Internship Report on Finding Near Cliques and Bicliques by Maximum K-Clique and PQ-Biclique Density Extraction in Large Graphs

A. Tarik Temur

June - July 2019

1 Introduction

This report is a summary of my work during a 30-day internship under Prof. Pinar Karagoz and Prof. I. Hakki Toroslu of Orta Dogu Teknik Universitesi, during my internship my main goal was to build a system capable of extracting meaningful relationships between vertices of a graph representing networks of people and products or services to then be used in recommendation systems.

Graphs are an extremely widely used mathematical representation for many networks and constructs, real-life or otherwise. Simply put, they consist of vertices representing any kind of data point or agent with possible connections, called edges, to other vertices.

Extraction of dense subgraphs has many real life applications from detection of protein complexes in bioinformatics to recommendation systems in streaming platforms such as Youtube or Netflix. Here a subgraph is a subset of the original graph's vertices, and dense refers to a set of vertices with a high number of connections or edges. As an example, acquiring information about the likelihood of a person preferring a certain movie over another in a graph representing a binary people versus movies table so as to be used in a recommendation system in the future can be done through detecting dense subgraphs of people with presumably similar taste in movies (their watched movie histories largely overlap). In such a scenario it would be highly likely that a person would enjoy a movie represented by a vertex within the persons dense subgraph that the person has not yet watched.

I should also mention that my work in particular aims to extract cliques from bipartite graphs (graphs with two distinct sets of vertices that are internally completely disconnected, the only edges in such graphs are from one set to the other) which are called p-q bicliques.

After some literature survey in an effort accomplish my task, I was lead to the k-clique densest subgraph which, given a graph, tries to find a subgraph with the highest k-clique density. Here a k-clique is a simple graph theory term for a fully connected subgraph with k vertices, and k-clique density refers to the number of k-cliques in a given subgraph over the number of vertices in the subgraph. I studied different approaches to the k-clique densest subgraph problem which is relevant due to the fact that even though k-cliques themselves only represent fully connected subgraphs, a high k-clique density implies near cliques of a higher order (each k-clique consists of k (k-1)-cliques).

It should be noted that dense subgraphs alone might not provide enough information for reasonably accurate recommendations with the rising standards of our day, so I would like to stress that my work during this internship should be taken as a contribution to a larger project which aims to

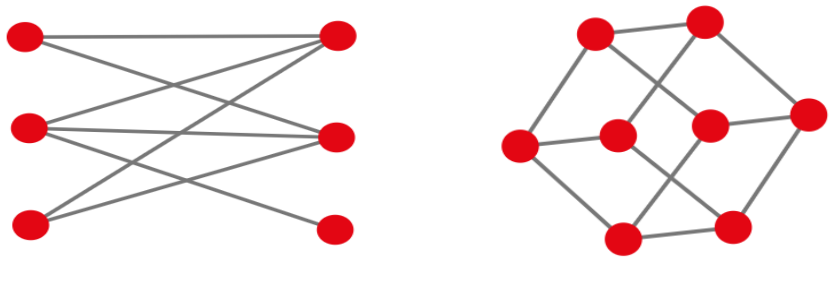

Figure 1: This figure shows a bipartite graph on left and a regular undirected graph on the right.

integrate a variety of methods to achieve the goal rather than an attempt at creating a complete and effective recommendations system. I have studied dense subgraph detection methods and written and experimented with a state-of-the-art software system due mainly to two 2015 papers [Tso15] [Mit+15], the system is my reproduction of algorithms proposed in these papers.

2 Information About the Project

This section provides a detailed analysis of the project in its many aspects categorized under four phases, namely, the Analysis, Design, Implementation and Testing phases in chronological order. I should note however that, in actuality many tasks listed under different phases were tackled in parallel.

2.1 Analysis Phase

As mentioned in the introduction, this work is intended to be a part of a broader project on recommendation systems, as interns in this project, we were assigned tasks aimed at studying certain aspects of recommendation systems. A graph representation for networks of people and provided products or services is natural, indeed many datasets available of such networks are essentially graphs (adjacency matrices).

With this in mind we first investigated extraction methods for relationships meaningful in a recommendations system application. We were directly lead to the densest subgraph problem which "lies at the core of large scale data mining" [BKV12]. A dense subgraph is meaningful in the recommendations systems sense because it implies near-cliques representing groups of people interested in similar products or services. The densest subgraph problem was shown to be polynomial time solvable in 1984 [Gol84] by a max flow based algorithm. The problem can be defined as follows:

Definition of DSP

Given a graph $G(V, E)$, find nonempty $S \subseteq V$ such that $d(S) \geq d(S')$ for all $S' \subseteq V$ where

$$d : \mathcal{V} - \emptyset \to \mathbb{I}_{\geq 0},\; d(S) = \frac{e(S)}{|S|}$$

note that $\mathcal{V}$ is the power set of V and $e : \mathcal{V} \to \mathbb{Z}_{\geq 0}$ is the number of edges induced by V.

After further investigation however, it was revealed to us that a simple densest subgraph solution was inadequate for a recommendations system due simply to the fact that its main goal is to return a subset with the maximum edge to vertex ratio which does not necessarily mean a near clique and in practice rarely does. For instance, on a huge football related graph with many near cliques of various sizes, it was observed that the dsp solution was the entire graph [Tso15].

Indeed different solutions were presented to this problem, one of the most interesting being the Triangle Densest Subgraph Problem in which the triangle density or 3-clique density was maximized, the algorithm looked for subgraphs dense in subsets of 3 vertices forming triangles. This problem was also polynomial time solvable and was solved using a max flow based algorithm [Tso14]. Triangle densest subgraphs yielded better results in terms of being near cliques due to the fact that any clique with more than 3 vertices consists of triangles.

This could be thought of as looking for a subgraph dense in a more complex building block of a clique than just an edge and agreeing with intuition, the results turn out to be much better. It also follows from this logic that for certain applications 4-cliques might make more sense than triangles, 5-cliques more than 4-cliques and so forth.

Which brings us to the K-Clique Densest Subgraph Problem, the solutions of this problem for different k values of a graph contains valuable information for a recommendations system simply due to the fact that highly accurate near clique detection is made possible by it. This problem was also shown to be polynomial time solvable for a constant k [Tso15]. During my internship my main focus was to implement a K-clique DSP solver, using methods proposed in the original paper as well as a follow up with a more efficient method [Mit+15].

We have also worked on a different, more straightforward matrix operation based approach to solving the same problem proposed in another paper, Dense Subgraph Extraction with Application to Community Detection [CS12], though this approach was explored by a fellow intern while I mainly worked on the K-Clique DSP solution.

Since the work presented in this report essentially aims to solve a number of decision and optimization problems, the definitions of said problems as well as of constructs and notions necessary for their understanding are first and foremost.

Definitions

Definition 1 (k-clique)

Let $G(V, E)$ be a graph, then a **k-clique** is defined to be a nonempty subset S of V such that $|S| = k$ and S is fully connected.

Definition 2 (pq-biclique)

Let $G(V, E)$ be a bipartite graph, such that $L, R \subseteq V$ are the left and right sets of vertices respectively, then a **pq-clique** is defined to be a nonempty subset $L' \cup R'$ of $L \cup R$, such that $|L'| = p$, $|R'| = q$ and L' and R' are fully connected (each vertex in L' is connected to each vertex in R').

Definition 3 (k-clique density)

Let $G(V, E)$ be a graph, and let $S \subseteq V$ be a nonempty subset of G's vertices, then the **k-clique density** of S is defined to be $\rho_k(S)$ such that:

$$\rho_k : \mathcal{V} - \emptyset \to \mathbb{I}_{\geq 0} \text{ and } \rho_k(S) = \frac{c_k(S)}{|S|}$$

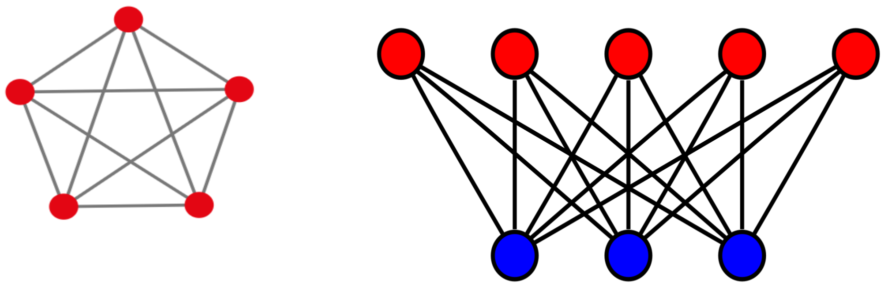

Figure 2: The figure on the left is a 5-clique on an undirected graph and the figure on the right is a 5-3 biclique on a bipartite graph.

where $\mathcal{V}$ is the power set of V, the function $c_k(S) : \mathcal{V} \to \mathbb{Z}_{\geq 0}$ represents the number of k-cliques in S, and $|S|$ is the cardinality of S. It should be noted that p_k maps any subset of V to positive rational numbers.

Definition 4 (pq-biclique density)

Let $G(V, E)$ be a bipartite graph with $L, R \subseteq V$ being the sets of left and right vertices respectively and let $L' \subseteq L, R' \subseteq R$, then the **pq-biclique density** of $S = L' \cup R'$ is defined to be $\rho_{p,q}(S)$ such that:

$$\rho_{p,q} : \mathcal{V} - \emptyset \to \mathbb{I}_{\geq 0} \text{ and } \rho_{p,q}(S) = \frac{c_{p,q}(S)}{|S|}$$

where $\mathcal{V}$ is the power set of V, the function $c_{p,q}(S) : \mathcal{V} \to \mathbb{Z}_{\geq 0}$ represents the number of pq-bicliques in S, and $|S|$ is the cardinality of S. It should be noted that $p_{p,q}$ maps any subset of V to positive rational numbers.

Problem 1 (Decision for K-clique DSP)

Given a graph $G(V, E)$ and a constant α, does there exist $S \subseteq V$ such that:

$$\rho_k(S) \geq \alpha$$

Problem 2 (Optimization for K-clique DSP)

Given a graph $G(V, E)$, find nonempty $S \subseteq V$ such that:

$$\rho_k(S) \geq \rho_k(S') \text{ for all } S' \subseteq V$$

Problem 3 (Decision for PQ-biclique DSP)

Given a bipartite graph $G(V, E)$ such that $L, R \subseteq V$ are the left and right sets of vertices of G respectively, and a constant α, does there exist $S = L' \cup R'$ for nonempty $L' \subseteq L$, $R' \subseteq R$ such that:

$$\rho_{p,q}(S) \geq \alpha$$

Problem 4 (Optimization for PQ-biclique DSP)

Given a bipartite graph $G(V, E)$ such that $L, R \subseteq V$ are the left and right sets of vertices of G respectively, find nonempty $S = L' \cup R'$ where $L' \subseteq L$ and $R' \subseteq R$ such that:

$$\rho_{p,q}(S) \geq \rho_{p,q}(S') \text{ for all } S' \subseteq V$$

With the necessary definitions made, hopefully the solution can now be more clearly understood.

2.2 Design Phase

The solution consists of a series of ideas, theorems and algorithms across three papers, namely, The Triangle Densest Subgraph Problem [Tso14], K-clique Densest Subgraph Problem [Tso15] and Scalable Large Near Clique Detection in Large-Scale Networks via Sampling [Mit+15] in chronological order. Although ultimately the main algorithm stems from the third paper, the groundwork is done in the previous ones and a careful study of all three papers is required for a complete understanding of the method.

Indeed many proofs and important points are left unexplained in the third paper with pointers to the second and the same is true for the second and the first. For the sake of completeness, I have compiled a step-by-step explanation of the methods as well as proofs concerning their correctness and termination. It should be noted that this explanation provides more detailed proofs than those available in the original papers, during my study I had to prove certain theorems in a more detailed fashion since I was unable to follow certain premises or conclusions that followed "we can see that".

Here I will provide a basic explanation for the overall method followed by a detailed examination of its sub-methods and algorithms as well as their correctness, termination, necessities and computational complexities.

Summary of the Method

Basically put, we can solve optimization problems 2 and 4 by repeatedly solving decision problems 1 and 3 for different α values. This is done by a binary search operation on all possible different density values, this set turns out to be discrete, finite and its cardinality polynomially bounded in the number of vertices. Therefore solving the decision problems leads to solving the optimization problems with just an extra logarithmic term in front. In order to solve the decision problems, a flow network is built using the α value as well as the vertices and (k-1)-cliques of the original graph. Listing (k-1)-cliques itself is a nontrivial task and has a large literature surrounding it, I did not have the time to delve into this new set of optimization problems and my implementation makes use of a pre-optimized k-clique listing algorithm written in C++ [Uno06]. Once the network is built, an (S, T) cut is found by running a max flow on the network, and the decision is made depending on what S is.

We will examine the K-clique DSP and PQ-biclique DSP separately for clarity.

K-Clique DSP:

Before we begin, two basic definitions need to be made:

$$|V| = n$$

$$|E| = m$$

Certainly the summary of the method requires elaboration. Let's begin with the first part, namely:

Theorem 1:

Given a solution for Problem 1, Problem 3 can be solved with an extra $O(log(n))$ computation for constant k.

Proof. First we need to define the maximum and minimum values for the k-clique density $\rho_k(S)$ of any subset S of V that we would like to include in our search, the minimum value is:

$$\rho_k(S) \geq \rho_k(V) \text{ for all } S \subseteq V \tag{1}$$

This is true because a subgraph with less density than the original graph is of no interest, simply returning the entire graph yields a better solution.

Now for the maximum value:

$$\rho_k(S) \leq \frac{1}{n}\binom{n}{k} \text{ for all } S \subseteq V \tag{2}$$

This is true because the maximum number of b-cliques in a graph with a vertices is $\binom{a}{b}$ (number of different subsets of b vertices) which is an increasing function in a, therefore its maximum value in our domain is at the maximum value of a, namely $a = n$.

Now from (1) and (2):

$$\frac{c_k(V)}{|V|} \leq \rho_k(S) \leq \frac{1}{n}\binom{n}{k} \text{ for all } S \subseteq V \tag{3}$$

We now need to prove a small claim: The minimum difference between two different density values is $\frac{1}{n(n-1)}$.

Proof. Let S_1, S_2 be two subsets of V with different densities such that,

$$\frac{|c_k(S_1)|S_2| - c_k(S_2)|S_1||}{|S_1||S_2|} > 0 \tag{4}$$

we will prove the claim under two different cases:
Case 1: $|S_1| = |S_2|$:
From (4), we have:

$$\frac{|c_k(S_1) - c_k(S_2)|}{|S_1|} \geq \frac{1}{n} \geq \frac{1}{n(n-1)} \tag{5}$$

Case 2: $|S_1| \neq |S_2|$:
Since c_k and set cardinalities are both integers, from (4), we have:

$$|c_k(S_1)|S_2| - c_k(S_2)|S_1|| \geq 1 \tag{6}$$

Then since the largest two subsets of V with different numbers of vertices have n and $(n-1)$ vertices, it follows from (6) that:

$$\frac{|c_k(S_1)|S_2| - c_k(S_2)|S_1||}{|S_1||S_2|} \geq \frac{1}{n(n-1)} \tag{7}$$

Therefore, the minimum difference between two subsets of V with different density values is $\frac{1}{n(n-1)}$ □

Now that we have boundaries and step sizes, let's find the number of different density values α we can have, from (2) and (7):

$$(max - min)/\text{step\_size} = (\frac{1}{n}\binom{n}{k} - \frac{c_k(V)}{n})(n(n-1)) \tag{8}$$

Which, since $\frac{c_k(V)}{n} \geq 0$, is clearly upper bounded by:

$$(\binom{n}{k})(n-1) = \frac{n!(n-1)}{k!(n-k)!} \tag{9}$$

So the total number of possible points is a polynomial of order $(k+1)$ and therefore a binary search operation on it has cost $O(log(n^{k+1}))$, which, for constant k, is $O(log(n))$. □

As a direct consequence of Theorem 1, we can build an algorithm for Problem 3 by only performing binary search on solutions of Problem 1 for different α values. We have shown the interval boundaries as well as step size for the binary search in equations (5) and (7) of the Proof of Theorem 1.

At this point a construct needs to be defined, this is a flow network that will be used in Problem 1's, and ultimately, Problem 3's solution.

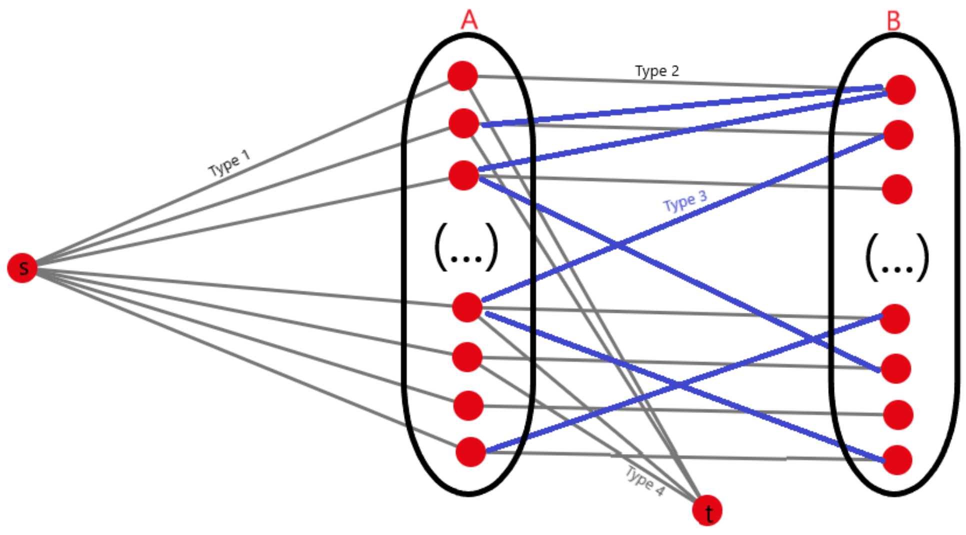

Figure 3: Construct 1 for some graph.

Construct 1:

Given some graph $G(V, E)$, an α value and the set of (k-1)-cliques in G, the construct is defined as the following:

$H(V, E, c)$ is a bipartite flow network with A, B being its left and right sets of vertices and $V = A \cup B \cup \{s, t\}$.

Each vertex in A represents a vertex in the original graph G.

Each vertex in B represents a (k-1)-clique in the original graph G.

There are four types of arcs in the network, namely:

type 1:

From the source s to each vertex $v \in A$, with a capacity of $f_k(v)$ where $f_k(v)$ is the number of k-cliques v participates in.

type 2:

From each vertex $v \in A$ to all vertices $u \in B$ iff v, together with u, form a k-clique, with capacity equal to 1. This type will be further explained below.

type 3:

From each vertex $u \in B$ to all vertices $v \in A$ if v participates in u. With infinite capacity. This type, though looks unnatural, is crucial for the correctness of the algorithm.

type 4:

From each vertex $v \in A$ to the sink t with capacity equal to $k\alpha$.

An explanation about arcs of type 2 is necessary here, we need establish a simple fact in graph theory:

In some k-clique Q, every vertex is fully connected to a unique (k-1)-clique together with which they form the k-clique. This fact simply comes from the definition of a clique. In arcs of type 2, the goal is to connect each vertex $v \in A$ to the unique (k-1)-clique $u \in B$ such that v and u together form a k-clique.

The next step is to find an (S, T) cut in H which is proven to solve the decision Problem 1.

Theorem 2:

Construct 1 can be used to solve Problem 1 in polynomial time (for constant k).

Proof. Problem 1 requires us to decide if some $S \subseteq G$ has k-clique density equal to or more than α. We will do so by running an (S,T) cut on instance of Construct 1, $H = A \cup B \cup \{s, t\}$ constructed using G and α.

We begin by finding an (S,T) cut on H. Let $\{s\} \cup A_1 \cup B_1 = S$ and $\{t\} \cup A_2 \cup B_2 = T$. Now consider the cost of such an (S,T) cut:

$$cost = \sum_{v \in A_2} f_k(v) + \alpha k|A_1| + \sum_{j=1}^{k-1} j c_k^{(j)} \tag{10}$$

The first term in this summation namely:

$$\sum_{v \in A_2} f_k(v)$$

is due to arcs of type 1. This term is fairly straightforward to justify since it simply follows from the definition of type 1 arcs, any vertex $v \in A_2$ has to be disconnected from the source s and cutting the edge connecting them incurs a $f_k(v)$ cost.

The second term namely,

$$\alpha k|A_1| \tag{11}$$

is due to arcs of type 4. This term is also straightforward to justify since it follows from the definition of type 4 arcs, any vertex $v \in A_2$ has to be disconnected from the sink t by definition of an (S,T) cut and the edge connecting v and t has capacity $k\alpha$.

Finally the third term namely,

$$\sum_{j=1}^{k-1} j c_k^{(j)} \tag{12}$$

is due to arcs of type 2 but requires elaboration.
The rather peculiar term $c_k^{(j)}$ is defined to be the number of k-cliques not induced by A_1 with exactly j vertices in A_1.

Arcs of type 2 exist between vertices and k-1 cliques when together they form a k-clique. If a vertex u representing a k-1 clique is in B_1, then all of its components must be in A_1 due to arcs of type 3. Though the opposite of this need not be true. So $u \in B_2$ does not imply that all vertices in u have to be in A_2.

So any k-clique not induced by A_1 with j elements in A_1 will result in the severing of one type 2 arc. Since for some $v \in A_1$ the k-1 clique u together with which v forms a k-clique is in B_2 (clearly had u been in B_1, all vertices in it would have to be in A_1, and the k-clique would have been induced by A_1).

Therefore, the third term represents the number of arcs of type 2 we have cut due to crossings. Now notice the following:

$$kc_k(A_1) + \sum_{j=1}^{k-1} jc_k^{(j)} = \sum_{j=1}^{k} jc_k^{(j)} = \sum_{v \in A_1} f_k(v) \tag{13}$$

And,

$$\sum_{v \in A_1} f_k(v) = kc_k - \sum_{v \in A_2} f_k(v) \tag{14}$$

We can finally substitute (13) and (14) into (10) to get:

$$cost = kc_k + \alpha k|A_1| - kc_k(A_1) \tag{15}$$

We also know that the maximum possible value of the (S,T) cut is kc_k since $S = \{s\}$ is a valid cut. Now if the following holds:

$$cost < kc_k \tag{16}$$

Then from (15), we can say:

$$kc_k(A_1) > \alpha k|A_1| \tag{17}$$

Or,

$$\frac{c_k(A_1)}{|A_1|} = \rho(A_1) > \alpha \tag{18}$$

Therefore, a subgraph $S \subseteq V$ with $\rho(S) > \alpha$ and furthermore $S = A_1$.

So by creating an instance of Construct 1 and finding an (S,T) cut on it, we can solve Problem 1. In fact the solution is as simple as checking if $S = s$.

If indeed $S = \{s\}$ then the answer is False, otherwise the answer is True and the subgraph in question is A_1.

$\square$

Since at this point we pretty much have a complete exact solution to Problem 1, it makes sense to examine the time complexity of the solution,

The solution constructed above has time complexity $O(c_k^2)$ and space complexity $O(c_k)$.

It's worth mentioning here that this is true in practice due to c_k terms dominating any other term in the more complicated complexity functions. And also that c_k for constant k is upper bounded by n^k.

I will not be providing a complete complexity analysis, you may refer to [Mit+15] for a more in depth examination.

Bipartite Graphs:

The algorithmic scheme built around Construct 1 above can also work on bipartite graphs, thus solving problems 2,4. Small clarifications may be necessary concerning the four types of arcs detailed above.

type 1:

From the source s to each vertex $v \in A$, with a capacity of $f_{p,q}(v)$ where $f_{p,q}(v)$ is the number of pq-bicliques v participates in.

type 2:

From each vertex $v \in A$ to all vertices $u \in B$ iff v, together with u, forms either a (p-1, q)-biclique or a (p, q-1)-biclique, with capacity equal to 1.

type 3:

From each vertex $u \in B$ to all vertices $v \in A$ if v participates in u. With infinite capacity.

type 4:

From each vertex $v \in A$ to the sink t with capacity equal to $k\alpha$.
Using Theorem 2, with the updated Construct 1, we can show that the method also works for bipartite graphs and also has polynomial time and space complexity.

2.3 Implementation Phase

I was not able to find any open source implementations of K-Clique DSP or PQ-Biclique DSP and therefore had to implement them. I'll begin by explaining my K-Clique DSP solution first.

2.3.1 Implementation of K-Clique DSP:

I represent graphs and flow networks as adjacency matrices, so 2D vectors of integers in C++. My implementation of K-Clique DSP is implemented as a set of methods under a class named KCDSPFinder (indeed the naming scheme stands witness to my limitless powers of imagination).

```
class KCDSPFinder{
    public:
        KCDSPFinder();
        KCDSPFinder(std::vector<std:vector<int>>&, unsigned int);
        double _get_density(std::vector<int>&);
        std::vector<int> _density_search(double);
        std::vector<std::vector<int>> _get_construct_one(double);
        std::vector<int> _get_alpha_dense_subgraph(double);
        std::vector<int> _get_densest_subgraph();
    private:
        std::vector<std::vector<int>> Graph;
        unsigned int K;

};
```

Below are explanations of KCDSPFinder's methods:

```
    KCDSPFinder();
```

This is basically the C++ default constructor for the class.

```
KCDSPFinder(std::vector<std::vector<int>>&, double);
```

An object of the class is constructed by providing two inputs where the 2D vector is the adjacency matrix of the graph and the unsigned integer is K.

```
double KCDSPFinder::_get_density(std::vector<int>&)
```

This function returns the K-Clique density of the subgraph (of the original graph) induced by its input vertices. The input vertices are provided as a one dimensional vector.

It should be noted that the counting of the number of K-Cliques is done by the K-Clique listing algorithm in the implementation by Uno [Uno06]. The main counting algorithm implemented by Uno is added as a friend of this class and is used to enumerate and list K-Cliques.

```
std::vector<std::vector<int>> KCDSPFinder::_get_constructor_one(double)
```

This function builds and returns construct 1 detailed under the constructive proof of the polynomial time solution for K-Clique DSP.
Details of the construct can be found in the proof above.
The return value of the function is the flow-network represented in adjacency matrix form.
The input of the function is the value alpha (the density value we are searching for).

```
std::vector<int> KCDSPFinder::_get_alpha_dense_subgraph(double);
```

This function solves the decision problem 1 detailed in the constructive proof of the polynomial time solution for K-Clique DSP.
The return value is the subset of vertices that induce a subgraph with at least alpha K-Clique density. If there is no such subgraph, the the return value is an empty vector.

This function uses the Min S-T cut algorithm implemented by Yuri Boykov as part of the Boost Graph Library freely available online [Boy06].

```
std::vector<int> KCDSPFinder::_get_densest_subgraph();
```

This is the main function that calls all the others, it performs a binary search over the set of possible values of alpha and returns the k-clique densest subgraph.

This function takes no inputs, though the object it is attached to must have a graph and a K value set.

The densest subgraph is returned as a subset of the vertices, and the complete subgraph can be obtained by finding the graph induced by this set of vertices.

2.3.2 PQ Biclique Densest Subgraph Problem

: This is basically analogous to the implementation of the K-Clique DSP above, however I will still take the time to explain the basic functionalities of the individual methods.

I represent graphs and flow networks as adjacency matrices, so 2D vectors of integers in C++. My implementation of PQ-Biclique DSP is implemented as a set of methods under a class named PQBicliqueDSPFinder.

```
class PQBicliqueDSPFinder{
    public:
        PQBicliqueDSPFinder();
        PQBicliqueDSPFinder(std::vector<std:vector<int>>&, std::vector<int>&,
            std::vector<int>&, unsigned int, unsigned int);
        double _get_density(std::vector<int>&);
        std::vector<int> _density_search(double);
        std::vector<std::vector<int>> _get_construct_two(double);
        std::vector<int> _get_alpha_dense_subgraph(double);
        std::vector<int> _get_densest_subgraph();
        std::vector<std::vector<int>> _enumerate_pq_bicliques(std::vector<int>&);
    private:
        std::vector<std::vector<int>> Graph;
        std::vector<int> R;
        std::vector<int> L;
        unsigned int P;
        unsigned int Q;

};
```

Below are explanations of PQBicliqueDSPFinder's methods:

```
    PQBicliqueDSPFinder();
```

This is basically the C++ default constructor for the class.

```
    PQBicliqueDSPFinder(std::vector<std::vector<int>>&, std::vector<int>&,
        std::vector<int>&, unsigned int, unsigned int);
```

An object of the class is constructed by providing two inputs where the 2D vector is the adjacency matrix of the graph, the unsigned integers are P and Q respectively and the 1D vectors are the left and right sets of vertices respectively.
The reason behind explicitly storing the left and right sets of vertices is optimization and reduced complexity in the implementations of certain methods.

```
    double PQBicliqueDSPFinder::_get_density(std::vector<int>&)
```

This function returns the PQ-Biclique density of the subgraph (of the original graph) induced by its input vertices. The input vertices are provided as a one dimensional vector.

I was not able to find an implementation online for the PQ Biclique counter and so implemented my own naive counter.

```
    std::vector<std::vector<int>> PQBicliqueDSPFinder::_get_constructor_two(double)
```

Creates and returns construct 2 for the given density value as per the instructions detailed in the previous section under Bipartite Graphs.

```
    std::vector<int> PQBicliqueDSPFinder::_enumerate_pq_bicliques(std::vector<int>&);
```

This function naively enumerates the PQ Bicliques in the given subgraph. The subgraph is provided as a subset of vertices, the graph is induced by these vertices.

```
    std::vector<int> PQBicliqueDSPFinder::_get_alpha_dense_subgraph(double);
```

This function solves the decision problem 1 detailed in the constructive proof of the polynomial time solution for PQ-Biclique DSP.

The return value is the subset of vertices that induce a subgraph with at least alpha PQ-Biclique density. If there is no such subgraph, the the return value is an empty vector.

This function uses the Min S-T cut algorithm implemented by Yuri Boykov as part of the Boost Graph Library freely available online [Boy06].

```
std::vector<int> PQBicliqueDSPFinder::_get_densest_subgraph();
```

This is the main method that calls all other methods, it performs a binary search over the set of possible values of alpha and returns the pq-biclique densest subgraph.

This function takes no inputs, though the object it is attached to must have a graph and a (p, q) value set.

The densest subgraph is returned as a subset of the vertices, and the complete subgraph can be obtained by finding the graph induced by this set of vertices.

2.4 Testing Phase

I have conducted my experiments on datasets (mostly social networks) taken from the Stanford Network Analysis Project SNAP.

The list of datasets I have experimented with is as follows:

Name	Graph Type	Number of Nodes	Number of Edges
com-Yoututbe	Undirected	1 135 890	2 987 624
email-Eu-core	Directed	1 005	25 571
com-Amazon	Undirected	334 000	925 000
wiki-Elec	Bipartite (directed)	7 000	100 000

2.4.1 On Signed Networks

A signed network is a network where edges can have positive and negative weights.

I should note that the bipartite wiki-Elec network is a signed network (where edge weights can be both negative and positive). I do not quite understand the implications of this for the algorithms I've implemented, therefore I have taken an extra step and changed all weights to positive using the following method:

- Find the minimum (most negative) edge weight.
- Add (actually subtract, since its negative) the minimum weight to every edge weight.

This process ensures that all weights are positive numbers and that the algorithms above work correctly.

2.4.2 Experimentation Hardware

All experiments listed below are conducted on a Dell Workstation with the following hardware specifications:

CPU: Intel Xeon W-2145 (11M Cache, 8 Cores, 16 Threads)

Memory: 32.0 GB DDR4, 2900Mhz Memory

2.4.3 An Introduction to the SNAP Library

The SNAP library [Les19], which stands for Stanford Network Analysis Platform, provides high quality datasets as well as classes and interfaces to efficiently use them. The interfaces are available in Python and C++, in this work I have used C++ for all of my implementations and therefore that's what I will be examining here.

The data structures supported by SNAP can be examined under two main categories: graphs and networks. Both of the data structures supported can be directed, undirected, weighted or unweighted. Various other types such as bipartite graphs or signed networks are also supported.

The functions I have listed under the Implementations section make use of the SNAP library classes and methods (I frequently convert SNAP variables representing graphs to C++ vectors mostly for convenience).

I will briefly explain some of the methods by which I have used SNAP library classes in my work. Firstly, I should list the ways in which these classes can be created and manipulated:

Creating and Manipulating SNAP Graphs and Networks:

```
    PNGraph G = TNGraph::New();
```

The prefix T stands for type and P for pointer. Therefore the object PNGraph is a pointer to the beginning of the TNGraph object G, iterators are provided for the Graph pointers in the classes.

There are many other important methods of these classes that I have worked with. As I have mentioned before, I worked mostly with adjacency matrix representations in 2D vector form in C++. Therefore my usage of most of these methods comes down to converting them to 2D vectors and passing them into some other function that does the job.

Below are some of the significant methods that I've used and their use cases:

```
PNGraph SubGraph = TSnap::GetSubGraph(G, TIntV::GetV(0,1,2));

{ TFOut FOut("test.graph"); G->Save(FOut); }
{ TFIn FIn("test.graph"); PNGraph G2 = TNGraph::Load(FIn); }

TSnap::SaveEdgeList(G2, "test.txt", "Save as tab-separated list of edges");
PNGraph G3 = TSnap::LoadEdgeList("test.txt", 0, 1);
```

Here the function in the first line returns the subgraph induced by vertices 0,1 and 2. It goes without saying that this was extremely useful for me (I do regularly need to work with induced graphs). The second and third lines are used to save to and read from a binary file (.graph format) and the fourth are fifth lines are used to read from and write to a txt file.

2.4.4 Experimental Results

Below are the results of my experiments conducted on the datasets listed in the table under 4.2.

Table 1: Number of Edges and Vertices in the K-Clique DSP for different values of K

Graph Name	**K**	**Vertex Count**	**Edge Count**	**K-Clique Density**	**K-Clique Count**
com-Youtube	3	1742	301 921	93.508	162 891
com-Youtube	4	1643	482 132	46.263	76 010
email-Eu-Core	3	643	15 021	14.504	9 326
email-Eu-Core	4	702	18 382	7.105	5 023
email-Eu-Core	5	121	3 792	1.694	205
com-Amazon	3	1992	102 321	71.423	142 275
com-Amazon	4	392	23 008	15.722	6 163

Here are a few observations that can be made from this table:

- com-Amazon and com-Youtube have similar distributions as far as can be seen from the K-Clique DS results, however due to the huge size of these graphs, I was not able to experiment with them extensively.
- As K increases the size of the K-Clique DS seems to decrease (this is not a strict necessity but the agrees with intuition).
- The maximum K-Clique Density seems to decrease as the value K goes up (this also agrees with intuition).

Experimentation was extremely slow for higher value of K and therefore I needed to keep K to small values. The computation times for each of the runs above are listed as follows:

Table 2: Computation times for the experiments in Table 1.

Graph Name	**K**	**Computation Time**
com-Youtube	3	13 minutes and 17 seconds
com-Youtube	4	26 hours 28 minutes and 32 seconds
email-Eu-Core	3	4 minutes and 56 seconds
email-Eu-Core	4	1 hour 10 minutes and 1 second
email-Eu-Core	5	36 hours 53 minutes and 10 seconds
email-Eu-Core	6	Cancelled after 48 hours of computation.
com-Amazon	3	12 minutes and 3 seconds
com-Amazon	4	24 hours 57 minutes and 44 seconds

The computation times clearly rise exponentially in the number of vertices, I was able to find the 5-Klique DS for the email-Eu-Core graph due to the fact that it only has 1000 vertices.

To further demonstrate the exponential nature of the problem for different values of K, I have drawn the graph below for the K=3,4,5 cases of the email-Eu-Core experiments:

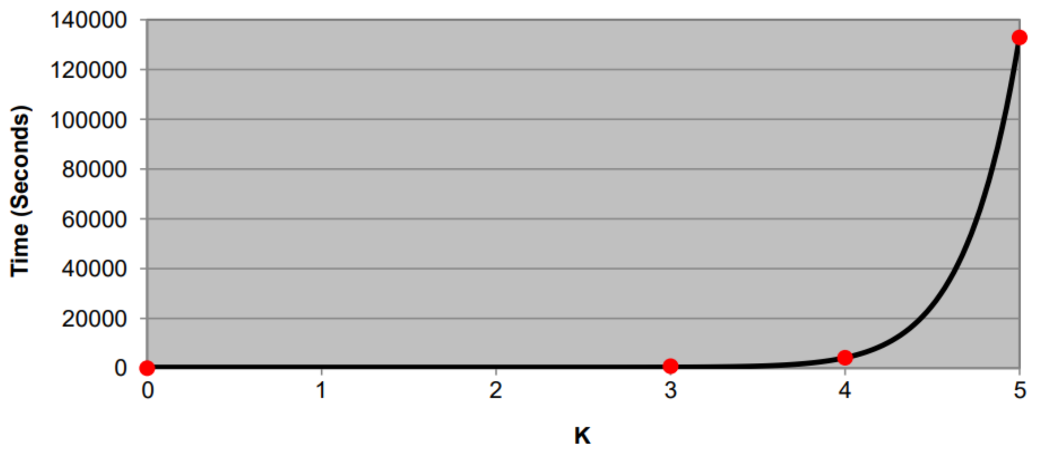

Figure 4: Computation times for the email-Eu-Core experiments for different values of K.

3 Organization

MIDDLE EAST TECHNICAL UNIVERSITY

Department of Computer Engineering

For this internship program, I worked as a research intern at the Department of Computer Engineering, Middle East Technical University. My supervisors were professors P. Karagoz and I. H. Toroslu.

3.1 Organization and Structure

Besides being an institution for higher education, the Department of Computer Engineering houses some of the best researchers and labs in the country. Throughout this internship, I practiced the research guidelines of the department (statements about research ethics, concerning plagiarism etc.).

I was in contact with my supervisors throughout the internship who answered any questions I had and with whom I met on at least a weekly basis.

3.2 Methodologies and Strategies Used in the Company

Due the internship program being research oriented, my methodologies are in fact research methodologies practiced at the Department of Computer Engineering, METU. However, for this internship program specifically, these can be listed as follows:

- Meet with supervisors for initial directives to resources and locations for a literature survey.
- Conduct a literature survey, get feedback regularly
- Present the workplan for implementation and testing of the picked work
- Implementation
- Initial testing, experiments and supervisor feedback for the implementation

- Complete and document experimentations
- Composure of a project report

The steps listed above were mostly followed with some place for overlap. As an example, I did continue my literature survey partly after a workplan was set (to provide an extended context for my work).

4 Conclusions and Future Work

As future work the applications of the K and PQ biclique problems discussed and implemented as part of this report will be examined within the following semester. One of the many possible applications of these algorithms is the detection of communities in social networks and other different graphs with node-to-node connection information.

One other thing that can be examined in the future is a set of heuristic algorithms proposed in the future work section of [Tso15] as well as in [Mit+15]. These algorithms will likely end up being much faster and may allow us to extract densest subgraphs for higher values of K.

References

[Gol84] A. V. Goldberg. *Finding a Maximum Density Subgraph.* Tech. rep. UCB/CSD-84-171. EECS Department, University of California, Berkeley, 1984. URL: `http://www2.eecs.berkeley.edu/Pubs/TechRpts/1984/5956.html`.

[Boy06] Yuri Boykov. *Boost C++ Libraries: Boykov Kolmogorov Max Flow Implementation.* 2006. URL: `https://www.boost.org/doc/libs/1_54_0/libs/graph/doc/boykov_kolmogorov_max_flow.html` (visited on 09/12/2019).

[Uno06] Takeaki Uno. *K-clique Lister: An Implementation of Makino and Uno Algorithm.* 2006. URL: `http://research.nii.ac.jp/~uno/codes.htm` (visited on 09/12/2019).

[BKV12] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. "Densest Subgraph in Streaming and MapReduce". In: *Proc. VLDB Endow.* 5.5 (Jan. 2012), pp. 454–465. ISSN: 2150-8097. DOI: `10.14778/2140436.2140442`. URL: `http://dx.doi.org/10.14778/2140436.2140442`.

[CS12] Jie Chen and Yousef Saad. "Dense Subgraph Extraction with Application to Community Detection". In: *IEEE Trans. on Knowl. and Data Eng.* 24.7 (July 2012), pp. 1216–1230. ISSN: 1041-4347. DOI: `10.1109/TKDE.2010.271`. URL: `http://dx.doi.org/10.1109/TKDE.2010.271`.

[Tso14] Charalampos E. Tsourakakis. "A Novel Approach to Finding Near-Cliques: The Triangle-Densest Subgraph Problem". In: *CoRR* abs/1405.1477 (2014). arXiv: `1405.1477`. URL: `http://arxiv.org/abs/1405.1477`.

[Mit+15] Michael Mitzenmacher et al. "Scalable Large Near-Clique Detection in Large-Scale Networks via Sampling". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* KDD '15. Sydney, NSW, Australia: ACM, 2015, pp. 815–824. ISBN: 978-1-4503-3664-2. DOI: `10.1145/2783258.2783385`. URL: `http://doi.acm.org/10.1145/2783258.2783385`.

[Tso15] Charalampos Tsourakakis. "The K-clique Densest Subgraph Problem". In: *Proceedings of the 24th International Conference on World Wide Web.* WWW '15. Florence, Italy: International World Wide Web Conferences Steering Committee, 2015, pp. 1122–1132. ISBN: 978-1-4503-3469-3. DOI: `10.1145/2736277.2741098`. URL: `https://doi.org/10.1145/2736277.2741098`.

[Les19] Jure Leskovec. *Stanford Network Analysis Project: SNAP*. 2019. URL: `https://snap.stanford.edu/snap` (visited on 09/12/2019).