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 good solutions. Our algorithm comes with novel performance bounds that depend on the graph spectrum. Our graph-dependent bounds are in practice significantly tighter than worst case a priori bounds: for most tested real-world graphs we find subgraphs with density provably within 70% of the optimum.

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 $G$ 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. The best known approximation ratio was $n^{1/3+\epsilon}$ (for some small $\epsilon$) due to (Feige et al., 2001). Recently, (Bhaskara et al. 2010) introduced an algorithm with approximation ratio $n^{1/4+\epsilon}$, that runs in time $n^{O(1/\epsilon)}$. Such results, where the approximation factor scales as a polynomial in the number of vertices, are too pessimistic for real-world applications. The resistance to better approximation 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 Spangram, which we define below. At its core our algorithm has an exact solver for constant rank bilinear programs with combinatorial constraints.

Our approximation guarantees are data-dependent: they are related to the spectrum of the adjacency matrix. Let opt denote the average degree (i.e., the density) of the densest $k$-subgraph, where $0 \leq \text{opt} \leq k-1$. Our algorithm takes as input the graph, the subgraph size $k$ and an accuracy parameter $d \in \{1, \ldots, n\}$ and runs in time $O(kn^{d+1})$. Note that under a spectral condition satisfied by many real graphs, the computational time is reduced to nearly-linear as we subse-
We found that the above spectral condition holds for log ratio: be positive and let the linear time.

randomized version of our algorithm runs in nearly-linear time approximation.

require the execution of an algorithm.

power of graph-dependent guarantees that, however, is provably within 70% from the upper bound of experimental section, where for many graphs our algorithm obtain a graph-dependent upper bound for the optimal theoretical guarantees, our analysis allows us to observe 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 $O(kn^{d+1})$, where $d$ controls the quality of the approximation. However, under some mild spectral assumptions, a randomized version of our algorithm runs in nearly-linear time.

Next, 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 “$c$-rank,” using an approximate DkS solver for low-rank perturbed versions of the adjacency matrix.

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

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

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

2. Proposed Algorithm

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

\[
\text{den}(S) = \frac{1^T S A 1_S}{|S|}
\]

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

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

\[
\text{DkS} : \quad \text{opt} = \left(\frac{1}{k}\right) \max_{|S|=k} 1^T S A 1_S
\]

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

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

\[
\text{DBkS} : \quad \text{opt} = \left(\frac{1}{k}\right) \max_{|X|=k} \max_{|Y|=k} X^T A Y
\]

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 can be found in the supplemental material, where we describe how to convert an algorithm for DBkS to one for DkS.

Lemma 1. A \( \rho \)-approximation algorithm for DBkS implies a \( 2\rho \)-approximation algorithm for DkS.

2.1. DBkS 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, as we show in the supplemental material.

The exact steps of our low-rank DBkS algorithm are given in the pseudo-code tables referred to as Algorithms 1 and 2.

Step 1: Obtain \( A_d = \sum_{i=1}^d \lambda_i v_i v_i^T \), a rank-\( d \) approximation of \( A \). Here, \( \lambda_i \) is the \( i \)-th largest in magnitude eigenvalue and \( v_i \) the corresponding eigenvector.

Step 2: Use \( A_d \) to obtain \( O(n^d) \) candidate subgraphs.

For any matrix \( A \) we can solve DBkS by exhaustively checking all \( \binom{n}{k} \cdot \binom{k}{d} \) pairs \((X, Y)\) of \( k \)-subsets of vertices. Surprisingly, if we want to find the \( X, Y \) pairs that maximize \( 1^T S A_d 1_Y \), i.e., the bilinear problem on the rank-\( d \) matrix \( A_d \), then we show that only \( O(n^d) \) candidate pairs need to be examined.

In the next section, we derive the constant rank-solver using two key facts. First, for each fixed vertex set \( X \), we show that it is easy to find the optimal set \( Y \) that maximizes \( 1^T Y A_d 1_Y \). Since this turns out to be easy, then the challenge is to find the number of different vertex sets \( Y \) that we need to check. Do we need to exhaustively check all \( \binom{n}{k} \) supports \( Y \)? We show that this question is equivalent to searching the span of the first \( d \) eigenvectors of \( A \), and collecting in a set \( S_d \), the \( k \)-th largest in magnitude eigenvector.

The worst-case time complexity of our algorithm is \( O(Tr_d + n^{d+1}) \), where \( Tr_d \) is the time to compute the first \( d \) eigenvectors of \( A \). Under conditions satisfied by many real world graphs, we show that the complexity reduces to nearly linear in the size of \( G \): \( O(m \cdot \log n + n \cdot \log^{d+1} n) \).

Algorithm 1 lowrankDBkS\((k, \ d, \ A)\)

1. \([V_d, A_d] = \text{EVD}(A, d)\)
2. \(S_d = \text{Spannogram}(k, V_d)\)
3. \([X_d, Y_d] = \text{arg max}_{|X|=k} \max_{|Y|=k} X^T A_d Y^T A_d Y\)
4. Output: \([X_d, Y_d]\)

1. \(\text{Spannogram}(k, V_d)\)
2. \(S_d = \{\text{top}_k(v) : v \in \text{span}(v_1, \ldots, v_d)\}\)
3. Output: \(S_d\).

Remark 1. In the following we present an exact solver for DBkS on constant rank approximations of \( A \). Our general algorithm for DBkS makes a number of \( O(k) \) calls to the DBkS low-rank solver, on slightly different matrices that are obtained by sub-sampling the
2.2. Approximation Guarantees

We approximate DBkS by finding a solution to the constant rank problem
\[
\max_{|X|=k} \max_{|Y|=k} 1^T_X A_d 1_Y.
\]
We output a pair of vertex sets, \(X_d, Y_d\), which we refer to as the \(k\)-subgraph optimal solution, that has density
\[
\text{opt}_d = (1/k) \cdot 1^T_{X_d} A_1 1_{Y_d}.
\]

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

**Lemma 2.** For any matrix \(A\)
\[
\text{opt}_d \geq \text{opt}^B - 2 \cdot |\lambda_{d+1}|.
\]
where \(\lambda_i\) is the \(i\)th largest eigenvalue of \(A\).

Using an appropriate pre-processing step and then running Algorithm 1 as a subroutine on a sub-sampled and low-rank version of \(A\), we output a \(k\)-subgraph \(Z_d\) that has density \(\text{opt}_d\). By essentially combining Lemmata 1 and 2 we obtain the following bounds.

**Theorem 1.** The \(k\)-subgraph indexed by \(Z_d\) has density
\[
\text{opt}_d \geq \frac{\text{opt}}{2 + o_1(1)} - 2 \cdot |\lambda_{d+1}|.
\]
where \(\lambda_i\) is the \(i\)th largest, in magnitude, eigenvalue of the adjacency matrix of the graph. Moreover, if the graph is bipartite, or if the largest \(d\) eigenvalues of the graph are positive, then \(\text{opt}_d \geq \text{opt} - 2 \cdot |\lambda_{d+1}|\).

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 \(\text{DBkS}\) within a factor of \(2 + \epsilon\), in time \(n^\Theta(1/\epsilon^2)\). If additionally the graph is bipartite, then we can approximate \(\text{DBkS}\) within a factor of \(1 + \epsilon\).

**Remark 2.** The above results are similar to the \(1 + \epsilon\) ratio of \(\text{Arora et al., 1995}\), which holds for graphs where the densest-\(k\)-subgraph contains \(\Omega(n^2)\) edges.
Clearly, this set contains all possible locally optimal $X$ sets of the form $\text{top}_k(v_{12})$. Therefore, we can rewrite DBkS on $A_d$ as

$$\max_{|\mathcal{Y}|=k} \max_{X \in \mathcal{S}_d} 1^T_X A_d 1_Y.$$  \hspace{1cm} (3)

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

Due to the above, the problem of solving DBkS on $A_d$ is equivalent to constructing the set of $k$-supports $\mathcal{S}_d$, 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.

**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 $\mathcal{S}_1 = \{\text{top}_k(c_1 \cdot v_1) : c_1 \in \mathbb{R}\}$. It is not hard to see that there are only two supports to include in $\mathcal{S}_1$: $\text{top}_k(v_1)$ and $\text{top}_k(-v_1)$. These two sets can be constructed in time in time $O(n)$, via a partial sorting and selection algorithm (Cormen et al., 2001). Hence, $\mathcal{S}_1$ 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

$$c = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \sin \phi \\ \cos \phi \end{bmatrix}.  \hspace{1cm} \text{(3)}$$

Then, we re-express $c_1 \cdot v_1 + c_2 \cdot v_2$ in terms of $\phi$ as

$$v(\phi) = \sin \phi \cdot v_1 + \cos \phi \cdot v_2.$$  \hspace{1cm} (4)

This means that we can rewrite the set $\mathcal{S}_2$ as:

$$\mathcal{S}_2 = \{\text{top}_k(\pm v(\phi)) : \phi \in \{0, \pi\}\}.$$  \hspace{1cm} (5)

Observe that each element of $v(\phi)$ is a continuous spectral curve in $\phi$: $[v(\phi)]_i = [v_1]_i \sin(\phi) + [v_2]_i \cos(\phi)$. Consequently, the top/bottom-$k$ supports of $v(\phi)$ (i.e., $\text{top}_k(\pm v(\phi))$) are themselves a function of $\phi$. How can we find all possible supports?

---

4Here 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).
and selection algorithm (Cormen et al., 2001). Since we perform this routine a total of $O\left(\binom{n}{2}\right)$ 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 2.

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

**Algorithm 2 Spannogram(k, V_d)**

1: $S_d = \emptyset$
2: for all $(i_1, \ldots, i_d) \in \{1, \ldots, n\}^d$ and $s \in \{-1, 1\}$ do
3:   $c = s \cdot \text{nullspace}(\begin{bmatrix} [V_d]_{i_1} \cdots [V_d]_{i_d} \end{bmatrix})$
4:   $v = V_d^T c$
5:   $S = \text{top}_k(v)$
6:   $T = S - \{i_1, \ldots, i_d\}$
7: for all $(d-1)\text{-subsets } J$ of $(i_1, \ldots, i_d)$ do
8:   $S_d = S_d \cup (T \cup J)$
9: end for
10: end for
11: Output: $S_d$.

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

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

**Theorem 3.** Let the $d$ largest eigenvalues of the graph be positive and let $\lambda_d/\lambda_{d+1} = \Theta(1)$. Then, using Algorithm 3 as a subroutine in Algorithm 1 yields, with high probability, a subgraph with density $(1 - \frac{1}{\log n})\cdot \text{opt}_d$ in nearly-linear time $O(m \cdot \log n + n \cdot \log^{d+1} n)$, where $m$ is the number of edges.

The main idea is that instead of checking all $O(n^d)$ possible $k$ sets in $S_d$, we can approximately solve the problem by randomly sampling $(\log n)$ vectors in the span of $v_1, \ldots, v_d$. Our proof is based on the fact that we can “approximate” the surface of the $d$-dimensional sphere with $M = O(\log^{d+1} n)$ randomly sampled vectors from the span of $v_1, \ldots, v_d$, which then allows us to identify, with high probability, near-optimal candidates in $S_d$. The modified algorithm is very simple and is given below; its analysis can be found in the supplemental material.

**Algorithm 3 Spannogram approx(k, V_d, A_d)**

1: for $i = 1 : d \cdot 2^d \cdot \log^{d+1} n$ do
2:   $v = (A_d^{1/2} \cdot V_d)^T \cdot \text{randn}(d, 1)$
3:   $S_i = S_i \cup \text{top}_k(v) \cup \text{top}_k(-v)$
4: end for
5: Output: $S_d$.

**4. Scaling up**

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

**4.1. Vertex Sparsification**

We introduce a very simple and efficient pre-processing step for discarding vertices that are unlikely to appear in a top $k$ set in $S_d$. This step runs after we compute $A_d$ and uses the leverage score, $\ell_i = \sum_{j=1}^d |V_d|_{i,j}^2 |\lambda_j|$, 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 $\tilde{n} \approx 10 \cdot k$ 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; Babani et al., 2012)) for a comprehensive treatment of the MapReduce paradigm. In short, the Hadoop/MapReduce infrastructure stores the input graph as a distributed file spread across multiple machines; it provides a tuple streaming abstraction, where each map and reduce function receives and emits tuples as (key, value) pairs. The role of the keys is to ensure information aggregation: all the tuples with the same key are processed by the same reducer.

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

Our MapReduce implementation of the rank-2 Spannogram is outlined in Algorithm 4. The Mapper is responsible for the duplication and dissemination of the eigenvectors, $V_2, U_2 = V_2A_2$, to all reducers. Line 3 emits the $j$-th row of $V_2$ and $U_2$ once for every node $i$. Since $i$ is used as the key, this ensures that every reducer receives $V_2, U_2$ in their entirety.

From the breakdown of the Spannogram in Section 3, it is understood that, for the rank-2 case, it suffices to solve a simple system of equations for every pair of nodes. The Reducer for node $i$ receives the full eigenvectors $V_2, U_2$ and is responsible for solving the problem for every pair $(i, j)$, where $j > i$. Then, Line 6 emits the best candidate computed at Reducer $i$. A
performed experiments on synthetic dense subgraphs

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.

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 MR-Job 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 distributed algorithm design. As is well-known, e.g., (Meng & Mahoney), 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
Finding Dense Subgraphs via Low-Rank Bilinear Optimization

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.

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/.


d’Aspremont, Alexandre et al. Weak recovery conditions using graph partitioning bounds. 2010.


Meng, Xiangrui and Mahoney, Michael W. Robust regression on mapreduce. ICML 2013, (to appear).


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

1. Proof of Lemma 4: Building the set $S_d$ for arbitrary $d$-dimensional subspaces

In our general case, we solve DBkS on

$$A_d = V_d U_d^T = \sum_{i=1}^{d} v_i u_i^T$$

where

$$V_d = [v_1 \ldots v_d] \text{ and } U_d = [\lambda_1 \cdot v_1 \ldots \lambda_d \cdot v_d].$$

Solving the problem on $A_d$ 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 v_1 + \ldots + c_d \cdot 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

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

Clearly we can express every vector in the span of $V_d$ as a linear combination $c_1 \cdot v_1 + \ldots + c_d \cdot v_d$ in terms of $\phi$:

$$v(\phi_1, \ldots, \phi_{d-1}) = (\sin \phi_1) \cdot v_1 + (\cos \phi_1 \sin \phi_2) \cdot v_2 + \ldots + (\cos \phi_1 \cos \phi_2 \ldots \cos \phi_{d-1}) \cdot v_d. \quad (1)$$

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 $S_d$ that contains all top $k$ coordinates in the span of $V_d$ as:

$$S_d = \{\text{top}_k(c_1 \cdot v_1 + \ldots + c_d \cdot v_d) : c_1, \ldots, c_d \in \mathbb{R}\}$$

$$= \{\text{top}_k(\varphi) : \varphi \in \Phi^{d-1}\}$$

$$= \{\text{top}_k((\sin \phi_1) \cdot v_1 + (\cos \phi_1 \sin \phi_2) \cdot v_2 + \ldots + (\cos \phi_1 \cos \phi_2 \ldots \cos \phi_{d-1}) \cdot v_d), \varphi \in \Phi^{d-1}\}$$

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

$$[v(\varphi)]_i = (\sin \phi_1) \cdot [v_1]_i + (\cos \phi_1 \sin \phi_2) \cdot [v_2]_i + \ldots + (\cos \phi_1 \cos \phi_2 \ldots \cos \phi_{d-1}) \cdot [v_d]_i.$$

Consequently, the top/bottom-$k$ supports of $v(\varphi)$ (i.e., top_k($\pm v(\varphi)$)) are themselves a function of the $d-1$ variables in $\varphi$. 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 $c_1 \cdot v_1 + \ldots + c_d \cdot v_d$ that we construct using our algorithm.
1.1. Ranking regions for a single coordinate $[v(\varphi)]_1$

We now show that for each single coordinate $[v(\varphi)]_1$, we can partition $\Phi^{d-1}$ in regions, wherein the $i$th coordinate $[v(\varphi)]_i$ retains the same ranking relative to the other $n-1$ coordinates in the vector $v(\varphi)$.

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

\[
\begin{align*}
[v(\varphi)]_1 & \geq [v(\varphi)]_2 \\
[v(\varphi)]_1 & \geq [v(\varphi)]_3 \\
& \vdots \\
[v(\varphi)]_1 & \geq [v(\varphi)]_n
\end{align*}
\]

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

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

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

Observe that these $n-1$ curves $B_{1,1}(\varphi), \ldots, B_{1,n}(\varphi)$ partition $\Phi$ in disjoint cells, $C_1^1, \ldots, C_1^T$, such that

\[
\bigcup_{i=1}^T C_i^1 = \Phi^{d-1}.
\]

Within each cell $C_i^1$, the first coordinate $[v(\varphi)]_1$ retains a fixed ranking relative to the rest of the elements in $v(\varphi)$, 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 $\varphi$ in a single cell, the respective ordering $[v(\varphi)]_1 \geq [v(\varphi)]_2$, \ldots, $[v(\varphi)]_1 \geq [v(\varphi)]_n$ remains the same.

If we have access to a single point, say $\varphi_0$, that belongs to a specific cell, say $C_j^1$, then we can calculate $[v(\varphi_0)]_1$ and find the ranking of the first coordinate $[v(\varphi)]_1$, that remains invariant for all $\varphi \in C_j^1$. Hence, if we visit all these cells, then we can find all possible rankings that the first coordinate $[v(\varphi)]_1$ takes in the $d$-dimensional span of $v_1, \ldots, v_d$. 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 $B_{1,i}(\varphi)$ has a one-to-one correspondence to an equation $[v(\varphi)]_1 = [v(\varphi)]_j$, which is linear in $c$:

\[
[v(\varphi)]_1 = [v(\varphi)]_j \Rightarrow e_i^T V_d c - e_j^T V_d c = 0 \Rightarrow (e_i - e_j)^T V_d c = 0.
\]

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

\[
\begin{align*}
[v(\varphi)]_1 &= [v(\varphi)]_1 \\
(v_1 - e_1)^T V_d c &= 0 \\
& \vdots \\
(v_d - e_d)^T V_d c &= 0
\end{align*}
\]

Let us denote the solution of the above linear inverse problem as $c_{1,i_1,\ldots,i_{d-1}}$. We refer to $c_{1,i_1,\ldots,i_{d-1}}$ as an intersection vector. For each intersection vector $c_{1,i_1,\ldots,i_{d-1}}$, we can compute its polar expression and solve for the angles $\varphi$ 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 $c_{1,i_1,\ldots,i_{d-1}}$, as $\varphi_{1,i_1,\ldots,i_{d-1}}$, which we refer to as the intersection point of the $d-1$ curves $B_{1,i_1}(\varphi), \ldots, B_{1,i_{d-1}}(\varphi)$.

\footnote{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.}
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 $V_d$.

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 $V_d$. To do so, it is sufficient to visit the cells where $[v(\varphi)]_1$ is the $k$-th largest coordinate, then the cells where $[v(\varphi)]_2$ is the $k$-largest, and so on. Within such cells, one coordinate (say $[v(\varphi)]_i$) 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

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

for all $\varphi$ in that cell and some specific $n-k$ 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 $[v(\varphi)]_i$.

Hence, for each coordinate $[v(\varphi)]_i$, we need to visit the cells wherein it is the $k$-th largest. We do this by examining all cells wherein $[v(\varphi)]_i$ 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 $B_{i,j}(\varphi)$ 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 $v(\varphi_0)$ with a $\varphi_0$ that lies in that cell. We include this top $k$ index set in $S_d$ 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 $C^i_j$. This means that this procedure will construct all possible supports in

$$S_d = \{\text{top}_k(c_1 \cdot v_1 + \ldots + c_d \cdot v_d) : c_1, \ldots, c_d \in \mathbb{R}\}$$

1.3. Constructing the set $S_d$

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

$$[v(\varphi)]_{i_1} = [v(\varphi)]_{i_2} = \ldots = [v(\varphi)]_{i_d}$$

$$\Leftrightarrow [v(\varphi)]_{i_1} = [v(\varphi)]_{i_2}, \ldots, [v(\varphi)]_{i_{d-2}} = [v(\varphi)]_{i_{d-1}}, [v(\varphi)]_{i_d}.$$ (4)

We can rewrite the above as

$$\begin{bmatrix}
    e^T_{i_1} - e^T_{i_2} \\
    \vdots \\
    e^T_{i_{d-2}} - e^T_{i_{d-1}}
\end{bmatrix} V_d c = \mathbf{0}_{(d-1) \times 1}$$

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 $\binom{d}{t}$ solution intersection vectors $c_{i_1,\ldots,i_d}$. On each intersection vector we need to compute the locally optimal support set

$$\text{top}_k (V_d c_{i_1,\ldots,i_d})$$

Then observe that the coordinates $i_1,\ldots,i_d$ of $V_d c_{i_1,\ldots,i_d}$ have the same value, since they all satisfy equation (5). Let us assume that $t$ of them appear in the set $\text{top}_k (V_d c_{i_1,\ldots,i_d})$. 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 $V_d c_{i_1,\ldots,i_d}$, while keeping the rest of the elements in $V_d c_{i_1,\ldots,i_d}$ in their original ranking, as computed in $\text{top}_k (V_d c_{i_1,\ldots,i_d})$. This, induces at most $O(\binom{d}{t/2})$ local sortings, i.e.,
top $k$ supports. All these sortings will eventually be the elements of the $S_d$ set. The number of all candidate support sets will now be $O(d^d (n-1)^{d-1}) = O(n^d)$ and the total computation complexity is $O(n^{d+1})$, 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 $S_d$ is given below.

### Algorithm 1 Spannogram Algorithm for $S_d$

1. $S_d = \emptyset$
2. for all $(i_1, \ldots, i_d) \in \{1, \ldots, n\}^d$ and $s \in \{-1, 1\}$ do
3. $c = s \cdot \text{nullspace} \left( \begin{bmatrix} (V_{d11} \ldots V_{d1d}) \\ \vdots \\ (V_{d1d} \ldots V_{ddd}) \end{bmatrix} \right)$
4. $v = V_T^T c$
5. $S = \text{top}_k(v)$
6. $T = S - \{i_1, \ldots, i_d\}$
7. for all $\binom{d-|T|}{d-1}$ subsets $J$ of $(i_1, \ldots, i_d)$ do
8. $S_d = S_d \cup (T \cup J)$
9. end for
10. end for
11. Output: $S_d$.

1.4. Resolution of singularities

In our proofs, we assumed that the curves in $v(\phi)$ 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 $V_d$ 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).

2. Going from D$k$S to DB$k$S and back

In this subsection we show how a $\rho$-approximation algorithm for D$k$S for arbitrary matrices, implies a $2\rho$-approximation for D$k$S. Our proof goes through a randomized sampling argument.

At the end of this section we also show a deterministic scheme that coerits a D$k$S algorithm to an algorithm for D$k$S. This deterministic translation only guarantees that a $\rho$-approximation algorithm for D$k$S for arbitrary matrices, can be converted to $4\rho$-approximation for D$k$S. Our deterministic method uses much simpler vertex pruning techniques.

### Algorithm 2 randombipartite($G$)

1. $L = \emptyset$, $R = \emptyset$
2. for Each vertex $v$ in $G$ do
3. $Z = \text{Bernoulli}(1/2)$.
4. if $Z == 1$ then
5. Put $v$ in $L$
6. else
7. Put $v$ in $R$
8. end if
9. end for
10. $G_B = G$
11. delete all edges in $G_B(L)$ and $G_B(R)$
12. Output: $G_B$
2.1. Proof of Lemma 1: Randomized Reduction

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

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

where $n_1 + n_2 = n$. In the following, we refer to $B$ as the bi-adjacency matrix of the bipartite graph $G_B$. Moreover, we denote as $L$ and $R$ 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 $A_B = \begin{bmatrix} 0_{n_1 \times n_2} & B \\ B^T & 0_{n_2 \times n_1} \end{bmatrix}$ be the adjacency matrix of a bipartite graph. Then, for any subset of vertices $S$, we have that $\overline{S} = S_L \cup S_R$, with $S_L = S \cap L$ and $S_R = S \cap R$. Moreover,

$$1_s^T A_B 1_S = 2 \cdot 1_s^T B 1_{S_R}.$$

**Proof.** It is easy to see that $S_L$ and $S_R$ are the vertex subsets of $S$ that correspond to either the left or right nodes of the bipartite graph. Since the two sets are disjoint, we have

$$1_s = 1_{S_L} + 1_{S_R}.$$

Then, the quadratic forms on $A_B$ can be equivalently rewritten as bilinear forms on $B$:

$$1_s^T A_B 1_S = (1_{S_L} + 1_{S_R})^T \begin{bmatrix} 0_{n_1 \times n_2} & B \\ B^T & 0_{n_2 \times n_1} \end{bmatrix} (1_{S_L} + 1_{S_R}) = 1_{S_L} B^T 1_{S_L} + 1_{S_R} B 1_{S_R} = 2 \cdot 1_{S_L} B 1_{S_R}. \square$$

Due to the above, we will consider the following $\text{DBkS}$ problem

$$\{X_B, Y_B\} = \arg \max_{|X| = k_1} \max_{|Y| = k_2} 1_x^T B 1_y,$$

where $k_1 + k_2 = k$. Due to Proposition 1, $X_B \cap Y_B = \emptyset$, since the columns and rows of $B$ index two disjoint vertex sets $L$ and $R$, respectively. Then, our approximate solution with respect to the original $\text{DbkS}$ problem on $A$ will then be

$$S_B = X_B \cup Y_B.$$

Clearly $|S_B| = k$.

**Proposition 2.** The density of the above approximate solution is

$$\text{den}(S_B) = \frac{1^T_{S_B} A_B 1_{S_B}}{k} \geq \frac{1^T_{S_B} A_B 1_{S_B}}{k} = 2 \cdot 1^T_{S_B} B 1_{Y_B}. \square$$

**Proof.** The result follows immediately by the nonnegativity of the entries in $A$, and the fact that $A_B$ contains a subset of the entries of $A$. The last equality follows from Proposition 1. \square

We will now show that $\text{den}(S_B)$ is at least $\text{opt}/(2 + \delta)$ with high probability, if we solve $\text{DBkS}$ on $\log n$ graphs independently created using $\text{randombipartite}(G)$, and by keeping the best solution among all $\log n$ extracted sets $S_B$.

**Proposition 3.** Let us partition the vertices of $G$ into two sets $L, R$ according to $\text{randombipartite}(G)$: We flip a fair coin for each vertex of $G$ and put it with probability $1/2$ in either of the two sets. Then, we create a bipartite graph $G_B$ that has as left and right vertex sets the sets $L$ and $R$, respectively. The edges that we maintain from the original graph, are only those that cross from $L$ to $R$, while we delete all edges in the subgraphs induced by both sets $L$ and $R$. Then, there exists in a $k$-subgraph in $G_B$, that contains $0.5 \cdot k \cdot \text{opt}$ edges, in expectation.
**Proof.** First observe that we can represent the edges of $\mathcal{G}_B$ 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 $\mathcal{G}_b$, with the same probability that one of its vertices lands in $L$, while the second is in $R$. It is easy to find that this probability is $\Pr\{X_{i,j} = 1\} = 1/2$. Hence,

$$X_{i,j} = \begin{cases} 0, & \text{if } (i,j) \text{ not an edge in } \mathcal{G}, \\ Z, & \text{if } (i,j) \text{ is an edge in } \mathcal{G}, \end{cases} \quad (6)$$

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

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

$$1^T_{\mathcal{S}_s}A_{\mathcal{S}_s} = \sum_{i,j \in \mathcal{S}_s} A_{i,j} = k \cdot \text{opt}. \quad (7)$$

Let $A_B$ denote the adjacency matrix of the bipartite graph $\mathcal{G}_B$. Then, we have that the expected quadratic form on the new adjacency $1^T_{\mathcal{S}_s}A_B1_{\mathcal{S}_s}$ is:

$$E\{1^T_{\mathcal{S}_s}A_B1_{\mathcal{S}_s}\} = E\left\{\sum_{i,j \in \mathcal{S}_s} X_{i,j}\right\} = E\left\{\sum_{i,j \in \mathcal{S}_s, \text{and } (i,j) \in \mathcal{G}} X_{i,j}\right\} = E\left\{\sum_{i,j \in \mathcal{S}_s, \text{and } (i,j) \in \mathcal{G}} Z\right\}$$

$$= \frac{1}{2} \sum_{i,j \in \mathcal{S}_s} A_{i,j} = 0.5 \cdot k \cdot \text{opt.} \quad (7)$$

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

**Proposition 4.** Let us run $\text{randombipartite}(\mathcal{G})$ $m = 3 \log n \cdot \log \log n$ times, and let us obtain each time a graph $\mathcal{G}_B$. For each bipartite graph let $A_B^i$ denote its adjacency and $B_i$ its bi-adjacency, let

$$\{X_i, Y_i\} = \arg \max_{k_1+k_2=k} \max_{|X|=k_1, |Y|=k_2} 1^T_X B_i 1_Y,$$

and let $\mathcal{S}_i = X_i \cup Y_i$ denote the optimal $k$-subgraph of $\mathcal{G}_B^i$. Then, with probability at least $1 - \frac{1}{\log n}$, we have

$$\max_{i=1, \ldots, m} \text{den}(\mathcal{S}_i) = \max_{i=1, \ldots, m} 1^T_{\mathcal{S}_i}A_B^i1_{\mathcal{S}_i} \geq \frac{\text{opt}}{2 + 1/\log n}.$$  

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

$$X = 1^T_{\mathcal{S}_s}A_B^11_{\mathcal{S}_s}.$$  

Due to Proposition 3, we have that $E\{X\} = 0.5 \cdot k \cdot \text{opt}$. Moreover, we know that

$$1^T_{\mathcal{S}_s}A_B^11_{\mathcal{S}_s} \leq 1^T_{\mathcal{S}_s}A1_{\mathcal{S}_s} = k \cdot \text{opt.}$$
Hence, we will set \( \delta = 1/\log n \) and use the reverse Markov inequality:

\[
\Pr \{ 1_T^S \mathbf{A}_B^1 1_{S_c} \leq k \cdot \text{opt} / (2 + 1/\log n) \} \leq \frac{k \cdot \text{opt} - k \cdot \text{opt}/2}{k \cdot \text{opt} - k \cdot \text{opt}/(2 + 1/\log n)} = \frac{1 - 1/2}{1 - 1/(2 + 1/\log n)} = \frac{1/2}{1 + 1/(2 + 1/\log n)} = \frac{2 + 1/\log n}{2 + 2/\log n} = \frac{2 \log n + 1}{2 \log n + 2} = 1 - \frac{1}{2(\log n + 1)} \leq 1 - \frac{1}{3 \log n}.
\]

Now, let \( B_i \) denote the event that \( \{ 1_T^S \mathbf{A}_B^1 1_{S_c} \leq k \cdot \text{opt} / (2 + 1/\log n) \} \). We would like to find a number of draws \( m \) such that, with high probability \( 1 - \frac{1}{\log n} \), we will have one graph \( G_B^i \) satisfying

\[
1_T^S \mathbf{A}_B^1 1_{S_c} > k \cdot \text{opt} / (2 + 1/\log n).
\]

Since these \( m \) events are independent for different graphs, it is easy to see find the number of draws \( m \) by solving the equation:

\[
\left( 1 - \frac{1}{3 \log n} \right)^m \leq \frac{1}{\log n} \Rightarrow m \cdot \log \left( 1 - \frac{1}{3 \log n} \right) \leq - \log(\log n) \Rightarrow m \geq \frac{- \log(\log n)}{\log \left( 1 - \frac{1}{3 \log n} \right)} \Rightarrow m \geq 3 \log n \cdot \log \log n.
\]

where in the last step we used the fact that \(- \log(1 - \epsilon) > \epsilon \), for \( \epsilon < 1 \).

The above derivations tell us that if we draw \( m = 3 \log n \cdot \log \log n \) graphs \( G_B^i \) using \text{randombipartite}(G), then at least one of them contains a subgraph of density

\[
\frac{\text{opt}}{2 + 1/\log n}
\]

Hence, the subgraph \( S_i = X_i \cup Y_i \) with the highest density among the \( m \) densest \( k \)-subgraph of the graphs \( G_B^i \) has density at least \( \frac{\text{opt}}{2 + 1/\log n} \).

\[\square\]

Using Proposition 4, we can now establish Lemma 1 in the following way. Suppose that we have an approximation algorithm such that for any \( k_1, k_2 \), we can obtain a subgraph with density \( A \)

\[
1 - \frac{1}{\rho} \cdot \arg \max_{|X|=k_1} \max_{|Y|=k_2} 1^T_X B^1 y \leq A \leq \arg \max_{|X|=k_1} \max_{|Y|=k_2} 1^T_X B^1 y
\]

Then, we could use it with Proposition 4, to obtain a \((2 + \delta)\rho\) approximation algorithm for \( \text{DkS} \), for any \( \delta > 1/n^7 \), and \( \gamma = \Theta(1) \), in polynomial time.

What we show in our approximation results is that we can compute a solution with density at least

\[
\arg \max_{|X|=k_1} \max_{|Y|=k_2} 1^T_X B^1 y - 2|\lambda_{d+1}|,
\]

where \( \lambda_{d+1} \) is the \( d + 1 \) absolutely largest eigenvalue of the adjacency matrix \( \mathbf{A} \).

\subsection{2.2. Deterministic Reduction}

Here, we show a simper translation of a \( \text{DBkS} \) algorithm to a \( \text{DkS} \) algorithm.

\textbf{Proposition 5.} \( \text{Let } X, Y \text{ be two vertex sets of size } k, \text{ and let } Z = X \cup Y \). Then, we can use \text{prune}(G, X, Y, k) to construct in time \( O(k^2) \) a vertex set \( Z' \) of size \( k \), that has density at least

\[
\left( \frac{1}{4} - O \left( \frac{1}{k} \right) \right) \text{den}(Z).
\]
Algorithm 3 prune($G, \mathcal{X}, \mathcal{Y}, k$)

1: Input: $G, \mathcal{X}, \mathcal{Y}$
2: $Z_1 = \mathcal{X} \cup \mathcal{Y}$, $k' = |Z_1|$
3: for $i = 1 : k' - k$ do
4: find $v$ smallest degree vertex in $G(Z_1)$
5: $Z_{i+1} = Z_i \backslash \{v\}$
6: end for
7: Output: $Z = Z_{k'-k}$

Proof. Let $k'$ denote the size of the set $Z_1 = \mathcal{X} \cup \mathcal{Y}$. Clearly $k \leq k' \leq 2k$. Let us denote as $d_{\text{min}}(S)$, the smallest degree in the subgraph induced by the vertex set $S$. Then, observe that its minimum degree can be bounded as

$$d_{\text{min}}(S) \leq \frac{\text{den}(S)}{|S|}.$$ 

During the first iteration of prune, we delete the smallest degree vertex from $Z_1$ and obtain $Z_2$. This leads to a loss in density that is equal to $2 \cdot d_{\text{min}}(Z_1)$. Hence, the density of $Z_2$ is lower bounded as

$$\text{den}(Z_2) = \text{den}(Z_1) - 2 \cdot d_{\text{min}}(Z_1) \geq \text{den}(Z_1) - 2 \cdot \frac{\text{den}(Z_1)}{k} = \frac{k - 2}{k} \cdot \text{den}(Z_1).$$

This means that for any iteration $i$ of the algorithm the density of the current subgraph induced by $Z_i$ is at least

$$\text{den}(Z_{i+1}) \geq \frac{(k' - i + 1) - 2}{k - i + 1} \cdot Z_i = \frac{k - i - 1}{k - i + 1} \cdot Z_i.$$ 

Hence, the output vertex set $Z$ satisfies

$$\text{den}(Z) \geq \left( \prod_{i=1}^{k' - k} \frac{k' - i - 1}{k' - i + 1} \right) \text{den}(Z_1).$$ 

We can now compute the product using factorials. For the numerator we have

$$(k' - 2) \cdot (k' - 3) \cdots (k' - l - 1) = \frac{k'!}{k' \cdot (k' - 1) \cdot [(k' - l - 2)!].}$$

And for the denominator

$$k' \cdot (k' - 1) \cdots (k' - l + 1) = \frac{k'^l}{(k' - l)!}.$$ 

Hence,

$$\prod_{i=1}^{l} \frac{k' - i - 1}{k' - i + 1} = \frac{(k' - l)!}{k' \cdot (k' - 1) \cdot [(k' - l - 2)!]} = \frac{(k' - l - 1)(k' - l)}{k' \cdot (k' - 1)}. $$

Hence, for $l = k' - k$ we get

$$\text{den}(Z) = \frac{k \cdot (k-1)}{k' \cdot (k' - 1)} \geq \frac{k \cdot (k-1)}{2k \cdot (2k - 1)} = 1 - \frac{1}{8k - 2}$$

which establishes the result. 

Hence, if we had an algorithm that obtains an approximate solution $\{\hat{\mathcal{X}}, \hat{\mathcal{Y}}\}$ such that

$$1_T^\mathcal{X} A_1 \hat{\mathcal{Y}} \geq \frac{1}{\rho} 1_T^\mathcal{X} A_1 \mathcal{Y},$$

Then, we could use prune to obtain a $k$-subgraph $\hat{Z}$, such that

$$\text{den}(\hat{Z}) \geq \frac{1}{4} \frac{1_T^\mathcal{X} A_1 \mathcal{Y}}{k} \geq \frac{1}{4 \cdot \rho} \frac{1_T^\mathcal{X} A_1 \mathcal{Y}}{k} \geq \frac{1}{4 \cdot \rho} \frac{1_T^\mathcal{X} A_1 \mathcal{Y}}{k} \geq \frac{1}{4 \cdot \rho} \text{opt}.$$
3. Approximation Guarantees

3.1. Proof of Theorem 1

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

**Proposition 6.** Let B be any matrix of size n_1 \times n_2 and let

\[ B_d = \sum_{i=1}^{d} \sigma_i v_i u_i^T \]

be its singular value decomposition, where v_i and u_i is the left and right singular vectors corresponding to the ith largest singular value \( \sigma_i(B) \). Then, we can solve the following problem

\[ \{X_d, Y_d\} = \arg \max_{|X|=k_1, |Y|=k_2} \max_{1 \leq d \leq n} \sum_{i \in X} v_i^T B_d Y_i. \]

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

\[ B_d = v_1 u_1^T + \ldots + v_d u_d^T. \]

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

We wish to solve:

\[ \max_{|X|=k_1, |Y|=k_2} \max_{1 \leq d \leq n} \sum_{i \in X} v_i^T (v_1 u_1^T + \ldots + v_d u_d^T) Y_i. \]

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

\[ \max_{|X|=k_1, |Y|=k_2} \max_{1 \leq d \leq n} \sum_{i \in X} v_i^T (v_1 u_1^T + \ldots + v_d u_d^T) Y_i = \max_{|Y|=k_2} \max_{|X|=k_1} \sum_{i \in X} v_i^T v_y, \]

where \( v_y = v_1 \cdot c_1 + \ldots + v_d \cdot c_d \) is an \( n_1 \)-dimensional vector generated by the \( d \)-dimensional subspace spanned by \( v_1, \ldots, v_d \).

We will now make a key observation: for every fixed vector \( v_y \), the index set \( X \) that maximizes \( \max_{X} \sum_{i \in X} v_i^T v_y \) can be easily computed. It is not hard to see that for any fixed vector \( v_y \), the \( k_1 \)-subset \( X \) that maximizes

\[ \sum_{i \in X} v_i^T v_y = \sum_{i \in X} |v_i|, \]

corresponds to either the set of \( k_1 \) largest or \( k_1 \) smallest signed coordinates of \( v_y \). That is, the locally optimal sets are either \( \text{top}_{k_1}(v_y) \) or \( \text{top}_{k_1}(-v_y) \).

We now wish to find all possible locally optimal sets \( X \). If we could possibly check all vectors \( v_y \), then we could find all locally optimal index sets \( \text{top}_{k_1}(\pm v_y) \).

Let us denote as \( S_d \) the set of all \( k_1 \)-sized sets \( X \) that are the optimal solutions of the inner maximization of in the above, for any vector \( v \) in the span of \( v_1, \ldots, v_d \)

\[ S_d = \{ \text{top}_{k_1}(\pm [v_1 \cdot c_1 + \ldots + v_d \cdot c_d]) : c_1, \ldots, c_d \in \mathbb{R} \}. \]

Clearly, this set contains all possible locally optimal \( X \) sets of the form \( \text{top}_{k_1}(v_y) \). Therefore, we can rewrite DBkS on \( B_d \) as

\[ \max_{|Y|=k_2} \max_{X \in S_d} \sum_{i \in X} v_i^T B_d Y_i. \]
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 \(\text{DBkS}\) 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 \(O\left(\binom{n}{d}\right)\) and can be constructed in time \(O(n^{d+1})\).

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 \(\text{DBkS}\)

1. lowrankDBkS\((k_1, k_2, d, \mathbf{B})\)
2. \([\mathbf{V}_d, \Sigma_d, \mathbf{U}_d] = \text{SVD} (\mathbf{B}, d)\)
3. \(\mathcal{S}_d = \text{Spannogram} (k_1, \mathbf{V}_d)\)
4. \(\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg \max_{|\mathcal{X}| = k_2} \max_{\mathcal{X} \in \mathcal{S}_d} \mathbf{1}_\mathcal{X}^T \mathbf{V}_d \Sigma_d \mathbf{U}_d^T \mathbf{1}_\mathcal{Y}\)
5. Output: \(\{\mathcal{X}_d, \mathcal{Y}_d\}\)

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:

\[
\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg \max_{|\mathcal{X}| = k_2, |\mathcal{Y}| = k} \mathbf{1}_\mathcal{X}^T \mathbf{A}_d \mathbf{1}_\mathcal{Y} \quad \text{and} \quad \text{opt}_d = \frac{\mathbf{1}_\mathcal{X}^T \mathbf{A}_d \mathbf{1}_\mathcal{Y}}{k},
\]

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

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

\[
\mathcal{S}_* = \arg \max_{|\mathcal{S}| = k} \mathbf{1}_\mathcal{S}^T \mathbf{A}_1 \mathcal{S} \quad \text{and} \quad \text{opt}_* = \frac{\mathbf{1}_\mathcal{S}^T \mathbf{A}_1 \mathcal{S}}{k},
\]

\[
\{\mathcal{X}_*, \mathcal{Y}_*\} = \arg \max_{|\mathcal{X}| = k_2, |\mathcal{Y}| = k} \mathbf{1}_\mathcal{X}^T \mathbf{A}_1 \mathcal{Y} \quad \text{and} \quad \text{opt}_* = \frac{\mathbf{1}_\mathcal{X}^T \mathbf{A}_1 \mathcal{Y}}{k},
\]

\[
\{\mathcal{X}_*, \mathcal{Y}_*\} = \arg \max_{k_1, k_2, k_1 + k_2 = k} \max_{|\mathcal{X}| = k_1, |\mathcal{Y}| = k_2} \mathbf{1}_\mathcal{X}^T \mathbf{B}_1 \mathcal{Y} \quad \text{and} \quad \text{opt}_* = \frac{2 \mathbf{1}_\mathcal{X}^T \mathbf{B}_1 \mathcal{Y}}{k}.
\]

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.

**Proposition 7.** For any matrix \(\mathbf{A}\), we have

\[
\text{opt}_d \geq \text{opt}_* - 2 \cdot |\lambda_{d+1}|. \quad (10)
\]
Moreover, for any rectangular matrix $B$, we have
\[
\text{opt}^B_d \geq \text{opt}^B - 2 \cdot \sigma_{d+1}.
\] (11)

**Proof.** Let $X_d, Y_d$ be the optimal solution of $DBkS$ on $A$ and let $X_a, Y_a$ be the optimal solution of $DBkS$ on the rank-$d$ matrix $A_d$. Then, we have
\[
\begin{align*}
\text{opt}^B_d &= \frac{1}{k} \sum_{i \leq k} (A_d + A - A_d) Y_d^i X_d^i = \frac{1}{k} \sum_{i \leq k} A_d Y_d^i X_d^i + \frac{1}{k} \sum_{i \leq k} (A - A_d) Y_d^i X_d^i \geq \frac{1}{k} \sum_{i \leq k} A_d Y_d^i X_d^i - |\lambda_{d+1}|,
\end{align*}
\]
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 $A - A_d$ is equal to the $d + 1$ largest eigenvalue of $A$.

Moreover, we have that
\[
\begin{align*}
\text{opt}^B &= \frac{1}{k} \sum_{i \leq k} (A_d + A - A_d) Y_d^i X_d^i = \frac{1}{k} \sum_{i \leq k} A_d Y_d^i X_d^i + \frac{1}{k} \sum_{i \leq k} (A - A_d) Y_d^i X_d^i \\
&\leq \frac{1}{k} \sum_{i \leq k} A_d Y_d^i X_d^i + \frac{1}{k} \sum_{i \leq k} |\lambda_{d+1}| \leq \frac{1}{k} \sum_{i \leq k} A_d Y_d^i X_d^i + |\lambda_{d+1}|
\end{align*}
\]
where the first inequality comes due to the fact that $\frac{1}{k} \sum_{i \leq k} A_d Y_d^i X_d^i \geq \frac{1}{k} \sum_{i \leq k} A_d Y_d^i X_d^i$ and the second and third are similar to the previous bound. We can now combine the above two bound to obtain:
\[
\text{opt}^B_d \geq \text{opt}^B - 2 \cdot |\lambda_{d+1}|.
\] (14)

In the exact same way, we can obtain the result for rectangular matrices. \hfill \Box

The above proposition, combined with Proposition 1 give us the bipartite part of Theorem 1, where $\text{opt}^B = \text{opt}$, that is
\[
\text{opt}^B_d \geq \text{opt}^B - 2 \cdot |\lambda_{d+1}| = \text{opt} - 2 \cdot |\lambda_{d+1}|.
\]
To establish the part about graphs with the $d$ largest eigenvalues being positive, we use the following result.

**Proposition 8.** If $A_d$ is positive semidefinite, then
\[
\max_{|X| = k} \frac{1}{k} \sum_{i \leq k} A_d Y_i X_i = \max_{|X| = k} \max_{|Y| = k} \frac{1}{k} \sum_{i \leq k} A_d Y_i X_i
\]

**Proof.** This is easy to see by the fact that for any two sets $X, Y$ we have
\[
\max_{|S| = k} \frac{1}{k} \sum_{i \leq k} A_d Y_i X_i \leq \max_{|X| = k, |Y| = k} \frac{1}{k} \sum_{i \leq k} A_d Y_i X_i = \max_{|X| = k, |Y| = k} \frac{1}{k} \sum_{i \leq k} A\frac{1}{2} V_d A_d^{1/2} V_d^T Y_i X_i \leq \max_{|X| = k, |Y| = k} \max_{|S| = k} \frac{1}{k} \sum_{i \leq k} A_d Y_i X_i
\]
where the second inequality comes due to the Cauchy-Schwarz inequality. \hfill \Box

We can combine the above proposition with the first part of Proposition 7 to obtain that
\[
\text{opt}^B_d \geq \text{opt} - 2|\lambda_{d+1}(A)|
\]
when $A_d$ is positive semidefinite.

In the next proposition, we show how to translate a low-rank approximation of $A$ after we used the random sampling of $\text{randombipartite}(G)$. 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 $DKS$.
The general algorithm is given below

Proposition 9. Let $A$ be the adjacency matrix of a graph. Moreover, let the matrices $P_1$ and $P_2$ be such that $B = P_1 A P_2$ is the bi-adjacency created by each loop of randombipartite$(G)$, where $P_1$ is an $n_1 \times n$ matrix indexing the left vertices of the graph, and $P_2$ is an $n \times n_2$ sampling matrix that indexes the right vertices of the sub-sampled graph. Then,

$$\text{opt}_{d}^B \geq \text{opt}^B - 2|\lambda_{d+1}(A)|,$$

where $\text{opt}^B$ is the maximum density on $B = P_1 A P_2$.

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

$$A = \begin{bmatrix} C & B \\ B^T & D \end{bmatrix} \quad (15)$$

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

$$P_1 = \begin{bmatrix} I_{n_1 \times n_1} & 0_{n_1 \times n_2} \end{bmatrix} \quad \text{and} \quad P_2 = \begin{bmatrix} 0_{n_1 \times n_2} \\ I_{n_2 \times n_2} \end{bmatrix}$$

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

$$B_d = P_1 A_d P_2.$$

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

$$\text{opt}_d = \frac{1^T_{X_d} B 1_{Y_d}}{k} = \frac{1^T_{X_d} P_1 (A_d + A - A_d) P_2 1_{Y_d}}{k} = \frac{1^T_{X_d} P_1 A_d P_2 1_{Y_d}}{k} + \frac{1^T_{X_d} P_1 (A - A_d) P_2 1_{Y_d}}{k} \geq \frac{1^T_{X_d} P_1 A_d P_2 1_{Y_d}}{k} - |\lambda_{d+1}|, \quad (16)$$

where the last step comes due to the fact that $P_1, P_2$ their singular values are 1. We can use a similar bound to obtain

$$\text{opt}^B \leq \frac{1^T_{X_d} P_1 A_d P_2 1_{Y_d}}{k} + |\lambda_{d+1}|,$$

where $\text{opt}^B$ is the density of the densest $k$-subgraph on the graph with bi-adjacency matrix $P_1 A P_2$. and combine the above to establish the result.

We can now use our random sampling Proposition 4 and combine that with Propositions 9, and 7, to establish Theorem 1 for arbitrary graphs.

The general algorithm is given below
Algorithm 5 low-rank approximations for $DkS$

1: $[V_d, \Lambda_d] = \text{EVD}(A, d)$
2: if $G$ is bipartite then
3: $B = \text{adjacency of } G$
4: $[V_d, \Sigma_d, U_d] = \text{SVD}(B, d)$
5: $B_d = V_d \Sigma_d U_d^T$
6: $\{X_d, Y_d\} = \arg \max_{|X|+|Y|=k} 1_X^T B_d 1_Y$
7: $Z_d = X_d \cup Y_d$
8: else if $G$ is not bipartite and first $d$ eigenvalues are positive then
9: $\{X_d, Y_d\} = \arg \max_{|X|=k} \max_{|Y|=k} 1_X^T A_d 1_Y$
10: $Z_d = X_d$
11: else if $G$ is not bipartite and first $d$ eigenvalues not all positive then
12: for $i = 1: O(\text{poly}(\log n))$ do
13: draw $n$ fair coins and assign them to vertices
14: $\mathcal{L}$ = the set of the vertices with heads.
15: $\mathcal{R} = \{1, \ldots, n\} - \mathcal{L}$
16: $B^i_d = [A_d]_{\mathcal{L}, \mathcal{R}}$
17: $\{X^i_d, Y^i_d\} = \arg \max_{|X|+|Y|=k} 1_X^T B^i_d 1_Y$
18: end for
19: $\{X^1_d, Y^1_d\} = \arg \max_{1 \leq i \leq n} 1_X^T B^i_d 1_Y$
20: $Z_d = X_d \cup Y_d$
21: end if
22: Output: $Z_d$

3.2. Proof of Theorem 2

We now proceed with with a simple bound that we get for the $d$-th largest eigenvalue of any adjacency matrix. This bound in combination with Theorem 1 will establish Theorem 2.

Proposition 10. For any un-weighted adjacency matrix $A$, we have that

$$\lambda_d \leq \sqrt{\frac{2E}{d}}.$$  

Moreover, for any un-weighted bi-adjacency matrix $B$, we have that

$$\sigma_d \leq \sqrt{\frac{E}{d}}.$$

Proof. First, observe that First, observe that

$$d \lambda^2 \leq \sum_{i=1}^d \lambda^2_i \leq \sum_{i=1}^n \lambda^2_i = \|A\|_F^2.$$  

Observe moreover that

$$\|A\|_F = \sqrt{\sum_{i,j} A_{i,j}^2} = \sqrt{\sum_{i,j} A_{i,j}} = \sqrt{2 \cdot E},$$

where the second equality comes due to the fact that $A_{i,j}^2$ can only be 1 or 0. We now combine the above inequalities to obtain $\lambda_d \leq \sqrt{\frac{2E}{d}}$. For the rectangular case, we similarly have

$$d \sigma^2 \leq \sum_{i=1}^d \sigma^2_i \leq \sum_{i=1}^{\min(n_1, n_2)} \lambda^2_i = \|B\|_F^2.$$  

(17)
Observe moreover that
\[ \|B\|_F = \sqrt{\sum_{i,j} B_{i,j}^2} = \sqrt{\sum_{i,j} B_{i,j}} = \sqrt{E}. \]

Hence, \( \sigma_d \leq \sqrt{\frac{E}{\pi}}. \)

We can now establish our \( 1 + \epsilon \) result for bipartite matrices.

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

**Proof.** Let
\[ A = \begin{bmatrix} 0_{n_1 \times n_2} & B \\ B^T & 0_{n_2 \times n_1} \end{bmatrix}, \]
where \( n_1 + n_2 = n \), and \( B \) is the bi-adjacency matrix of the bipartite graph \( G \). Then, as we showed in Section 7
\[ \text{opt} = \max_{|S| = k} 1_S^T A 1_S = 2 \max_{k_1, k_2 : k_1 + k_2 = k} \max_{|X| = k_1, |Y| = k_2} 1_X^T B 1_Y. \]

We approximate the above optimization, by solving the one on the rank-\( d \) matrix \( B_d \):
\[ \max_{k_1, k_2 : k_1 + k_2 = k} \max_{|X| = k_1, |Y| = k_2} 1_X^T B_d 1_Y. \]
and among all \( O(k) \) pairs for all cases \( k_1 + k_2 = k \), we output the one that is best. For each pair \( k_1 \times k_2 \), we solve
\[ \max_{|X| = k_1, |Y| = k_2} 1_X^T B_d 1_Y. \]

Let \( \{X_d, Y_d\} \) denote the solution, among all pairs of sparsity levels \( k_1, k_2 \).

Then, due to Proposition 10, we have that
\[ 1_{X_d}^T B_d 1_{Y_d} \geq 1_{X_d}^T B 1_{Y} - 2 \cdot \sigma_{d+1}. \]

We can now combine the spectral bound of Proposition 10 \( \sigma_i \leq \sqrt{\frac{E}{\pi}} \) with our assumption that the optimal \( k \)-subgraph contains a number of edges equal to \( k \cdot \text{opt} = c \cdot E \), for some constant \( c > 0 \). Since \( \sigma_d \leq \sqrt{\frac{E}{\pi}} \) and \( k = c\sqrt{E} \) we have
\[ 2 \text{opt} = 21_{X_d}^T B 1_{Y} \geq 2(1_{X_d}^T B 1_{Y} - 2 \cdot \sigma_{d+1}) \geq \text{opt} - 4 \cdot \sigma_{d+1} \geq c' \cdot \sqrt{E} - 4\sqrt{\frac{E}{d}} = \left( 1 - 4\sqrt{\frac{2}{c' \pi}} \right) \text{opt} \]

Hence, if we want \( 4\sqrt{\frac{2}{c' \pi}} = \epsilon \), we need
\[ d = \frac{16}{c' \pi^2} = O(1/\epsilon^2), \]
which establishes the result. \( \square \)

Now, it is simple to establish our \( 2 + \epsilon \) result on general graphs that contain a dense subgraph.

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

**Proof.** A simple combination of Proposition 10 and Proposition 4, give us our result. We first convert \( G \) into a bipartite with our random sampling Algorithm of Proposition 4. Then, we solve the bipartite problem on each of the sampled graphs. The solution to the bipartite will be within \( 1 + \epsilon \), since the bipartite graph will contain a constant fraction of the edges. Then converting it to a solution for the original graph, we lose an extra factor of at most \( 2 + \delta \), as we showed in Proposition 4. \( \square \)
3.3. Proof of Lemma 3: the data dependent bound

Finally, we prove our data dependent bounds, that bound the performance of opt, relative to polynomially computable quantities.

**Claim 3.** The optimal density of can be bounded as

\[
\text{opt} \leq \min \left\{ \frac{2 + \alpha_n(1)}{k} \cdot 1_T^T A_d 1_{Z_d} - |\lambda_{d+1}|, \ k - 1, \ \lambda_1 \right\}.
\]

Additionally, if the graph is bipartite, or if the largest \(d\) eigenvalues of the graph are positive, then

\[
\text{opt} \leq \min \left\{ \frac{1}{k} \cdot 1_T^T A_d 1_{Z_d} - |\lambda_{d+1}|, \ k - 1, \ \lambda_1 \right\}.
\]

**Proof.** Let \(X, Y\) be the optimal solution of DBkS on \(A\) and let Let \(X_d, Y_d\) be the optimal solution of DBkS on the rank-\(d\) matrix \(A_d\). Then, for the first bound we have

\[
\text{opt} = \frac{1^T_X A_1 Y}{k} = \frac{1_T^T (A_d + A - A_d) 1_{Y_d}}{k} = \frac{1_T^T A_d 1_{Y_d}}{k} + \frac{1_T^T (A - A_d) 1_{Y_d}}{k}
\]

\[
\leq \frac{1_T^T A_d 1_{Y_d}}{k} + \frac{1_T^T (A - A_d) 1_{Y_d}}{k} \leq \frac{1_T^T A_d 1_{Y_d}}{k} + \frac{1_T^T (A - A_d) 1_{Y_d}}{k} \leq \frac{1_T^T A_d 1_{Y_d}}{k} + |\lambda_{d+1}|.
\]

Applying this bound for all cases of graphs, as described by Theorem 1, and using the same bounds in the proofs of Proposition 7-9, we obtain the result. The upper bound \(k - 1\) is trivial by the fact that for any \(X\) and \(Y\) we have

\[
\frac{1_T^T A_1 Y}{k} \leq \frac{1_T^T (11 - I) 1_{Y}}{k} \leq \frac{k(k - 1)}{k} = k - 1.
\]

The last bound is simply due to the spectral bound on the bilinear form \(\frac{1_T^T A_1 Y}{k} \leq \frac{||1_Y||_2 ||A||_2}{k} \leq \lambda_1\). If we combine now the above with the PSD part of Theorem 1, and the results of Proposition 4, we obtain the claim. \(\square\)

4. Proof of Theorem 3: Nearly-linear Time Solver for Positive Semidefinite Matrices

When the matrix \(A_d\) has mixed signs of eigenvalues, then we have to go through the route of DBkS. However, when \(A_d\) has only positive eigenvalues, then it is easy to show that solving the bilinear problem on \(A_d\) is equivalent to solving

\[
\max_{|X| = k} \frac{1_T^T A_d 1_X}{k}.
\]

This is the \(\text{DiS}\) 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 DiS are equivalent for positive semidefinite matrices. We will show here how \(\max_{|S| = k} 1_S^T A_d 1_S\) can be approximately in time nearly-linear in \(n\), by only introducing a small relative approximation error. Our approximation will use \(\epsilon\)-nets.

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

\[
\forall c \in S^d \exists \hat{c} \in N^d_{\epsilon} : ||c - \hat{c}||_2 \leq \epsilon.
\]

We now show that we can solve our optimization, via the use of \(\epsilon\)-nets.

**Proposition 11.** Let \(N^d_{\epsilon}\) be an \(\epsilon\)-net of \(S^d\). Then,

\[
(1 - \epsilon)^2 \cdot \max_{|S| = k} 1_S^T A_d 1_S \leq \max_{c \in N^d_{\epsilon}, |S| = k} \left( c^T A_d^{1/2} V_d^{1/2} 1_S \right)^2 \leq \max_{|S| = k} 1_S^T A_d 1_S.
\]
Proof. Let $c$ be a $d \times 1$ unit length vector, i.e., $\|c\|_2 = 1$. Then, by the Cauchy-Schwartz inequality we have
\[ (c^T \Lambda_d^{1/2} V_d^T 1_S)^2 \leq \|\Lambda_d^{1/2} V_d^T 1_S\|^2_2. \]

We can get equality in the previous bound for a unit norm $c$ co-linear to $\Lambda_d^{1/2} V_d^T 1_S$. Therefore, we have
\[ \|\Lambda_d^{1/2} V_d^T 1_S\|^2_2 = \max_{|S| = k} (c^T \Lambda_d^{1/2} V_d^T 1_S)^2. \] (22)

Hence,
\[ \max_{|S| = k} 1_S^T A_d 1_S = \max_{|S| = k} \|\Lambda_d^{1/2} V_d^T 1_S\|^2_2 = \max_{|S| = k} \max_{|c| = 1} (c^T \Lambda_d^{1/2} V_d^T 1_S)^2. \] (23)

We can now obtain the upper bound of the proposition, since $N^d_c \subseteq S^d$. Now for the lower bound, let $(1_{S_d}, c_d)$ denote the optimal solution of the above maximization, such that
\[ \max_{|S| = k} 1_S^T A_d 1_S = (c_d V_d^T 1_{S_d})^2. \]

Then, there exists a vector $\tilde{c}$ in the $\epsilon$-net $N^d_c$, such that $c_d = \tilde{c} + r$, with $\|r\| \leq \epsilon$. Then,
\[ \sqrt{\max_{|S| = k} 1_S^T A_d 1_S} = c_d^T V_d^T 1_{S_d} = (\tilde{c} + r)^T V_d^T 1_{S_d} \leq \tilde{c}^T V_d^T 1_{S_d} + \epsilon \cdot \|V_d^T 1_{S_d}\| = \tilde{c}^T V_d^T 1_{S_d} + \epsilon \cdot \max_{|S| = k} 1_S^T A_d 1_S, \]

where (a) is due to the triangle inequality, then the Cauchy-Schwartz inequality, and then the fact that $\|r\| \leq \epsilon$. From the above inequality we get
\[ (1 - \epsilon)^2 \max_{|S| = k} 1_S^T A_d 1_S \leq (\tilde{c}^T V_d^T 1_{S_d})^2 \leq \max_{c \in N^d_c} \max_{|S| = k} (c^T V_d^T 1_S)^2 \]

which concludes the proof.

The importance of the above proposition is because for a fixed $c$ we can easily solve the problem
\[ \max_{|S| = k} (c^T V_d^T 1_S)^2. \]

Observe that the above optimization is the same problem that we had to solve for a fixed $Y$ in Section 3. This inner product is maximized when $S$ picks the largest, or smallest $k$ elements of the $n$-dimensional vector $c^T V_d^T$. 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 $N^d_c$ is important. In (Wyner, 1967), it is shown how to randomly create an $\epsilon$-net on the sphere, by randomly drawing $m$ vectors on the sphere, where
\[ m_\epsilon = \epsilon^{-d} \cdot e^o(d). \]

This approach gives with probability $1 - o_d(1)$ an $\epsilon$-net. Let $M_\epsilon$ be a set of $m_\epsilon$ vectors drawn uniformly on the sphere. Our algorithm operates as follows: First we draw if we draw a set of $M_\epsilon$ random vectors, and then we find the corresponding optimal $S$, by solving
\[ \max_{\epsilon \in M_\epsilon} \max_{|S| = k} \left( c^T \Lambda_d^{1/2} V_d^T 1_S \right)^2. \]

This can be done in time $O(m_\epsilon n)$. Then, we draw the set $M_\epsilon$ a total of $\log n$ times, and iterate this process of finding the best $S$ for each of these sets $M_\epsilon$. Then, among all these solutions, with probability $1 - c/n$, for some $c > 0$, the best solution satisfies the bound of the proposition. The randomized algorithm is given below for completeness.
Algorithm 6 Randomized Spannogram

1: Spannogram\_approx(k, V_d, A_d)
2: S_d = \emptyset
3: for i = 1 : m \cdot \log n do
4: v = (A_d^{1/2} \cdot V_d)^T \cdot \text{randn}(d, 1)
5: S_d = S_d \cup \text{top}_k(v) \cup \text{top}_k(-v)
6: end for
7: Output: arg max_{S \in S_d} \|A_d^{1/2}V_d^T 1_s\|

Computing $A_d$ in the first step of the algorithm, can also be done in nearly linear-time in the size of the input $A$. There is extensive literature on approximating $A_d$ by a rank-$d$ matrix $\hat{A}_d$ such that $\|A_d - \hat{A}_d\|_2 \leq \delta$, in time proportional to the nonzero entries of $A$ times a poly-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).

By further setting $\epsilon = \frac{1}{\log n}$, we obtain Theorem 3.

5. Vertex Sparsification via Simple Leverage Score Sampling

Our algorithm comes together with a vertex elimination step: after we compute the low-rank approximation matrix $A_d$, we discard rows and columns of $A_d$, 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, we can only introduce a provably small error.

Let us define as

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

the weighted leverage score of a vertex $i$. Then, our elimination step is simple. Let $\hat{A}_d$ be a subset of $A_d$, where we have eliminated all vertices with $\ell_i \leq \frac{n}{\sqrt{\log t}}$. Let $P_H$ be a diagonal matrix of 1s and 0s, with a 1 only in the $(i, i)$ indices such that $\ell_i > \frac{n}{\sqrt{\log t}}$. Then,

$$\hat{A}_d = P_H A_d P_H.$$

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

**Proposition 12.** Let $\hat{A}_d$ be created as above,

$$\hat{A}_d = P_H A_d P_H,$$

where $P_H$ is a diagonal matrix of 1s and 0s, with a 1 on $(i, i)$ indices such that $\ell_i > \frac{n}{\sqrt{\log t}}$. Then,

$$\frac{1}{k} \hat{A}_d 1_y - \eta \leq \frac{1}{k} A_d 1_y \leq \frac{1}{k} \hat{A}_d 1_y + \eta,$$

for all subsets of $k$ vertices $X, Y$.

**Proof.** Let for brevity $\theta = \frac{n}{\sqrt{\log t}}$. Moreover, Let $P_H$ be a diagonal matrix of 1s and 0s, with a 1 on $(i, i)$ indices such that $\ell_i > \frac{n}{\sqrt{\log t}}$, and let $P_L$ be a diagonal matrix of 1s and 0s, with a 1 on $(i, i)$ indices such that $\ell_i \leq \frac{n}{\sqrt{\log t}}$. Clearly,

$$P_H + P_L = I_{n \times n}.$$

Then, we can rewrite $A_d$ as:

$$A_d = (P_H + P_L) A_d (P_H + P_L) = P_H A_d P_H + P_L A_d P_L + P_H A_d P_L + P_L A_d P_H.$$
Then, we have the following

\[ |1^T_y (A_d - \hat{A}_d) 1_y| = |1^T_x (P_C A_d P_C + P_H A_d P_C + P_C A_d P_H) 1_y| \]

\[ \leq |1^T_x P_C A_d P_C 1_y| + |1^T_x P_H A_d P_C 1_y| + 2 |1^T_x P_C A_d P_H 1_y| \]  

(25)  

(26)  

(27)

Observe that for the first error term we have

\[ |1^T_x P_C A_d P_C 1_y| = |1^T_x P_C V_d A_d^{1/2} S A_d^{1/2} V_d^T P_C 1_y| \]

\[ \leq \|1^T_x P_C V_d A_d^{1/2}\| \cdot \|S A_d^{1/2} V_d^T P_C 1_y\| \leq \|1^T_x P_C V_d A_d^{1/2}\| \cdot \|S\| \cdot \|A_d^{1/2} V_d^T P_C 1_y\| \]

\[ = \|1^T_x P_C V_d A_d^{1/2}\| \cdot \|S\| \cdot \|A_d^{1/2} V_d^T P_C 1_y\| \leq k \cdot \theta \cdot k \cdot \theta = k^2 \cdot \theta^2. \]  

(28)  

(29)  

(30)

where the second and third inequalities come due to the Cauchy-Schwarz inequality. In the above \( S \) 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 \( 1^T_x \cdot 1_y \) have \( k \) entries with 1, and each picks the rows of \( V_d A_d^{1/2} \) with the highest leverage score. Then, due to the triangle inequality on the \( k \)-largest row norms (i.e., leverage scores) of \( V_d A_d^{1/2} \) we get the final result.

Similarly, we can bound the remaining two error terms

\[ |1^T_x P_C A_d P_H 1_y| = |1^T_x P_C V_d A_d^{1/2} S A_d^{1/2} V_d^T P_H 1_y| \]

\[ \leq \|1^T_x P_C V_d A_d^{1/2}\| \cdot \|S A_d^{1/2} V_d^T P_H 1_y\| \leq \|1^T_x P_C V_d A_d^{1/2}\| \cdot \|S\| \cdot \|A_d^{1/2} V_d^T P_H 1_y\| \]

\[ = \|1^T_x P_C V_d A_d^{1/2}\| \cdot \|S\| \cdot \|A_d^{1/2} V_d^T P_H 1_y\| \leq k \cdot \theta \cdot k \cdot \ell_1 = k^2 \cdot \theta \cdot \ell_1. \]  

(31)  

(32)  

(33)

Since, \( \eta \leq \ell_1 \), we conclude that the above error can be bounded as

\[ \frac{|1^T_x (A_d - \hat{A}_d) 1_y|}{k} \leq 3 \cdot k \cdot \theta \cdot \ell_1 = \eta. \]

(34)

Hence, we obtain the proposition.

6. NP-hardness of \( \text{DkS} \) on rank-1 matrices

In this section, we establish the hardness of the quadratic formulation of \( \text{DkS} \), 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 4.** \( \text{DkS} \) is NP-hard for rank-1 matrices \( A \) with one negative eigenvalue.

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

\[ A = -vv^T \]

where \( \lambda_1 = \|v\|^2 \). Then, see that

\[ \max_{|S|=k} 1^T_S A 1_S = \max_{|S|=k} -1^T_S vv^T 1_S = \min_{|S|=k} 1^T_S vv^T 1_S = \left( \min_{|S|=k} |1^T_S v| \right)^2 = \left( \min_{|S|=k} \sum_{i \in S} v_i \right)^2. \]  

(35)  

(36)  

(37)

An algorithm that can solve the above problem, can be used to solve \( \text{SubsetSum} \). In \( \text{SubsetSum} \) 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 \( \min_{|S|=k} \sum_{i \in S} v_i \) for all values of \( k \). If for some value of \( k \) the sum in the minimization is zero, then we decide YES as the output for the \( \text{SubsetSum} \). Hence, solving \( \max_{|S|=k} 1^T_S A 1_S \) is NP-hard even for rank-1 matrices, in the general case.

7. 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.
Algorithm 7 SubsetSum via rank-1 DkS

1: Input: \( v = [v_1, \ldots, v_n] \)
2: for \( i = 1 : k \) do
3: \( s_i = \min_{|S|=k} |\sum_{i \in S} v_i| \)
4: if \( s_k = 0 \) then
5: Output: YES
6: else
7: Output: NO
8: end if
9: end for

Table 1. Datasets used in our experiments

<table>
<thead>
<tr>
<th>Data set</th>
<th>Nodes</th>
<th>Edges</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com-DBLP</td>
<td>317,080</td>
<td>1,049,866</td>
<td>DBLP collaboration network</td>
</tr>
<tr>
<td>com-LiveJournal</td>
<td>3,997,962</td>
<td>34,681,189</td>
<td>LiveJournal online social network</td>
</tr>
<tr>
<td>web-NotreDame</td>
<td>325,729</td>
<td>1,497,134</td>
<td>Web graph of Notre Dame</td>
</tr>
<tr>
<td>ego-Facebook</td>
<td>4,039</td>
<td>88,234</td>
<td>Social circles from Facebook (anonymized)</td>
</tr>
<tr>
<td>ca-AstroPh</td>
<td>18,772</td>
<td>396,160</td>
<td>Collaboration network of Arxiv Astro Physics</td>
</tr>
<tr>
<td>ca-HepPh</td>
<td>12,008</td>
<td>237,010</td>
<td>Collaboration network of Arxiv High Energy Physics</td>
</tr>
<tr>
<td>ca-CondMat</td>
<td>23,133</td>
<td>186,936</td>
<td>Collaboration network of Arxiv Condensed Matter</td>
</tr>
<tr>
<td>ca-GhQC</td>
<td>5,242</td>
<td>28,980</td>
<td>Collaboration network of Arxiv General Relativity</td>
</tr>
<tr>
<td>ca-HepTh</td>
<td>9,877</td>
<td>51,971</td>
<td>Collaboration network of Arxiv High Energy Physics Theory</td>
</tr>
<tr>
<td>roadNet-CA</td>
<td>58,228</td>
<td>214,078</td>
<td>Brightkite location based online social network</td>
</tr>
<tr>
<td>email-Enron</td>
<td>1,965,206</td>
<td>5,533,214</td>
<td>Road network of California</td>
</tr>
<tr>
<td>com-Orkut</td>
<td>36,692</td>
<td>367,662</td>
<td>Email communication network from Enron</td>
</tr>
<tr>
<td>flickr</td>
<td>105,938</td>
<td>2,316,948</td>
<td>Network of Flickr images sharing Common Metadata</td>
</tr>
<tr>
<td>FBMedium</td>
<td>63,731</td>
<td>817,090</td>
<td>Friendship Data of Facebook Users</td>
</tr>
</tbody>
</table>

References


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.