal{X}_d}^T\mathbf{P}_1\mathbf{A}_d\mathbf{P}_2\mathbf{1}_{\mathcal{Y}_d}}{k} - |\lambda_{d+1}|, \end{aligned} \tag{16}$$

where the last step comes due to the fact that $\mathbf{P}_1, \mathbf{P}_2$ their singular values are 1. We can use a similar bound to obtain

$$\mathsf{opt}^{\mathsf{B}} \leq \frac{\mathbf{1}_{\mathcal{X}_d}^T\mathbf{P}_1\mathbf{A}_d\mathbf{P}_2\mathbf{1}_{\mathcal{Y}_d}}{k} + |\lambda_{d+1}|,$$

where $\mathsf{opt}^{\mathsf{B}}$ is the density of the densest k-subgraph on the graph with bi-adjacency matrix $\mathbf{P}_1\mathbf{A}\mathbf{P}_2$. and combine the above to establish the result. □

We can now use our random sampling Proposition 4 and combine that with Propositions 8, and 6, to establish Theorem 1 for arbitrary graphs.

4. Proof of Theorem 2: graphs with highly dense k-subgraphs

We now establish the following a priori spectral bound that holds for any graph.

Lemma 1. *For any unweighted graph $\mathcal{G}$, we have that*

$$|\lambda_d| \leq \sqrt{\frac{2 \cdot m}{d}} \tag{17}$$

where m is the number of edges in $\mathcal{G}$.

Proof. Observe that

$$d \cdot \lambda_d^2 \leq \sum_{i=1}^{d} \lambda_i^2 \leq \sum_{i=1}^{n} \lambda_i^2 = \|\mathbf{A}\|_F^2 = \sum_{i,j} A_{i,j}^2 = \sum_{i,j} A_{i,j} = 2 \cdot m, .$$

where the second to last equality comes due to the fact that $A_{i,j}^2$ can only be 1 or 0. □

We use this bound and Theorem 1, to obtain a the following result, which is a restatement of Theorem 2.

Proposition 9. *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 $1+\epsilon$.*

Proof. For any arbitrary graph, due to Theorem 1, we have

$$\mathsf{opt}_d \geq \left(\frac{1-\delta}{2}\right) \cdot \mathsf{opt} - 2 \cdot |\lambda_{d+1}|.$$

Since, we assumed that the densest k-subgraph contains a constrant fraction of the edges, this means that $k \cdot \mathsf{opt} = c_1 \cdot m$ for some constant $c > 0$. Moreover, we assumed that $k = c_2 \cdot \sqrt{m}$, for some constant $c_2 > 0$. Hence,

$$c_2 \cdot \sqrt{m} \cdot \mathsf{opt} = c_1 \cdot m \Rightarrow \mathsf{opt} = \frac{c_1}{c_2} \cdot \sqrt{m}.$$

Using Lemma 1, we also get

$$|\lambda_d| \leq \sqrt{\frac{2m}{d}} = \sqrt{\frac{2}{d}} \cdot \frac{c_2}{c_1} \cdot \mathsf{opt}.$$

Combining the above gives us

$$\mathsf{opt}_d \geq \left(\frac{1-\delta}{2}\right) \cdot \mathsf{opt} - 2|\lambda_{d+1}| \geq \left(\frac{1-\delta}{2} - \sqrt{\frac{2}{d}} \cdot \frac{c_2}{c_1}\right) \cdot \mathsf{opt}$$

Hence, if we want $\sqrt{\frac{2}{d}} \cdot \frac{c_2}{c_1} = \frac{\delta}{2}$, we need to set $d = \left\lceil \frac{1}{2} \cdot \frac{c_2^2}{c_1^2} \cdot \frac{1}{\delta^2} \right\rceil = O(\frac{1}{\delta^2})$. Setting $\delta = \frac{\epsilon}{2}$ establishes the result. In a similar way, we obtain the $1+\epsilon$ approximation for bipartite graphs, by using the second bound of Theorem 1.

□

5. Proof of Lemma 3: Data Dependent Bounds

Proof. Let $\mathcal{X}_*, \mathcal{Y}_*$ be the optimal solution of DBkS on $\mathbf{A}$ and let Let $\mathcal{X}_d, \mathcal{Y}_d$ be the optimal solution of DBkS on the rank-d matrix $\mathbf{A}_d$. Then, for the first bound we have

$$\begin{aligned}\mathsf{opt}^{\mathsf{B}} &= \frac{\mathbf{1}_{\mathcal{X}_*}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}_*}}{k} = \frac{\mathbf{1}_{\mathcal{X}_*}^T (\mathbf{A}_d + \mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_*}}{k} = \frac{\mathbf{1}_{\mathcal{X}_*}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_*}}{k} + \frac{\mathbf{1}_{\mathcal{X}_*}^T (\mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_*}}{k} \\ &\leq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} + \frac{\mathbf{1}_{\mathcal{X}_*}^T (\mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_*}}{k} \leq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} + \frac{\|\mathbf{1}_{\mathcal{X}_*}\|_2 \, \|(\mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_*}\|_2}{k} \leq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} + |\lambda_{d+1}|. \quad (18)\end{aligned}$$

The upper bound $k-1$ is trivial by the fact that for any $\mathcal{X}$ and $\mathcal{Y}$ we have

$$\frac{\mathbf{1}_{\mathcal{X}}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}}}{k} \leq \frac{\mathbf{1}_{\mathcal{X}}^T (\mathbf{1}\mathbf{1}^T - \mathbf{I}_n) \mathbf{1}_{\mathcal{Y}}}{k} \leq \frac{k(k-1)}{k} = k - 1.$$

The last bound is simply due to the spectral bound on the bilinear form $\frac{\mathbf{1}_{\mathcal{X}}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}}}{k} \leq \frac{\|\mathbf{1}_{\mathcal{X}}\|_2 \cdot \|\mathbf{A}\mathbf{1}_{\mathcal{Y}}\|_2}{k} \leq \lambda_1$.

□

6. Proof of Theorem 3: Nearly-linear Time Algorithm

When the matrix $\mathbf{A}_d$ has mixed signs of eigenvalues, then we have to go through the route of DBkS. However, when $\mathbf{A}_d$ has only positive eigenvalues, then it is easy to show that solving the bilinear problem on $\mathbf{A}_d$ is equivalent to solving

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

This is the DkS low-rank problem, that now can be solved by our algorithm. We show that when this spectral scenario holds, we can speed up computations tremendously, by the use of a simple randomization.

Let us first remind the fact that DBkS and DkS are equivalent for positive semidefinite matrices. We will show here how $\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}}$ can be approximately in time nearly-linear in n, by only introducing a small relative approximation error. Our approximation will use ϵ-nets.

Definition 1. *(ϵ-net) Let $\mathbb{S}^d = \{\mathbf{c} \in \mathbb{R}^d : \|\mathbf{c}\|^2 = 1\}$ be the surface of the d-dimensional sphere. An ϵ-net of $\mathbb{S}^d$ is a finite set $\mathcal{N}_\epsilon^d \subset \mathbb{S}^d$ such that*

$$\forall \mathbf{c} \in \mathbb{S}^d \;\; \exists \;\; \widehat{\mathbf{c}} \in \mathcal{N}_\epsilon^d : \|\mathbf{c} - \widehat{\mathbf{c}}\|_2 \leq \epsilon.$$

We now show that we can solve our optimization, via the use of ϵ-nets, which we construct in the next subsection.

Proposition 10. *Let $\mathcal{N}_\epsilon^d$ be an ϵ-net of $\mathbb{S}^d$. Then,*

$$(1-\epsilon)^2 \cdot \max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}} \leq \max_{\mathbf{c} \in \mathcal{N}_\epsilon^d} \max_{|\mathcal{S}|=k} \left(\mathbf{c}^T \mathbf{\Lambda}_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}\right)^2 \leq \max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}}. \tag{19}$$

Proof. Let $\mathbf{c}$ be a $d \times 1$ unit length vector, *i.e.*, $\|\mathbf{c}\|_2 = 1$. Then, by the Cauchy-Schwartz inequality we have

$$(\mathbf{c}^T \mathbf{\Lambda}_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}})^2 \leq \|\mathbf{\Lambda}_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}\|_2^2.$$

We can get equality in the previous bound for a unit norm $\mathbf{c}$ co-linear to $\mathbf{\Lambda}_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}$. Therefore, we have

$$\|\mathbf{\Lambda}_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}\|_2^2 = \max_{\|\mathbf{c}\|_2=1} (\mathbf{c}^T \mathbf{\Lambda}_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}})^2. \tag{20}$$

Hence,

$$\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}} = \max_{|\mathcal{S}|=k} \|\mathbf{\Lambda}_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}\|_2^2 = \max_{|\mathcal{S}|=k} \max_{\|\mathbf{c}\|_2=1} (\mathbf{c}^T \mathbf{\Lambda}_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}})^2. \tag{21}$$

We can now obtain the upper bound of the proposition, since $\mathcal{N}_\epsilon^d \subseteq \mathbb{S}^d$. Now for the lower bound, let $(\mathbf{1}_{\mathcal{S}_d}, \mathbf{c}_d)$ denote the optimal solution of the above maximization, such that

$$\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}} = (\mathbf{c}_d {\mathbf{V}_d}^T \mathbf{1}_{\mathcal{S}_d})^2.$$

Then, there exists a vector $\widehat{\mathbf{c}}$ in the ϵ-net $\mathcal{N}_\epsilon^d$, such that $\mathbf{c}_d = \widehat{\mathbf{c}} + \mathbf{r}$, with $\|\mathbf{r}\| \leq \epsilon$. Then,

$$\sqrt{\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}}} = {\mathbf{c}_d}^T \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}_d} = (\widehat{\mathbf{c}} + \mathbf{r})^T \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}_d} \overset{(\alpha)}{\leq} \widehat{\mathbf{c}}^T \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}_d} + \epsilon \cdot \|\mathbf{V}_d^T \mathbf{1}_{\mathcal{S}_d}\| = \widehat{\mathbf{c}}^T \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}_d} + \epsilon \cdot \sqrt{\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}}},$$

where (α) is due to the triangle inequality, then the Cauchy-Schwartz inequality, and then the fact that $\|\mathbf{r}\| \leq \epsilon$. From the above inequality we get

$$(1-\epsilon)^2 \max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}} \leq \left(\widehat{\mathbf{c}}^T \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}_d}\right)^2 \leq \max_{\mathbf{c} \in \mathcal{N}_\epsilon^d} \max_{|\mathcal{S}|=k} \left(\mathbf{c}^T \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}\right)^2$$

which concludes the proof. □

The importance of the above proposition lies in the fact that for a fixed $\mathbf{c}$ we can easily solve the problem

$$\max_{|\mathcal{S}|=k} \left(\mathbf{c}^T \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}\right)^2.$$

Observe that the above optimization is the same problem that we had to solve for a fixed $\mathcal{Y}$ in Section 3. This inner product is maximized when $\mathcal{S}$ picks the largest, or smallest k elements of the n-dimensional vector $\mathbf{c}^T\mathbf{V}_d^T$. The complexity to do that is linear $O(n)$ (Cormen et al., 2001).

It is now obvious that the number of elements, and the complexity to construct $\mathcal{N}_\epsilon^d$ is important. In the next subsection, we show how to build such a net, using similar random coding arguments to (Wyner, 1967). Let $\mathcal{N}_\epsilon^d$ be a set of vectors drawn uniformly on the sphere (the cardinality is determined in the next subsection and will be $O(\frac{1}{\epsilon^d} \cdot \log(\frac{1}{\epsilon \cdot \delta}))$. Our algorithm operates as follows: First we draw a set of $|\mathcal{N}_\epsilon^d|$ random vectors, and then we find the corresponding optimal $\mathcal{S}$, by solving

$$\max_{\mathbf{c} \in \mathcal{M}_\epsilon} \max_{|\mathcal{S}|=k} \left(\mathbf{c}^T \mathbf{\Lambda}_d^{1/2} \mathbf{V}_d^T \mathbf{1}_\mathcal{S} \right)^2 .$$

This can be done in time $O(|\mathcal{N}_\epsilon^d| \cdot n)$. Then, among all these solutions, with probability $1-\delta$, the best solution satisfies the bound of the proposition. The randomized algorithm is given below for completeness.

Algorithm 5 Randomized Spannogram

1: `Spannogram_approx(`k, $\mathbf{V}_d, \mathbf{\Lambda}_d$`)`
2: $\mathcal{S}_d = \emptyset$
3: **for** $i = 1 : |\mathcal{N}_\epsilon^d|$ **do**
4: $\quad \mathbf{v} = (\mathbf{\Lambda}_d^{1/2} \cdot \mathbf{V}_d)^T \cdot \texttt{randn}(d, 1)$
5: $\quad \mathcal{S}_d = \mathcal{S}_d \cup \text{top}_k(\mathbf{v}) \cup \text{top}_k(-\mathbf{v})$
6: **end for**
7: **Output:** $\arg\max_{\mathcal{S} \in \mathcal{S}_d} \left\| \mathbf{\Lambda}_d^{1/2} \mathbf{V}_d^T \mathbf{1}_\mathcal{S} \right\|$.

Computing $\mathbf{A}_d$ in the first step of the algorithm, can also be done in nearly linear-time in the size of the input $\mathbf{A}$. There is extensive literature on approximating $\mathbf{A}_d$ by a rank-d matrix $\widehat{\mathbf{A}}_d$ such that $\|\mathbf{A}_d - \widehat{\mathbf{A}}_d\|_2 \leq \delta$, in time proportional to the nonzero entries of $\mathbf{A}$ times a logarithmic term, as long as $|\lambda_d/\lambda_{d+1}|$ is at least a constant (Rokhlin et al., 2009; Halko et al., 2011; Gittens et al., 2013).

6.1. A simple ϵ-net construction via random coding principles

We construct an ϵ-net of the d-dimensional sphere by randomly and independently drawing a sufficient number of uniformly distributed points. This construction is essentially studied by Wyner (Wyner, 1967) in the asymptotic $d \to \infty$ regime, under a different question: how many random spherical caps are needed to cover a sphere?

The idea behind our construction is simple, and uses to ingredients. First, by a lemma of (Vershynin, 2010) (p.8, Lemma 5.2) we know that there exist an ϵ-net on the d-dimensional sphere of size at most

$$|\mathcal{N}_\epsilon^d| \leq \left(1 + \frac{2}{\epsilon}\right)^d .$$

Then, we use an elementary balls-and-bin arguments to find the number of vectors that we need to draw at random, so that each point of the net $\mathcal{N}_\epsilon^d$, is ϵ-close to at least one of the vectors that we drew. This set of random vectors will then correspond to a $2 \cdot \epsilon$-net.

More formally, let

$$\mathbb{C}^d(\mathbf{c}_0, \epsilon) = \left\{ \mathbf{c} \in \mathbb{S}^d : \|\mathbf{c} - \mathbf{c}_0\| \leq \epsilon \right\},$$

be a spherical cap of $\mathbb{S}^d$ centered at $\mathbf{c}_0$, that includes all vectors within distance ϵ from $\mathbf{c}_0$. Let us now define a set of $m_{\epsilon,d}$ spherical caps for each point $\mathbf{c}_0$ of the set $\mathcal{N}_\epsilon^d$. Then, by the definition of an ϵ-net, the caps centered at the vectors of $\mathcal{N}_\epsilon^d$ cover the sphere:

$$\bigcup_{\mathbf{c}_0 \in \mathcal{N}_\epsilon^d} \mathbb{C}^d(\mathbf{c}_0, \epsilon) = \left\{ \mathbf{c} \in \mathbb{S}^d : \|\mathbf{c} - \mathbf{c}_0\| \leq \epsilon \right\} = \mathbb{S}^d.$$

Now, consider an arbitrary point $\widehat{\mathbf{c}_0}$ in some spherical cap $\mathbb{C}^d(\mathbf{c}_0, \epsilon)$. By the triangle inequality, any other point $\mathbf{c}$ in $\mathbb{C}^d(\mathbf{c}_0, \epsilon)$, satisfies

$$\|\mathbf{c} - \widehat{\mathbf{c}}_0\| \leq \|\mathbf{c} - \mathbf{c}_0\| + \|\mathbf{c}_0 - \widehat{\mathbf{c}}_0\| \leq 2\epsilon.$$

The above implies that if we construct a set that contains at least one point from each of the $m_{\epsilon,d}$ caps centered on the vectors of $\mathcal{N}_\epsilon^d$, then this set of points forms a 2ϵ-net. In the following, we do this by randomly drawing a sufficient number of vectors on the sphere.

Let us draw random points uniformly distributed over $\mathbb{S}^d$, by normalizing randomly generated Gaussian vectors distributed according to $N(\mathbf{0}, \mathbf{I}_d)$. A random point falls in one particular cap with probability at least $\frac{1}{m_{\epsilon,d}}$. This is true, since $m_{\epsilon,d}$ spherical caps suffice to cover the surface of the sphere. The probability that some of the caps is empty after we throw m vectors is

$$\Pr\left\{\bigcup_{i=1}^{|\mathcal{N}_\epsilon^d|}\{\text{cap } i \text{ is empty after } m \text{ vector draws}\}\right\} \leq |\mathcal{N}_\epsilon^d| \cdot \left(1 - \frac{1}{|\mathcal{N}_\epsilon^d|}\right)^m. \tag{22}$$

If we wish the probability of this "bad event" to be δ, then we get that the number of m vectors that we need to throw has to satisfy

$$\begin{aligned}
&|\mathcal{N}_\epsilon^d| \cdot \left(1 - \frac{1}{|\mathcal{N}_\epsilon^d|}\right)^m \leq \delta \Rightarrow \log(|\mathcal{N}_\epsilon^d|) + m \cdot \log\left(1 - \frac{1}{|\mathcal{N}_\epsilon^d|}\right) \leq \log(\delta) \\
\Rightarrow &m \geq \frac{\log(\delta) - \log(|\mathcal{N}_\epsilon^d|)}{\log\left(1 - \frac{1}{|\mathcal{N}_\epsilon^d|}\right)} = \frac{\log(|\mathcal{N}_\epsilon^d|/\delta)}{-\log\left(1 - \frac{1}{|\mathcal{N}_\epsilon^d|}\right)} \geq \frac{d\log\left(\frac{1+2\epsilon}{\delta}\right)}{\frac{1}{|\mathcal{N}_\epsilon^d|}} \\
\Rightarrow &m \geq \left(1 + \frac{2}{\epsilon}\right)^d \cdot \left(d \cdot \log\left(1 + \frac{2}{\epsilon}\right) + \log\delta\right)
\end{aligned}$$

Hence, we get the following lemma.

Lemma 2. *Let us draw uniformly at random*

$$\left(1 + \frac{2}{\epsilon}\right)^d \cdot \left(d \cdot \log\left(1 + \frac{2}{\epsilon}\right) + \log\delta\right) = O\left(\frac{1}{\epsilon^d}\log\left(\frac{1}{\epsilon \cdot \delta}\right)\right)$$

vectors on the d-dimensional sphere. Then, with probability at least $1 - \delta$, this set is a $2 \cdot \epsilon$-net of the sphere.

7. Vertex Sparsification via Simple Leverage Score Sampling

Our algorithm comes together with a vertex elimination step: after we compute the low-rank approximation matrix $\mathbf{A}_d$, we discard rows and columns of $\mathbf{A}_d$, i.e., vertices, depending on their *weighted leverage scores*. Leverage score sampling has been extensively studied in the literature, for many different applications, where it can provably provide small error bounds, while keeping a small number of features from the original matrix (Mahoney & Drineas, 2009; Boutsidis et al., 2009).

As we see in the following, this pre-processing step comes with an error guarantee. We show that by throwing away vertices with small leverage scores, can only introduce a provably small error.

Let us define as

$$\ell_i = \left\| \left[\mathbf{V}_d |\mathbf{\Lambda}|^{1/2}\right]_{i,:} \right\| = \sqrt{\sum_{j=1}^{d} [\mathbf{V}_d]_{i,j}^2 |\lambda_j|},$$

the weighted leverage score of a vertex i. Then, our elimination step is simple. Let $\hat{\mathbf{A}}_d$ be a subset of $\mathbf{A}_d$, where we have eliminated all vertices with $\ell_i \leq \frac{\eta}{3k\ell_1}$. Let $\mathbf{P}_\mathcal{H}$ be a diagonal matrix of 1s and 0s, with a 1 only in the (i,i) indices such that $\ell_i > \frac{\eta}{3k\ell_1}$. Then,

$$\hat{\mathbf{A}}_d = \mathbf{P}_\mathcal{H} \mathbf{A}_d \mathbf{P}_\mathcal{H}.$$

We can now guarantee the following upper bound on the error introduced by the elimination.

Proposition 11. *Let $\hat{\mathbf{A}}_d$ be created as above,*

$$\hat{\mathbf{A}}_d = \mathbf{P}_{\mathcal{H}}\mathbf{A}_d\mathbf{P}_{\mathcal{H}}.$$

where $\mathbf{P}_{\mathcal{H}}$ is a diagonal matrix of 1s and 0s, with a 1 on (i,i) indices such that $\ell_i > \frac{\eta}{3k\ell_1}$. Then,

$$\frac{\mathbf{1}_{\mathcal{X}}^T\mathbf{A}_d\mathbf{1}_{\mathcal{Y}}}{k} - \eta \leq \frac{\mathbf{1}_{\mathcal{X}}^T\hat{\mathbf{A}}_d\mathbf{1}_{\mathcal{Y}}}{k} \leq \frac{\mathbf{1}_{\mathcal{X}}^T\mathbf{A}_d\mathbf{1}_{\mathcal{Y}}}{k} + \eta, \tag{23}$$

for all subsets of k vertices $\mathcal{X}, \mathcal{Y}$.

Proof. Let for brevity θ be a user tuned threshold. Moreover, let $\mathbf{P}_{\mathcal{H}}$ be a diagonal matrix of 1s and 0s, with a 1 on (i,i) indices such that $\ell_i > \theta$, and let $\mathbf{P}_{\mathcal{L}}$ be a diagonal matrix of 1s and 0s, with a 1 on (i,i) indices such that $\ell_i \leq \theta$. Clearly,

$$\mathbf{P}_{\mathcal{H}} + \mathbf{P}_{\mathcal{L}} = \mathbf{I}_{n\times n}.$$

Then, we can rewrite $\mathbf{A}_d$ as:

$$\mathbf{A}_d = (\mathbf{P}_{\mathcal{H}} + \mathbf{P}_{\mathcal{L}})\mathbf{A}_d(\mathbf{P}_{\mathcal{H}} + \mathbf{P}_{\mathcal{L}}) = \mathbf{P}_{\mathcal{H}}\mathbf{A}_d\mathbf{P}_{\mathcal{H}} + \mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{L}} + \mathbf{P}_{\mathcal{H}}\mathbf{A}_d\mathbf{P}_{\mathcal{L}} + \mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{H}}.$$

Then, we have the following

$$\begin{aligned}
|\mathbf{1}_{\mathcal{X}}^T(\mathbf{A}_d - \hat{\mathbf{A}}_d)\mathbf{1}_{\mathcal{Y}}| &= |\mathbf{1}_{\mathcal{X}}^T(\mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{L}} + \mathbf{P}_{\mathcal{H}}\mathbf{A}_d\mathbf{P}_{\mathcal{L}} + \mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{H}})\mathbf{1}_{\mathcal{Y}}| && (24)\\
&\leq |\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{L}}\mathbf{1}_{\mathcal{Y}}| + |\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{H}}\mathbf{A}_d\mathbf{P}_{\mathcal{L}}\mathbf{1}_{\mathcal{Y}}| + |\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{H}}\mathbf{1}_{\mathcal{Y}}| && (25)\\
& && (26)
\end{aligned}$$

Observe that for the first error term we have

$$\begin{aligned}
|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{L}}\mathbf{1}_{\mathcal{Y}}| &= |\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\mathbf{\Lambda}_d^{1/2}\mathbf{S}\mathbf{\Lambda}_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{L}}\mathbf{1}_{\mathcal{Y}}| && (27)\\
&\leq \|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\mathbf{\Lambda}_d^{1/2}\| \cdot \|\mathbf{S}\mathbf{\Lambda}_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{L}}\mathbf{1}_{\mathcal{Y}}\| \leq \|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\mathbf{\Lambda}_d^{1/2}\| \cdot \|\mathbf{S}\| \cdot \|\mathbf{\Lambda}_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{L}}\mathbf{1}_{\mathcal{Y}}\| && (28)\\
&= \|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\mathbf{\Lambda}_d^{1/2}\| \cdot 1 \cdot \|\mathbf{\Lambda}_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{L}}\mathbf{1}_{\mathcal{Y}}\| \leq \sqrt{k}\cdot\theta\cdot\sqrt{k}\cdot\theta = k\cdot\theta^2. && (29)
\end{aligned}$$

where the second and third inequalities come due to the Cauchy-Schwarz inequality. In the above $\mathbf{S}$ denotes the diagonal matrix that contains the signs of the eigenvalues. Clearly, its operator norm is 1. Hence, the last inequality in the above is due to the fact that $\mathbf{1}_{\mathcal{X}}, \mathbf{1}_{\mathcal{Y}}$ have k entries with 1, and each picks the rows of $\mathbf{V}_d\mathbf{\Lambda}_d^{1/2}$ with the highest leverage score. Then, due to the triangle inequality on the k-largest row norms (i.e., leverage scores) of $\mathbf{V}_d\mathbf{\Lambda}_d^{1/2}$ we get the final result.

Similarly, we can bound the remaining two error terms

$$\begin{aligned}
|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{H}}\mathbf{1}_{\mathcal{Y}}| &= |\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\mathbf{\Lambda}_d^{1/2}\mathbf{S}\mathbf{\Lambda}_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{H}}\mathbf{1}_{\mathcal{Y}}| && (30)\\
&\leq \|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\mathbf{\Lambda}_d^{1/2}\| \cdot \|\mathbf{S}\mathbf{\Lambda}_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{H}}\mathbf{1}_{\mathcal{Y}}\| \leq \|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\mathbf{\Lambda}_d^{1/2}\| \cdot \|\mathbf{S}\| \cdot \|\mathbf{\Lambda}_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{H}}\mathbf{1}_{\mathcal{Y}}\| && (31)\\
&= \|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\mathbf{\Lambda}_d^{1/2}\| \cdot 1 \cdot \|\mathbf{\Lambda}_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{H}}\mathbf{1}_{\mathcal{Y}}\| \leq \sqrt{k}\cdot\theta\cdot\sqrt{k}\cdot\ell_1 = k\cdot\theta\cdot\ell_1. && (32)
\end{aligned}$$

Since, $\theta \leq \ell_1$, we conclude that the above error can be bounded as

$$\frac{|\mathbf{1}_{\mathcal{X}}^T(\mathbf{A}_d - \hat{\mathbf{A}}_d)\mathbf{1}_{\mathcal{Y}}|}{k} \leq 3\cdot k\cdot\theta\cdot\ell_1 = \eta.$$

Hence, we obtain the proposition. □

8. NP-hardness of DkS on rank-1 matrices

In this section, we establish the hardness of the quadratic formulation of DkS, even for rank-1 matrices. Interestingly the problem is not hard when we relax it to its bilinear form as we showed in our main result. The claim follows.

Claim 1. *DkS is NP-hard for rank-1 matrices $\mathbf{A}$ with one negative eigenvalue.*

Proof. Observe that a rank-1 matrix with 1 negative eigenvalue can be written as

$$\mathbf{A} = -\mathbf{v}\mathbf{v}$$

where $\lambda_1 = \|\mathbf{v}\|^2$. Then, see that

$$\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A} \mathbf{1}_{\mathcal{S}} = \max_{|\mathcal{S}|=k} -\mathbf{1}_{\mathcal{S}}^T \mathbf{v}\mathbf{v}^T \mathbf{1}_{\mathcal{S}} = \min_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{v}\mathbf{v}^T \mathbf{1}_{\mathcal{S}} = \left(\min_{|\mathcal{S}|=k} |\mathbf{1}_{\mathcal{S}}^T \mathbf{v}| \right)^2 = \left(\min_{|\mathcal{S}|=k} \left| \sum_{i \in S} v_i \right| \right)^2 .$$

An algorithm that can solve the above problem, can be used to solve SUBSETSUM. In SUBSETSUM we are given a set of integers and we wish to decide whether there exists a non-empty subset of these integers that sums to zero. In the following algorithm we show how this can be trivially done, by solving $\min_{|\mathcal{S}|=k} \left| \sum_{i \in S} v_i \right|$ for all values of k. If for some value of k the sum in the optimizaton is zero, then we decide YES as the output for the SUBSETSUM. Hence, solving

Algorithm 6 SubsetSum via rank-1 DkS

Input: $\mathbf{v} = [v_1, \ldots, v_n]$
for $i = 1 : k$ **do**
  $s_i = \min_{|\mathcal{S}|=k} \left| \sum_{i \in S} v_i \right|$
  if $s_i == 0$ **then**
    Output: `YES`
  else
    Output: `NO`
  end if
end for

$\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A} \mathbf{1}_{\mathcal{S}}$ is NP-hard even for rank-1 matrices, in the general case. □

9. Additional Experiments

In Fig. 1, we show additional experiment on 9 more large-graphs. The description of the graphs can be found in Table 1. Moreover, in Fig. 2. The description of the experiments can be found in the figure captions.

Table 1. Datasets used in our experiments

DATA SET	NODES	EDGES	DESCRIPTION
COM-DBLP	317,080	1,049,866	DBLP COLLABORATION NETWORK
COM-LIVEJOURNAL	3,997,962	34,681,189	LIVEJOURNAL ONLINE SOCIAL NETWORK
WEB-NOTREDAME	325,729	1,497,134	WEB GRAPH OF NOTRE DAME
EGO-FACEBOOK	4,039	88,234	SOCIAL CIRCLES FROM FACEBOOK (ANONYMIZED)
CA-ASTROPH	18,772	396,160	COLLABORATION NETWORK OF ARXIV ASTRO PHYSICS
CA-HEPPH	12,008	237,010	COLLABORATION NETWORK OF ARXIV HIGH ENERGY PHYSICS
CA-CONDMAT	23,133	186,936	COLLABORATION NETWORK OF ARXIV CONDENSED MATTER
CA-GRQC	5,242	28,980	COLLABORATION NETWORK OF ARXIV GENERAL RELATIVITY
CA-HEPTH	9,877	51,971	COLLABORATION NETWORK OF ARXIV HIGH ENERGY PHYSICS THEORY
LOC-BRIGHTKITE	58,228	214,078	BRIGHTKITE LOCATION BASED ONLINE SOCIAL NETWORK
ROADNET-CA	1,965,206	5,533,214	ROAD NETWORK OF CALIFORNIA
EMAIL-ENRON	36,692	367,662	EMAIL COMMUNICATION NETWORK FROM ENRON
COM-ORKUT	3,072,441	117,185,083	ORKUT ONLINE SOCIAL NETWORK
FLICKR	105,938	2,316,948	NETWORK OF FLICKR IMAGES SHARING COMMON METADATA
FBMEDIUM	63,731	817,090	FRIENDSHIP DATA OF FACEBOOK USERS

References

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.

Gittens, Alex, Kambadur, Prabhanjan, and Boutsidis, Christos. Approximate spectral clustering via randomized sketching. *arXiv preprint arXiv:1311.2854*, 2013.

Halko, Nathan, Martinsson, Per-Gunnar, and Tropp, Joel A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

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.

Papailiopoulos, Dimitris S, Dimakis, Alexandros G, and Korokythakis, Stavros. Sparse pca through low-rank approximations. *arXiv preprint arXiv:1303.0551*, 2013.

Rokhlin, Vladimir, Szlam, Arthur, and Tygert, Mark. A randomized algorithm for principal component analysis. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1100–1124, 2009.

Vershynin, Roman. Introduction to the non-asymptotic analysis of random matrices. *arXiv preprint arXiv:1011.3027*, 2010.

Wyner, Aaron D. Random packings and coverings of the unit n-sphere. *Bell System Technical Journal*, 46(9):2111–2118, 1967.

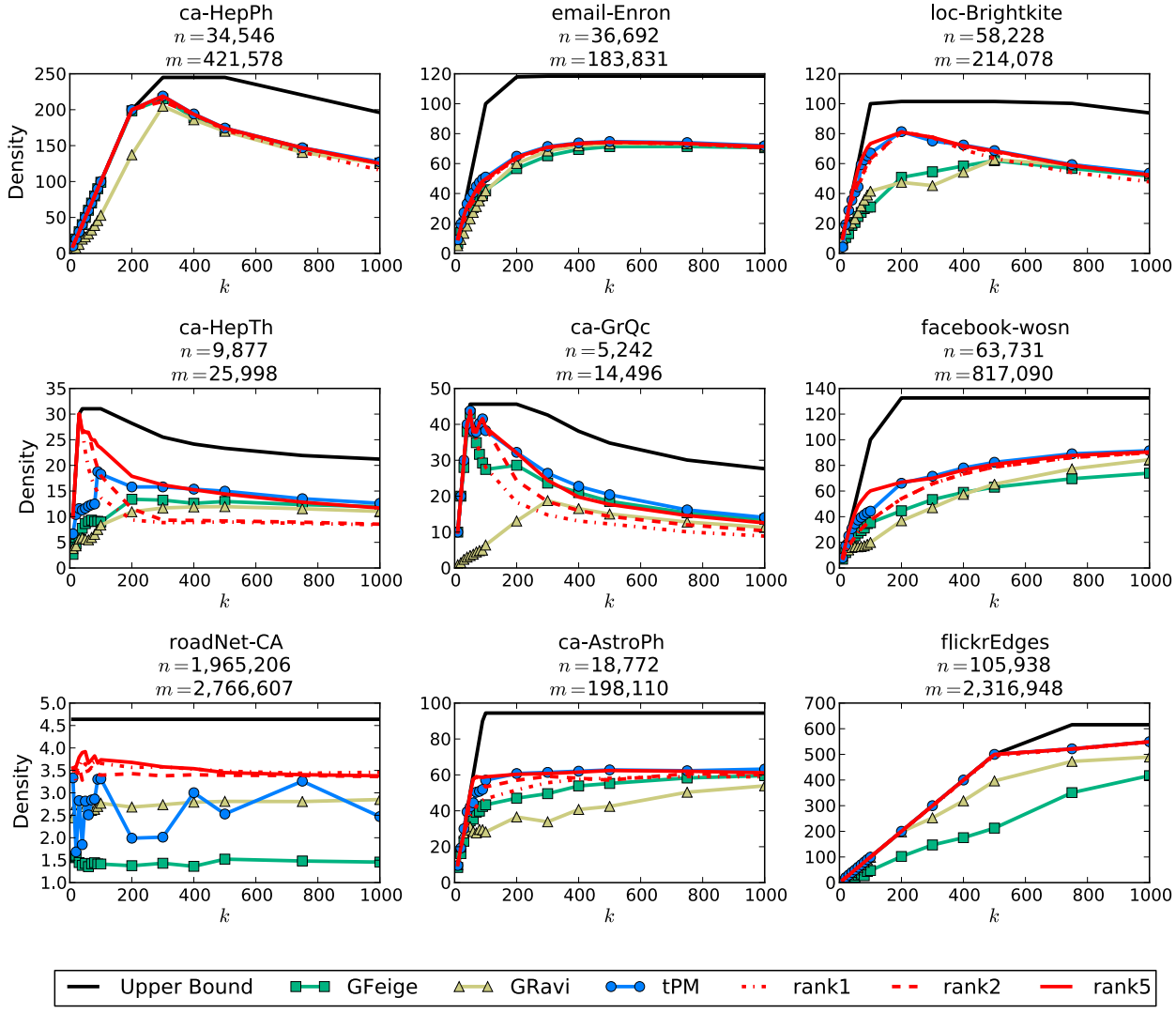


Figure 1. Subgraph density vs. subgraph size (k). We show the comparison of densest subgraph algorithms on several additional datasets: Academic collaboration graphs from Arxiv (ca-HepTh, ca-HepPh, ca-GrQc, Ca-Astro), Geographic location-based networks (roadNet, loc-Brightkite), The Enron email communication graph (email-Enron) and a facebook subgraph (facebook-wosn). The number of vertices and edges are shown in each plot. As can be seen, in almost all cases rank-2 and rank-5 spannograms match or outperform previous algorithms. One notable exception is the ca-GrQc where, for subgraphs of size above $k = 400$ or above, T-power performs better. Another observation is that the spannogram benefits are often more significant for smaller subgraph sizes. It can also be seen that the tightness of our data-dependent bound (solid black line) varies for different data sets and subgraph sizes.

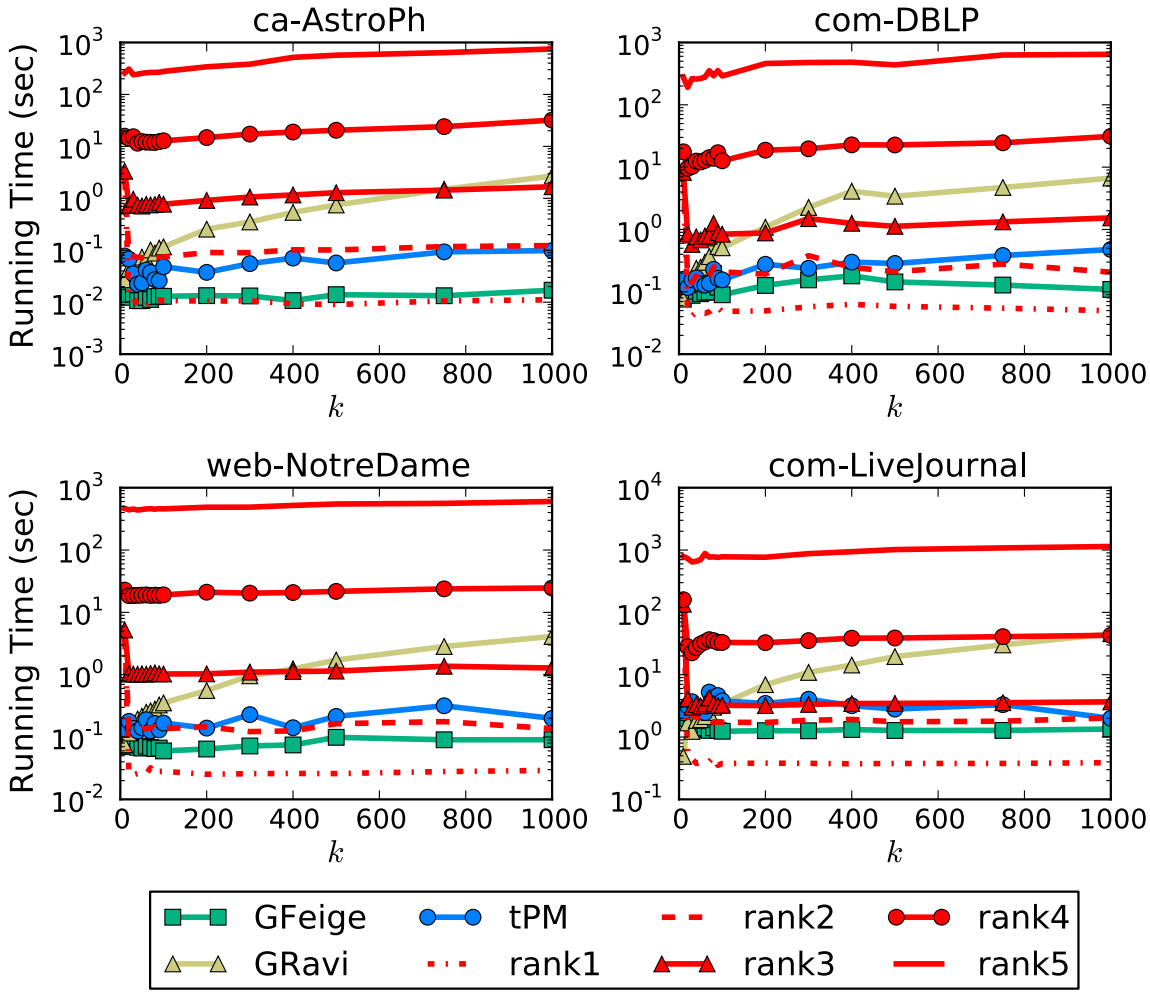


Figure 2. Running times on a MacBook Pro 10.2, with Intel Core i5 @ 2.5 GHz (2 cores), 256 KB L2 Cache per core, 3 MB L3 Cache, and 8 GB RAM. Experiments were run on MATLAB R2011b (7.13.0.564). As can be seen, Rank-1 is significantly faster than all other algorithms for all tested cases. Rank-2 is comparable to prior work, having running times of a few seconds. Rank-5 was the highest accuracy setting we tested. It can take several minutes on large graphs and seems useful only when high accuracy is desired or other methods are far from the upper bound. The approximation error in the ϵ-net was set to 0.1.