ALGORITHMS FOR COMPUTING EDGE-CONNECTED SUBGRAPHS

A Thesis

by

XU WEN

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Sergiy Butenko
Committee Members,	Lewis Ntaimo
	Ursula Müller-Harknett
Head of Department,	Mark Lawley

December 2018

Major Subject: Industrial Engineering

ABSTRACT

This thesis concentrates on algorithms for finding all the maximal k-edge-connected components in a given graph $G = (V, E)$ where V and E represent the set of vertices and the set of edges, respectively, which are further used to develop a scale reduction procedure for the maximum clique problem. The proposed scale-reduction approach is based on the observation that a subset C of $k + 1$ vertices is a clique if and only if one needs to remove at least k edges in order to disconnect the corresponding induced subgraph $G[C]$ (that is, $G[C]$ is k-edge-connected). Thus, any clique consisting of $k + 1$ or more vertices must be a subset of a single k-edge connected component of the graph. This motivates us to look for subgraphs with edge connectivity at least k in a given graph G, for an appropriately selected k value.

We employ the method based on the concept of the auxiliary graph, previously proposed in the literature, for finding all maximal k-edge-connected subgraphs. This method processes the input graph G to construct a tree-like graphic structure A, which stores the information of the edge connectivity between each pair of vertices of the graph G. Moreover, this method could provide us the maximal k-edge-connected components for all possible k and it shares the same vertex set V with the graph G.

With the information from the auxiliary graph, we implement the scale reduction procedure for the maximum clique problem on sparse graphs based on the k-edge-connected subgraphs with appropriately selected values of k. Furthermore, we performed computational experiments to evaluate the performance of the proposed scale reduction and compare it to the previously used k-core method. The comparison results present the advancement of the scale reduction with k-edge-connected subgraphs. Even though our scale reduction algorithm based has higher time complexity, it is still of interest and deserves further investigation.

DEDICATION

To all the people who have helped me, who come into my life and inspire me to a be better person.

ACKNOWLEDGMENTS

I would like to express my deepest gratefulness to my advisor, Professor Sergiy Butenko. He has inspired my great interest in Linear Programming and Network Flows areas. Besides the courses, he has helped even more in research. He encouraged me to choose the research topic and always gave me some advises when I was stuck in some points. Without his persistent guidance, encouragement and help this thesis could not be completed.

I would like to thank my committee members, Professor Lewis Ntaimo and Professor Ursula Müller-Harknett, whose valuable advice and support makes this thesis possible as well.

I thank Mykyta Makovenko, who helped me in coding with C++. I am also grateful to all my friends and classmates at Industrial & Systems Engineering of Texas A&M University. The time here is one of most valuable and precious treasures in my life.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Sergiy Butenko and Professor Lewis Ntaimo of the Department of Industrial and Systems Engineering and Professor Ursula Müller-Harknett of the Department of Statistics.

The experiments depicted in Chapter 4 were conducted in part by Joachim Haakonsen from the Univeristy of Bergen for his thesis, and by Anurag Verma, Austin Buchanan and Sergiy Butenko of the Department of Industrial & Systems Engineering and were published in 2015 in an article listed in the INFORMS Journal on Computing. The data analyzed for Chapter 4 comes from the SuiteSparse Matrix Collection (former UF Sparse Matrix Collection) by Tim Davis.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

No outside funding was received for the research and compilation of this document.

NOMENCLATURE

MCP	Maximum Clique Problem
BFS	Breadth-First Search
DFS	Depth-First Search
LP	Linear Program

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This thesis focuses on algorithms for computing maximal k-edge-connected subgraphs in a given graph and the use of such subgraphs in solving the classical maximum clique problem. We deal with a simple undirected graph $G = (V, E)$, where $V = \{v_1, ...v_n\}$ and $E = \{(v_i, v_j) : v_i \neq v_j\}$ represent the set of vertices and the set of edges, respectively. The number of vertices and edges are denoted by $|V|$ or n and $|E|$ or m, respectively. If $H = (U, F)$, where $U \subseteq V$ and $F \subseteq E$ and all edges in F have both endpoints in U, then H is a subgraph of G. If F includes all the edges from E that have both endpoints in U then H is the subgraph of G *induced* by U, denoted $G[U]$.

Two vertices are said to be *connected* when there exists one or more paths from one vertex to the other. A graph is called *connected* when every pair of vertices are connected; otherwise, the graph is *disconnected*; see Figure 1.1.

The *vertex connectivity*, denoted by $\kappa(G)$, is the minimum number of vertices which must be eliminated to disconnect the graph. The edge connectivity $\lambda(G)$ is the least number of edges that must be deleted for disconnecting the graph. Figure 1.2 shows examples of a 2-edge-connected graph and a 3-edge-connected graph. Moreover, the local vertex connectivity, denoted by $\kappa_G(s, t)$, is the minimum number of vertices that must be removed to destroy all paths from the vertex s to

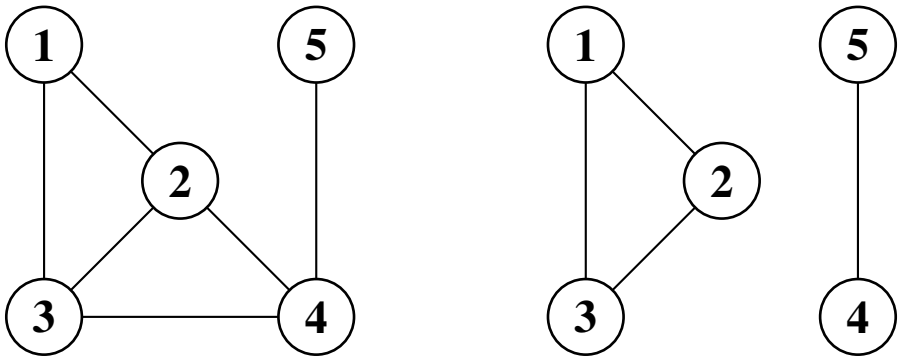

Figure 1.1: A connected graph (left) and a disconnected graph (right).

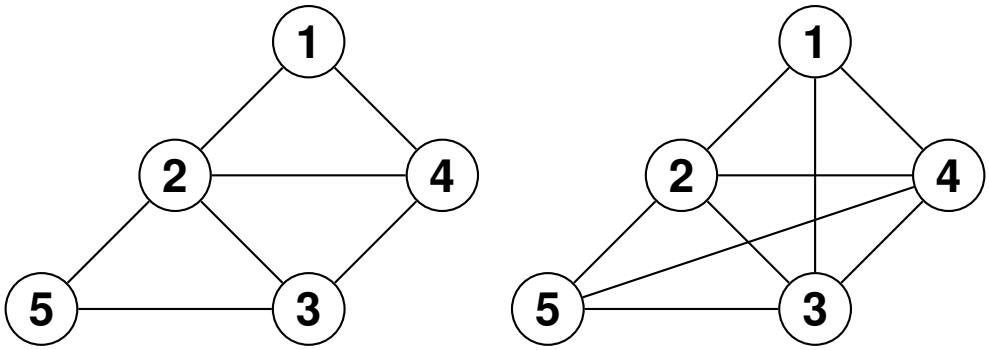

Figure 1.2: A 2-edge-connected graph (left) and a 3-edge-connected graph (right).

the vertex t, and the local edge connectivity $\lambda_G(s,t)$ is the least number of edges to be removed so that no path from the vertex s to the vertex t left.

Introduced by Luce and Perry [2], the clique concept formalizes the notion of a group of people who all know each other in graph-theoretic terms. Since then, this concept has been playing a more and more important role in Graph Theory, Computer Science, and Operations Research.

Formally, a subset of vertices C of the graph G is a clique if any two vertices from C are connected by an edge; see Figure 1.3 for an illustration. In particular, C is a maximal clique if it is not the subset of a larger clique, while C is a maximum clique if there does not exist a clique whose cardinality is larger than $|C|$. Moreover, the cardinality of a maximum clique in the graph G is called the clique number of graph G, written as $\omega(G)$.

A clique can be equivalently characterized using the so-called *elementary clique-defining prop-*

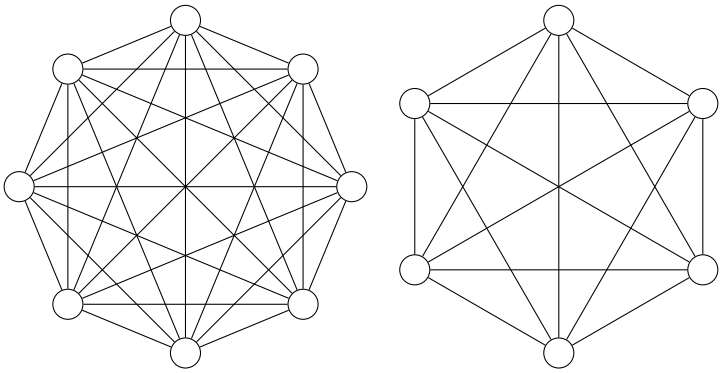

Figure 1.3: Example of cliques consisting of 8 and 6 vertices, respectively.

erties, which provide alternative definitions of a clique using other graph-theoretic concepts, such as distance, diameter, degree, etc. Pattillo, Youssef and Butenko [1] proposed a framework that uses such elementary clique-defining properties for describing a variety of *clique relaxation models*, aiming to address the overly restrictive nature of a clique in modeling clusters in complex networks as illustrated in Table 1.1.

Table 1.1: Alternative clique definitions based on elementary clique-defining properties [1].

Parameter	**Definition**
Distance	$d_G(v, V') = 1$,for any $v, V'; \in C$
Diameter	$diam(G[C]) = 1$;
Domination	$D = \{v\}$ is a dominating set in $G[C]$,for any $v \in C$;
Degree	$\delta(G[C]) = \|C\| - 1$;
Density	$\rho(G[C]) = 1$;
Independence Number	$\alpha(G[C]) = 1$;
Vertex Cover Number	$\tau(G[C]) = \|C\| - 1$;
Chromatic Number	$\chi(G[C]) = \|C\|$;
Clique Cover Number	$\overline{\chi}(G[C]) = 1$;
Vertex Connectivity	$\kappa(G[C]) = \|C\| - 1$;
Edge Connectivity	$\lambda(G[C]) = \|C\| - 1$;

In particular, the connectivity of graph can also be used to express an elementary clique-defining property. The elementary clique-defining properties based on the edge connectivity is obtained using the following observation. If a subset of vertices C is a clique, then the corresponding induced subgraph has the edge connectivity of $|C| - 1$. Thus, the graph $G[C]$ requires removal of at least $|C| - 1$ edges to disconnect it,

$$\lambda(G[C]) = |C| - 1.$$

Here, $\lambda(G)$ denotes the edge connectivity of graph G and $G[C]$ is the subgraph induced by C in G. Moreover, if we have a subset of vertices C such that $G[C]$ has the edge connectivity $|C| - 1$, then C is a clique. Hence, we can state the following elementary clique defining property.

Proposition: A subset C of k vertices is a clique if and only if $\lambda(G[C]) = k - 1$.

In this thesis we are interested in finding large subgraphs of a given graph G with edge connectivity of at least k in G, where k is a fixed positive integer. This corresponds to relaxing the elementary clique defining property in the proposition above to a subset of vertices of an arbitrary cardinality. Moreover, the connectivity of the subset C is computed with respect to graph G rather than the induced subgraph $G[C]$, making the k-edge-connected subgraph a *weak* clique relaxation according to the terminology used in [1].

Definition: Given a simple, undirected graph $G = (V, E)$ and a fixed positive integer k, a subset of vertices C is called a k-edge-connected set if $\lambda_G(s, t) \geq k$ for any pair of vertices $s, t \in C$. A k-edge-connected set is maximal (corresponds to a k-edge-connected component) if it is not s subset of a larger k-edge-connected set in G.

Problem Statement: Given the graph $G = (V, E)$ and the positive integer k, the maximum k-edge-connected set problem is to find a k-edge-connected set of maximum cardinality in the graph. This maximum cardinality will be denoted by $\bar{\omega}_\lambda(G)$.

The objectives of this thesis include:

1. Designing, implementing, analyzing, and testing algorithms for the maximum k-edge-connected set problem and the problem of finding all maximal k-edge-connected sets in a graph;

2. Developing methods for calculating an appropriate k in a practical setting;

3. Establishing methods for using k-edge-connected sets (with a suitable choice of k) in scale-reduction and upper-bounding schemes for the maximum clique problem.

Intuitively, the choice of k should be related to the clique number $\omega(G)$. Indeed, we are guaranteed to have a nonempty k-edge-connected set in the graph if $k < \omega(G)$, since a maximum clique is a k-edge-connected set in this case. Picking a higher value of k could result in infeasibility of the maximum k-edge-connected set problem. On the other hand, picking a small value for k may result in a very large maximal k-edge-connected sets, lacking in cohesiveness properties expected

of an interesting network cluster in practice. Determining a suitable, practically relevant value for the parameter k is an interesting and challenging problem.

On the one hand, as we just pointed out, the clique number can be useful in selecting an appropriate value of k. On the other hand, computing maximal k-edge-connected sets for different values of k could be used for designing scale-reduction and upper-bounding schemes for the maximum clique problem. More specifically, if we know that a clique of cardinality ω exists in the graph (e.g., found using some effective heuristic for the maximum clique problem), then all cliques of cardinality at least ω, including all the maximum cliques, are guaranteed to be subsets of the k-edge connected components of the graph with $k = \omega - 1$. Since the k-edge connected components correspond to maximal k-edge-connected sets, this implies that the problem of finding a maximum clique in the graph reduces to the problem of finding a maximum clique in each maximal k-edge-connected set of the graph. As for upper-bounding, selecting a (high) value of k, such that the graph has no k-edge-connected sets, shows that there is no clique of cardinality $k + 1$ in the graph; i.e., such a k provides an upper bound on the clique number.

We will use the following definitions in this thesis. A *tree* is an simple undirected graph without cycles, in which any pair of vertices are connected by only one path. A union or a collection of trees is called a *forest*.

A *cut* is defined as a partition which splits the set of vertices into two groups, usually noted as $[S, \overline{S}]$. An *s-t* cut is a cut that separates vertices s,*the source vertex*, and t,*the sink vertex*, so that they belong to two different parts.

The *edge density* (or, simply, *density*) is one of the characteristics of a graph. A graph is considered to be *dense* if its number of edges is relatively high and the graph is closer to a complete graph; it is considered to be *sparse* if the number of edges in the graph is comparatively small. For an undirected graph G, its density $\rho(G)$ is calculated by:

$$\rho(G) = \frac{2|E|}{|V|(|V| - 1)}.$$

The chapters of this thesis are structured as follows. The remainder of this chapter discusses some applications and provides a brief literature review on k-edge-connected set and the maximum clique problem. Chapter 2 describes the auxiliary graph method for finding maximal k-edge-connected sets. The new scale reduction approach based on k-edge-connected sets for the maximum clique problem is detailed in Chapter 3. Chapter 4 reports the computational results of finding k-edge-connected sets and the effect of the auxiliary graph in a scale reduction procedure for the maximum clique problem. Chapter 5 summarizes the results we have obtained and discusses several further research directions.

1.1 Applications

The concept of a k-edge-connected set is a flexible and important bridge connecting the edge connectivity and clique-like structures in subgraphs. Thus, k-edge-connected set is of a great potential value for graph-based analysis applications, especially for network structure detection and connectivity evaluation in areas like social networks, transportation systems, web networks, and modern biology research.

One application of the k-edge-connected set is detecting clique-like structures (clusters) whose members are densely related to each other. In some fields which can be represented by network systems, like railways, shipping schedules, airlines, etc., clique-like structures can show the frequent commuted city blocks. In the social and economic networks, clique-like structures stand for the cohesive communication communities or frequent trading groups. Moreover, clique-like structures symbolize the specific structures for cells in some research areas for the modern biology.

Another application refers to the evaluation of the connectivity for a network. "Graphs operate at the interface of structure and function" [3]. We could consider the vertices in V as the individuals of interest, and the edges in E as the relationships between the individuals. Therefore, the graph-theoretic connectivity analysis can operate at both entire network and individual perspectives. An example from land ecology can account for both perspectives [4]. For applications at the level of networks, functional connectivity analysis has been employed for evaluating connectivity conversion of a specific species. From the individual level, the functional connectivity analysis is

also a great tool for selecting the most important patches in a habitat network.

Furthermore, if we know the edge connectivity of the subgraphs, then the upper bound of edge connectivity for graph G is also known. As a result, the sampling approximation and some other statistical methods can be applied for providing reasonable estimations of the edge connectivity of graph G through computing the edge connectivity of subgraphs. This process is less time-consuming and more flexible for implementation than interpreting measurement data of the entire graph.

1.2 Literature Review

1.2.1 Edge Connectivity

Connectivity is one of the basic definitions of graph theory as well as a significant measure of the reliability and stability of the network, in Mathematics, Computer Science and Operation Research areas [5].

The concept of a k-edge-connected graph was first introduced by Jordan in 1869 [6]. During the past 50 years, the methods for searching k-edge-connected components and computing edge-connectivity algorithms have been extensively researched and studied.

The first method which has been applied to compute the edge-connectivity is depth-first search (DFS). In 1972, Tarjan proposed an algorithm with DFS for finding all components with $\lambda(G) = 2$ in a graph G in linear time [7]. In a few years, the problem of computing 3-edge-connected components was solved by reducing the edge connectivity problem to the vertex connectivity problem and employing Tarjan's method on the reduced problem.

In 1975, Even and Tarjan [8] used the *min-cut* to compute the edge-connectivity.

Theorem 1 (Max-Flow Min-Cut Theorem)**.** *The maximum value of the flow from a source vertex s to a sink vertex t in a capacitated network equals the minimum capacity among all $s - t$ cuts.*

Most edge connectivity algorithms proposed later were based on this result, according to which the min-cut can be calculated using the max-flow algorithms. This approach computes $\lambda(G)$ of a graph $G = (V, E)$ through comparing all possible values of $\lambda(v, w)$, $v, w \in V$ and finding the

minimum one:

$$\lambda(G) = \min\{\lambda(v, w) \,|v, w \, \in V\}$$

Esfahanian and Hakimi [9] utilized the structure of the spanning tree to reduce the number of calls to the max-flow algorithm to less than $n/2$. In 1987, the dominating set method was introduced by Matula [10], who succeeded in reducing the complexity of edge connectivity algorithms to $O(mn)$, the same complexity level of procedure for computing $\lambda(v, w)$. Moreover, any improvement of the max-flow algorithms naturally contributes to the improvement for computing the edge connectivity as well. In 1992, Nagamochi and Watanabe [11] proposed an algorithm which can capture all k-edge-connected components in $O(mn(min(k, m, \sqrt{n}))$ time.

In general, the basic cut framework for finding k-connected components or k-edge-connected sets could be improved by making reductions on the size of graph and reducing the number of calls to the min-cut algorithm. Chang et al. [12] proposed a method named graph decomposition tree, in which the leaves represent the corresponding subgraphs generated by iteratively cutting the edges in the min-cut. This method runs in $O(hlm)$ time (h is the level of the decomposition tree and l is a value smaller than m).

The cut tree (Gomory-Hu tree) is another way of summarizing the information regarding the minimum cut for every pair of vertices [13]. The weight of each edge in the cut tree describes the cardinality of the minimum cut between corresponding pair of vertices in the original graph G. Moreover, the auxiliary graph introduced by Wang et al. [14] is also a tree structure, but the information on the min-cut is replaced with the information on edge connectivity in this case. These two methods store key information regarding the structure of graph, saving large amount of time from repetitive computing applied in other cases.

1.2.2 The Maximum Clique Problem

The maximum clique problem (MCP) is one of Karp's 21 original NP-hard problems [15]. Hence, there is no polynomial-time approach that could solve this problem, unless $P = NP$.

However, the maximum clique problem has been broadly applied for numerous practical set-

tings, i.e., economic and financial networks, scheduling optimization, bioinformatics research, transmission analysis, etc. Over the past years, a large variety of methods for solving the MCP have been proposed, and most of them could be divided into two types [16].

The first type is represented by effective exact methods which ensure optimality of the result but are of huge computation complexity. Moreover, the majority of algorithms of this type are developed with the branch-and-bound (B&B) structure. A famous algorithm of this type was developed by Carraghan and Pardalos in 1990 [17]. Many other B&B algorithms have been proposed and succesfully tested, including B&B algorithm by Babel and Tinhofer [18], Östergad's algorithm [19], MCQ by Tomita and Seki [20], MCR by Tomita and Kameda [21], MCS by Tomita et al [22] among many others.

The algorithms of the second type utilize the heuristic and metaheuristic methods, which could run with reasonable time complexity but cannot guarantee the optimal solutions. Some of algorithms for MCP of this type are greedy algorithms, like QUALEX-MS by Busygin [23] and the deep adaptive greedy search (DAGS) by Grosso et al. [24]. Besides, some other heuristic approaches, like local search heuristics and various metaheuristics have been proposed [25].

2. THE METHOD OF AUXILIARY GRAPH

To compute the maximal k-edge-connected sets, the algorithms proposed by Wang et al. [14] based on the auxiliary graph, will be used.

Given a directed or undirected, simple or multi-graph $G = (V, E)$, *Auxiliary Graph* $A = (G, E_A)$ is an undirected tree which stores information of graph G in the weight of edges of A. The information of graph G is the number of edges from the min-cut between each pair of vertices $\{v_i, v_j | v_i, v_j \in E, v_i \neq v_j\}$.

According to the paper by Wang et al. [14], the auxiliary graph is built on the basis of the hierarchical structure of k-edge-connected components. Every k-edge-connected component is actually the union of $(k+1)$-edge-connected components. Hence, a natural tree-like graphic structure A can be developed, in which the weight of an edge $\{s, t\}$ represents the edge-connectivity $\lambda_G(s, t)$.

Like most of the algorithms for searching k-edge-connected subgraphs or computing the edge-connectivity, this algorithm is also dependent on the max-flow (or the min-cut). With the demand for computing the edge connectivity between every pair of vertices, the min-cut algorithm executes $2n - 2$ times. Thus, the time complexity for constructing the auxiliary graph is $O(Fn)$, where F represents the time for executing one round of s-t max-flow algorithm. For example, for the algorithm proposed by Goldberg and Tarjan [26], this time is given by $F = O(n^3)$. In the implementation part of this chapter, the maximum flow algorithm proposed by Edmonds-Karp [27] with running time of $O(nm^2)$ will be used. After the construction phase, a traversal of the auxiliary graph in time $O(n)$ is required for determining the k-edge-connected components.

2.1 Auxiliary Graph

As mentioned above, the auxiliary graph is a tree-like structure built on the basis of hierarchical structure of k-edge-connected components [14]. A k-edge-connected component in a graph $G = (V, E)$ is a maximal subset of vertices $V' \subseteq V$ such that the edge connectivity between any two vertices in V' is at least k, thus every k-edge-connected component is actually the union of $(k+1)$-

Algorithm 1 Max FLow Algorithm by Edmonds-Karp [27]

Input: Given a flow graph G, source vertex s, sink vertex t
Create a residual graph G, from G, and set $maxF = 0$
while $\exists$ a *path* from source to sink in G_r **do**
    set $f = \infty$
    for all e $\in$ path **do**
        if $c_e < f$ **then**
            $f = c_e$
        end if
    end for
    $maxF+ = f$
end while
return $maxF$

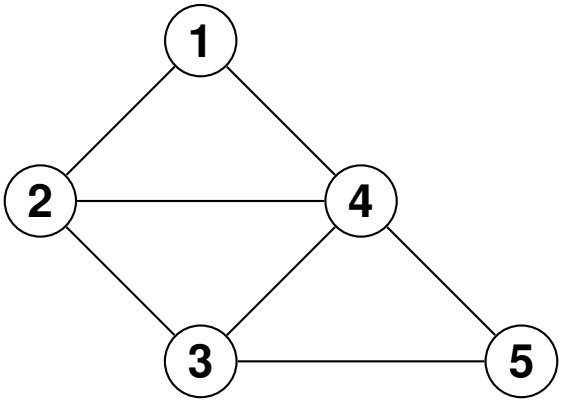

Figure 2.1: An example of undirected connected graph.

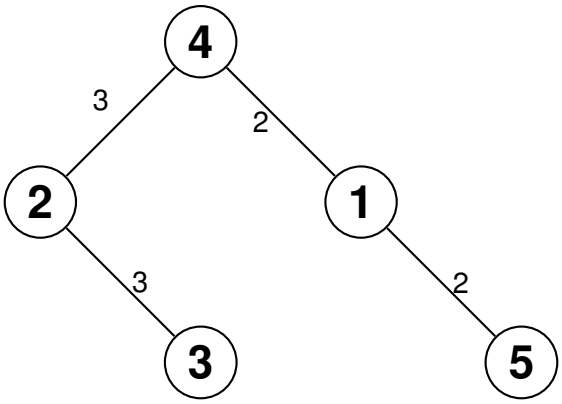

Figure 2.2: The corresponding auxiliary graph for the graph in Figure 2.1 described in a rooted tree.

Table 2.1: Information from the auxiliary graph.

Weight of Edges	Corresponding Set of Vertices
1	$\{1, 2, 3, 4, 5\}$
2	$\{1, 2, 3, 4, 5\}$
3	$\{1\}, \{2, 3, 4\}, \{5\}$

edge-connected components (it should be noted that a set consisting of a single vertex (singleton) is assumed to be k-connected for any k). The collections of k-edge-connected components compose a hierarchical structure.

Consider the example graph in Figure 2.1. Evidently, when $k = 1$, the 1-edge-connected component is just the set of V itself (assuming the graph is connected). When $k = 2$, the 2-edge-connected component is $\{1, 2, 3, 4, 5\}$ which is the union of all the 3-edge-connected components $\{1\}, \{2, 3, 4\}$, and $\{5\}$. Furthermore, a singleton which is the set with only one element is also the union of itself. However, singletons are not the connected components, so we could ignore them.

The natural hierarchical structure could also be explained from the perspective of the traditional cut framework. The cardinality for the min-cut is always getting higher and higher when we iteratively cut out and search in the subgraphs with sufficient number of vertices, i.e., $|V| \geq k, |E| \geq k$, generated from previous cutting. It is known that the cardinality of the min-cut for the entire graph G is always the smallest one among all the s-t cuts (by the definition of the min-cut). Thus, the smallest carnality of remaining s-t-cuts is definitely higher than that of the min-cut.

The definition of the auxiliary graph tells us that A uses the same vertex set V with the graph $G = (V, E)$. In the initial stage, the edge set of the auxiliary graph is empty $E_A = \{\}$. The edges with weights are continuously added to A for building a tree-like graph and avoiding cycles. If A_k is the spanning forest of A based on edges of weight at least k, then the collection of vertex sets of connected components in A_k exactly corresponds to the collection of k-edge-connected components of graph G (with the components containing k or less vertices split into singletons). Furthermore, we could obtain A_{k-1} by adding the edges with weight of $k - 1$. A_{k-1} could lead

us to all the $(k-1)$-edge-connected components of graph G. It is obvious that the maximum weight of all the edges, denoted as h, is the minimum k value such that all $(k+1)$-edge-connected components are singletons. The fact that two graphs share the vertex set makes this auxiliary graph method much more efficient and user-friendly. In particular, we could create the sought induced subgraphs using the auxiliary graph.

Furthermore, the auxiliary graph as a compact storage of the information, can be applied for so many cases, i.e. for heuristic searching and optimization based on local connectivity. In particular, we will use it for scale reduction procedure for the maximum clique problem in Chapter 3.

2.2 Description of the Algorithm

The input consists of a graph $G = (V, E)$, a randomly picked source vertex s, and a sink vertex candidate pool N. The CONSTRUCTION procedure firstly picks a sink vertex t from all possible candidates $N - \{s\}$. Then the max-flow algorithm runs for the min-cuts between s and t, (S, T) and (S', T') (the results of s-t max-flow and t-s max-flow). Here, it needs attention that the max-flow algorithm only runs on directed graph. Thus, each edge of an undirected graph G needs to be changed to two directed edges which point in the opposite directions. Next, the cardinalities of two min-cut are compared and the edge (s, t) is added to the graph A with weight of the smaller cardinality of two min-cuts. After that, the procedure calls itself with two versions of input: the first one is with s as source vertex and $N \cap S$ as sink candidate pool and the second is with t as source vertex and $N \cap T$ as sink vertex candidate pool. The CONSTRUCTION procedure only ends when $N = \{s\}$, that is, no sink vertex is available.

Wang et al. [14] proved the transitivity of the connectivity between edges, that the edge connectivity between two vertices is actually the minimum weight from all the edges on the path connecting them. This serves as theory foundation of retrieving edge connectives of graph G from the auxiliary graph A. For computing the edge connectivity of the graph G, we just need a traverse of A and find the minimum weight of all the edges; For k-edge-connected components, we are supposed to keep the edges with weight at least k and the collection of connected components with at least $k + 1$ vertices is exactly what we want.

Algorithm 2 Construction$(G(V, E), s, N)$ [14]

if $N = \{s\}$ **then**
  return
end if

Randomly pick a vertex t from $N - \{s\}$
$(x, S, T) := s$-t MAX-FLOW (G, s, t).
$(x', S', T') := s$-t MAX-FLOW (G, t, s).

if $x' \leq x$ **then**
  $x := x'$, $S := S'$,$T := T'$
end if
Add edge (s, t) with weight x to A
CONSTRUCTION$(G, s, N \cap S)$
CONSTRUCTION$(G, t, N \cap T)$

2.3 Implementation and Analysis of the Algorithm

2.3.1 Finding the Min-Cut

For finding the min-cut for graph G, we operate a maximum flow algorithm and a depth first search (DFS) [28]. With the input of graph G, the source vertex s and the sink vertex t, the maximum flow algorithm only runs when DFS could find next augmenting path. The next augmenting path is the shortest path from the source vertex s to the sink vertex t with a positive residual value. Then we try to push flow from the sink vertex t back to the source vertex s and keep updating the residual capacities of edges in that path. This maximum flow algorithm terminates when no path from s to t with positive residual value is left. After that, a DFS is used to find the cut $[S, T]$. It is known that the running time for the Edmonds-Karp maximum flow algorithm is $O(nm^2)$. The DFS here runs in time of $O(m + n)$. Hence, the running complexity for findig the min-cut is the same with Edmonds-Karp maximum flow algorithm of $O(nm^2)$.

2.3.2 Constructing the Auxiliary Graph

In this stage, we build a rooted tree for describing the auxiliary graph. This rooted tree could help us trace the vertices that could be connected in the original graph G. The auxiliary graph A

Algorithm 3 Minimum s-t cut [28]

Input: Given a flow graph $G = (V, E)$, a source vertex s, target vertex t
function MINIMUM s-t CUT(G, s, t)
set $flow = 0$, vertex set $S = \{\}$, vertex set $T = \{\}$
while BFS reaches t through edges with residual capacity > 0 **do**
set $x = t$
while $x \neq s$ **do**
choose a neighbour y of x closer to the source
set capacity of edge $(x, y) += 1$
set capacity of edge $(y, x) -= 1$
$x = y$
end while
end while
push s onto stack L
mark s
while L not empty **do**
set x to top element of L
remove top element from L
for all neighbours y of x **do**
if edge(x, y) capacity >0 and y not marked **then**
push y onto stack L
mark y
end if
end for
end while
for all $y \in V$ **do**
if y is marked and $y \in N_s$ **then**
Add y to S
else
if $y \in N_s$ **then**
Add y to T
end if
end if
end for
end function

begins with the vertex set V which is inherited from graph $G = (V, E)$, and an empty edge set $E' = \{\}$. Similar to what we discussed in previous section, a root vertex is chosen firstly and all the vertices are added as candidates to the pool of the sink N_{root}.

Then we iteratively choose a source vertex s and a sink vertex t from their own candidate pools (if not empty), find the minimum s-t cut, add the edge (s, t) with weight of cardinality of the min-cut to the graph A, and update the pools for the source and the sink vertices respectively.

The Construction procedure runs the min-cut function n times. Thus, the time complexity for the auxiliary graph algorithm is $O(n^2m^2)$ (the multiple of $O(n)$ and $O(nm^2)$).

Algorithm 4 Constructing the Auxiliary Graph [28]

Input: Given graph $G = (V, E)$
$A = (V, E')$ where $E' = \{\}$
Choose a root vertex for A
for all $x \in V$ **do**
    $N_x = \{\}$
end for
$N_{root} = V$
add *root* to Q
while Q is not empty **do**
    set s to first element of Q
    if size of $N_s < 2$ **then**
        remove first element of Q
    else
        set t to be a vertex from $N_s \backslash \{s\}$
        construct a flow graph F of G
        $C(c, S, T) =$ MINIMUM S-T CUT(F, s, t)
        add the directed edge(s, t) to E' with weight $= c$
        add t to Q
        for all $x \in N_s$ **do**
            if $x \in T$ **then**
                add x to N_t
                remove x from N_s
            end if
        end for
    end if
end while

3. NEW SCALE REDUCTION APPROACH FOR THE MAXIMUM CLIQUE PROBLEM

3.1 Scale Reduction for the MCP

Large sparse graphs are very common in real-life applications, like social networks, citation networks, communications, and web graphs.

The technique named “peeling" was proposed by Abello et al. [29], which could be considered as a preprocessing before doing the exact searching for maximum clique. The peeling procedure recursively deletes the vertices and the edges which are evidently not useful for the exact maximum clique algorithms, i.e. the vertices whose degrees are smaller than a certain number k. Because any vertex with degree lower than k could not exist in a clique with clique number of $k+1$ or higher. Thus, "peeling" could help us sharply reduce the size of the large sparse graph G, where most of the vertices are of low degrees.

Verma et at. [30] proposed a new method for finding the maximum cliques using a scale reduction based on k-community structures. A k-community is defined as a subset vertices V' such that every pair of endpoints for edges E' have at least k common neighbors in its induced subgraph $G' = (V', E')$.

3.2 Scale Reduction Algorithm Based on k-Community

In this algorithm, the lower bound of the clique number ω_{lower} is first captured by a greedy heuristic through searching for the maximum degree of vertices in the graph G. After that, the upper bound of the clique number ω_{upper} will be obtained through the linear or binary search in UpperBound$(G, \omega_{lower}, \Delta)$. Moreover, the UpperBound is found by incrementing k from ω_{lower} until that k-community is an empty set.

Then a scale reduction procedure ScaleReduction1 (G, ω_{upper}) is implemented for the maximum $(\omega_{upper} - 2)$-community of graph G which could dramatically cut down the size of $|V|$. If the resulting graph G_{upper} is small enough, an exact MCP algorithm can be employed for a ω_{lower}. Here, the exact maximum clique algorithm proposed by Östergard in 2002 [19] is used. The same

process composed of scale reduction and exact MCP algorithm for a ω_{lower} is also employed for each connected component of G_{lower}. Moreover, the highest clique number among G_{upper} and all the connected components of G_{lower} gives the clique number of graph G.

Algorithm 5 Algorithm to find a maximum clique of G [30]

$\omega_{lower} \leftarrow$ HeuristicClique
$\omega_{upper} \leftarrow$ UpperBound $(G, \omega_{lower}, \Delta)$
if $\omega_{lower} < \omega_{upper}$ **then**
  $G_{upper} \equiv (V_{upper}, E_{upper}) \leftarrow$ ScaleReduction1 (G, ω_{upper})
  if $|V_{upper}| \leq K$ **then**
    $\omega_{lower} \leftarrow$ max $\{\omega_{lower}$, FindMaxCliqueExact$(G_{upper})\}$
  end if
end if

if $\omega_{lower} < \omega_{upper}$ **then**
  $G_{lower} \equiv (V_{lower}, E_{lower}) \leftarrow$ ScaleReduction2 (G, ω_{lower})
  for each connected component $G' = (V', E')$ of G_{lower} **do**
    if $|V'| \leq K$ **then**
      $\omega_{lower} \leftarrow$ max $\{\omega_{lower}$, FindMaxCliqueExact(G')
    end if
  end for
  if the clique number of each connected component is found **then**
    $\omega_{upper} \leftarrow \omega_{lower}$
  end if
end if
return $[\omega_{lower}, \omega_{upper}]$

3.3 Scale Reduction Based on k-Edge-Connected Sets

This thesis aims to replace the ScaleReduction procedure from original k-community based search to k-edge-connected set based search. Similar to the original peeling procedure, it selects the vertices which may be of use for the exact maximum clique algorithms, i.e. $(k-1)$-edge-connected set or k-edge-connected components from the auxiliary graph. Because any vertex from a subgraph whose edge connectiviy is less than k could never exist in a clique with clique number of $k+1$ or higher. Hence, this new “peeling" procedure could also make a help in shrinking the

size of the large sparse graph G which enables us to use the exact algorithm for the optimal result.

As mentioned before, all k-edge-connected sets (k is a fixed integer) and i-edge-connected sets with $i \leq k$, could be found using the auxiliary tree algorithm. Thus, G_{upper} and all connected components for G_{lower} can be found at the same time, which could save time from repetitive searching in ScaleReduction procedure.

Moreover, the upperBound$(G, \omega_{lower}, \Delta)$ can be achieved from finding the maximum weight for the edges in the auxiliary graph $A = (G, E_a)$, avoiding significant amount of memory and time for binary or linear search of the graph G. Hence, we could modify the original algorithm with combining two ScaleReduction procedures together and obtaining the upperBound of ω from the corresponding auxiliary graph as illustrated in Algorithm 6 below.

3.4 Implementing Scale Reduction Based on k-Edge-Connected Set

The algorithm is modified with a substitute of the k-edge-connected set to the k-core or k-community in the ScaleReduction procedure. Moreover, the linear search or binary search for the upper bound for the clique number has been removed because we could use the maximum k number captured from the auxiliary graph.

Furthermore, we expect to observe the performance of new ScaleReduction procedure for a large sparse graph G. The test result analysis and the comparison between different methods could motive us for new ideas for improvements.

3.4.1 Heuristic Clique Function

This greedy heuristic is used in [30]. It searches for a clique beginning from the vertex with the maximum degree. Then, we add all the neighbours of this vertex into candidate sets. After that, we repeat the same process for the elements in candidate set until is empty. If the size of candidate set is greater than a certain number, we record the degree of that elements; otherwise, the number of common neighbours between this element and the whole candidate sets is recorded.

Finally, the clique built on the vertex with maximum degree is found along with its size. Moreover, this clique number can be used as the lower bound in the maximum clique algorithm, ω_{lower}.

Algorithm 6 Algorithm to find a maximum clique of G

$\omega_{lower} \leftarrow$ HeuristicClique
$\omega_{upper} \leftarrow$ UpperBound $(A$,max$\{weight[edges]\})$
if $\omega_{lower} < \omega_{upper}$ **then**
  $G_{lower} \equiv (V_{lower}, E_{lower}) \leftarrow$ ScaleReduction (A, ω_{lower})
  for each connected component $G' = (V', E')$ of G_{lower} **do**
    if $|V'| \leq K$ **then**
      $\omega_{lower} \leftarrow$ max $\{\omega_{lower}$, FindMaxCliqueExact$(G')\}$
    end if
  end for
  if the clique number of each connected component is found **then**
    $\omega_{upper} \leftarrow \omega_{lower}$
  end if
end if
return $[\omega_{lower}, \omega_{upper}]$

Thus, the input number of k for the scale reduction procedure equals to $\omega_{lower} - 1$.

3.4.2 ScaleReduction with the k-Edge-Connected Sets

Given the information from the auxiliary graph, the subset of vertices involved in edges with weight at least k can be obtained. Hence, we could create the induced subgraphs of the graph G with that set of vertices. However, some induced subgraphs still cannot satisfy the need that the edge connectivity is at least k. Thus, one more step is required for removing the components with less than $k + 1$ vertices.

The original graph G is reduced through this process to the graph G', whose size can be achieved for comparing the peeling or scale reduction effect. Moreover, operating a BFS can trace all the connected components with edge connectivity no less than k for us.

An example with the input graph G shown in Figure 3.1, will be used to illustrate this process.

- Step 1:

  Build the corresponding auxiliary graph based on graph G and obtain the table summarizing the edge weight information (Table 3.1).

- Step 2:

Algorithm 7 Greedy heuristic algorithm for a maximum clique of G [30]

```
function FINDGREEDYCLIQUE(G)
    maxdegree := 0
    for i := 1 to n do
        if G.degree[i] > maxDegree then
            maxDegree := G.degree[i]
            next := i
        end if
    end for
    candidates := G.adj[next]
    maxclique := 0
    while candidates ≠ ∅ do
        maxdegree := 0
        next := candidate[0]
        for i := 0 to cand_size(cand_size := candidates.size) do
            if cand_size > 10000 then
                temp := G.degree[candidates[i]]
            else
                temp := |CommonNeighbour(candidates[i], candidates)|
            end if

            if temp > maxDegree then
                maxDegree := temp
                next := candidates[i]
            end if
        end for
        candidates := CommonNeighbourList(next, candidates)
    end while
    if Clique.size > maxclique then
        maxclique := clique.size
    end if
    return clique
end function FindGreedyClique(G)
```

Table 3.1: Information from the auxiliary graph

Weight of Edges	Corresponding Set of Vertices
1	{1, 2, 3, 4 ,5 ,6, 7}
2	{1, 2, 3, 4 ,5 ,6, 7}
3	{1, 2, 3, 4 ,7}, {5}, {6}
4	{1}, {2}, {3}, {4}, {5}, {6}, {7}

Given an input k (e.g., $k = 3$), create induced subgraph of graph G with the vertex set corresponding to edges of weight at least k, as illustrated in Figure 3.2.

- Step 3: Remove vertices with less than k neighbours and all the edges incident to them (Figure 3.3).

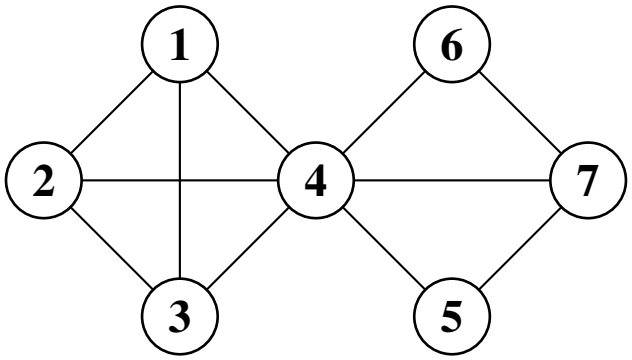

Figure 3.1: An input graph G

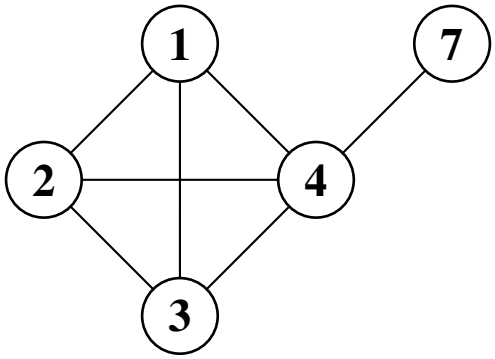

Figure 3.2: Created induced subgraph with $k = 3$

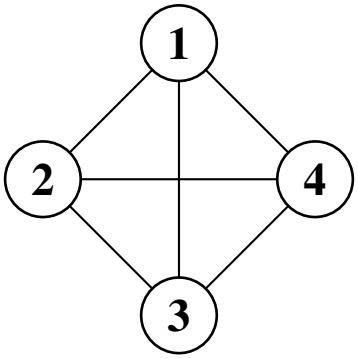

Figure 3.3: A 3-edge-connected set of graph G

4. COMPUTATIONAL RESULTS

This chapter describes the results of computational experiments conducted in order to 1) find maximal k-edge-connected sets 2) evaluate the effect of the auxiliary graph method on scale reduction for the maximum clique problem. First, we describe the experimental setting.

- Test Environment:

  Tools: C++ with MinGW (Minimalist GNU for Windows).

  Computer specs: Intel Xeon E5600 2.40GHz 2 Processors, 12.0GB RAM.

- Test Instances:

  A variety of graphs from the SuiteSparse Matrix Collection (former UF Sparse Matrix Collection) by Tim Davis have been used for testing. We conduct the tests on the sparse and connected graphs, mainly arising in the power industry. Table 4.1 provides the names, number of vertices, number of edges and the edge density of the graphs used for testing.

- Test Scenario:

  Applying the scale reduction approach, we first implement the greedy heuristic for a lower bound of the clique number ω_{lower} for the graphs we have chosen. Then we set the value of k to be $\omega_{lower} - 1$. Next, we compare the scale reduction effect of k-core peeling and the reduction based on k-edge-connected sets (k-ECS). In k-edge-connected peeling method, we report all connected components obtained in the residual auxiliary graph.

4.1 Scale Reduction Effect

The result of scale reduction with peeling methods based on k-edge-connected sets and k-core is shown is Table 4.5.

The comparison of number of vertices and number of edges in original graph G and in G'_{ECS}

Table 4.1: Graph cases for testing. Source: the SuiteSparse Matrix Collection (former UF Sparse Matrix Collection) by Tim Davis.

Graph Name	Number of Vertices (n)	Number of Edges (m)
bcspwr04	274	943
bcspwr05	443	1033
bcspwr08	1624	3837
bcspwr09	1723	4117
bcspwr10	5300	13571
power	4941	6594
494_bus	494	1666
685_bus	685	1967
1138_bus	1138	2596
qh768	768	2934
qh882	882	3354

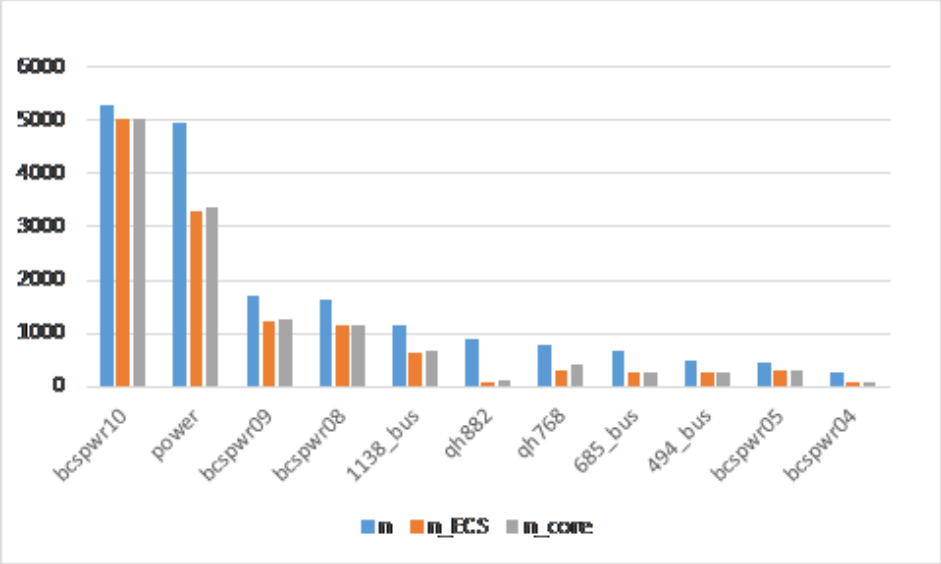

Figure 4.1: Scale reduction on the number of vertices.

are shown in Figure 4.1 and Figure 4.2 respectively. The scale reduction with k-ECS method shows an evident effect of cutting down the size of graphs.

Table 4.2: Scale reduction effect based on different peeling methods.

		G		G'_{ECS}		G'_{core}	
Graph	ω_{lower}	n	m	n	m	n	m
bcspwr04	6	274	943	66	239	73	259
bcspwr05	3	443	1033	309	405	313	460
bcspwr08	3	1624	3837	1139	1721	1158	1747
bcspwr09	3	1723	4117	1241	1910	1247	1918
bcspwr10	3	5300	13571	5016	7985	5027	7998
power	3	4941	6594	3293	4930	3353	5006
494_bus	3	494	1666	270	361	277	369
685_bus	4	685	1967	254	546	255	557
1138_bus	3	1138	2596	652	963	671	991
qh768	5	768	2934	316	621	400	779
qh882	4	882	3354	84	222	100	254

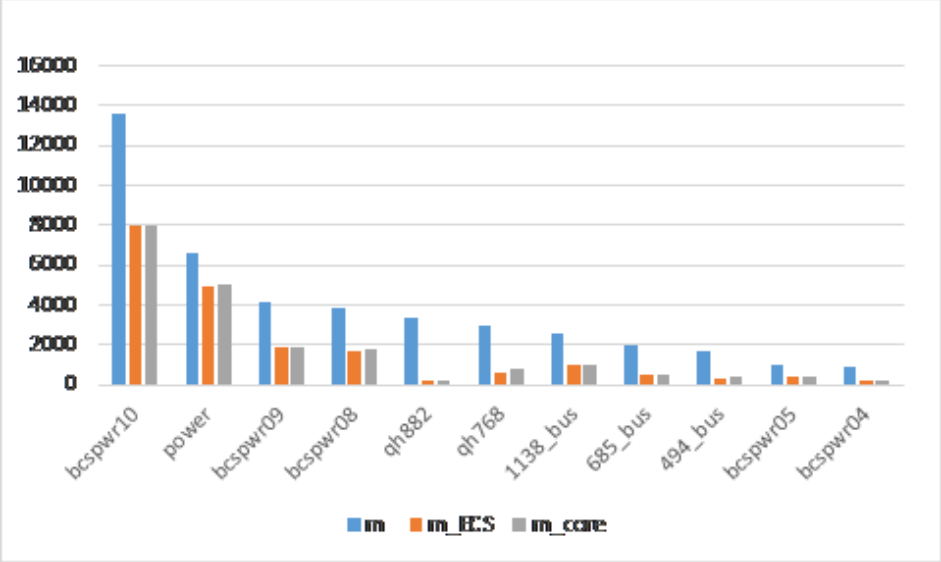

Figure 4.2: Scale reduction on the number of edges.

Moreover, the scale reduction effect differs. On the one hand, the reduction percentage differs for different graphs. For example, for the graph qh_882, around 90% of vertices of G are deleted, while only less than 5% of vertices of graph bcspwr10 are removed. On the other hand, the reduction scale are not the same for the same graph in $|V|$ and $|E|$. However, we could observe the similar trend for the scale reduction using the k-core method. Hence, both situations are normal

and may be caused by some other natural factors like structure, density, etc.

4.1.1 Comparison with k-Core

Figure 4.3 presents the difference between the size after the peeling procedure by k-ECS and k-core. This figure shows that the scale reduction based on the k-ECS method does improve the performance of reducing the size of graphs compared to k-core.

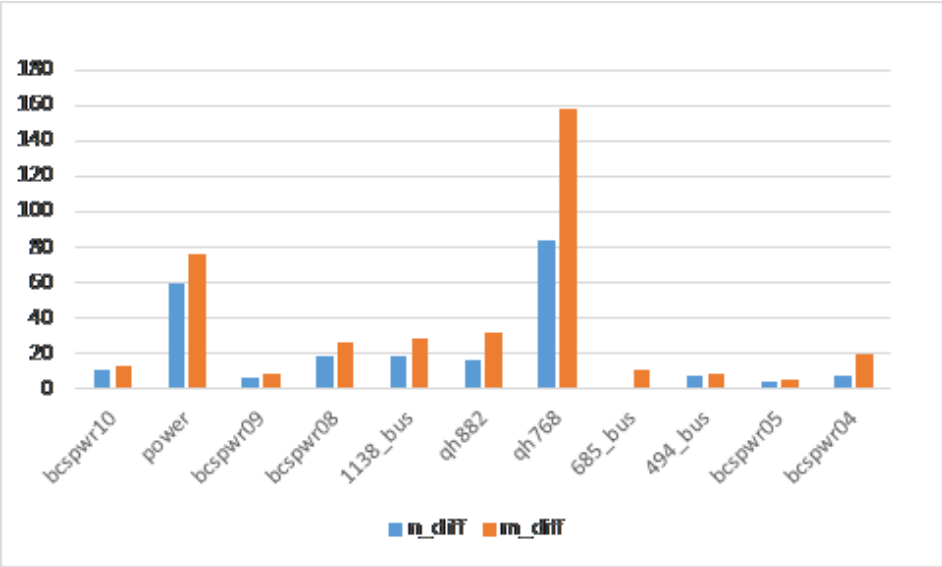

Figure 4.3: Difference between the number of vertices and edges left from scale reduction.

4.2 Connected Components

Table 4.3 summarizes the values of ω_{lower} and the number of k-edge-connected components for each graph, where $k = \omega_{lower} - 1$. Table 4.4 lists the number of vertices and edges in each k-edge-connected component in the graph after peeling with the corresponding k value.

Evidently, most of the graphs that have more than one k-edge-connected components have $\omega_{lower} > 3$. This is reasonable because we do the scale reduction based on $k = \omega_{lower} - 1$.

Table 4.3: Summary of ω_{lower} values and the number of k-ECS components, where $k = \omega_{lower} - 1$.

Graph Name	ω_{lower}	#k-ECS Components	Running Time (CPU seconds)
bcspwr04	6	2	0.079
bcspwr05	3	1	0.177
bcspwr08	3	1	4.078
bcspwr09	3	1	4.663
bcspwr10	3	1	127.487
power	3	1	100.614
494_bus	3	1	0.213
685_bus	4	6	0.448
1138_bus	3	2	1.482
qh768	5	2	0.532
qh882	4	6	0.751

Table 4.4: List of connected components after scale reduction with ($\omega_{lower} - 1$)-ECS.

	G'_{ECS}		Connected Components (n, m)					
Graph	n	m	#1	#2	#3	#4	#5	#6
bcspwr04	66	239	(12, 41)	(54, 198)				
bcspwr05	309	455	(309, 455)					
bcspwr08	1139	1721	(1139, 1721)					
bcspwr09	1241	1910	(1241,1910)					
bcspwr10	5016	7985	(5016, 7985)					
power	3293	4930	(3293, 4930)					
494_bus	270	361	(270,361)					
685_bus	254	546	(81, 279)	(29, 63)	(34, 85)	(62,137)	(23,42)	(25, 40)
1138_bus	652	963	(648,959)	(4, 4)				
qh768	316	621	(282 , 564)	(34, 57)				
qh882	84	222	(14,37)	(14,37)	(14,37)	(14,37)	(14,37)	(14,37)

Table 4.5: Scale reduction effect of k-ECS.

		G		G'_{ECS}		Diff	
Graph	ω_{lower}	n	m	n	m	n	m
bcspwr04	6	274	943	66	239	208	704
bcspwr05	3	443	1033	309	405	134	578
bcspwr08	3	1624	3837	1139	1721	485	2116
bcspwr09	3	1723	4117	1241	1910	482	2207
bcspwr10	3	5300	13571	5016	7985	284	5586
power	3	4941	6594	3293	4930	1684	1664
494_bus	3	494	1666	270	361	224	1305
685_bus	4	685	1967	254	546	431	1421
1138_bus	3	1138	2596	652	963	486	1633
qh768	5	768	2934	316	621	798	3132
qh882	4	882	3354	84	222	452	2313

5. DISCUSSIONS AND CONCLUSIONS

The proposed new scale reduction approach based on the k-edge-connected sets presents improvement in performance of scale reduction effect compared with the previous one based on k-core method. Though the improvement effect varies in different cases, the similar trend discloses that the variations may depend on the structure or some other factors of the graph. While this new scale reduction approach requires longer computational time compared with previous one based on k-core method.

Considering the existing possibilities of improving the proposed approach, as well as the parallelization, we could deal with the setback of larger time complexity in scale reduction with k-edge-connected sets. Several further ideas could be used for optimizing the proposed algorithm for finding the maximum clique in large sparse graphs. On the one hand, we could reduce the time complexity of the algorithm for building the auxiliary graph. This task could be accomplished in at least two ways. One of them is replacing Edmonds-Karp's max-flow function with time complexity of $O(nm^2)$ with a more up-to-date and faster max-flow function. The other way is parallelizing the implementation. For example, Haakonsen [28] proposed a parallel version of the construction procedure and achieved an obvious improvement in the running time.

Thus, though we have shown the improvement of performance of this approach compared to the previous ones, future research could yield further improvements.

REFERENCES

[1] J. Pattillo, N. Youssef, and S. Butenko, "On clique relaxation models in network analysis," *European Journal of Operational Research*, vol. 226, pp. 9–18, 2013.

[2] R. D. Luce and A. D. Perry, "A method of matrix analysis of group structure," *Psychometrika*, vol. 14, pp. 95–116, 1949.

[3] D. Urban and T. Keitt, "Landscape connectivity: a graph-theoretic perspective," *Ecology*, vol. 82, pp. 1205–1218, 2001.

[4] A. Laita, J. S. Kotiaho, and M. Monkkonen, "Graph-theoretic connectivity measures: what do they tell us about connectivity," *Landscape Ecol*, vol. 26, pp. 951–967, 2011.

[5] S. B. Seidman, "Network structure and minimum degree," *Social Networks*, vol. 5, pp. 269 – 287, 1983.

[6] C. Jordan, "Sur les assemblages de lignes," *Journal fÃijr die reine und angewandte Mathematik (In French)*, vol. 2, pp. 185–190, 1869.

[7] R. E. Tarjan, "A note on finding the bridges of a graph," *Information Processing Letters*, vol. 2, pp. 160–161, 1974.

[8] S. Even and R. E. Tarjan, "Network flow and testing graph connectivity," *SIAM Journal of Computing,*, vol. 4, pp. 507–518, 1975.

[9] A. H. Esfahanian and S. L.Hakimi, "On computing the connectivities of graphs and digraphs," *NETWORKS*, vol. 14, pp. 355–366, 1984.

[10] D. W. Matula, "Determining edge connectivity in o(mn)," *Proceedings of 28th Symp. On Foundations of Computer Science*, pp. 249–251, 1987.

[11] H. Nagamochi and T. Watanabe, "Computing k-edge-connectee components of a multigraphs," *Trans. IEICE of Japan*, vol. 4, pp. 513–517, 1993.

[12] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang, "Efficiently computing k-edge connected components via graph decomposition," *In Proc. Of SIGMOD*, pp. 205–216., 2013.

[13] R. E. Gomory and T. C. Hu, "Multi-terminal network flows," *J. Soc. Indust, Appl. Math*, vol. 9, pp. 551–570, 1961.

[14] T. Wang, Y. Zhang, F. Y. Chin, H. F. Ting, Y. H. Tsin, and S. H. Poon, "A simple algorithm for finding all k-edge-connected components," *PLoS ONE*, vol. 10, pp. 1–10., 2015.

[15] R. M. Karp, "Reducibility among combinatorial problems," *Complexity of Computer Computations*, no. R. E. Miller and J. W. Thatcher, Eds., pp. 85–103, 1972.

[16] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo, "The maximum clique problem," in *Handbook of Combinatorial Optimization* (D.-Z. Du and P. M. Pardalos, eds.), (Dordrecht, The Netherlands), pp. 1–74, Kluwer Academic Publishers, 1999.

[17] R. Carraghan and P. Pardalos, "An exact algorithm for the maximum clique problem," *Operations Research Letters*, vol. 9, pp. 375–382, 1990.

[18] L. Babel and C. Tinhofer, "A branch and bound algorithm for the maximum clique problem," *Methods and Models of Operational Research*, vol. 34, pp. 207–217, 1990.

[19] P. R. J. Östergård, "A fast algorithm for the maximum clique problem," *Discrete Applied Mathematics*, vol. 120, pp. 197–207, 2002.

[20] E. Tomita and T. Seki, "An efficient branch-and-bound algorithm for finding the maximum clique," *Lecture Notes in Computer Science*, vol. 2371, pp. 278–289, 2003.

[21] E. Tomita and T. Kameda, "An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments," *Journal of Global Optimization*, vol. 37, pp. 95–111, , 2007.

[22] E. Tomita and T. Seki, "An efficient branch-and-bound algorithm for finding a maximum clique," in *Proceedings of the 4th International Conference on Discrete Mathematics and Theoretical Computer Science*, DMTCS'03, pp. 278–289, 2003.

[23] S. Busygin, "A new trust region technique for the maximum weight problem," *Discrete Applied Mathematics*, vol. 154, pp. 2080–2096, 2006.

[24] F. D. Croce, A. Grosso, and M. Locatelli, "Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem," *Journal of Heuristics*, vol. 10, pp. 135–152, 2004.

[25] O. Yezerska and S. Butenko, "The maximum clique and vertex coloring," in *Handbook of Heuristics* (R. Martí, P. M. Pardalos, and M. G. C. Resende, eds.), Springer, 2018. DOI: 10.1007/978-3-319-07153-4_47-1.

[26] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *J. ACM*, vol. 35, pp. 921–940, Oct. 1988.

[27] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, pp. 248–264, Apr. 1972.

[28] J. Haakonsen, "Parallel algorithms for computing k-connectivity.," *University of Bergen*, 2017.

[29] J. Abello, P. Pardalos, and M. Resende, "On maximum clique problems in very large graphs," in *External Memory Algorithms and Visualization* (J. Abello and J. Vitter, eds.), pp. 119–130, Boston: American Mathematical Society, 1999.

[30] A. Verma, A. Buchanan, and S. Butenko, "Solving the maximum clique and vertex coloring problems on very large sparse networks," *INFORMS Journal on Computing*, vol. 27, pp. 164–177, 2015.