On the densest k-subgraph problem

Uriel Feige and Michael Seltser
Department of Applied Math and Computer Science
The Weizmann Institute
Rehovot, Israel

September 8, 1997

Abstract

Given an n-vertex graph G and a parameter k, we are to find a k-vertex subgraph with the maximum number of edges. This problem is NP-hard. We show that the problem remains NP-hard even when the maximum degree in G is three. When G contains a k-clique, we give an algorithm that for any $\epsilon > 0$ finds a k-vertex subgraph with at least $(1-\epsilon)\binom{k}{2}$ edges, in time $n^{O((1+\log \frac{n}{k})/\epsilon)}$.

We study the applicability of semidefinite programming for approximating the dense k-subgraph problem. Our main result in this respect is negative, showing that for $k \simeq n^{1/3}$, semidefinite programs fail to distinguish between graphs that contain k-cliques and graphs in which the densest k-vertex subgraph has average degree below $\log n$.

1 Introduction

We study the *Densest k-subgraph* (*DkS*) maximization problem, of computing the densest k-vertex subgraph of a given simple graph. That is, given an input graph G and a parameter k, we are interested in finding a set of k vertices with maximum number of edges in the subgraph induced by this set.

This is a search problem. The corresponding decision problem (DkS_d) (given graph G, numbers k, m does G have k-vertex subgraph with at least m edges?) is NP-complete as shows the following reduction from $CLIQUE$ (which is known to be NP-complete): a graph G has a clique on k vertices iff G has k-vertex subgraph with at least $\binom{k}{2}$ edges. It follows that *DkS* is NP-hard.

No efficient algorithm for solving NP-hard problems is known and therefore there has been a lot of work in exploring the possibility of obtaining efficiently near-optimal solutions. The usual metric for measuring the nearness to optimality of a solution is by the ratio of its value to that of an optimal solution. The performance of an approximation algorithm is measured then by its worst-case ratio over all inputs of a given size. An algorithm achieves ratio $r \geq 1$ if for every instance it computes a solution whose value is at most r times smaller than the value of the optimal solution. (This is for maximization problems; in the case of minimization ones, the value of the computed solution must be at most r times the optimal value). Sometimes we are interested only in the value of the optimal solution, rather than actually finding a solution. Then we use the previous definition for estimation algorithms that try to answer the question "What is the value of the optimal solution?".

Feige, Kortsarz and Peleg showed in [8] that *DkS* has an approximation algorithm with approximation ratio $O(n^{1/3-\epsilon})$, for some $\epsilon > 0$. They were unable to present a complementary negative result giving evidence that for some $\epsilon > 0$, achieving an approximation ratio of $O(n^{\epsilon})$ is NP-hard. In fact, it is not even known whether achieving an approximation ratio of $(1+\epsilon)$ is NP-hard.

Arora, Karger and Karpinsky [3] obtained that if $k = \Omega(n)$ and the number of edges in the graph is $\Omega(n^2)$, then the problem has a polynomial time approximation scheme (PTAS). (That is, for every ϵ, there is a polynomial time algorithm that approximates *DkS* within ratio $1+\epsilon$).

Some problems are related to *DkS*.

- The *Densest Subgraph* (*DSG*) problem concerns choosing a subset V' (of arbitrary size) such that the vertex induced subgraph has maximum average degree. This problem can be solved polynomially using flow techniques (cf. Chapter 4 of [17]). One may hope that some algorithmic techniques used in solving *DSG* can help approximate *DkS*, but there seem to be major difficulties involved. Consider for example the case of regular graphs. The densest subgraph of a regular graph is the graph itself, and hence no algorithmic ideas are involved in solving *DSG* on this graph. On the other hand, finding the densest k-vertex subgraph remains NP-hard, because *CLIQUE* on regular graphs is NP-hard (proof omitted).

- The *Minimum Flux Cut* (*FLUX*) problem concerns choosing a cut C with minimum ratio between the number of edges that cross the cut and the number of vertices in the smaller side of the cut. This is a measure of the edge expansion of the graph. *FLUX* on regular graphs is related to *DkS* in the following sense: by solving *DkS* optimally for all values of k in a regular graph, one can deduce the optimal solution to the *FLUX* on this graph. *FLUX* can be approximated within a factor of $O(\log n)$ [18], but no hardness results are known for approximating *FLUX*.

We study further the problem of *DkS*. Our goal was either to provide evidence that *DkS* is hard to approximate, or to obtain improved approximation algorithms for the problem. Our results provide some modest steps towards this goal.

We show that if G has a clique on k vertices, then for each $1 > \epsilon > 0$ it is possible to obtain a k-vertex subgraph with at least $(1-\epsilon)\binom{k}{2}$ edges by an $n^{O((1+\log\frac{n}{k})/\epsilon)}$-time algorithm.

We also show that *DkS* remains NP-hard when restricted to graphs of maximum degree 3. This result is best possible, as the problem is solvable in polynomial time on graphs with maximum degree 2.

We study the use of *semidefinite programming* (SP) for approximating *DkS*. Though SP does give approximation ratios of roughly $\frac{n}{k}$ for the problem, our main result in this respect is negative. We construct an example where the gap between the optimum value of the SP and the true value of the *DkS* instance is $\tilde{\Omega}(n^{\frac{1}{3}})$.

2 *DkS* when G has a k-clique

NP-hardness of *DkS* can be proved by a reduction from *CLIQUE*. Can this reduction be used directly in order to show that *DkS* is hard to approximate? That is, is it NP-hard to distinguish between graphs that have a k-clique and graphs in which the densest k-vertex subgraph contains only $(1-\epsilon)\binom{k}{2}$ edges for some particular ϵ? We show that this problem is not "very" hard.

Examine the *DkS*, when we know that G contains K_k - clique on $k \geq 1$ vertices. Consider the following algorithm **DenseSubgraph**$(G = (V, E), k)$.

1. Denote by n the number of vertices of G. If $n \leq k$ then check whether $G = K_k$. If so then return G, otherwise return the empty graph.

2. Find the vertex v of smallest degree. Let h denote its degree.

3. If $h < (1-\epsilon)n$ then create two graphs: G_v and G_{-v}. G_v is a subgraph of G induced by v and its neighbours. G_{-v} is a graph which remains when we remove v from G. Call **DenseSubgraph**$(G_v,\ k)$. If **DenseSubgraph** returns nonempty output then stop. Else call **DenseSubgraph**$(G_{-v},\ k)$.

4. If $h \geq (1-\epsilon)n$ then repeatedly remove from G a vertex with the least degree in the currently remaining graph, until exactly k vertices are left. (This method is due to [5]). Denote by H the graph which is induced by these vertices. Return H.

We shall call a graph with k vertices and at least $(1-\epsilon)\binom{k}{2}$ edges a *desired* graph.

Claim 2.1 *If G contains K_k then* **DenseSubgraph***$(G,\ k)$ returns a desired subgraph of G.*

Proof: Fix k. We shall prove a stronger claim, namely that

1. if G contains K_k then the output of the algorithm is a desired graph;
2. if the output of **DenseSubgraph**$(G,\ k)$ is nonempty graph, then this is a desired graph.

We prove the claim by induction on n. The claim is true when $n \le k$, as in this case $G = K_k$ (otherwise the output is empty). Suppose that for all $n' < n$ the claim is true and prove for n. First, assume that

$$h \ge (1-\epsilon)n. \tag{*}$$

and hence step 4 of the algorithm is followed. Then we use the following auxiliary lemma.

Lemma 2.1 *If graph $U = (V, E), |V| = q$ has at least $(1-\epsilon)\binom{q}{2}$ edges for some $1 > \epsilon > 0$ and we remove from U the least degree vertex then the remaining graph U' has at least $(1-\epsilon)\binom{|V(U')|}{2}$ edges.*

Note that if $(*)$ holds then G has as at least $(1-\epsilon)\binom{n}{2}$ edges and so, repeatedly applying the lemma, we obtain that H is desired.

Now we prove lemma 2.1.

Proof: If we remove the least degree vertex then we loose at most $\frac{2}{q}|E|$ edges. So we remain with at least

$$(1-\epsilon)\binom{q}{2}\frac{q-2}{q} = (1-\epsilon)\binom{q-1}{2} = (1-\epsilon)\binom{|V(U')|}{2} \text{ edges.}$$

■

It follows from lemma 2.1 that whenever the algorithm returns a nonempty graph, this graph is desired. Now assume that $h < (1-\epsilon)n$. Note that the number of vertices both in G_v and in G_{-v} is strictly less than n. In addition, at least one of these graphs must contain K_k by construction.

So, we can apply the inductive hypothesis for each one of these graphs. It follows that the output of **DenseSubgraph**$(G,\ k)$ is a desired graph. Therefore the claim is true for n.

■

So, we have a deterministic algorithm which given k, G with K_k inside finds a k-vertex subgraph with at least $(1-\epsilon)\binom{k}{2}$ edges. Let us analyze its running time.

Step 4 of the algorithm takes polynomial time. In step 3 we divide the problem into 2 subproblems, one of size $(n-1)$ and one of size most $(1-\epsilon)n+1$. Thus, we have the following recurrence, where $T(n)$ is the running time of the algorithm on graphs with n

vertices, and the term n^c accounts for the polynomial computations of different stages of the algorithm.

$$T(n) \leq n^c + T(n-1) + T((1-\epsilon)n + 1).$$

It follows that the running time of the algorithm is at most $n^{O((1+\log\frac{n}{k})/\epsilon)}$. (Going from a problem of size n to a problem of size k involves at most $n^{O((1+\log\frac{n}{k})/\epsilon)}$ occasions in which the problem size shrinks by a factor of $(1-\epsilon)$). So we have the following theorem:

Theorem 2.1 *Given k, $\epsilon > 0$, and graph G with a k-clique, there is an $n^{O((1+\log\frac{n}{k})/\epsilon)}$ - time algorithm that returns a vertex induced subgraph on k vertices and at least $(1-\epsilon)\binom{k}{2}$ edges.*

Hence unless $NP \subseteq TIME(n^{O(\log n)})$, it is not NP-hard to distinguish between graphs that contain K_k and graphs in which the densest k-vertex subgraph has only $(1-\epsilon)\binom{k}{2}$ edges.

3 The problem on sparse graphs

The reduction given in Section 1 from *CLIQUE* to *DkS*$_d$ gives graphs which have vertices of degrees at least $k-1$. We shall show that *DkS* remains NP-hard on graphs with maximum degree 3. For graphs with maximal degree 2, there is a polynomial time algorithm for *DkS*, based on dynamic programming.

Consider *DkS* when the input graph has maximum degree 3. Call this problem *3DkS* (we use the same name both for the decision and the search problems). We shall show that the decision problem is NP-complete. It will follow that the search problem is NP-hard.

It is known that *CLIQUE* is NP-complete. We reduce from it to *3DkS*. Suppose that we have (G,k) - an instance of *CLIQUE* , $G = (V, E)$, $|V| = n$, $V = (v_1, \ldots, v_n)$. Create an instance of *3DkS* as follows.

For every $1 \leq i \leq n$ create C^i, a chordless cycle on n vertices $(v_1^i, \ldots, v_n^i)$. We shall call its vertices "circle" vertices. For each edge of G, $e = \{v_i, v_j\}$, $i < j$, create a simple path P^{ij} of size $kn+3$, $P^{ij} = (v_j^i, u_1^{ij}, \ldots, u_{kn+1}^{ij}, v_i^j)$. We call its vertices "path" vertices. Let the union of all the defined paths and cycles be the graph H. We take the triple $(H, t := kn + \binom{k}{2}(kn+1), m := kn + \binom{k}{2}(kn+2))$ as the *3DkS*-instance, in which we seek a subgraph with t vertices and m edges.

It is clear that the construction is polynomial and that each vertex of H has at most 3 neighbours. We shall show that in G there is clique on k vertices iff H has t-vertex subgraph with at least m edges.

Suppose that G has a clique on k vertices. Without loss of generality let them be $v_1, \dots, v_k$. Then the subgraph of H induced by the union of

$$C^1, \dots, C^k, \bigcup_{1 \le i < j \le k} P^{ij}$$

has t vertices and m edges.

Now we prove the converse direction. Assume that H has an induced subgraph with $kn + \binom{k}{2}(kn+1)$ vertices and at least $kn + \binom{k}{2}(kn+2)$ edges. Let $S = (V(S), E(S))$ denote such a subgraph. In the following part of the proof all the reasoning about vertex degrees will be with respect to S.

We shall use the following auxiliary fact:

Claim 3.1 *There are no two different vertices $v_1, v_2 \in V(S)$ of degree 3, and a vertex $u \in V(S)$ with degree 1, such that $v_1 \rightsquigarrow u$, $v_2 \rightsquigarrow u$. Here $a \rightsquigarrow b$ means that there is a simple path $a \equiv x_1, \dots, x_k \equiv b \in V(S)$ such that the degree of every vertex along the path, (except for a and b) is 2.*

We call a vertex v of degree three in S *good* if there is no vertex u of degree one in S such that $v \rightsquigarrow u$.

Proposition 3.2 *Let $U \subseteq V(S)$ be the set of good vertices of degree 3 in S. Then $|U| \ge 2\binom{k}{2}$.*

Proof: For the sake of contradiction, suppose that $m_1 = |U| < 2\binom{k}{2}$. Then calculate D – the sum of degrees of the vertices of S. By definition, $D = 3n_3 + 2n_2 + n_1$, where n_i is the number of vertices of degree i. $n_3 = m_1 + m_2$, where m_2 is a number of degree 3 vertices that are not good. By claim 3.1 $m_2 \le n_1$, so

$$D = 3m_1 + 3m_2 + 2n_2 + n_1 = 3m_1 + 2m_2 + 2n_2 + m_2 + n_1 \le$$
$$3m_1 + 2m_2 + 2n_2 + 2n_1 \le$$
$$2t + m_1 <$$
$$2\binom{k}{2} + 2(kn + \binom{k}{2}(kn+1)) =$$
$$2(kn + \binom{k}{2}(kn+2)) = 2m.$$

It follows that the number of edges of S is $\frac{D}{2} < m$, but it is a contradiction to the choice of S. So, the proposition is correct. ∎

Due to the construction of H, only vertices that belong to $\bigcup_{i=1}^{n} C^i$ may have degree 3. Therefore every vertex of U lies on one of these circles.

Let $v \in U$. By construction there are i, j such that $v \in C^i$, $v \in P^{ij}$. Let u_1 be the neighbour of v on this path (there is only one such vertex). $\mathrm{Deg}(v) = 3$, so $u_1 \in S$. Due to the choice of v and the structure of S, $\deg(u_1) = 2$. Hence the second neighbour of u_1, u_2, also belongs to S. Proceeding in such a way we discover that all the path P^{ij} is in S. Observe that S contains at most $\binom{k}{2}$ such paths, because the length of each path (without its "circle" vertices) is $kn + 1$.

On the other hand, as mentioned above, S must contain at least one such path for every vertex from U and each path contains at most two vertices from U. So we conclude that S contains exactly $\binom{k}{2}$ such paths, and among "circle" vertices only vertices of U may be "path" vertices.

Now turn to the neighbours of v on the circle C^i. The fact that $v \in U$ implies that both of them are in S, and that each of them has degree at least 2. For each of them there are only two possibilities:

- To be a "path" vertex. Then, as mentioned above, it must belong to U and, thus, have degree 3. Hence, its "circle" neighbours are in S.
- Not to be a "path" vertex. Then its degree is exactly 2 and so its "circle" neighbours also are in S.

Proceeding along C^i we finally get that all its vertices belong to S. Each such circle C^i corresponds to the vertex v_i of G and each P^{ij} corresponds to the edge $\{v_i, v_j\}$ of G. As follows from the above discussion, along with every such "edge", S contains both of the "edge" "vertices". We have $\binom{k}{2}$ "edges". It follows that S must contain at least k different "vertices". But every such "vertex" means n vertices of S and a counting argument allows us to conclude that S consists of exactly k such "vertices" - circles, and between each two of them there is one "edge". It follows that G has a clique on k different vertices. This completes the proof.

Note that splitting every edge of H by an additional vertex and slightly changing of t and m we immediately get the following theorem:

Theorem 3.1 DkS *is NP-hard on bipartite graphs with maximum degree 3.*

In Section 2 we saw that in order to prove hardness of approximation results, it is more promising to look at sparse instances of *DkS*. Unfortunately, despite of the low degree 3, we still proved NP-hardness using reduction from *CLIQUE*. Therefore we do not expect to get hardness of approximation results for *DkS* on the particular sparse graphs constructed in the proof of the theorem 3.1.

Theorem 3.1 has the following corollary.

Corollary 3.1 *For each* $1 > \epsilon > 0$ DkS_d *is* NP*-complete when* $m = k + k^{\epsilon}$*.*

Proof: Assume that

$$z = \frac{\binom{k}{2}^{1/\epsilon} - kn}{\binom{k}{2}}$$

is natural number (otherwise, round z up to the nearest positive integer). Now let the length of P^{ij}, paths in the proof of the correctness of the reduction, be $z+3$ instead $kn+3$. Then $m = k + k^{\epsilon}$. ■

This result improves upon a result of Asahiro and Iwama from [4], who show that *DkS* is NP-hard when $m = k^{1+\epsilon}$. In addition our proof is very simple and the vertices of our graphs have degrees bounded by 3.

4 The use of semidefinite programming

The following are three definitions of positive semidefinite matrices. Throughout we assume that the matrix Y is symmetric of order n.

1. All the eigenvalues of Y are nonnegative. (Recall that the eigenvalues of any symmetric matrix are real).
2. For any $x \in \mathcal{R}^n$, $x^T Y x \geq 0$.
3. There is an $n \times n$ matrix B such that $Y = B^T B$.

A *semidefinite program* is an optimization problem of the following form:

$$\text{Find reals } y_{1,1}, y_{1,2}, \ldots, y_{n,n},$$

$$\text{so as to maximize } c^T y,$$

$$\text{where } y = (y_{1,1}, y_{1,2}, y_{1,3}, \ldots, y_{n,n})^T, \tag{I}$$

$$\text{subject to}$$

$$\left.\begin{array}{l}\text{a finite number of linear constraints } a_i^T y \geq b_i,\\ \text{and such that the } n \times n \text{ matrix } Y = (y_{i,j})\\ \text{is symmetric and positive semidefinite.}\end{array}\right\} (\nabla)$$

Semidefinite Programming (SP) is an extension of linear programming (LP). Specifically if the condition that Y is a diagonal matrix is added to the constraint set (∇) then (**I**) reduces to linear programming.

Semidefinite programs are polynomial time solvable if an *a priori* bound on the size of their solution is known (see Grötchel, Lovàsz and Schrijver, [13]). Polynomial time solvability of SP is a direct consequence of the general results based on the ellipsoid method for convex programming. The main point essentially is that optimization of a linear function over a convex set endowed with a separation oracle and an *a priori* bound on the objective can be achieved in polynomial time using the ellipsoid method. See [14] for a thorough treatment. The ellipsoid method, however, has not proven practical in most applications, including SP. A more recent development is the possibility of using interior point methods to obtain polynomial time algorithms for semidefinite programs. See the work by Nesterov and Nemirovskii [20], and by Alizadeh [1].

4.1 Semidefinite Programming - how it was used

The application of *Semidefinite Programming* to obtaining approximation algorithms for NP-hard problems was pioneered by Goemans and Williamson [12]. This technique involves relaxing an integer quadratic program (which is NP-hard) to a semidefinite program (which can be solved with a sufficiently small error in polynomial time). In a remarkable breakthrough, Goemans and Williamson showed how this technique can be used to give a randomized approximation algorithm for the *MAX-CUT* problem with an approximation ratio of 0.878. The previously best known approximation ratio for *MAX-CUT* was 0.5. Subsequently, semidefinite programming based techniques have led to randomized algorithms with substantially better approximation ratios for a number of fundamental problems [12, 16, 7, 9, 2]. In all above applications of SP, the authors also studied the limitations of the SP approach, and showed that the particular SP formulations that they use do not in fact give approximation ratios that are significantly better than those proved. See also [6].

All the new developments mentioned above are *randomized* algorithms. All of them share the following common paradigm. First, a semidefinite program is solved to obtain Y as defined in (**I**). Then Y which is a Gram matrix is decomposed (using *Incomplete Cholesky Decomposition* [12]) to a collection of n vectors in n dimensional space satisfying some properties, depending upon the particular problem in question. This vector solution, which is obtained via a deterministic polytime algorithm is converted to a 0/1 solution by a randomized procedure. A set of independent random vectors is generated, and the 0/1 solution is obtained using some computation on the n given vectors and the random vectors.

One can derandomize the above randomized algorithms, i.e., obtain a "good" set of random vectors deterministically. The derandomization approach uses the method of Conditional Probabilities. The problem that occurs is to compute the conditional probabilities in polynomial time. Mahajan and Ramesh [19] present a technique which does this, and enables derandomization of all the semidefinite programming based approximation algorithms referenced above. This leads to deterministic approximation algorithms for *MAX-CUT*, *MAX k-CUT*, *MAX-DICUT*, *MAX-BISECTION*, *MAX-2SAT*, *MAX-SAT*, *GRAPH k-COLOURING* and *INDEPENDENT SET* with the same approximation ratios as their randomized counterparts.

4.2 The use of semidefinite programming for *DkS*

Rephrase *DkS* as follows. Given $(G = (V, E), V = \{1, \ldots, n\}, k > 0)$ - an instance of the problem, assign an integer $y_i \in \{0, 1\}$ to each vertex, 0 representing the vertices which are not in the subgraph and 1 representing the ones which are in the subgraph, so as the number of the subgraph vertices will be k and the sum, over all $i < j$, of $y_i y_j$ will be as big as possible. The reason this works is that $y_i y_j$ is 1 if both i and j belong to the subgraph and 0 if it is not the case. So, we have a following quadratic program (QP) in which the optimum is the maximum possible number of edges in k-vertex subgraph of G:

$$\text{maximize} \quad \sum_{\{i,j\}\in E} y_i y_j, \tag{$QP.1$}$$

$$\text{subject to} \quad y_i \in \{0, 1\}, \tag{$QP.2$}$$

$$1 \leq i \leq n,$$

$$\sum_{i=1}^{n} y_i = k. \tag{$QP.3$}$$

Since *DkS* is NP-hard, clearly one cannot solve this problem exactly in polynomial time (unless $P{=}NP$). However, one can relax this program to a polynomial-time solvable one, as follows. Let $\vec{u}_i$ be an n-dimensional vector whose first component is y_i and whose other components are 0, i.e., actually, a one-dimensional vector. Clearly

$$\forall 1 \leq i, j \leq n, \qquad 0 \leq (\vec{u}_i, \vec{u}_j) \leq 1 \tag{*}$$

($(\vec{x}, \vec{y})$ denotes scalar product of two vertices) and the objective function $(QP.1)$ can be rewritten as $\sum_{(i,j)\in E}(\vec{u}_i, \vec{u}_j)$ for these vectors, where the usual multiplication of integers y_i

and y_j has been replaced by the scalar product of vectors $\vec{u}_i, \vec{u}_j \in \mathcal{R}^n$. We also may note that for these vectors $\{\vec{u}_i\}_{i=1}^n$ the following equations are correct:

$$\sum_{i=1}^{n}(\vec{u}_i, \vec{u}_i) = k, \tag{**}$$

$$\forall 1 \leq i \leq n \sum_{j=1}^{n}(\vec{u}_i, \vec{u}_j) = k(\vec{u}_i, \vec{u}_i). \tag{***}$$

Now consider this problem (VP):

$$\text{Find vectors} \quad \vec{u}_1, \vec{u}_2, \ldots, \vec{u}_n \in \mathcal{R}^n,$$

$$\text{to maximize} \quad \sum_{\{i,j\}\in E}(\vec{u}_i, \vec{u}_j), \tag{$VP.1$}$$

$$\text{subject to} \quad 0 \leq (\vec{u}_i, \vec{u}_i) \leq 1, \tag{$VP.2$}$$

$$i, j \in \{1, \ldots, n\},$$

$$\sum_{i=1}^{n}(\vec{u}_i, \vec{u}_i) = k, \tag{$VP.3$}$$

$$\forall 1 \leq i \leq n \quad \sum_{j=1}^{n}(\vec{u}_i, \vec{u}_j) = k(\vec{u}_i, \vec{u}_i). \tag{$VP.4$}$$

Since the vectors u_i given above satisfy ($VP.1$) - ($VP.4$), they are feasible for this program. Allowing the $\vec{u}_i$ to be vectors in n dimensions makes VP a relaxation of *DkS*, and thus the optimal value of this program is at least as large as the number of edges in a densest k-subgraph of G.

In order to solve VP, we turn to SD-program, which can be (almost exactly) solved in polynomial time. By the third definition of "positive semidefinite", we see that if Y is symmetric and positive semidefinite, then there is an $n \times n$ matrix B such that $Y = B^T B$, and if B is $n \times n$ matrix, then $Y = B^T B$ is positive semidefinite. To solve the problem on variables $\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_n \in \mathcal{R}^n$, we instead solve this problem:

$$\text{Find a symmetric positive semidefinite matrix} \quad U = (U_{i,j}),$$

$$\text{to maximize} \quad \sum_{\{i,j\}\in E} U_{i,j}, \qquad (SDP.1)$$

$$\text{subject to} \quad 0 \le U_{i,j}, \qquad (SDP.2)$$

$$U_{i,j} \le 1, \qquad (SDP.3)$$

$$i, j \in \{1, \dots, n\},$$

$$\sum_{i=1}^{n} U_{i,i} = k, \qquad (SDP.4)$$

$$\forall 1 \le i \le n \quad \sum_{j=1}^{n} U_{i,j} = kU_{i,i}, \qquad (SDP.5)$$

Given U, we can find a B such that $U = B^T B$ in polynomial time as is explained in the section 4.1. Letting v_i be the ith column of B, we have $(\vec{v}_i, \vec{v}_j) = U_{i,j}$ and it is easy to see that the set of vectors $\{\vec{v}_i\}_{i=1}^n$ meets conditions $(*), (**)$. Given these vectors, we can build a matrix B whose ith column is u_i, and then let $U = B^T B$; we shall have U symmetric and positive semidefinite, the conditions ($SDP.2$ - $SDP.5$) are met. That is, VP can be phrased as a semidefinite program (SDP) and thus (almost exactly) solved in polynomial time.

Constraint ($SDP.5$) has no counterpart in the original quadratic program. There it was implied by the other constraints. But after relaxation to SDP this becomes a new constraint that is not implied by the other constraints.

Note that the value of the SD-program can not be more that $k^2 - k$ due to constraints ($SDP.4$, $SDP.5$).

The geometric interpretation of SDP's solution U is as follows. U is the Gram matrix of some vectors $\{\vec{u}_i\}_{i=1}^n$ (from VP). ($SDP.2$) means that the angle between any two of these vectors is at most 90°. ($SDP.3$) means in particular that the norm of each vector u_i is at most 1. Let v be $\sum_{i=1}^n \vec{u}_i$. Then $\forall 1 \le i \le n$ $\frac{||\vec{u}_i||}{cos(\widehat{\vec{u}_i, \vec{v}})} = \frac{||\vec{v}||}{k}$, because of condition ($SDP.5$). Here $cos(\widehat{\vec{a}, \vec{b}})$ denotes cosine of the angle between vectors $\vec{a}$ and $\vec{b}$. Thus, among vectors $\{\vec{u}_i\}_{i=1}^n$ the greater the norm of the vector u_i, the "closer" it is to the vector $\vec{v}$.

We would like to study the "integrality gap", the minimum over all graphs, of the ratio between the number of edges in the densest k-vertex subgraph and the optimal value of the semidefinite relaxation for that graph. In the following discussion we, on the base of above, shall speak about vectors of the SDP's solution, rather than its matrix.

4.3 Upper bound

Now we give an upper bound on the integrality gap. The analysis here is based on an approach of Goemans [11]. Consider the following randomized approximation polynomial algorithm.

1. Given $(G = (V, E), k)$ solve the corresponding SDP and obtain a vector set solution, $\vec{v}_1, \ldots, \vec{v}_n$.
2. Denote a sum $\sum_{i=1}^{n} \sqrt{(\vec{v}_i, \vec{v}_i)}$ by s. For each $i \in V$ pick vertex i with probability $\frac{k}{s}\sqrt{(\vec{v}_i, \vec{v}_i)}$. Denote by $H = (V(H), E(H))$ the subgraph of G which is induced by the picked vertices.
3. If $|V(H)| < k$ then add $k - |V(H)|$ vertices arbitrarily. If $|V(H)| > k$ then remove the $|V(H)| - k$ vertices of lowest degree. Denote the new graph by $P = (V(P), E(P))$. Return P.

For the analysis of the algorithm let us use the following version of the *Chernoff inequality*.

Lemma 4.1 *Let X_i, $1 \le i \le n$ be independent Bernoulli random variables with $Pr[X_i = 1] = p_i$. Then*

$$Pr[\sum_i X_i - \sum_i p_i > \alpha \sum_i p_i] < e^{-\frac{\alpha^2}{3}\sum_i p_i}, \tag{1}$$

when $0 < \alpha < \frac{1}{3}$.

It is easy to see that the algorithm is indeed polynomial and $|V(P)| = k$. Calculate $\mathcal{E}$, the expectation of $|E(H)|$.

$$\mathcal{E} = \sum_{e:=\{i,j\} \in E} Pr[e \in E(H)] = \frac{k^2}{s^2} \sum_e \sqrt{U_{i,i} U_{j,j}} \ge \frac{k^2}{s^2} \mathrm{OPT}_{SD},$$

where OPT_{SD} denotes the value of the SDP solution. Note that s is maximal when all the vectors of the optimum of SDP have the same norm, namely $\sqrt{\frac{k}{n}}$. Then $s = \sqrt{kn}$. It follows that

$$\mathcal{E} \ge \frac{k}{n} \mathrm{OPT}_{SD}.$$

Denote by q_1 the probability that $|E(H)| \le \mathcal{E} - 1$.

By definition of $\mathcal{E}$,

$$(\mathcal{E} - 1) q_1 + (1 - q_1)|E| \ge \mathcal{E},$$

because $|E(H)| \le |E|$.

It follows that

$$q_1 \le \frac{|E| - \mathcal{E}}{|E| - (\mathcal{E} - 1)} < 1 - \frac{1}{n^2}.$$

The last inequality is true because $|E| < n^2$.

If $|V(H)| < k$ then we add vertices and it obviously does not decrease the number of taken edges. Note that the expectation of number of vertices in H, $\mathcal{E}(|V(H)|)$, is exactly k. Let us calculate q_2, the probability that $|V(H)| > k(1 + k^{-\frac{1}{3}})$. Here we use lemma 4.1, and we shall assume $k > n^{\frac{1}{3}}$.

$$q_2 = Pr[|V(H)| > k(1 + k^{-\frac{1}{3}})] < e^{-\frac{k^{\frac{1}{3}}}{3}} < e^{-\frac{n^{\frac{1}{9}}}{3}} < \frac{1}{2n^2}.$$

$q_1 + q_2 < 1 - \frac{1}{2n^2}$. It follows that with probability $q \ge \frac{1}{2n^2}$ $|E(H)| > \mathcal{E} - 1$ and $|V(H)| \le k(1 + k^{-\frac{1}{3}})$. In this case we have to remove at most $k^{\frac{2}{3}}$ least degree vertices. Hence we loose at most $2\frac{|E(H)|}{k}k^{\frac{2}{3}}$ edges. Therefore the number of edges in the produced solution, $|E(P)|$ is at least

$$\frac{k}{n}(1 - \frac{2}{k^{\frac{1}{3}}})\mathrm{OPT}_{SD} \ge \frac{k}{n}(1 - \frac{2}{k^{\frac{1}{3}}})\mathrm{OPT},$$

where OPT denotes the number of edges in the densest k-vertex subgraph of G.

Suppose now that $k < n^{\frac{1}{3}}$. Then $k^2 < \frac{n}{k}$. The integrality gap is at most k^2 - optimal algorithm takes at least one edge. So the gap is at most $\frac{n}{k}$. It gives us the following theorem:

Theorem 4.1 *The integrality gap between the value of SDP-solution and the densest k-vertex subgraph of G is at most*

$$\frac{n}{k}(1 + \frac{O(1)}{k^{\frac{1}{3}}}) \tag{2}$$

Independently performing the randomized rounding of SDP-solution for a polynomial number of times and returning the densest among those k-vertices subgraphs found, we obtain a randomized polynomial time algorithm whose expectation of approximation ratio is at most $\frac{n}{k}(1 + \frac{O(1)}{k^{\frac{1}{3}}})$.

If $k < n^{\frac{1}{3}}$ then we can take $\frac{k}{2}$ arbitrary edges from G and return the set of vertices incident with these edges, adding arbitrary vertices to this set if its size is smaller than k. The approximation ratio of such algorithms is at most $k < n^{\frac{1}{3}} < \frac{n}{k}(1 + \frac{O(1)}{k^{\frac{1}{3}}})$.

4.4 Lower bound of the norm-based rounding algorithms

The vector rounding technique of subsection 4.3 was based only on the norm of vectors. Now we show the limitations of rounding methods that are based on the norm of the vectors disregarding their directions. We suppose that if we have to choose between two vectors of the same norm then we do our choice randomly.

Consider G – a graph on the n vertices $\{1,\ldots,n\}$ which is the union of $\frac{n}{k}$ disjoint unconnected cliques, $K_1,\ldots,K_{\frac{n}{k}}$, each one on k vertices, and we have to find the densest k-vertex subgraph of G. Then, of course, the optimal solution is to take one of the cliques (thus we shall have $\binom{k}{2}$ edges).

Look at our SP-program. Consider the following vectors assignment: if a vertex j belongs to the clique K_i then its vector $\vec{v}_j$ will be n-dimensional, whose ith component is $\sqrt{\frac{k}{n}}$ and whose other components are 0. These vectors give the optimum of the SDP because

$$\sum_{\{i,j\}\in E} (\vec{v}_i, \vec{v}_j) = k^2 - k$$

and this value is the maximum possible, as mentioned in section 4.2.

However all vectors are of the same norm. Hence, we need to select at random k of them (that is, of the corresponding vertices) and we are able to get in this way roughly $\frac{k^2}{n}$ vertices from each of these $\frac{n}{k}$ cliques. Thus we get roughly $\frac{n}{k}\binom{k^2/n}{2}$ edges.

Proposition 4.1 *Choosing vertices based only on the norms of the respective vectors one cannot get an approximation ratio better than by $\Omega(\frac{n}{k})$.*

4.5 A large integrality gap

We demonstrate that for some input (G, k), SDP gives an answer that is more than the optimum by a multiplicative factor of $\tilde{\Omega}(n^{\frac{1}{3}})$.

For each large enough n $V := \{1,\ldots,n\}$. Decide independently at random for each $i < j$ if $\{i,j\} \in E$, with probability $p = n^{-\frac{1}{3}} \ln n$. Let k be $49n^{1/3}$. This defines a random input for *DkS*.

Füredi and Komlós show in [10] that with high probability the absolute value of the most negative eigenvalue of adjacency matrix A for such G is at most $O(\sqrt{pn}) \equiv O(n^{\frac{1}{3}}\sqrt{\ln n})$.

Let us now to use one more version of the *Chernoff inequality.*

Lemma 4.2 *Let X_i, $1 \le i \le n$ be independent Bernoulli random variables with $Pr[X_i = 1] = p_i$. Then*

$$Pr[\sum_i p_i - \sum_i X_i > \alpha \sum_i p_i] < e^{-\frac{\alpha^2}{2}\sum_i p_i} \tag{3}$$

By lemma 4.2 we have that

$$\forall 1 \leq i \leq n \quad Pr[deg(i) < \frac{1}{2}pn] \leq e^{-\frac{1}{8}pn}. \tag{4}$$

It follows that with probability at least $1 - ne^{-\frac{1}{8}n^{\frac{2}{3}}\ln n} >> \frac{1}{2}$

$$\forall 1 \leq i \leq n \quad \frac{pn}{2} = \frac{n^{\frac{2}{3}}\ln n}{2} \leq deg(i) \tag{5}$$

Let us to use the following theorem of Weyl [15]:

Theorem 4.2 *Let $n \times n$ matrices A, B be Hermitian and the eigenvalues $\lambda_i(A)$, $\lambda_i(B)$, $\lambda_i(A+B)$ are ordered as $\lambda_{min} = \lambda_1 \leq \ldots \leq \lambda_n = \lambda_{max}$. Then $\forall k \in \{1, \ldots, n\}$ holds*

$$\lambda_k(A) + \lambda_1(B) \leq \lambda_k(A+B) \leq \lambda_k(A) + \lambda_n(B). \tag{6}$$

Define J be a following diagonal matrix: $\forall 1 \leq i \leq n \;\; J_{i,i} = \frac{deg(i)}{k-1}$. The least eigenvalue of the matrix $C = A + J$ by theorem 4.2 is at least $\Omega(n^{\frac{1}{3}}\ln n - n^{\frac{1}{3}}\sqrt{\ln n}) \equiv \Omega(n^{\frac{1}{3}}\ln n) > 0$. Define s to be $\frac{k^2-k}{2|E|}$. So, the matrix

$$B = sC \tag{7}$$

meets properties ((SDP.2) - (SDP.5)) from section 4.2. B gives us solution with value $2s|E| = k^2 - k$ - upper bound on the value of SDP-solution.

Therefore the value of SDP-solution is exactly

$$k^2 - k = \Omega(n^{\frac{2}{3}}) \tag{8}$$

On the other hand, by application of lemma 4.1 we can conclude for any particular H - k-vertex subgraph of G - that it has at most $\frac{3}{4}pk^2$ edges with probability at least $1 - e^{-\frac{1}{48}pk^2}$.

We have at most $\binom{n}{k}$ k-vertex subgraphs of G. So the probability that at least one of them has more than $\frac{3}{4}pk^2$ edges is bounded by

$$n^k e^{-\frac{1}{48}pk^2} = e^{k\ln n - \frac{1}{48}pk^2} = e^{49n^{\frac{1}{3}}\ln n - 49\frac{49}{48}n^{\frac{1}{3}}\ln n} << 1.$$

Hence, the optimal value is at most

$$O(n^{\frac{1}{3}}\ln \; n) \tag{9}$$

From (8), (9) we get

Theorem 4.3 *For any sufficiently large n there exists graph on n vertices such that if we use it as input for the SP-program then we shall get an answer that will be at least $\tilde{\Omega}(n^{\frac{1}{3}})$ times more than the optimal solution.*

This means that above *semidefinite programming* techniques do not improve the approximation ratio of $O(n^{\frac{1}{3}-\epsilon})$, which was obtained in [8].

Alternatively, we can consider as an input a random graph as above, and place in it a k-clique. Then an optimal solution of the SDP need not distinguish between this new graph and the graph in the proof of theorem 4.3, and hence might be useless in detecting the k-clique.

References

[1] F. Alizadeh, "Interior point methods in semidefinite programming with applications to combinatorial optimization", *SIAM Journal on Optimization*, 5 (1995), 13–51.

[2] N. Alon, N. Kahale, "Approximating the independence number via the θ-function", To appear in *Math. Programming.*

[3] S. Arora, D. Karger, M. Karpinski, "Polynomial Time Approximation Schemes for Dense Instances of $\mathcal{NP}$-Hard Problems", *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, 1995, pp. 284–293.

[4] Y. Asahiro, K. Iwama, "Finding Dense Subgraphs", *Proc. International Symposium on Algorithms and Computation '95*, Lecture Notes in Computer Science 1004, pp. 102–111, 1995.

[5] Y. Asahiro, K. Iwama, H. Tamaki, T. Tokuyama, "Greedily finding a dense subgraph", *Proc SWAT 96, Lecture Notes in Computer Science 1097*, 1996, pp. 136–148.

[6] U. Feige, "Randomized Graph Products, Chromatic Numbers, and the Lovász θ-Function", *in Proc. 27th Annual ACM Symposium on Theory of Computing*, 1995, pp. 635–640.

[7] U. Feige, M. Goemans, "Approximating the Value of Two Prover Proof Systems, With Applications to *MAX-2SAT* and *MAX-DICUT*", *Proc. Third Israel Symposium on Theory of Computing and Systems*, Israel, 1995.

[8] U. Feige, G. Kortsarz, D. Peleg, "The Dense k-Subgraph Problem", *A preliminary version appeared in the proceedings of the 34th Annual Symposium on Foundations of Computer Science*, 1993, pp. 692–701.

[9] A. Frieze, M. Jerrum, "Improved Approximation Algorithms for *MAX k-CUT* and *MAX BISECTION*", *Integer Programming and Combinatorial Optimization*, 1995.

[10] Z. Füredi, J. Komlós " The eigenvalues of random symmetric matrices", *Combinatorica*, 1 (1981), pp. 233–241.

[11] M. Goemans, *Private communication.*

[12] M. Goemans, D. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming", *Journal of the Association for Computing Machinery*, 42(6), 1115–1145, 1995.

[13] M. Grötchel, L. Lovász, A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization", *Combinatorica*, 1:pp. 169–197, 1981.

[14] M. Grötchel, L. Lovász, A. Schrijver, "Geometric Algorithms and Combinatorial Optimization", *Springer Verlag*, 1988.

[15] R. A. Horn, C. R. Johnson, "Matrix Analysis", *Cambridge University Press, Cambridge*, 1986.

[16] D. Karger, R. Motwani, M. Sudan, "Approximate *GRAPH COLOURING* by Semidefinite Programming", *35th IEEE Symposium on Foundations of Computer Science*, Santa Fe, pp. 1–10, October 1994.

[17] E. L. Lawler, "Combinatorial Optimization: Networks and Matroids.", *Holt, Rinehart and Winston*, 1976.

[18] F. T. Leighton and S. Rao, " An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Applications to Approximation Algorithms.", *29th Symp. on Foundations of Computer Science, IEEE*, pp. 422–431, 1988.

[19] S. Mahajan, H. Ramesh, "Derandomizing Semidefinite Programming Based Approximation Algorithms" *36th IEEE Symposium on Foundations of Computer Science*, pp. 162–169, October 1995.

[20] Y. Nesterov, A. Nemirovskii, "Self-Concordant Functions and Polynomial Time Methods in Convex Programming", *Central Economical and Mathematical Institute, USSR Academy of Science, Moscow*, 1990.