Finding Dense Subgraphs through Low-rank Approximations

Dimitris S. Papailiopoulos, Ioannis Mitliagkas,
Alexandros G. Dimakis, and Constantine Caramanis
The University of Texas at Austin

Abstract

Given a graph G and a parameter k, we are interested in finding the subgraph of size k with the highest number of edges. This is called the Densest-k-Subgraph (DkS) problem and is fundamental for many applications including community detection. We present a novel spectral algorithm with provable approximation guarantees for DkS. We establish data-dependent approximation bounds, related to the spectrum of the adjacency matrix. This is important, as existing results have only provided worst-case guarantees – far too conservative to provide guidance on real-world problems. Our algorithm exploits a combinatorial structure called the *Spannogram*, related to a low-rank approximation of the adjacency matrix.

Our algorithm is highly suitable for parallel implementation: we implement it in MapReduce, run it on a large computer cluster, and test it on artificial and real data sets. We show that it is possible to find dense subgraphs in massive graphs that have billions of edges.

1 Introduction

Given a graph G and a parameter k, we are interested in finding the induced subgraph of G on k vertices that contains the largest number of edges. This is the *Densest k-Subgraph* (DkS)– a fundamental problem in combinatorial optimization with applications in social sciences, communication networks, and biology [SHK+10, BKV12]. Algorithms for solving DkS are very useful for communities and spam detection and other graph analysis tasks that require well-connected subgraphs.

Unfortunately, DkS is a notoriously hard problem. It is easily seen that it is NP-hard since the maximum clique problem can be solved if one had access to a DkS solver. Khot [Kho04] showed 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 ϵ) for n the number of nodes, by Feige *et al.* [FPK01]. Recently, Bhaskara *et al.* [BCC+10] introduced an algorithm that has an approximation ratio of $n^{1/4+\epsilon}$ and runs in time $n^{O(1/\epsilon)}$. As we discuss in the related work section, there are several other algorithms for DkS and the related clique finding problem but no general approximation bounds.

Our Contributions. We present a novel algorithm for DkS with provable approximation guarantees. Our algorithm has three key features: first, it obtains *data-dependent* approximation bounds, related to the spectrum of the adjacency matrix. This is important, since, as it turns out, worst-case analysis may be too conservative for real-world graphs. Second, our spectral algorithm relies on manipulating candidate subgraphs obtained from vectors lying in a low-rank approximation to A. This is accomplished through an object called the *spannogram*, which we define below. While our overall algorithm is efficient, in the sense that it runs in polynomial time, in real-world examples that we

consider, it can be dramatically sped up by means of a vertex elimination step. The third and perhaps most important feature is that our algorithm is *fully-parallelizable*. This means that if we execute it on N cores, the CPU running time (which is the main bottleneck) is reduced by a factor of N. We illustrate this by implementing our algorithm on MapReduce and executing it on Elastic MapReduce (EMR) on Amazon. We were able to scale our implementation to run on thousands of mappers and reducers in parallel over 800 cores. Our implementation allows us to find large dense subgraphs in graphs with billions of edges.

To describe our theoretical contributions in more detail, we introduce some notation. We are given a graph $G(V, E)$ on n vertices and an integer k. We would like to find a subgraph supported on k vertices of G with the largest number of edges, or equivalently, with the largest average subgraph degree. Let δ_{OPT} denote the average degree of the densest subgraph. Our algorithm takes as input the graph, the preferred subgraph size k, and an accuracy parameter d, and outputs a subgraph on k vertices and of density δ_d. Our main theorem shows that δ_d is within an additive error from δ_{OPT}. Specifically

$$\delta_d \geq \delta_{OPT} - \gamma, \tag{1}$$

where the error term is bounded by:

$$\gamma \leq \lambda_{d+1} + |\lambda_n|. \tag{2}$$

Here, λ_{d+1} is the $d+1$ largest and λ_n the smallest eigenvalue of the adjacency matrix A. Here, the eigenvalue ordering retains the signs: λ_{d+1} can be negative (which helps our bound) and λ_n is the negative eigenvalue with largest magnitude. The accuracy parameter d controls the running time of our algorithm which is $O(n^{d+1})$. The parameter d also controls the rank of the approximation matrix we create for A before the combinatorial part of our algorithm. In all our experiments, d was set to 2 or 3 which shows that rank two, or three, approximations of the adjacency matrix can be used to find very good dense subgraphs. To understand the orders of magnitude, note that $0 < \delta_{OPT} \leq k - 1$ since the average degree of any subgraph on k vertices can be at most $k - 1$. The eigenvalues of the adjacency matrix lie in the interval $[-d_{max}, d_{max}]$ where d_{max} is the largest degree of G. Therefore, there are graphs where the error bound becomes trivial, when $\gamma > \delta_{OPT}$. However, if all the eigenvalues after the d-th have small magnitudes (this happens for example in some power-law graphs[][]), namely much smaller than the average degree of the dense subgraph we are looking for, our bound implies a tight approximation.

Guarantees on random graphs. We provide recovery guarantees on the famous hidden planted clique problem [AKS98, McS01, FR10, Ame11, DM13]. We generate a random graph $G(n, p = 1/2)$ and add a fully connected subgraph of size k. The goal is for an algorithm to discover this *planted clique* despite the "noise" created by the random edges. Clearly, if the size of the planted clique k is large, the problem is easier. In fact, when $k > c\sqrt{n}\log n$, a simple degree test will identify the hidden clique, due to standard binomial concentrations. For $k = C\sqrt{n}$ the problem of finding the hidden clique has been extensively studied. Several families of algorithms have been developed for this problem including combinatorial [FR10, DGGP10] and spectral algorithms [AKS98, McS01] as well as convex relaxations [FK00, Ame11, JMBD12] and message passing algorithms [DM13]. Some of the above approaches give explicit bounds on the constant C, with the best known being $1/\sqrt{e}$, due to a very recent message passing algorithm [DM13]. Within the family of spectral algorithms, the smallest identifiable clique size is $k = C\sqrt{n}$ for $C > 10$ due to [AKS98]. We show that our spectral algorithm can improve this to $C = 9.15$. Moreover, we show that for any $C > 4$ our algorithm recovers a constant fraction α of the clique with high probability, where $\alpha = \sqrt{1 - 4/C}$.

Elimination and MapReduce Implementation. The worst case running time of our algorithm is $O(n^{d+1})$, which is quite problematic for large graphs, even if $d = 2$. We argue that this worst-case running time metric is actually irrelevant to the scalability of the algorithm. The first reason

is the elimination step. As we explain below, one of the key novelties of our algorithm relates to combinatorial properties of curves that are generated from the top eigenvectors of A. We implement a vertex elimination step that takes advantage of combinatorial properties of these spectral curves and is able to eliminate most candidate subgraphs in all the tested data sets. This reduces the running time to $O(\hat{n}^{d+1})$ where, for all the real-world data sets we tried, $\hat{n}$ was smaller than 1000 for subgraph sizes $k \leq 100$, *even for data sets with tens of millions of nodes*. Unfortunately one cannot obtain general bounds on $\hat{n}$ without introducing extra assumptions on the data. In fact, we can construct artificial data sets where the vertex elimination step does not reduce the complexity; nevertheless, this is very different from what we observed in practice.

Related work. In the following we give a brief overview of the related algorithmic work.

DkS algorithms: There is a vast literature on algorithms for detecting communities and well-connected subgraphs [MBW10]. The literature includes optimization approaches [JMBD12], greedy schemes [FPK01] [RRT94], and the Truncated power (Tpower) method [YZ11]. We compare with various of these algorithms in our evaluation section.

The Spannogram framework: The idea behind the Spannogram framework is inspired by the related work of [APK11] and the consequent work in [PDK13]. There, the problem of sparse PCA is studied which is itself a quadratic form maximization, like the DkS; unlike the DkS problem it comes without the $0,1$ constraints on the variables, which require a fundamental modification to the theoretical analysis. The key idea of the spannogram framework is a low-rank spherical transformation of the problem using a low-dimensional auxiliary vector. This machinery was introduced by Karystinos et al. [KL10].

MapReduce algorithms for graphs: The design of MapReduce algorithms for massive graphs is a very active research area as Hadoop MapReduce becomes the *de-facto* standard for storing large data sets. The related paper by Bahmani *et al.* [BKV12] designs a novel MapReduce algorithm for finding dense subgraphs but without enforcing a specific subgraph size k. Surprisingly, without a restriction on k the densest subgraph problem becomes polynomially solvable and therefore the problem is fundamentally different.

2 Proposed Algorithm

We start by expressing the DkS problem as an optimization over the *sparse quadratic forms* of the adjacency matrix A. Consider a vector x of length n with elements that take values in $\{0, 1/\sqrt{k}\}$. Moreover, assume that x has k nonzero entries, i.e., $\|x\|_0 = k$. Let $G(x)$ denote the subgraph induced by the vertices indexed by the non-zero entries in x. Observe that the average degree of the subgraph indexed by the support of x is equal to

$$\delta(G(x)) = x^\intercal A x. \tag{3}$$

We refer to this quadratic objective function as the density for a fixed vector x. We can now express DkS as the solution to the following quadratic form maximization:

$$\mathcal{Q}: \qquad q_* = \arg\max_x x^\intercal A x, \text{ subject to: } x_i \in \left\{0, 1/\sqrt{k}\right\}, \|x\|_0 = k. \tag{4}$$

Here, the vector x ranges over all k-sparse vectors that have $1/\sqrt{k}$ in their non-zero entries. The supports of these vectors correspond to candidate k-subgraphs. Throughout the manuscript, we refer to this set of vectors as $\mathbb{S}_k$, that is

$$\mathbb{S}_k = \left\{ x \in \mathbb{R}^n : x_i = \left\{0, \frac{1}{\sqrt{k}}\right\}, \|x\|_0 = k \right\}. \tag{5}$$

We continue with a high-level description of our algorithm. For any given accuracy parameter d, let us assume that the first d largest eigenvalues of A are positive.[1] Our algorithm consists of the following steps:

Step 1: *Obtain A_d, a rank-d, positive-semidefinite (PSD) approximation of A.*

We create A_d, the rank-d approximation of A obtained by the first d eigenvectors corresponding to the d largest positive eigenvalues:

$$A_d = \sum_{i=1}^{d} \lambda_i v_i v_i^T, \tag{6}$$

where λ_i is the i-th largest eigenvalue of A and v_i the corresponding eigenvector.

Algorithm 1 Densest subgraph through a rank-d approximation

1: **Input:** k, d, A
2: $V_d = \left[\sqrt{\lambda_1} v_1 \dots \sqrt{\lambda_d} v_d\right] \leftarrow \text{SVD}(A, d)$
3: $\mathcal{S}_d \leftarrow \text{Spannogram}(k, p, V_d)$
4: $I_d = \arg\max_{I \in \mathcal{S}_d} \text{density}(G(I))$
5: $\delta_d = \text{density}(G(I_d))$
6: **Output:** $G(I_d)$, δ_d

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

For any matrix A, we can exhaustively solve DkS by checking all $\binom{n}{k}$ possible k-subgraphs and finding the one that maximizes $x^T A x$. Surprisingly, if we want to find the subgraph that maximizes $x^T A_d x$, only $O(n^d)$ candidate subgraphs need to be examined, in a similar manner to [APK11, PDK13].[2] Specifically, we show that a *set of candidate k-subgraphs* $\mathcal{S}_d = \{\mathcal{I}_1, \dots, \mathcal{I}_T\}$, where $\mathcal{I}_t$ is a subset of k vertices from $\{1, \dots, n\}$, contains the subgraph that maximizes $x^T A_d x$. We prove that the number of these candidate subgraphs is

$$|\mathcal{S}_d| \leq 2^d \binom{n}{d}, \tag{7}$$

which surprisingly does not depend on the subgraph size k and is polynomial in n for any constant d. This set of candidate subgraphs $\mathcal{S}_d$ is created in time $O(n^{d+1})$ by the *Spannogram algorithm* described subsequently.

Step 3: *Check each candidate subgraph from $\mathcal{S}_d$ on A.*

In this step, we check all the candidate subgraphs and output the densest subgraph in $\mathcal{S}_d$.

We refer to this approximate DkS solution as the *rank-d optimal k-subgraph*. In the next section, we describe our approximation guarantees, and then provide the details of the spannogram algorithm. The exact steps of our algorithm are given in the pseudo-code tables and the detailed description of the rank-d case can be found in the Appendix.

3 Approximation Guarantees

In this section, we state our dataset dependent guarantees of our DkS algorithm. The proofs can be found in the Appendix.

The desired k-subgraph is induced by the vertices indexed by the optimal k-sparse binary vector

$$q_* = \arg\max_{x \in \mathbb{S}_k} \quad x^T A x.$$

[1] This is usually the case for small d, however, if it does not hold, we can adjust the algorithm by shifting the spectrum of A by a constant identity matrix and then work on $\hat{A} = A + |\lambda_n| I_n$. Observe that for any unit norm vector x we have that $x^T \hat{A} x = x^T A x + |\lambda_n|$.

[2] Although we know that adjacency matrices cannot be PSD, the problem of solving the DkS optimization on the constant rank PSD matrix A_d is relevant to our approximation bounds.

We instead obtain the k-sparse, binary vector q_d which gives an objective

$$\delta_d = q_d^T A q_d = \max_{\mathcal{I} \in \mathcal{S}_d} \delta(A_{\mathcal{I}}).$$

We would like to bound the error of our approximation:

$$\gamma = q_*^T A q_* - q_d^T A q_d. \tag{8}$$

Let us define the optimization on the constant rank PSD matrix A_d

$$\mathcal{Q}_d : \quad q_d = \arg\max_{x \in \mathbb{S}_k} x^\intercal A_d x \tag{9}$$

and let $\delta_d = q_d^T A q$. The following theorem bounds the performance of our algorithm with respect to the optimal solution using q_d, that is part of the candidate solutions that the Spannogram routine outputs: this means that Algorithm 1 outputs a solution that is as good as q_d.

Theorem 1. *The optimization problem $\mathcal{Q}_d$, for fixed d, can be solved in time $O(n^{d+1})$. Let q_d be the optimal solution of $\mathcal{Q}_d$. Then, $G(q_d)$ is a subgraph with average degree $\delta_d = q_d^\intercal A q_d$ such that*

$$\delta_d \geq \delta_{OPT} - \lambda_{d+1} - |\lambda_n|. \tag{10}$$

We would like to note that we can actually show a potentially stronger bound that replaces the eigenvalues of A with "sparse" eigenvalues: we can replace λ_{d+1} with the maximum k-sparse eigenvalue of $A - A_d$ and λ_n with the smallest (signed) k-sparse eigenvalue of A. However these are hard to compute or tightly bound. We use the eigenvalues of A to make our our bounds tractable.

Using the above approximation bounds and given a family of matrices that have a specific spectral profile, we can calculate graph specific guarantees. For example, for a random $G(n, p = 1/2)$ graph with a planted clique of size k, denoted by $G(n, 1/2, k)$, we obtain the following result:

Theorem 2. *Consider a $G(n, 1/2, k)$ planted clique random graph. For any $k > C\sqrt{n}$, with $C > 4$, Algorithm 1, for $d = 2$, recovers a dense subgraph that overlaps with the clique on at least αk vertices, $\alpha = \sqrt{1 - 4/C}$, with high probability. Further, for any $k \geq 9.15\sqrt{n}$ a simple degree test allows the reconstruction of the full hidden clique based on q_2.*

4 Finding the Candidate Subgraphs

In this section, we describe how the spannogram routine builds the set of candidate subgraphs $\mathcal{S}_d$. We show that this set has tractable size. 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 rank 1 case where $d = 1$. For this case we use the rank-1 approximation to A that is equal to

$$A_1 = \lambda_1 v_1 v_1^T$$

and solve (4) on A_1:

$$\max_{x \in \mathbb{S}_k} x^T A_1 x = \lambda_1 \cdot \max_{x \in \mathbb{S}_k} \left(v_1^T x \right)^2 = \lambda_1 \cdot \max_{x \in \mathbb{S}_k} \left| \sum_{i=1}^{n} v_{1,i} x_i \right|^2, \tag{11}$$

where, as we discussed, $\mathbb{S}_k$ is the set of all k sparse vectors with non-zero entries equal to $1/\sqrt{k}$. We can equivalently think of x as being a vector with exactly k ones and zeros elsewhere. In this problem, we are trying to find which such $(0,1)$ vector maximizes the absolute value of the inner product with a given vector v_1. In other words, which k entries of v_1 should we choose so as to maximize the magnitude of the sum? It is not hard to see that there are two choices: make x choose the k largest positive entries, or the k largest (in absolute value) negative entries of v_1. In general, (even if the vector does not have k entries for both signs), one should sort the signed entries of v_1 and keep two supports: the ones that indexes the k largest entries and the one indexing the k smallest entries of v_1. Therefore, for the rank $d = 1$ problem, there are two candidate subgraphs corresponding to the top k and the bottom k elements of the sorted eigenvector v_1. In our notation, the set $\mathcal{I}_{\overline{k}}(v_1)$ is the support of the top k entries of v_1 and $\mathcal{I}_{\underline{k}}(v_1)$ the support of the bottom k entries. The set of candidate subgraphs for rank 1 is $\mathcal{S}_1 = \left\{\mathcal{I}_{\overline{k}}(v_1), \mathcal{I}_{\underline{k}}(v_1)\right\}$.

Rank-2 case. Now we describe how the Spannogram algorithm builds $\mathcal{S}_2$. The $d = 2$ case is the first nontrivial d which exhibits the details of the Spannogram algorithm. Here, we work on the rank 2 matrix

$$A_2 = \lambda_1 v_1 v_1^T + \lambda_2 v_2 v_2^T = V_2 V_2^T, \tag{12}$$

where $V_2 = \left[\sqrt{\lambda_1} \cdot v_1 \;\; \sqrt{\lambda_2} \cdot v_2\right]$. We can now rewrite (4) on A_2 as

$$\max_{x \in \mathbb{S}_k} x^T A_2 x = \max_{x \in \mathbb{S}_k} \left\| V_2^T x \right\|_2^2. \tag{13}$$

For the rank-1 case, we could write the quadratic form maximization as a simple maximization of an inner product: $\max_{x \in \mathbb{S}_k} x^T A_1 x = \max_{x \in \mathbb{S}_k} \left(v_1^T x\right)^2$. Similarly, as we show in the following, in the rank-2 case we can write

$$\max_{x \in \mathbb{S}_k} x^T A_2 x = \max_{x \in \mathbb{S}_k} \left(v_c^T x\right)^2,$$

for some *specific* vector v_c in the span of the eigenvectors v_1, v_2; this proves very helpful in solving the problem efficiently.

To see this, let c be a 2×1 unit length vector, i.e., $\|c\|_2 = 1$. Using the Cauchy-Schwartz inequality for the inner product of c and $V_2^T x$, we obtain $\left(c^T V_2^T x\right)^2 \leq \|V_2^T x\|_2^2$, where equality holds, if and only if, c is co-linear to $V_2^T x$. By the previous fact, we have a variational characterization of the ℓ_2-norm:

$$\|V_2^T x\|_2^2 = \max_{\|c\|_2 = 1} \left(c^T V_2^T x\right)^2. \tag{14}$$

We can use (14) to rewrite (13) as:

$$\max_{x \in \mathbb{S}_k} \max_{\|c\|_2 = 1} \left(c^T V_2^T x\right)^2 = \max_{\|c\|_2 = 1} \max_{x \in \mathbb{S}_k} \left(v_c^T x\right)^2, \tag{15}$$

where $v_c = V_2 c$. We would like to note two important facts here. The first is that for all unit vectors c, $v_c = V_2 c$ generates all vectors in the span of V_2 (up to scaling factors). The second fact is that if we fix c, then the maximization $\max_{x \in \mathbb{S}_k} \left(v_c^T x\right)^2$ is a rank-1 instance, similar to (11). Therefore, for each fixed unit vector c there are two candidate subgraphs (denote it by $\mathcal{S}_1(V_2 c) = \left\{\mathcal{I}_{\overline{k}}(V_2 c), \mathcal{I}_{\underline{k}}(V_2 c)\right\}$) to be added in $\mathcal{S}_2$. If we could collect all possible candidate subgraphs for each vector $v_c = V_2 c$ in the span of v_1, v_2, then we could solve exactly the k-densest subgraph problem on A_2: we would simply test all locally optimal solutions obtained from each support in $\mathcal{S}_2$ and keep the one with the highest density. The issue is that there are infinitely many v_c vectors to check. Naively, one could think that

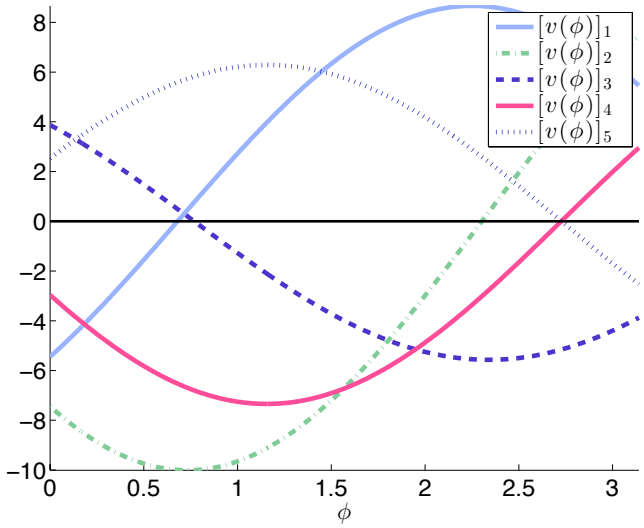


Figure 1: A rank-2 spannogram for $n = 5$. Notice that every two curves intersect in exactly two points. The curve intersection points define intervals where the candidate subgraphs can change.

all possible $\binom{n}{k}$ subgraphs could appear for some v_c vector. The key combinatorial fact is that if a vector v_c lives in a two dimensional subspace, there are tremendously fewer possible subgraphs:

$$|\mathcal{S}_2| \leq 2\binom{n}{2} + 2.$$

The bound on the number of candidate subgraphs for rank $d = 2$ is a special case of our general bound that is shown in the appendix. We prove the special case to simplify the presentation. We use a transformation of our problem space into spherical variables, as in [KL10] and [APK11], that enable us to visualize the 2-dimensional span of V_2. For the rank-2 case, we use a single phase variable $\phi \in \Phi = [0,\ \pi)$ and use it to rewrite c, without loss of generality, as

$$c = \begin{bmatrix} \sin\phi \\ \cos\phi \end{bmatrix},$$

which is again unit norm and for all ϕ it scans all 2×1 unit vectors. Under this characterization, we can express v_c in terms of ϕ as

$$v(\phi) = V_2 c = \sin\phi \cdot \sqrt{\lambda_1} v_1 + \cos\phi \cdot \sqrt{\lambda_2} v_2. \tag{16}$$

Observe that each element of $v(\phi)$ is a continuous *spectral curve* in ϕ:

$$[v(\phi)]_i = \left[\sqrt{\lambda_1} v_1\right]_i \sin(\phi) + \left[\sqrt{\lambda_2} v_2\right]_2 \cos(\phi), \tag{17}$$

for all $i = 1, \ldots, n$, where $[x]_i$ denotes the i-th element of the vector x. Therefore, the support sets of the k largest and smallest elements of $v(\phi)$ (*i.e.*, $\mathcal{I}_{\overline{k}}(v(\phi)), \mathcal{I}_{\underline{k}}(v(\phi))$) are themselves a function of ϕ.

The Spannogram. In Fig. 1, we draw an example plot of five curves $[v(\phi)]_i$, $i = 1, \ldots, 5$, from a randomly generated matrix V_2. We call this a *spannogram*, because at each ϕ, the values of curves correspond to the values of the elements in the column span of V_2. From the spannogram in Fig. 1, we can see that the continuity of the curves implies a local invariance property of the candidate

subgraphs $\mathcal{I}(v(\phi))$, around a given ϕ. In fact, a candidate subgraph $\mathcal{I}_k(v(\phi))$ changes *if and only if* two curves cross, *i.e.*, the respective order of two elements $[v(\phi)]_i$ and $[v(\phi)]_j$ changes.

Now observe that there are n curves and each pair intersects at exactly one point in the Φ domain[3]. Therefore, there are exactly $\binom{n}{2}$ intersection points. These $\binom{n}{2}$ intersection points define $\binom{n}{2}+1$ intervals within which the candidate subgraph set $S_1(v(\phi))$ remains the same. That is, each interval defined by two intersection points corresponds to exactly two candidate subgraphs (the indices of the largest k and smallest k curves of v_cs in this interal). Therefore there are in total $2\binom{n}{2}+2$ candidate subgraphs. To identify them, we simply need to compute all intersection points. The curve intersection points can be easily computed by solving linear equations of the form

$$[v(\phi)]_i = [v(\phi)]_j \Rightarrow (e_i^T V_2 - e_j^T V_2)c = 0_{2\times 1} \tag{18}$$

for the unknown c and all pairs $i, j \in [n]$, for $i \neq j$, where e_i is the i-th column of the $n \times n$ identity matrix I_n.

Once we generate all intersection vectors $c_{i,j}$, for all pairs of i and j, we can produce the candidate subgraphs that lie within the intervals specified by the intersection points. On each intersection point, i.e., for each intersection vector $c_{i,j}$, we calculate $S_1(i,j) = S_1(V_2 c_{i,j})$. Observe that each intersection point is associated with 2 intervals, the "left" adjacent one and "right" adacent one. This means that the vector $V_2 c_{i,j}$ has two curves with equal value. To ensure that all candidates subgraphs are included in S_2, i.e., that all intervals are visited, we will produce two candidate sets for a single vector $c_{i,j}$: one where curve i is larger than j and vice versa. Lets call these two sets $S_1(i,j)$ and $S_1(j,i)$. This means that for each vectors $V_2 c_{i,j}$, we generate two sets of candidate subgraph sets $S_1(v_{i,j})$ and $S_1(v_{j,i})$: in total 4 candidate subgraphs. With this method we compute all possible $\mathcal{I}_k(V_2 c)$ rank-2 subgraph candidates and we create the set of candidate subgraphs $\mathcal{S}_2$ where $|\mathcal{S}_2| \leq 2\binom{n}{2}+1 = O(n^2)$..

Observe that on each intersection point we need to compute the top-k and bottom-k indices of $V_2 c_{i,j}$. This can be done in time $O(n)$ using a partial sorting and selection algorithm [CLRS01]. Since we perform this sorting a total of $2\binom{n}{2}+2$ times, the total complexity of our rank-2 algorithm is $O(n^3)$.

Remark 1. *Observe that even if we omit half the Φ angle domain $[\pi, 2\pi)$, we skip no possible set $S_1(V_2 c)$. This is due to the fact that $\sin(\phi+\pi) = -\sin(\phi)$ and $\cos(\phi+\pi) = -\cos(\phi)$, which implies $v(\phi) = -v(\phi+\pi)$ and hence $S_1(v(\phi)) \equiv S_1(v(\phi+\pi))$. Therefore, collecting all sets $S_1(v(\phi))$, for $\phi \in [0,\pi)$, is equivalent to collecting all sets $S_1(v(\phi))$, for $\phi \in [\pi, 2\pi)$, and thus sufficient to collect all possible candidate subgraphs.*

General Rank-d case. The algorithm generalizes from the rank 2 case into an arbitrary dimension d and can be found in the Appendix. The spherical coordinates are in d dimensions and the general bound we prove is that if v_c is in a d dimensional subspace, there are at most $2^d\binom{n}{d}$ candidate subgraphs.

Elimination pre-processing: It is easy to verify that the amplitude of the curve corresponding to coordinate j is

$$\|[V_2]_{j,:}\|_2 = \sqrt{\sum_{i=1}^{r} (\sqrt{\lambda_i}[v_i]_j)^2}. \tag{19}$$

As discussed, within each interval, we create two candidate subgraphs from the largest and smallest k curves. Therefore, if a curve has very small amplitude, it may never appear in the largest or smallest k sets. This means that the vertex corresponding to that coordinate j never appears in a candidate subgraph and can be eliminated. We use a quick algorithm that eliminates coordinates without searching through all intersection points: we first sort the curves by their magnitude and

[3]Note that here we assume that the curves are in *general position*. This is without loss of generality because we can always add small perturbations to the curves. The details of this argument are in the appendix.

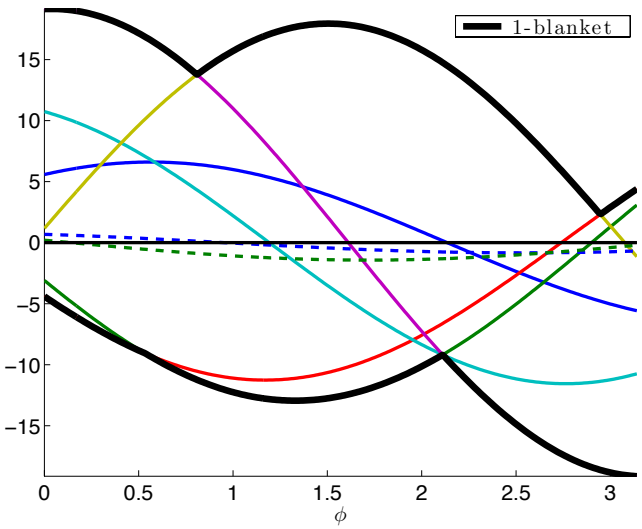


Figure 2: The concept of our elimination in a rank-2 spannogram for $k = 1$. Notice that here we plit the 1-blanket boundary, which is constructed by the intersection points that "mark" the generation of a new 1-largest/smallest vertex set. Observe how the curves marked with dashed lines have amplitude less than the minimum amplitude points of the 1-blanket. This means that the corresponding vertices can be safely eliminated, since they can never appear in a top-1/bottom-1 set.

start constructing a list of the intersection points of the k largest magnitude curves. We call this a k-blanket and it is a bound for curves in the spherical domain. For a new curve, if its amplitude is smaller than all the intersection points, it means that it provably cannot appear in a candidate subgraph. Further, all other curves with smaller amplitudes can also be eliminated automatically. We describe the details of our elimination algorithm in the Appendix, a sketch of the idea can be found in Fig. 2. We note that this step is remarkably effective in real data sets and decimates the number of nodes to be examined. It is a generalization of the heuristic that eliminates nodes with small degrees since they are unlikely to participate in dense subgraphs.

5 MapReduce Algorithm

Here we describe our MapReduce design for the rank-2 Spannogram algorithm. We assume that the reader is familiar with the MapReduce paradigm (e.g. see [MM, BKV12]).

The Hadoop MapReduce infrastructure stores the input graph as a distributed file spread across multiple machines. The MapReduce infrastructure provides a tuple streaming abstraction where each `map` and `reduce` function receive and emit tuples that $(key, value)$ pairs. The role of the keys is to ensure information aggregation: all the tuples with the same key are processed by the same reducer.

How to parallelize. Depending on the performance bottleneck (RAM/Network/CPU/DiskIO), different map/reduce functions can be more appropriate. In our problem, we can divide the construction of the $O(n^2)$ sets to $O(n^\alpha)$ reducers ($0 \leq \alpha \leq 2$); each of them would compute $O(n^{2-\alpha})$ candidate subgraphs. Note here that n is the number of vertices that remain after the elimination preprocessing and in all our experiments was smaller than 10000.

The total communication cost for such a parallelization scheme is $O(n^{1+\alpha})$: n^α reducers need to have access to the entire V_2. Moreover, the total computation cost for each reducer would be $O(n^{3-\alpha})$ ($O(n)$ per candidate subgraph).

For example, we could parallelize the Spannogram in a way that *each* candidate subgraph is produced by a single machine. However, this would require enormous communication in the orders of n^3 (n rows of V_2 communicated to n^2 machines). There is a clear trade-off between the cpu-time spent by each reducer and the cost to communicate V_2. In our case we found that communication was the main bottleneck and set $\alpha = 1$.

Algorithm 2 `MR_Spannogram`(V_2)

Map$(\{[V_2]_{j,:}, j\})$:
for $i = 1 : n$ **do**
  emit: $\langle i, \{[V_2]_{j,1}, [V_2]_{j,2}, j\}\rangle$
end for

Reduce$_i(\langle i, \{[V_2]_{j,1}, [V_2]_{1,2}, j\}\rangle, \forall j)$:
`metric` $= 0$, `optS` $= \emptyset$
for each $j \geq i+1$ **do**
  $x = \begin{bmatrix} [V_2]_{i,1} - [V_2]_{j,1} \\ [V_2]_{i,2} - [V_2]_{j,2} \end{bmatrix}$
  $c_{i,j} = \begin{bmatrix} x_2 \\ -x_1 \end{bmatrix}$
  $y = Vx$
  $I_{\text{top}} =$`top_k_set`(Vx)
  $I_{\text{bottom}} =$`top_k_set`$(-Vx)$
  `optS` $= \underset{I \in \{\texttt{optS}, I_{\text{top}}, I_{\text{bottom}}\}}{\arg\max} \left\| \sum_{i \in I} V_{i,:} \right\|^2$
  `metric` $= \left\| \sum_{i \in \texttt{optS}} V_{i,:} \right\|^2$
end for
emit: $\langle i,$ `optS`, `metric`$\rangle$

The Spannogram Map function. The mapper takes as inputs tuples that are the rows of the V_2 matrix. It then emits each row n times: for each row of V_2 the mapper emits the row itself and its index i as a value, with keys being set to all possible values of i. For example, assume that a mapper receives as input the ith row of V_2, call it v_i. Then, it will emit $\langle 1, v_i, i\rangle$,$\langle 2, v_i, i\rangle$, ..., $\langle n, v_i, i\rangle$. This will force each reducer to receive the entire V_2 matrix; moreover, due to the addition of the row index for each key, value pair, the reducer knows which row is which. This step costs $O(n^2)$ in communication.

The Spannogram Reduce function. The ith reducer routine, will receive as inputs $\langle i, v_1, 1\rangle$, ..., $\langle i, v_n, n\rangle$. It will use these rows of V_2 to compute the intersections of curve i with $n - i$ curves $i+1, i+2, \ldots, n$. This is done by calculating the nullspace of $v_i - v_j$ as we show in our theoretical analysis. Here, since for the $d = 2$ case the nullspace is just a vector $c_{i,j}$ othogonal to $y = v_i - v_i$, we just set $c_{i,j} = [y_2, -y_1]^T$. Then, for each intersection vector c_{ij}, we compute the two locally optimal candidate graphs. These are indexed by the top and bottom elements of the n dimensional vector $V_2 c$.[4] At each candidate calculation, we check if it offers a better metric compared to the previous solution on that reducer. Observe that here, we do not calculate the density, but rather a "rank-2 approximation" of it ($x^T V_2 V_2^T x$); this complies exactly with the performance bounds derived above. This is done to avoid communicating the entire A matrix to all reducers. If we would like to communicate A to each reducer then an enormous cost arises, which in practice is proved to be unbearable with respect to running times.

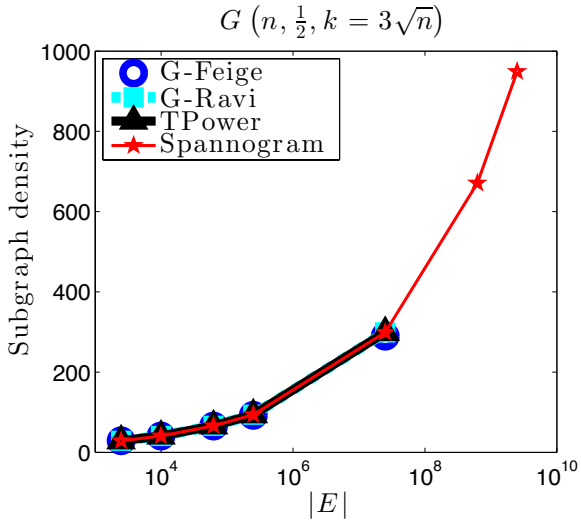
After all n reducers emit their locally optimal candidate k-subgraphs, the rank-2 optimal subgraph can be obtained by simply keeping the one with the highest metric, in nearly linear time $O(n \log n)$.

6 Experimental Evaluation

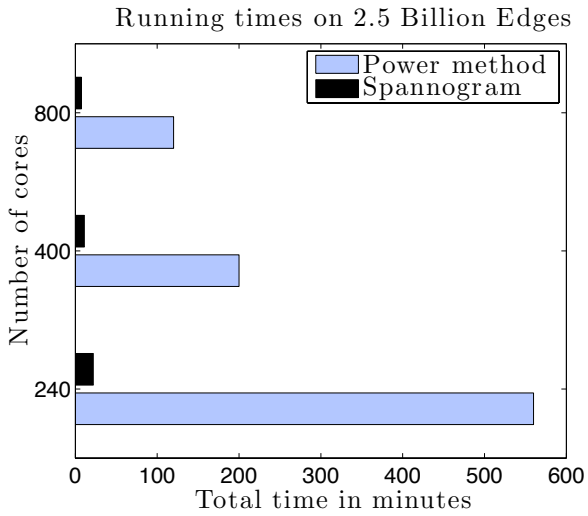
We experimentally evaluate the performance of our algorithm and compare it to the truncated power method (TPower) of [YZ11], a greedy algorithm by Feige et al. [FPK01] (G-Feige) and another greedy algorithm by Ravi et al. [RRT94] (G-Ravi).[5]

[4] Observe that here, for simplicity, we ignore the subgraphs introduced by switching the sorting of the i and j entry. This only incurs a cost of $\frac{2}{k}$ in our metric, that is vanishingly small compared to k, when k is large.

[5] The implementation of these algorithms can be found in `https://sites.google.com/site/xtyuan1980/publications`



(a) We plot the densities of the recovered subgraphs recovered by all algorithms v.s. the expected number of edges in the graph.



(b) We compare the running time of the Spannogram to the time it took power iteration to estimate the two first eigenvectors of A.

Figure 3: Planted clique experiments for random graphs.

We performed experiments on artificial dense subgraphs and also massive real graphs from multiple applications. In all the experiments we compare the density of the subgraph obtained by the Spannogram to the density of the output subgraphs given by the other algorithms. We further provide plots with the number of candidate subgraphs that our algorithm needs to examine (equal to $\binom{\hat{n}}{d}$), posterior to the elimination step. This is a good measure of the complexity of the Spannogram on the given data set.

Planted clique. We start with a synthetic experiment: we seek to recover a planted clique from a random $G\left(n, \frac{1}{2}, k\right)$ graph. We scale our randomized experiments from $n = 100$ up to $100,000$. In all experiments we set the size of the planted clique to be $k = 3 \cdot \sqrt{n}$.

This clique size is below our theoretical guarantees for full recovery, however the experiments manifest the tightness of our approximation even for this scenario. We plot the performance of our algorithm in comparison to TPower, and the two greedy algorithms. In all our experiments, G-Ravi, the TPower, and the Spannogram were successfully recovering the hidden clique.

However, as can be seen in Fig. 2, the Spannogram algorithm was the only one that was able to scale for $n = 100$k. This is because the Spannogram was implemented in MapReduce and was executed on 800 cores. If the adjacency matrix of the input graph can fit in main memory, this is not needed: all algorithms can be executed in a single server and Tpower, G-Feige and G-Ravi were faster than the proposed Spannogram algorithm (All running times took up to a few minutes). For $n \geq 30.000$, however, the graph does does not fit in RAM since it is in the order of multiple GB. Note that for $n = 100.000$, the $G\left(n, \frac{1}{2}, k\right)$ graph has approximately 2.5 billion edges and cannot fit in main memory.

The running times of our experiments over MapReduce are shown in Fig. 2(b). As can be seen, the main bottleneck is the computation of the first two eigenvectors which was done by repeating the power method for very few $(3-6)$ iterations. The Spannogram algorithm was significantly faster and the benefits of parallelization are clear since the Spannogram is CPU intensive. We note that other low-rank approximations of A can be used as inputs to the Spannogram and we are still investigating this issue.

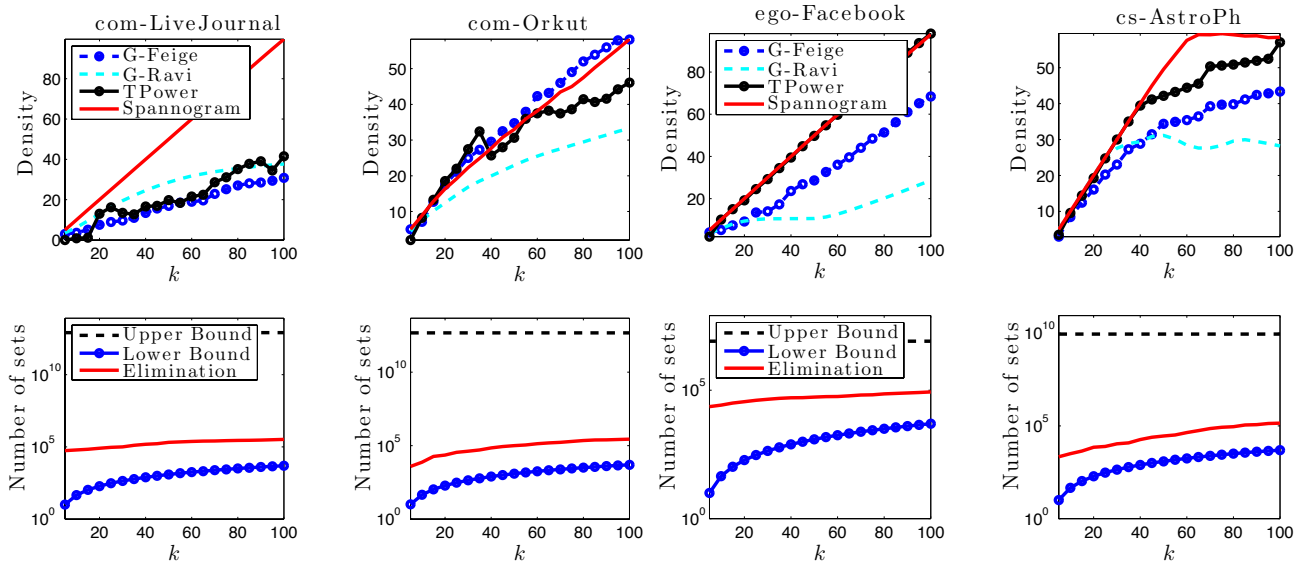


Figure 4: Subgraph density and number of candidate subgraphs $\binom{\tilde{n}}{d}$ vs. k. A lower bound for the elimination algorithm is also plotted and is $\binom{k}{d}$.

In principle, the other algorithms could be also implemented over MapReduce but that requires non-trival distributed algorithm design. As is well-known (*e.g.*, [MM]) implementing iterative machine learning algorithms over MapReduce can be a significant problem 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. We also run our algorithm in various large scale real data sets that we summarize in the following table; the findings are reported in Fig. 4. The data sets can be found at `http://snap.stanford.edu/data/`.

Data set	$\|V\|$	$\|E\|$	description
ego-Facebook	4,039	88,234	Social circles from Facebook
ca-AstroPh	18,772	396,160	Collaboration network of Arxiv Astro Physics
com-LiveJournal	3,997,962	34,681,189	LiveJournal online social network
com-Orkut	3,072,441	117,185,083	Orkut online social network

In our experiments, we run our algorithm for values of $k = 2, \ldots, 100$. For these subgraph sizes we observe performance at least similar to the prior state of the art. In some datasets, we were able to identify subgraphs with much higher density compared to what the other algorithms were reporting. We also observed several cases where Tpower and G-Feige found denser subgraphs than Spannogram. As we mentioned, our algorithm is highly scalable due to the elimination step. For example, on some large graphs (livejournal, orkut), we observe that the number of candidate subgraphs that the spannogram produces after the elimination can be even 10 orders of magnitude smaller than the worst case bound provided by our analysis. Note also that if the size of the desired subgraph k is large, elimination becomes harder. We note that for high values of k competing algorithms were significantly faster. We conclude that our algorithm is highly suitable for finding dense subgraphs in massive graphs if the size k of the desired subgraphs is not too large (e.g. below thousand nodes).

References

[AKS98] Noga Alon, Michael Krivelevich, and Benny Sudakov. Finding a large hidden clique in a random graph. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, page 594–598, 1998.

[Ame11] Brendan PW Ames. *Convex relaxation for the planted clique, biclique, and clustering problems*. PhD thesis, University of Waterloo, 2011.

[APK11] Megasthenis Asteris, Dimitris S Papailiopoulos, and George N Karystinos. Sparse principal component of a rank-deficient matrix. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pages 673–677. IEEE, 2011.

[BCC+10] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities: an o (n 1/4) approximation for densest k-subgraph. In *Proceedings of the 42nd ACM symposium on Theory of computing*, pages 201–210. ACM, 2010.

[BKV12] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5):454–465, 2012.

[CL06] Fan Rong K Chung and Linyuan Lu. *Complex graphs and networks*. Number 107. American Mathematical Soc., 2006.

[CLRS01] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2001.

[DGGP10] Yael Dekel, Ori Gurel-Gurevich, and Yuval Peres. Finding hidden cliques in linear time with high probability. *arXiv preprint arXiv:1010.2997*, 2010.

[DM13] Yash Deshpande and Andrea Montanari. Finding hidden cliques of size $\sqrt{N/e}$ in nearly linear time. 2013.

[FK00] Uriel Feige and Robert Krauthgamer. Finding and certifying a large hidden clique in a semirandom graph. *Random Structures and Algorithms*, 16(2):195–208, 2000.

[FPK01] Uriel Feige, David Peleg, and Guy Kortsarz. The dense k-subgraph problem. *Algorithmica*, 29(3):410–421, 2001.

[FR10] Uriel Feige and Dorit Ron. Finding hidden cliques in linear time. *DMTCS Proceedings*, (01):189–204, 2010.

[JMBD12] Vinay Jethava, Anders Martinsson, Chiranjib Bhattacharyya, and Devdatt Dubhashi. The lovasz theta function, svms and finding large dense subgraphs. In *NIPS*, 2012.

[Kho04] Subhash Khot. Ruling out ptas for graph min-bisection, densest subgraph and bipartite clique. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 136–145. IEEE, 2004.

[KL10] George N Karystinos and Athanasios P Liavas. Efficient computation of the binary vector that maximizes a rank-deficient quadratic form. *Information Theory, IEEE Transactions on*, 56(7):3581–3593, 2010.

[MBW10] B Miller, N Bliss, and P Wolfe. Subgraph detection using eigenvector l1 norms. In *NIPS*, volume 2010, pages 1633–1641, 2010.

[McS01] Frank McSherry. Spectral partitioning of random graphs. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 529–537. IEEE, 2001.

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

[MU05] M. Mitzenmacher and E. Upfal. Probability and computing: Randomized algorithms and probabilistic analysis. 2005.

[PDK13] Dimitris S Papailiopoulos, Alexandros G Dimakis, and Stavros Korokythakis. Sparse pca through low-rank approximations. *arXiv preprint arXiv:1303.0551*, 2013.

[RRT94] Sekharipuram S Ravi, Daniel J Rosenkrantz, and Giri K Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.

[SHK+10] Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. Dense subgraphs with restrictions and applications to gene annotation graphs. In *Research in Computational Molecular Biology*, pages 456–472. Springer, 2010.

[YZ11] Xiao-Tong Yuan and Tong Zhang. Truncated power method for sparse eigenvalue problems. *arXiv preprint arXiv:1112.2679*, 2011.

7 Appendix

7.1 Rank-d General Algorithm pseudo-code

In the geenral rank-d case, we use the best rank-d PSD approximation of A that is equal to $A_d = V_d V_d^T$ where $V_d = \left[\sqrt{\lambda_1}\cdot v_1 \ \ \ldots \ \ \sqrt{\lambda_d}\cdot v_d\right]$. The DkS on A_d is written as

$$\max_{x\in\mathbb{S}_k} x^T A_d x = \max_{x\in\mathbb{S}_k} \left\| V_d^T x \right\|_2^2 = \max_{x\in\mathbb{S}_k, \|c\|_2=1} \left(c^T V_d^T x\right)^2 = \max_{x\in\mathbb{S}_k, \|c\|_2=1} \left(v_c^T x\right)^2, \tag{20}$$

where $v_c = V_d c$. Following the work of [APK11] and [KL10], we introduce $d-1$ angles $\varphi = [\phi_1, \ldots, \phi_{d-1}] \in \Phi^{d-1}$ and use them to introduce the auxiliary spherical vector

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

In this case, an element of $v_c = V_d c$ is a continuous d-dimensional function on $d-1$ variables $\varphi = [\phi_1, \ldots, \phi_{d-1}]$, i.e., it is a d-dimensional hypersurface:

$$[v(\varphi)]_i = \sin\phi_1 \cdot [\sqrt{\lambda_1} v_1]_i + \cos\phi_1 \sin\phi_2 \cdot [\sqrt{\lambda_2} v_2]_i + \ldots + \cos\phi_1 \cos\phi_2 \ldots \cos\phi_{d-1} [\sqrt{\lambda_d} v_d]_i.$$

As with the rank-2 case, here for each vector c we can calculate the locally optimal sets $S_1(V_d c)$. All we need to do then, is find all possible subgraphs induced by the different vectors. In this case, the curves are d-dimensional, and every d of them intersect in a single point. These points define cells, wherein the candidate subgraphs remain the same.

A difference with the rank-2 case, is that here, the solution to the equation $[v(\varphi)]_i = [v(\varphi)]_j$ is not a single point (i.e., a single solution vector c), but a $(d-1)$ dimensional space of solutions. Let

$$\mathcal{X}_{i,j} = \left\{ v(\varphi), \varphi \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]^{d-1} : [v(\varphi)]_i = [v(\varphi)]_j \right\}$$

be the the set of all $v(\varphi)$ vectors where $[v(\varphi)]_i = [v(\varphi)]_j$ in the φ domain. Then, the elements of the set $\mathcal{X}_{i,j}$ are exactly the points of the intersection between hypersurfaces i and j.

Since locally optimal support sets change only with the local sorting changes, the intersection points defined by all $\mathcal{X}_{i,j}$ sets are the only points of interest.

Identifying all intersection vectors. We will now work on the $(d-1)$-dimensional space $\mathcal{X}_{i,j}$. For the vectors in this space, there are again critical φ points, where both the i and j coordinates become members of the top- or bottom-k set. These are the points when locally optimal support sets change. This happens when the i and j coordinates become equal with another l-th coordinate of $v(\phi)$. This new space of $v(\phi)$ vectors where coordinates i,j,and l are equal will be the set $\mathcal{X}_{i,j,k}$, this will now be a $(d-2)$-dimensional subspace. We can proceed in that manner until we reach the single-element set $\mathcal{X}_{i_1,i_2,\ldots,i_d}$ where there is only one vector $v(\phi)$ that satisfies

$$[v(\varphi)]_{i_1} = [v(\varphi)]_{i_2} = \ldots = [v(\varphi)]_{i_d}. \tag{21}$$

Therefore, the total number of intersection vectors c is equal to $\binom{n}{d}$. These are the only vectors that need to be checked, since they are the potential ones where d-tuples of elements become part of the top k support set.

Algorithm 3 Rank-d Spannogram

Input: k, p, V_d
Initialize $\mathcal{S}_d \leftarrow \emptyset$
for all $\binom{n}{d}$ subsets $(i_1, \ldots, i_d)$ from $\{1, \ldots, n\}$ **do**
  $c \leftarrow \text{nullspace}\left(\begin{bmatrix} e_{i_1}^T - e_{i_2}^T \\ \vdots \\ e_{i_1}^T - e_{i_d}^T \end{bmatrix} V_d \right)$
  $\mathcal{I} \leftarrow$ {indices of the k-top elements of Vc} $\cup$ {indices of the k-top elements of $-Vc$}
  $l \leftarrow 1$
  $\mathcal{J}_1 \leftarrow \mathcal{I}_{1:k}$
  $r \leftarrow |\mathcal{J}_1 \cap (i_1, \ldots, i_d)|$
  if $r < d$ **then**
    for all r-subsets $\mathcal{M}$ from $(i_1, \ldots, i_d)$ **do**
      $l \leftarrow l + 1$
      $\mathcal{J}_l \leftarrow \mathcal{I}_{1:k-r} \cup \mathcal{M}$
    end for
  end if
  $\mathcal{S}_d \leftarrow \mathcal{S}_d \cup \mathcal{J}_1 \ldots \cup \mathcal{J}_l$.
end for
Output: $\mathcal{S}_d$.

Building $\mathcal{S}_d$. This part of our algorithm and its analysis is similar to the work in [APK11, PDK13]. To visit all possible sorting candidates, we now have to check the intersection points, which are obtained by solving the system of $d-1$, linear in $c(\phi_{1:d-1})$, equations

$$\begin{aligned} &[v(\varphi)]_{i_1} = [v(\varphi)]_{i_2} = \ldots = [v(\varphi)]_{i_d} \\ \Leftrightarrow &[v(\varphi)]_{i_1} = [v(\varphi)]_{i_2}, \ldots, [v(\varphi)]_{i_1} = [v(\varphi)]_{i_d}. \end{aligned} \tag{22}$$

Then, we can rewrite the above as

$$\begin{bmatrix} e_{i_1}^T - e_{i_2}^T \\ \vdots \\ e_{i_1}^T - e_{i_d}^T \end{bmatrix} Vc = 0_{(d-1)\times 1} \tag{23}$$

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

To explore all possible candidate vectors, we need to compute all possible $\binom{n}{d}$ solution intersection vectors c. On each intersection vector we need to compute the locally optimal support set $\mathcal{I}_k(v_c)$. Then, due to the fact that the $i_1, \ldots, i_d$ coordinates of v_c have the same absolute value, we need to compute, on c, at most $\binom{d}{\lceil \frac{d}{2} \rceil}$ tuples of elements that are potentially in the boundary of the bottom

k elements. This, induces at most $\binom{d}{\lceil \frac{d}{2} \rceil}$ local sortings, i.e., rank-1 instances. All these sorting will eventually be the elements of the $\mathcal{S}_d$ set. The number of all candidate support sets will now be $\binom{d}{\lceil \frac{d}{2} \rceil}\binom{n}{d}$ and the total computation complexity is $O\left(n^{d+1}\log n\right)$.

7.2 Approximation Guarantees

Here we prove the following lemma introduced in the main text of our manuscript. The lemma bounds the performance of our algorithm with respect to the optimal solution using q_d, which is part of the candidate solutions that the Spannogram routine outputs.

Lemma 1. *Let q_d be the optimal solution of $\mathcal{Q}_d$. Then, $G(q_d)$ is a subgraph with average degree $\delta_d = q_d^\intercal A q_d$ such that*

$$\frac{\delta_d}{\delta_{OPT}} \geq 1 - \frac{\lambda_{d+1} - |\lambda_n|}{\delta_{OPT}} \tag{24}$$

Proof. Let

$$\mathrm{OPT} = \max_{x \in \mathbb{S}_k} x^\intercal A x, \;\; \mathrm{OPT}_d = q_d^\intercal A_d q_d$$

First, we show that $q_*^T A q_*$ is upper bounded by $q_d^T A_d q_d$ and an additive factor λ_{d+1}, and eventually that $q_d^T A q_d$ is lower bounded by $q_*^T A_d q_d$and again with an additive factor $-|\lambda_n|$. Observe that

$$\mathrm{OPT} = \max_{x \in \mathbb{S}_k} \left(x^\intercal A_d x + x^\intercal (A - A_d) x\right). \tag{25}$$

Then, we have

$$\mathrm{OPT} \leq \max_{x \in \mathbb{S}_k} x^\intercal A_d x + \max_{x \in \mathbb{S}_k} x^\intercal (A - A_d) x \leq \mathrm{OPT}_d + \lambda_{d+1}. \tag{26}$$

Now, we will calculate the sparse quadratic of q_d on A:

$$\begin{aligned} q_d^\intercal A q_d \geq (q_d)^\intercal A q_d &= (q_d)^\intercal (A_d + (A - A_d)) q_d = (q_d)^\intercal A_d q_d + (q_d)^\intercal (A - A_d)) q_d \\ &\geq \mathrm{OPT}_d + \min_{\|x\|_2 = 1} x^T (A - A_d) x = \mathrm{OPT}_d - |\lambda_n| \\ &\geq \mathrm{OPT} - \lambda_{d+1} - |\lambda_n|, \end{aligned} \tag{27}$$

where in the last step we used the inequality from (26). Hence, we can combine the following inequalities to conclude our proof

$$\frac{\delta_d}{\delta_{\mathrm{OPT}}} \geq 1 - \frac{\lambda_{d+1} - |\lambda_n|}{\delta_{\mathrm{OPT}}} \tag{28}$$

□

Now, given a family of matrices that have a specific spectral profile, we can calculate our guarantees by just plugging in the bounds obtained by eigenvalue laws in the above bound.

7.3 Finding Planted Cliques

7.3.1 The planted clique random graph model and spectra of random graphs

The problem of finding a planted clique of size k can be recast as finding the densest subgraph of size k in a graph G: if a clique of size k exists, then it is the densest subgraph of size k. We consider the following probabilistic model of a graph on n vertices. We assume that there exist a random subset of k nodes in the graph that form a clique, i.e., all $\binom{k}{2}$ in-between edges of these nodes exist and are equal to 1. The rest of the edges in the remaining graph are drawn with probability p. Here, we will assume that this probability is $p = \frac{1}{2}$, however our work generalizes to the case where $0 < p < 1$.

We will use the following property[AKS98].

Lemma 2. *Let $\lambda_1(A) \geq \ldots \geq \lambda_n(A)$ be the eigenvalues of the adjacency matrix of $G\left(n, \frac{1}{2}, k\right)$. Then, with probability at least $1 - e^{-Cn}$, for some constant $C > 0$, we have that*

$$\lambda_1 = \left(\frac{1}{2} + o(1)\right) n$$
$$|\lambda_i(A)| \leq (1 + o(1))\sqrt{n},\ \forall i \geq 3. \tag{29}$$

Algorithm 4 Degree test

1: Input G and G_{I_d} of Algorithm 1.
2: Initialize $\hat{Q} = \emptyset$
3: **for** each vertex $v \in \mathcal{V}$ **do**
4: $\quad d_W(v) \leftarrow$ number of edges shared with vertices in $G(I_d)$.
5: $\quad$ **if** $d_W(v) > k/2 + (1-a)k$ **then**
6: $\quad\quad \hat{Q} \leftarrow \hat{Q} \cup v$
7: $\quad$ **end if**
8: $\quad$ Output $\hat{Q}$.
9: **end for**

7.3.2 Analysis of the Algorithm

In this section we will prove the following theorems.

Theorem 3. *For any $k > 4\sqrt{n}$, $\mathcal{G}(q_2)$ contains a constant fraction of the clique of size k, with high probability.*

Theorem 4. *For any $k \geq 9.15\sqrt{n}$ Algorithm 1, recovers the hidden clique $\mathcal{G}(q_*)$, with high probability.*

We continue by establishing all Lemmas that yield the tightness theorems.

Lemma 3. *Let $k = c\sqrt{n}$. Then, $\mathcal{G}(q_2)$ is a subgraph with average degree at least $(c-2)\sqrt{n}$, with high probability. In other words, It holds true that with probability at least $1 - e^{-Cn}$ for some constant $C > 0$,*

$$q_2^T A q_2 \geq k - 2\sqrt{n} = (c-2)\sqrt{n}.$$

Proof. We obtain the above lemma, by simply plugging the bounds of Lemma 2 on the eigenvalues λ_i, for $i \geq 3$, in Lemma 1 □

Then, we will use the following lemma to establish the consequent results.

Lemma 4 ([CL06])**.** *Let X be a binomial random variable on N trials with probability of success at each trial $p = \frac{1}{2}$. Then,*

$$\Pr\{X \geq E\{X\} + \delta\} \leq \exp\left(-\frac{\delta^2}{N + 2/3 \cdot \delta}\right). \tag{30}$$

Lemma 5. *Let $k = (c + \eta)\sqrt{n}$, for $c \geq 4$ and any, arbitrarily small, constant $\eta > 0$. Then the solution, q_2 of $\mathcal{Q}_2$, attains an overlap of at least*

$$\alpha = \sqrt{1 - \frac{4}{c}}$$

with the true clique, i.e. $q_2^T q \geq \alpha k$, with probability at least $1 - e^{-Cn}$, for some constant C.

Proof. By Lemma 3 we know that, for any solution q_2 of the algorithm, $q_2^T A q_2 \geq k - 2\sqrt{n}$, with exponentially small probability of failure. Then, since

$$q_2 \in \left\{ x : x_i \in \{0, 1/\sqrt{k}\}, \|x\|_0 = k, x^T A x \geq k - 2\sqrt{n} \right\}, \quad \text{w.h.p.},$$

we get

$$\Pr\left\{ q_2^T q \leq \alpha k \right\} \leq \Pr\left\{ \exists\, x, x_i \in \{0, 1/\sqrt{k}\}, \|x\|_0 = k : x^T A x \geq k - 2\sqrt{n} \text{ and } x^T q \leq \alpha k \right\}. \tag{31}$$

In other words, we are looking for the probability that any subgraph of size k attains both an average degree of at least $k - 2\sqrt{n}$ and an overlap with the true clique of at most αk. We denote that probability by p_B:

$$\begin{aligned}
p_{\mathrm{B}} &\triangleq \Pr\left\{ \exists\, x, x_i \in \{0, 1/\sqrt{k}\}, \|x\|_0 = k : x^T A x \geq k - 2\sqrt{n} \text{ and } x^T q \leq \alpha k \right\} \\
&\leq \sum_{m=1}^{\lfloor \alpha k \rfloor} \Pr\left\{ \exists\, x, x_i \in \{0, 1/\sqrt{k}\}, \|x\|_0 = k : x^T A x \geq k - 2\sqrt{n} \text{ and } x^T q = m \right\} \\
&\leq \sum_{m=1}^{\lfloor \alpha k \rfloor} \binom{k}{m} \binom{n-k}{k-m} \Pr\left\{ x, x_i \in \{0, 1/\sqrt{k}\}, \|x\|_0 = k : x^T A x \geq k - 2\sqrt{n} \text{ and } x^T q = m \right\} \\
&\leq \sum_{m=1}^{\lfloor \alpha k \rfloor} \binom{k}{m} \binom{n-k}{k-m} \Pr\left\{ x, x_i \in \{0, 1/\sqrt{k}\}, \|x\|_0 = k : x^T A x \geq k - 2\sqrt{n} \;\middle|\; x^T q = m \right\}
\end{aligned}$$

Here we used consecutive union bounds over all possible choices for the value of the overlap and the vertices in the set represented by the vector x. For the last inequality we used the fact that $\Pr(A, B) = \Pr(A|B)\Pr(B) \leq \Pr(A|B)$, for any pair of events A, B.

Now, let us define

$$p_C(m) \triangleq \Pr\left\{ x, x_i \in \{0, 1/\sqrt{k}\}, \|x\|_0 = k : x^T A x \geq k - 2\sqrt{n} \;\middle|\; x^T q = m \right\}, \quad m = 1, \ldots, \lfloor \alpha k \rfloor,$$

i.e. the probability that an arbitrary vertex set of cardinality k, which overlaps with the clique over exactly m vertices, satisfies the average degree condition. We can now use the principle of deferred decisions to get an upper bound for $p_C(m)$. The condition $x^T A x \geq k - 2\sqrt{n}$ implies that, in the subgraph defined by x there's a total of at least $k(k - 2\sqrt{n})/2$ edges. There are three types of edges in the subgraph $\mathcal{G}(x)$: edges between the $k - m$ non-clique vertices, edges across non-clique and clique vertices and edges between the m clique vertices. The first two types of edges appear independently with probability $1/2$, while the clique vertices form a full subgraph. So we can rewrite the same condition as

$$p_{\mathrm{C}}(m) = \Pr\left\{ \mathrm{Bi}\left(\binom{k-m}{2} + m(k-m), \frac{1}{2} \right) + \binom{m}{2} \geq \frac{k(k - 2\sqrt{n})}{2} \right\}. \tag{32}$$

At this point we notice that the left-hand side of the inequality in the condition, is surely non-decreasing in m; the fact that $p_C(m)$ is also non-decreasing in m follows. From now on, we will use $p_C(m) \leq p_C(\alpha k)$ for all $m = 1, \ldots, \lfloor \alpha k \rfloor$ and focus on $p_C(\alpha k)$.

$$\begin{aligned}
p_C(m) &\leq p_C(\alpha k) && (33) \\
&\leq Pr\left\{ \mathrm{Bi}\left(\binom{k-\alpha k}{2} + \alpha k(k - \alpha k), \frac{1}{2} \right) + \frac{\alpha k(\alpha k - 1)}{2} \geq \frac{k(k - 2\sqrt{n})}{2} \right\} && (34) \\
&= Pr\left\{ \mathrm{Bi}\left(\binom{(1-\alpha)k}{2} + \alpha(1-\alpha)k^2, \frac{1}{2} \right) \geq \frac{k^2 - 2k\sqrt{n} - \alpha^2 k^2 + \alpha k}{2} \right\} && (35)
\end{aligned}$$

Now we will try to bound the tail of the probability of the binomial random variable in the inequality using Lemma 4. We denote the binomial by $X \sim \text{Bi}(N, \frac{1}{2})$ and calculate its expectation. The associated number of trials is

$$N = \left(\frac{(1-\alpha)^2}{2} + \alpha(1-\alpha) \right) k^2 - \frac{1-\alpha}{2} k = \frac{1-\alpha^2}{2} k^2 - \frac{1-\alpha}{2} k \tag{36}$$

and

$$E\{X\} = \frac{1}{2} N = \frac{1-\alpha^2}{4} k^2 - \frac{1-\alpha}{4} k. \tag{37}$$

Now, let

$$\Delta = \frac{k^2 - 2k\sqrt{n} - \alpha^2 k^2 + \alpha k}{2} \tag{38}$$

So, we need to calculate δ, which is

$$\begin{aligned} \delta = \Delta - E\{X\} &= \frac{k^2 - 2k\sqrt{n} - \alpha^2 k^2 + \alpha k}{2} - \frac{1-\alpha^2}{4} k^2 + \frac{1-\alpha}{4} k \\ &= \frac{k^2 - 2k\sqrt{n} - \alpha^2 k^2}{2} - \frac{1-\alpha^2}{4} k^2 + \frac{1-\alpha+2\alpha}{4} k \\ &= \frac{2k^2 - 4k\sqrt{n} - 2\alpha^2 k^2 - k^2 + \alpha^2 k^2}{4} + \frac{1+\alpha}{4} k. \end{aligned}$$

Now, setting $k = c'\sqrt{n}$, where $c' = c + \eta$,

$$\delta = \frac{2c'^2 n - 4c'n - 2\alpha^2 c'^2 n - c'^2 n + \alpha^2 c'^2 n}{4} + \frac{1+\alpha}{4} c'\sqrt{n} \tag{39}$$

$$= \frac{c'^2 - 4c' - \alpha^2 c'^2}{4} n + \frac{1+\alpha}{4} c'\sqrt{n} \tag{40}$$

$$\geq \frac{c'^2 - 4c' - \alpha^2 c'^2}{4} n \triangleq \delta' \tag{41}$$

Let $C_1 = (c'^2 - 4c' - \alpha^2 c'^2)/4$ so that $\delta' = C_1 n$. Then, we have

$$Pr\left\{ \text{Bi}\left(N, \frac{1}{2}\right) \geq \Delta \right\} = Pr\left\{ \text{Bi}\left(N, \frac{1}{2}\right) \geq EX + \delta \right\} \tag{42}$$

$$\leq Pr\left\{ \text{Bi}\left(N, \frac{1}{2}\right) \geq EX + \delta' \right\} \tag{43}$$

$$\leq \exp\left(-\frac{\delta'^2}{N + 2/3 \cdot \delta'} \right) \tag{44}$$

$$= \exp\left(-\frac{C_1^2 n^2}{\frac{1-\alpha^2}{2} c^2 n - \frac{1-\alpha}{2} c\sqrt{n} + \frac{2}{3} C_1 n} \right) \tag{45}$$

$$\leq \exp\left(-Cn\right), \tag{46}$$

for some constant $C > 0$, as long as $C_1 > 0$. The latter is true from the assumption that

$$\alpha = \sqrt{1 - \frac{4}{c}}, \text{ and } \eta > 0.$$

Putting it all together,

$$\Pr\left\{q_2^T q\right\} \leq \sum_{m=1}^{\lfloor \alpha k \rfloor} \binom{k}{m}\binom{n-k}{k-m} p_C(m) \tag{47}$$

$$\leq \sum_{m=1}^{\lfloor \alpha k \rfloor} \binom{k}{m}\binom{n-k}{k-m} p_C(\alpha k) \tag{48}$$

$$\leq \lfloor \alpha k \rfloor \cdot 2^{kH(\alpha)} \left(\frac{(n-k)e}{(1-\alpha)k}\right)^{(1-\alpha)k} p_C(\alpha k) \tag{49}$$

$$\leq \lfloor \alpha k \rfloor \cdot 2^{kH(\alpha)} \left(\frac{(n-k)e}{(1-\alpha)k}\right)^{(1-\alpha)k} \exp\left(-Cn\right) \tag{50}$$

$$\leq \exp\left(-C'n\right), \tag{51}$$

for some constant $C' > 0$, which concludes the proof. □

Lemma 6. *Let $\mathcal{C} \subseteq V$ be the true clique, and $\mathcal{W} \subseteq V$ be any set of $|\mathcal{W}| = k$ vertices such that*

$$\mathcal{W} \cap \mathcal{C} \geq \alpha k$$

for any constant $\alpha > 3/4$. Then, the clean-up step of Algorithm 4 will succeed with high probability.

Proof. First, we will prove that, with overwhelmingly high probability, there exists no vertex outside the true clique that would be erronesously included in the final step of the algorithm. That is,

$$\mathbb{P}\left(\exists i \notin \mathcal{C},\ d_{\mathcal{W}}(i) \geq \alpha k\right) = o(1),$$

where $d_{\mathcal{W}}$ denotes the degree of node i, restricted on the set $\mathcal{W}$, i.e.

$$d_{\mathcal{W}}(i) = \left|\{j \in \mathcal{W} : (i,j) \in E\}\right|.$$

We first provide a simple upper bound for the restricted degree of a node $i \notin \mathcal{C}$:

$$\begin{aligned} d_{\mathcal{W}}(i) &= d_{\mathcal{W}\cap\mathcal{C}}(i) + d_{\mathcal{W}\setminus\mathcal{C}}(i) \\ &\leq d_{\mathcal{C}}(i) + d_{\mathcal{W}\setminus\mathcal{C}}(i) \\ &\leq d_{\mathcal{C}}(i) + |\mathcal{W}\setminus\mathcal{C}| \\ &\leq d_{\mathcal{C}}(i) + (1-\alpha)k. \end{aligned} \tag{52}$$

Now, notice that for any $i \notin \mathcal{C}$, $d_{\mathcal{C}}(i)$ is a binomial random variable of $|\mathcal{C}|$ trials and success probability of $p = 1/2$. Using standard Chernoff bounds for binomial random variables [MU05], we get

$$\mathbb{P}\left(d_{\mathcal{C}}(i) \geq (1+\delta)\frac{|\mathcal{C}|}{2}\right) \leq e^{-\frac{\delta^2|\mathcal{C}|}{6}}$$

for any $i \notin \mathcal{C}$ and after a union bound,

$$\mathbb{P}\left(\exists i \notin \mathcal{C}, d_{\mathcal{C}}(i) \geq (1+\delta)\frac{|\mathcal{C}|}{2}\right) \leq |i \notin \mathcal{C}| e^{-\frac{\delta^2|\mathcal{C}|}{6}} \leq n e^{-\frac{\delta^2|\mathcal{C}|}{6}}.$$

or

$$\mathbb{P}\left(\exists i \notin \mathcal{C}, d_{\mathcal{C}}(i) \geq (1+\delta)\frac{k}{2}\right) \leq n e^{-\frac{\delta^2 k}{6}}. \tag{53}$$

At last, using inequalities (52) and (53), the original error event probability can be upper bounded as

$$\begin{aligned}\mathbb{P}\left(\exists i \notin \mathcal{C},\, d_{\mathcal{W}}(i) \geq \alpha k\right) &\leq \mathbb{P}\left(\exists i \notin \mathcal{C},\, d_{\mathcal{C}}(i) + (1-\alpha)k \geq \alpha k\right) \\ &= \mathbb{P}\left(\exists i \notin \mathcal{C},\, d_{\mathcal{C}}(i) \geq [1 + (4\alpha - 3)]\frac{k}{2}\right) \\ &\leq n e^{-\frac{(4\alpha-3)^2 k}{6}},\end{aligned}$$

and, since we have assumed $\alpha > 3/4$, the statement is proved. □

Now, we simply need to plug in $3/4$ in the c bound, to obtain that we need a $c > 9.146...$ to have a high probability of success in recovering the clique.

7.4 Vertex Elimination

In this subsection we present our vertex elimination algorithm. This elimination was originally introduced in [PDK13] and used in the context of sparse PCA approximations. A slight modification here, yields provably rank-2 optimal elimination of candidate vertices in G. The main difference is here we are going to work on absolute curves. This elimination is also suited for the DkS problem, since we can consider the absolyte $|[v(\varphi)]_i|$ curves and fix $k' = 2k$. This will secure that only vertices that will never appear in a candidate subgraph will be eliminated. The reason behind altering k to k' can be seen if we consider the intersection points that are constructed by the non-absolute curves. We could "mirror" all negative points on the positive side of the y-axis. Then, a curve would not make it in the top k non-absolute ones, if it doesn't make it in the top $2k$ ones.

The vertex elimination step reduces the dimension n of the problem and this reduction in practice is empirically shown to be significantly and allows us to run our algorithm for very large matrices A. Our elimination algorithm is combinatorial and is based on sequentially checking the rows of V_d, depending on the value of their norm. This step is based again on the spannogram framework used in our approximation algorithm for DkS. In Fig. 5, we sketch the main idea of our elimination step.

The essentials of the elimination. First note that a locally optimal support set $\mathcal{I}_k(Vc)$ for a fixed c in (20), corresponds to the top k elements of v_c. As we mentioned before, all elements of v_c correspond to hypersurfaces $|[v(\varphi)]_i|$ that are functions of the $d-1$ spherical variables in φ. For a fixed $\varphi \in \Phi^{d-1}$, the candidate support set corresponds exactly to the top k (in absolute value) elements in $v_c = v(\varphi)$, or the top k surfaces $|[v(\varphi)]_i|$ for that particular φ. There is a very simple observation here: a surface $|[v(\varphi)]_i|$ belongs to the set of top k surfaces if $|[v(\varphi)]_i|$ is below at most $k-1$ other surfaces on that φ. If it is below k surfaces at that point φ, then $|[v(\varphi)]_i|$ does not belong in the set of k top surfaces.

A second key observation is the following: the only points φ that we need to check are the critical intersection points between d surfaces. For example, we could construct a set $\mathcal{Y}_k$ of all intersection points of all d sets of curves, such that for any point in this set the number of curves above it is at least $k-1$. In other words, $\mathcal{Y}_k$ defines a boundary: if a curve is above this boundary then it may become a top k curve; if not it can never appear in a candidate set. This means that we could test each curve against the points in $\mathcal{Y}_k$ and discard it if its amplitude is less than the amplitudes of all intersection points in $\mathcal{Y}_k$. However, the above elimination technique implies that we would need to calculate all intersection points on the n surfaces. Our goal is to use the above idea by serially checking one by one the intersection points of high amplitude curves.

Elimination algorithm description. We use the above ideas, to build our elimination algorithm. We first compute the norms of each row $\|[V_d]_{:,i}\|_2$ of V_d. This norm corresponds to the amplitude of $[v(\varphi)]_i$. Then, we sort all n rows according to their norms. We first start with the $k+d$ rows of V_d (i.e., surfaces) of highest norm (i.e., amplitude) and compute their $\binom{k+d}{d}$ intersection points. For each

intersection point, say ϕ, we compute the number of $|[v(\varphi)]_i|$ surfaces above it. If there are less than $k-1$ surfaces above an intersection point, then this means that such point is a potential intersection point where a new curve enters a local top k set. We keep all these points in a set $\mathcal{P}_k$.

We then move to the $(k+d+1)$-st surface of highest amplitude; we test it against the minimum amplitude point in $\mathcal{P}_k$. If the amplitude of the $(k+d+1)$-st surface is less than the minimum amplitude point in $\mathcal{P}_k$, then we can safely eliminate this surface (i.e., this row of V_d), and all surfaces with maximum amplitude smaller than that (i.e., all rows of V_d with norm smaller than the row of interest). If its amplitude is larger than the amplitude of this point, then we compute the new set of $\binom{k+d+1}{d}$ intersection points obtained by adding this new surface. We check if some of these can be added in $\mathcal{P}_k$, using the test of whether there are at most $k-1$ curves above each point. We need also re-check all previous points in $\mathcal{P}_k$, since some may no longer be eligible to be in the set; if some are not, then we delete them from the set $\mathcal{P}_k$. We then move on the next row of V_d, and continue this process until we reach a row with norm less than the minimum amplitude of the points in $\mathcal{P}_k$.

A pseudo-code for our vertex elimination algorithm can be found as Algorithm 1. In Fig. 6, we give an example of how our elimination works.

Algorithm 5 Elimination Algorithm.

Input: k, p, $V_d = \left[\sqrt{\lambda_1}v_1 \dots \sqrt{\lambda_1}v_1\right]$
Initialize $\mathcal{P}_k \leftarrow \emptyset$, $k = 2k$
Sort the rows of V_d in descending order according to their norms $\|e_i^T V_d\|$.
$\tilde{n} \leftarrow k+d+1$.
$\tilde{V} \leftarrow V_{1:\tilde{n},:}$.
for all $\binom{\tilde{n}}{d}$ subsets $(i_1, \dots, i_d)$ from $\{1, \dots, \tilde{n}\}$ **do**
  for all sequences $(b_1, \dots, b_{d-1}) \in \mathcal{B}$ **do**
    $c \leftarrow \text{nullspace}\left(\begin{bmatrix} e_{i_1}^T - b_1 \cdot e_{i_2}^T \\ \vdots \\ e_{i_1}^T - b_{d-1} \cdot e_{i_d}^T \end{bmatrix} V_d\right)$
    if there are $k-1$ elements of $|v_c|$ greater than $|e_1 V_d c|$ **then**
      $\mathcal{P}_k \leftarrow \mathcal{P}_k \cup \{|e_1 V_d c|\}$
    end if
    if $\|V_{\tilde{n}+1,:}\| < \min\{x \in \mathcal{P}_k\}$ **then**
      STOP ITERATIONS.
    end if
    $\tilde{n} \leftarrow \tilde{n}+1$
    $\tilde{V} \leftarrow V_{1:\tilde{n},:}$
    for each element x in $\mathcal{P}_k$ **do**
      check the elements $|v_c|$ greater than x
      if there are more than $k-1$ **then**
        discard it
      end if
    end for
  end for
end for
Output: $\tilde{A}_d = \tilde{V}_d \tilde{V}_d^T$, where $\tilde{V}_d$ comprises of the first $\tilde{n}$ rows of V_d of highest norm.

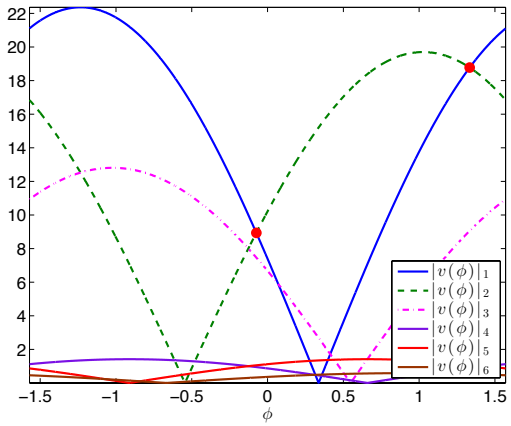
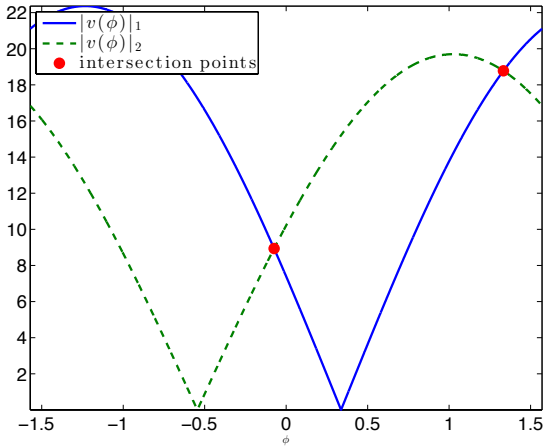
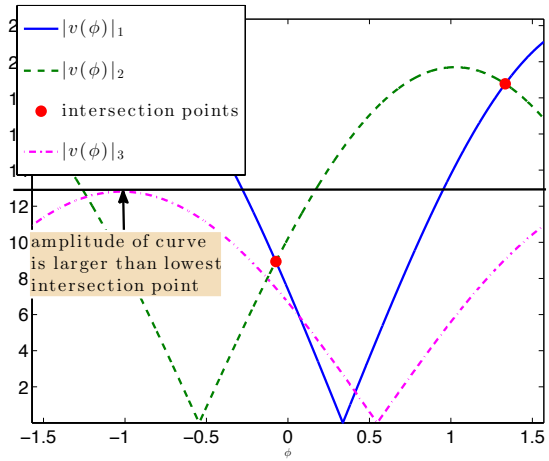


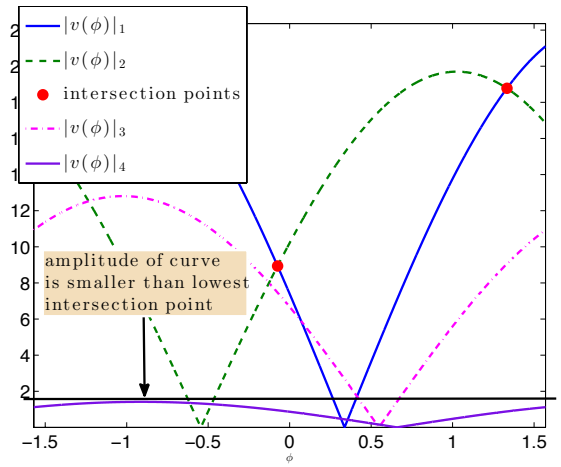
Figure 5: An example of a spannogram for $n = 6$, $d = 2$. Assume that $k = 1$. Then, the candidate optimal supports are $\mathcal{S}_2 = \{\{1\}, \{2\}\}$, that is either the blue curve ($i = 1$) is the top one, or the green curve ($i = 2$) becomes the top one, depending on the different values of ϕ. Finding the intersection points between these two curves is sufficient to recover these optimal supports. The idea of the elimination is that curves with (maximum) amplitude less than the amplitude of these types of intersection points can be safely discarded. In our example, after considering the blue and green curves and obtaining their intersection points, we can see that all other curves apart from the purple curve can be discarded; their amplitudes are less than the lowest intersection point of the blue and green curves. Our elimination step formalizes this idea.



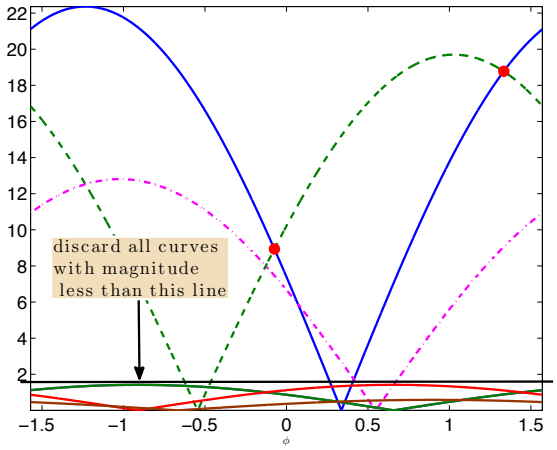
(a) Start with the curves of highest amplitude. Then, find their intersection points (red dots) and put them in the set $\mathcal{P}_1$.



(b) Examine the curve with amplitude that is the largest among the ones not tested yet. If the curve has amplitude less than the minimum intersection point in $\mathcal{P}_1$, discard it. Also, discard all curves with amplitude less than that. If it has amplitude higher than the minimum point in $\mathcal{P}_1$, then compute its intersection points with the curves already examined. For each new intersection point check whether it has $k-1$ curves above it. If yes, add it to $\mathcal{P}_1$. Retest all points in $\mathcal{P}_1$; if there is a point that has more than $k-1$ curves above it, discard it from $\mathcal{P}_1$.



(c) Repeat the same steps. Check if the amplitude of the lowest intersection point is higher than the amplitude of the curve next in the sorted list.



(d) Eventually this process will end by finding a curve with amplitude less than the intersection points in $\mathcal{P}_1$. It will then discard all curves with amplitude less, as shown in the figure above.

Figure 6: A simple elimination example for $n = 6$, $d = 2$, and $k = 1$.